Monte-Carlo Tree Search in Continuous Action Spaces with Value Gradients

Jongmin Lee,[1] **Wonseok Jeon,**[3,4] **Geon-Hyeong Kim,**[1] **Kee-Eung Kim**[1,2]
[1]School of Computing, KAIST, Daejeon, Republic of Korea
[2]Graduate School of AI, KAIST, Daejeon, Republic of Korea
[3]Mila, Quebec AI Institute, Montreal, Canada
[4]McGill University, Montreal, Canada
jmlee@ai.kaist.ac.kr, jeonwons@mila.quebec, ghkim@ai.kaist.ac.kr, kekim@kaist.ac.kr

Abstract

Monte-Carlo Tree Search (MCTS) is the state-of-the-art online planning algorithm for large problems with discrete action spaces. However, many real-world problems involve continuous action spaces, where MCTS is not as effective as in discrete action spaces. This is mainly due to common practices such as coarse discretization of the entire action space and failure to exploit local smoothness. In this paper, we introduce Value-Gradient UCT (VG-UCT), which combines traditional MCTS with gradient-based optimization of action particles. VG-UCT simultaneously performs a global search via UCT with respect to the finitely sampled set of actions and performs a local improvement via action value gradients. In the experiments, we demonstrate that our approach outperforms existing MCTS methods and other strong baseline algorithms for continuous action spaces.

1 Introduction

Many real-world problems require planning to select actions sequentially in continuous action spaces. For example, robotic manipulation is achieved by the torque command sequence, playing billiards requires computing precise force and angle of the cue, and aerospace navigation needs to choose target velocities. Physical systems often involve differential equations for their dynamics that are differentiable with respect to states and actions.

Monte-Carlo Tree Search (MCTS) (Kocsis and Szepesvári 2006; Coulom 2006; Browne et al. 2012) is a highly effective online *closed-loop* planning algorithm for large search space problems and has shown great promise especially when the action space is discrete such as playing board games (Silver et al. 2016; 2017), real-time games (Balla and Fern 2009; Pepels, Winands, and Lanctot 2014) and combinatorial optimization (Sabharwal, Samulowitz, and Reddy 2012). MCTS performs quite well with a black-box simulator, without any heuristic function. However, it is generally not a method of preferred choice when it comes to planning in continuous actions. This is mainly because the number of possible actions is infinite, thus coarse-grained discretization of the action space is unavoidable for a straightforward tree search. Taking only rough actions can render the search useless in many real-world problems that require precise control.

Existing MCTS methods deal with continuous actions in two main ways. Progressive widening (Coulom 2007; Couëtoux et al. 2011a), also known as progressive unpruning (Chaslot et al. 2008), gradually increases the number of available actions at each node based on the visitation count. The other approach is hierarchical optimistic optimization (HOO) (Bubeck et al. 2009; Mansley, Weinstein, and Littman 2011), which splits the action space and gradually increases the resolution of actions. While these approaches may show sufficient performance for selecting moderately good actions, they require a huge number of samples to achieve highly accurate controls. In contrast, gradient-based optimization is very effective for finding exact local optima but cannot be made reachable to global optima without careful initialization.

In this paper, we present Value-Gradient UCT (VG-UCT), an MCTS algorithm equipped with the gradient-based fine-tuning of the finite set of action samples. The algorithm jointly performs global (but coarse-grained) search via MCTS and local (but fine-grained) search via action-value gradient ascent. Our algorithm is based on the observation that many real-world problems with continuous action space have locally differentiable dynamics (with respect to state and action) thus make gradient ascent highly effective.

In addition, it has been shown that the domain-specific knowledge can be leveraged to significantly improve the performance of MCTS (Enzenberger et al. 2010; Arrington, Langley, and Bogaerts 2016). One of the promising algorithms is kernel-regression UCT (KR-UCT) (Yee, Lisy, and Bowling 2016; Lee et al. 2018), which has been successfully applied to challenging continuous action problems such as curling. KR-UCT can be regarded as leveraging the smoothness in continuous dynamical systems, i.e. similar actions have similar values thus it is a good idea to share information of values between actions. In VG-KR-UCT, we further incorporated our idea of using the first-order (i.e. gradient) information of values to boost the performance of KR-UCT.

2 Background

Notations and Basic Formulation

We consider continuous MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r \rangle$, where $\mathcal{S} \subset \mathbb{R}^n$ is a set of states s, $\mathcal{A} \subset \mathbb{R}^m$ is a set of actions a, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Pr(\mathcal{S})$ is a transition function, and $r(s, a) \in \mathbb{R}$ is the immediate reward for taking action a in state s. We consider the transition $s' \sim \mathcal{P}(\cdot|s, a)$ whose samples are generated by a deterministic function ρ of an input noise variable ξ and conditioning variables s, a: $s' = \rho(s, a, \xi)$ where $\xi \sim p(\xi)$ with a fixed noise distribution. A policy $\pi_t : \mathcal{S} \to \mathcal{A}$ is a mapping from state to action at time step t. The goal is to maximize the expected cumulative rewards $\mathbb{E}\left[\sum_{t=0}^{\infty} r(s_t, a_t)\right]$, which is assumed to be finite for all policies. We also assume the differentiable transition and reward function.

Online planning methods compute a policy for a limited horizon T, treating the current state as the initial state s_0. Specifically, they solve the following optimization problem within the limited computational budget to obtain optimal sequence of per-step policies $\pi_0^*, \ldots, \pi_T^*$:

$$\underset{\pi_0,\ldots\pi_T}{\arg\max}\ V(s_0) = \mathbb{E}_{\mathcal{P}}\left[\sum_{t=0}^{T} r(s_t, a_t) \mid s_0\right] \qquad (1)$$
$$\text{s.t. } a_t = \pi_t(s_t) \quad \forall t = 0, \ldots, T$$

After solving Eq. (1), the first action $a_0^* = \pi_0^*(s_0)$ is executed, and then re-planning is performed at the next time step. The planning horizon T trades off between searching for high-quality long-term policies and computation time required to solve the optimization problem (Jiang et al. 2015). We assume a *black-box* forward simulator $\mathcal{G}$ that generates samples of the next state s' and reward r, given the current state s, action a, and noise ξ to be used with the planner, i.e. $(s', r) = \mathcal{G}(s, a, \xi)$ where $\xi \sim p(\xi)$ is a known noise density function. We use a notation $x_{i:j}$ to denote a list of $\{x_i, x_{i+1}, \ldots, x_{j-1}, x_j\}$.

Monte-Carlo Tree Search

Monte-Carlo tree search (MCTS) (Browne et al. 2012) is a generic online closed-loop planning algorithm that combines random sampling and tree search. Starting from an empty tree, it iteratively performs finite-horizon simulations from the current state and selects a leaf node for expansion. UCT (Kocsis and Szepesvári 2006) is an MCTS algorithm that adopts UCB1 (Auer, Cesa-Bianchi, and Fischer 2002) as the action selection rule at each internal *decision node* ν^D *(associated to state)* of the search tree:

$$\underset{\nu^C \in \text{CHILDREN}(\nu^D)}{\arg\max}\ Q(\nu^D, \nu^C) + c\sqrt{\frac{\log N(\nu^D)}{N(\nu^D, \nu^C)}} \qquad (2)$$

where $\text{CHILDREN}(\nu^D)$ is a set of child nodes of ν^D (or equivalently a set of possible actions at the current state), $Q(\nu^D, \nu^C)$ is the average of the sampled rewards when the child *chance node* ν^C *(associated to action)* is selected in ν^D, $N(\nu^D)$ is the number of simulations performed through ν^D, $N(\nu^D, \nu^C)$ is the number of times ν^C is selected in ν^D, c is the exploration constant to adjust the exploration-exploitation trade-off. After ν^C is selected using Eq. (2), the state transition occurs by taking the action stored in ν^C (i.e. $s' = \rho(s, \text{ACTION}(\nu^C), \xi)$ where $\xi \sim p(\xi)$). Then, the simulation continues with the child node $\nu^{D\prime}$, which corresponds to the sampled noise ξ (or equivalently the sampled state s'). UCT expands the search tree non-uniformly, focusing more search efforts into promising nodes. UCT solves Eq. (1) in an anytime manner, and it can be formally shown that UCT chooses an optimal action at the root node $a_0^* = \pi_0^*(s_0)$ with probability 1 as the number of simulations goes to infinity in *finite* MDPs.

Progressive Widening

In *continuous* MDPs, the vanilla UCT no longer works since each action should be tried at least once but there are infinitely many actions. One of the widely adopted approaches to the problem is *progressive widening* (Coulom 2007; Chaslot et al. 2008), which maintains a finite list of chance nodes (or equivalently list of actions) to be searched and incrementally adds a new child chance node ν^C to the list based on the visitation counts. Specifically, a new child node is added every time the following is satisfied:

$$\lfloor N(\nu^D)^\alpha \rfloor \geq |\text{CHILDREN}(\nu^D)| \qquad (3)$$

where $\alpha \in (0, 1)$ is a parameter that controls the (sublinear) growth rate, which enables accumulate enough information for the best action on the list. As a new node ν^C is created, a new action particle a to be stored in ν^C is either sampled from a probability distribution $a \sim \pi_{\text{sampler}}(\cdot|s)$ or generated deterministically (e.g. in the direction of increasing the coverage of action space). The newly generated action particle a is then stored in $\text{ACTION}(\nu^C)$ (and in $\text{INIT-ACTION}(\nu^C)$ too). However, this approach still suffers from coarse-grained discretization and requires a vast amount of action particles to precisely select optimal actions. In the continuous and stochastic environment, there are also infinitely many possible transitions. In order to make MCTS be consistent, we also gradually add a new child decision node $\nu^{D\prime}$ to $\text{CHILDREN}(\nu^C)$ when the following inequality is satisfied, which is *double progressive widening* (Couëtoux et al. 2011a) technique:

$$\lfloor N(\nu^D, \nu^C)^\beta \rfloor \geq |\text{CHILDREN}(\nu^C)| \qquad (4)$$

where $\beta \in (0, 1)$ is a (sublinear) growth rate parameter. As a new decision node $\nu^{D\prime}$ is created, a new noise is sampled from the noise distribution $\xi \sim p(\xi)$ and is stored to $\nu^{D\prime}$. Then, the next state can always be obtained deterministically from this noise $s' = \rho(s, a, \xi)$. If the condition (4) is not met, the least visited decision node is traversed (Auger, Couëtoux, and Teytaud 2013) or any decision node can be selected uniformly. From now, we will refer to $\hat{p}(\xi)$ as a noise sampling distribution, which follows $p(\xi)$ if the condition (4) is satisfied or selects an existing noise in $\nu^{D\prime} \in \text{CHILDREN}(\nu^C)$ by the predefined rule otherwise.

Other Related Work

There is a significant body of important work on extending MCTS to continuous action spaces.

HOOT (Mansley, Weinstein, and Littman 2011) replaces the UCB1 action selection rule by HOO (Bubeck et al. 2009) in UCT to deal with continuous actions in discrete state environments. HOLOP (Weinstein and Littman 2012) takes an alternative approach of leveraging HOO by representing the entire planning problem as a continuous bandit problem, where actions correspond to plans.

The progressive widening approach has been extended to stochastic continuous state and action planning problems through double progressive widening (Couëtoux et al. 2011a), where the progressive widening is applied to states as well as actions. In order to improve the performance of MCTS with progressive widening, cRAVE (Couëtoux et al. 2011b) adopts the RAVE heuristic (Gelly and Silver 2011) using the Gaussian convolution, encouraging information sharing among actions in the entire subtree. KR-UCT (Yee, Lisy, and Bowling 2016) also encourages information sharing between actions in the same node through kernel regression, where new action generation is guided by kernel density estimation. KR-UCT has outperformed cRAVE in the simulated curling domain and is regarded as the state-of-the-art.

Another simple online planning algorithm for continuous control is Random Shooting (Nagabandi et al. 2018). It is a simple sampling-based planning method that generates multiple action sequences, computes the rewards for each trajectory and chooses the first action of the trajectory with the highest cumulative rewards. Cross-entropy method (CEM) (Rubinstein and Kroese 2004; Weinstein and Littman 2013) is also popular method for online planning. It iterates the procedure of sampling action sequences and fitting a new distribution (e.g. Gaussian) to top-K action sequences with the highest rewards alternatively.

3 VG-UCT for Continuous Actions

Many real-world problems with continuous state and action spaces have locally smooth dynamics. Nonetheless, to the best of our knowledge, using the first-order information (i.e. gradients) of the environment dynamics in MCTS has been mostly unexplored. In this section, we establish Value-Gradient UCT (VG-UCT), an MCTS algorithm that effectively uses this additional information. We first introduce the algorithm that uses the exact gradients of the dynamics with white-box model that can yield analytic derivatives, and then present its practical version that requires only black-box simulator $(s', r) = \mathcal{G}(s, a, \xi)$.

(Stochastic) Value Gradients

The goal of online planning is to directly optimize the sequence of policies at each time step $\pi_0, \ldots \pi_T$, rather than optimizing a single parameter θ of the parameterized policy π_θ. Suppose that the initial state s_0 and the sequence of policies $\pi_0, \ldots, \pi_T$ are given. Then, as described in Figure 4 in Appendix A, we can sequentially obtain rewards $r_{0:T}$ at every time step through intermediate states $s_{0:T}$ and actions $a_{0:T}$, which are computed by $a_t = \pi_t(s_t)$, $r_t = r(s_t, a_t)$, and $s_{t+1} = \rho(s_t, a_t, \xi_t)$ where $\xi_t \sim p(\xi)$. Then, differentiating the planning objective in Eq. (1) with respect to s_t and π_t yields the stochastic value-gradient (Heess et al. 2015):

$$\frac{\partial V(s_t, \pi_{t:T})}{\partial s_t} = \frac{\partial r_t}{\partial s_t} + \frac{\partial \pi_t}{\partial s_t}\frac{\partial r_t}{\partial \pi_t} + \mathbb{E}_{\xi_t \sim p(\xi)}\left[\left(\frac{\partial s_{t+1}}{\partial s_t} + \frac{\partial \pi_t}{\partial s_t}\frac{\partial s_{t+1}}{\partial \pi_t}\right)\frac{\partial V(s_{t+1}, \pi_{t+1:T})}{\partial s_{t+1}}\right] \quad (5)$$

$$\frac{\partial V(s_t, \pi_{t:T})}{\partial \pi_t} = \frac{\partial r_t}{\partial \pi_t} + \mathbb{E}_{\xi_t \sim p(\xi)}\left[\frac{\partial s_{t+1}}{\partial \pi_t}\frac{\partial V(s_{t+1}, \pi_{t+1:T})}{\partial s_{t+1}}\right] \quad (6)$$

for $t \leq T$. Therefore, if the gradients of environment dynamics, (i.e. $\frac{\partial r(s,a)}{\partial s}$, $\frac{\partial r(s,a)}{\partial a}$, $\frac{\partial \rho(s,a,\xi)}{\partial s}$, $\frac{\partial \rho(s,a,\xi)}{\partial a}$) are available, on the sampled trajectory $\tau = (s_0, a_0, \xi_0, s_1, a_1, \xi_1, \ldots, s_T)$ for any given policy $\pi_{0:T}$, Monte-Carlo estimates of the value gradients can be computed by the backward recursions, starting from $v^a_{T+1} = 0$ and $v^s_{T+1} = 0$:

$$v^s_t = \left[\frac{\partial r(s,a)}{\partial s} + \frac{\partial \pi(s)}{\partial s}\frac{\partial r(s,a)}{\partial a} + \left(\frac{\partial \rho(s,a,\xi)}{\partial s} + \frac{\partial \pi(s)}{\partial s}\frac{\rho(s,a,\xi)}{\partial a}\right)v^s_{t+1}\right]\Bigg|_{s=s_t, a=a_t, \xi=\xi_t} \quad (7)$$

$$v^a_t = \left[\frac{\partial r(s,a)}{\partial a} + \frac{\partial \rho(s,a,\xi)}{\partial a}v^s_{t+1}\right]\Bigg|_{s=s_t, a=a_t, \xi=\xi_t} \quad (8)$$

Remark. *If π_t is not a function that outputs a transformation of input s_t (e.g. MCTS or methods that directly optimizes the sequence of actions $a_{0:T}$), then $\frac{\partial \pi_t}{\partial s_t} = 0$.*

We can directly use these estimates of the value gradients $[v^a_0, \ldots, v^a_T]$ to locally improve the given policies $\pi_{0:T}$:

$$\pi_t(s_t) \leftarrow \pi_t(s_t) + \eta v^a_t \text{ for } t = 0, \ldots, T \quad (9)$$

where η is a step-size for the stochastic gradient ascent.

Value-Gradient UCT (VG-UCT)

UCT is an anytime planning algorithm that performs combinatorial optimization with respect to $\pi_0, \ldots, \pi_T$ in finite set of actions to maximize the expected sum of rewards $\mathbb{E}[\sum_{t=0}^{T} r(s_t, a_t)]$. It samples a trajectory $\tau = (s_0, a_0, \xi_0, s_1, a_1, \xi_1, \ldots, s_T)$ via UCB1 inside the tree or via rollout policy outside the tree at every simulation: starting from the root node ν^D_0 and initial state s_0:

(Tree policy: inside the tree) (10)

$$\nu^C_t = \arg\max_{\nu^C_t \in \text{CHILDREN}(\nu^D)}\left[Q(\nu^D_t, \nu^C_t) + c\sqrt{\frac{\log N(\nu^D)}{N(\nu^D, \nu^C)}}\right]$$

$$a_t = \text{ACTION}(\nu^C_t),\ \ \xi_t \sim \hat{p}(\xi),\ \ s_{t+1} = \rho(s_t, a_t, \xi_t)$$

$$\nu^D_{t+1} = (\text{a child of } \nu^C_t \text{ that corresponds to } \xi_t)$$

(Rollout policy: outside the tree)

$$a_t \sim \pi_{\text{rollout}}(\cdot|s_t),\ \ \xi_t \sim p(\xi),\ \ s_{t+1} = \rho(s_t, a_t, \xi_t)$$

UCT asymptotically converges to the globally optimal sequence of policies $\pi^*_{0:T}$ when the state and action space are *finite*. The main idea behind our proposed method, Value-Gradient UCT (VG-UCT), is to combine UCT's global search with respect to the (coarse-grained) discretized actions and local search via value-gradient ascent for fine-tuning. VG-UCT essentially consists of the following three steps in each simulation:

Algorithm 1 Value-Gradient UCT (VG-UCT)

```
function SEARCH($s_0$)
    Create a root node $\nu^D$.
    repeat
        SIMULATE($s_0$, $\nu^D$, 0)
    until TIMEOUT()
    $\nu^C \leftarrow \arg\max_{\nu^C \in \text{CHILDREN}(\nu^D)} Q(\nu^D, \nu^C)$
    $a^* \leftarrow \text{ACTION}(\nu^C)$
    return $a^*$
end function

function SIMULATE($s$, $\nu^D$, $t$)
    if $t =$ (planning horizon $T$) then
        return $[0, \emptyset, \emptyset]$
    end if
    $[\nu^C, a, \text{rollout}] \leftarrow$SELECTACTION($s$, $\nu^D$, $c$)
    $[\nu^{D\prime}, \xi] \leftarrow$SAMPLENOISE($\nu^D$, $\nu^C$)
    $[r, s'] \leftarrow \mathcal{G}(s, a, \xi)$
    if rollout then
        $[R', \{a_{t+1:T}\}, \{\xi_{t+1:T}\}] \leftarrow$ROLLOUT($s'$, $t+1$)
    else
        $[R', \{a_{t+1:T}\}, \{\xi_{t+1:T}\}] \leftarrow$SIMULATE($s'$, $\nu^{D\prime}$, $t+1$)
    end if
    $R \leftarrow r + R'$
    $N(\nu^D) \leftarrow N(\nu^D) + 1$
    $N(\nu^D, \nu^C) \leftarrow N(\nu^D, \nu^C) + 1$
    $Q(\nu^D, \nu^C) \leftarrow Q(\nu^D, \nu^C) + \frac{R - Q(\nu^D, \nu^C)}{N(\nu^D, \nu^C)}$
    // After simulation, perform value-gradient ascent.
    for $j = 1, \ldots, m$ do
        $\tilde{a}^j \leftarrow a + \epsilon e_j$
        $[g]_j = (\text{RETURN}(s, \{\tilde{a}^j, a_{t+1:T}\}, \{\xi, \xi_{t+1:T}\}) - R)/\epsilon$ (Eq. (12))
    end for
    $\text{ACTION}(\nu^C) \leftarrow \text{ACTION}(\nu^C) + \eta g$ // $a$ remains same.
    $\text{ACTION}(\nu^C)$ is clipped within:
            $\{\hat{a} \in \mathcal{A} : \|\hat{a} - \text{INIT-ACTION}(\nu^C)\| \leq \Delta\}$
    return $[R, \{a, a_{t+1:T}\}, \{\xi, \xi_{t+1:T}\}]$
end function

function SELECTACTION($s$, $\nu^D$, $c$)
    if $\lfloor N(\nu^D)^\alpha \rfloor \geq |\text{CHILDREN}(\nu^D)|$ then
        Create a new chance node $\nu^C$
        Add $\nu^C$ to $\text{CHILDREN}(\nu^D)$
        $a \sim \pi_{\text{sampler}}(\cdot|s)$
        $\text{ACTION}(\nu^C) \leftarrow a$, $\text{INIT-ACTION}(\nu^C) \leftarrow a$
        $N(\nu^D, \nu^C) \leftarrow 0$, $Q(\nu^D, \nu^C) \leftarrow 0$
        rollout $\leftarrow$ true
    else
        $\nu^C \leftarrow \arg\max_{\nu^C \in \text{CHILDREN}(\nu^D)} Q(\nu^D, \nu^C) + c\sqrt{\frac{\log N(\nu^D)}{N(\nu^D, \nu^C)}}$
        $a \leftarrow \text{ACTION}(\nu^C)$
        rollout $\leftarrow$ false
    end if
    return $[\nu^C, a, \text{rollout}]$
end function

function SAMPLENOISE($\nu^D$, $\nu^C$)
    if $\lfloor N(\nu^D, \nu^C) \rfloor^\beta \geq |\text{CHILDREN}(\nu^C)|$ then
        Create a new decision node $\nu^{D\prime}$
        Add $\nu^{D\prime}$ to $\text{CHILDREN}(\nu^C)$
        $\text{NOISE}(\nu^{D\prime}) \leftarrow p(\xi)$ and $N(\nu^{D\prime}) \leftarrow 0$
    else
        $\nu^{D\prime} \leftarrow$ (least visited node in $\text{CHILDREN}(\nu^C)$)
    end if
    $\xi \leftarrow \text{NOISE}(\nu^{D\prime})$
    return $[\nu^{D\prime}, \xi]$
end function

function ROLLOUT($s$, $t$)
    if $t =$ (planning horizon $T$) then
        return $[0, \emptyset, \emptyset]$
    end if
    $a \sim \pi_{\text{rollout}}(\cdot|s)$ and $\xi \sim p(\xi)$
    $[r, s'] \leftarrow \mathcal{G}(s, a, \xi)$
    $[R', \{a_{t+1:T}\}, \{\xi_{t+1:T}\}] \leftarrow$ROLLOUT($s'$, $t+1$)
    $R \leftarrow r + R'$
    return $[R, \{a, a_{t+1:T}\}, \{\xi, \xi_{t+1:T}\}]$
end function
```

1. Sample a trajectory $\tau = (s_0, a_0, \xi_0, s_1, a_1, \xi_1, \ldots, s_T)$ via UCT. (Eq. (10))
2. Compute Monte-Carlo estimate of the value gradients $v^a_{0:T}$ w.r.t. the sampled τ. (Eq. 7-8)
3. Update $\pi_{0:T}$ via gradient ascent: $\text{ACTION}(\nu^C_t) \leftarrow \text{ACTION}(\nu^C_t) + \eta v^a_t, \forall t$ (Eq. (9))

The first UCT step generates a tree-guided trajectory while exploring the combinatorial space of finite (coarse-grained) actions. Using the sampled trajectory, we then locally fine-tunes the trajectory-generating policies π_t (i.e. the actual action $\text{ACTION}(\nu^C_t)$ stored at each node ν^C_t) in the direction of maximizing the expected cumulative rewards. By combining UCT search and gradient-based fine-tuning, we can expect to overcome both local optima problem in gradient-based method and coarse-grained solution problem in tree search.

However, care must be taken when we update $\text{ACTION}(\nu^C_t)$ via value-gradient ascent in the chance node since the slight perturbation of the action can have potentially large impact on the future trajectory in the subtree. If the influence by action perturbation is severely large, then there is a risk that information accumulated in the tree becomes invalid. Fortunately, we can show that this influence is bounded by the degree of action perturbation and the Lipschitz continuity of the environment dynamics.

Theorem 1. *Assume transition function is induced by a Lipschitz model class (Asadi, Misra, and Littman 2018) Φ_g, and reward function is Lipschitz continuous. If two Lipschitz policies $\pi_{0:T}$ and $\hat{\pi}_{0:T}$ are Δ-close, i.e. $\sup_{t,s} d(\pi_t(s), \hat{\pi}_t(s)) \leq \Delta$, then for all s_0, the gap between two value functions are bounded by Δ:*

$$|V(s_0, \pi_{0:T}) - V(s_0, \hat{\pi}_{0:T})| \leq L_v \Delta$$

where $L_v = L_r\left(1 + T + \frac{L_a(1+L_\pi)}{1-L_s}\left(T - \frac{L_s(1-L_s^T)}{1-L_s}\right)\right)$, *$L_s, L_a$ are Lipschitz constants for transition function, L_r is a Lipschitz constant for reward function, and L_π is a Lipschitz constant for policy.*

Proof. Details on the definitions, assumptions and the proofs are in Appendix B. □

By Theorem 1, we can guarantee that the degree of the change of average rewards to be stored in each node is bounded by $L_v\Delta$ when we confine the maximum perturbation of $\textsc{action}(\nu^C)$ from the initially generated value $\textsc{init-action}(\nu^C)$ by Δ. Therefore, we additionally introduce the following procedure for the stability of the tree search, which restricts actions to be improved only within the Δ-bounded region.

4. Clip $\textsc{action}(\nu_t^C)$ within $\{a \in \mathcal{A} : d\left(\textsc{init-action}(\nu_t^C), a\right) \leq \Delta\}\ \forall t$

Here, we can further show that under some assumptions detailed in Appendix C, VG-UCT asymptotically improves the best sequence of actions toward the maximally achievable rewards within Δ-bounded region, despite the possibly occasional negative updates (Theorem 2 in Appendix C).

VG-UCT with Finite Differences

The vanilla VG-UCT introduced in Section 3 requires additional $O(n^2+nm)$ computational time at each time-step for matrix-multiplication (Eq. (7-8)) in order to obtain value-gradients $v^a_{0:T}$. A more fundamental issue in VG-UCT is that it requires *white-box* model of the transition function ρ and reward function r to compute $\left[\frac{\partial\rho}{\partial s}, \frac{\partial\rho}{\partial a}, \frac{\partial r}{\partial s}, \frac{\partial r}{\partial a}\right]$, which is usually not accessible. In contrast, existing MCTS methods work only with the black-box simulator $(s', r) \sim \mathcal{G}(s, a)$.

One way to circumvent the need for a (gradient-yielding) white-box model is to use *finite differences*,

$$\left[\frac{\partial f(x)}{\partial x}\right]_j \approx \frac{f(x+\epsilon e_j) - f(x)}{\epsilon} \qquad (11)$$

where e_j is an one-hot vector whose j-th value is one, and ϵ is a small number, e.g. 10^{-7}. It only requires evaluating the function, thus we can approximate the Jacobians of the dynamics using a black-box forward simulator ρ and r on the sampled trajectory $\tau = (s_0, a_0, \xi_0, s_1, a_1, \xi_1, \ldots, s_T)$ using the finite differences. Then, we can apply approximate $\left[\frac{\partial\rho}{\partial s}, \frac{\partial\rho}{\partial a}, \frac{\partial r}{\partial s}, \frac{\partial r}{\partial a}\right]$ to Eq. (7-8) for computing value-gradients. However, while this approach is applicable, it requires $O(n^2+nm)$ times more queries to the black-box simulator at every time step. Given that the forward-simulation of physical simulators is usually the main bottleneck of the overall search procedure, this approach is infeasible for *online* planning in high-dimensional state space with a very limited search time.

Thus, we take instead a more direct approach to obtaining value gradients $[v_0^a, \ldots, v_T^a]$, which circumvents estimating Jacobian of the dynamics. Let $\textsc{return}(s_t, \{a_{t:T}\}, \{\xi_{t:T}\})$ be the sum of rewards with respect to the initial state s_t and a sequence of actions $\{a_{t:T}\}$ and noises $\{\xi_{t:T}\}$:

$$\textsc{return}(s_t, \{a_{t:T}\}, \{\xi_{t:T}\}) = \sum_{k=t}^{T} r(s_k, a_k)$$
$$\text{s.t. } s_{k+1} = \rho(s_k, a_k, \xi_k),$$

which is a deterministic function of the given inputs. Then, the value gradient can be computed by[1]

$$[\tilde{v}_t^a]_j = \Big(\textsc{return}(s_t, \{a_t + \epsilon e_j, a_{t+1:T}\}, \{\xi_{t:T}\}) - \textsc{return}(s_t, \{a_{t:T}\}, \{\xi_{t:T}\})\Big)/\epsilon \qquad (12)$$

for all $j = 1, \ldots, m$, where $\tilde{v}_t^a$ becomes equivalent to v_t^a in Eq. (8) as $\epsilon \to 0$.

Complexity The additional time complexity and the number of additional queries to the simulator of Eq. (12) is now $O\big((T-t)m\big)$. Here, note that $\tilde{v}_t^a$ does not depend on the past or future value-gradients unlike Eq. (8), thus their computations are fully parallelizable for each time step. When fully parallelized, the overall complexity of Eq. (12) is $O\big(Tm\big)$ whereas Eq. (7-8) requires $O\big(T(n^2+nm)\big)$. The final pseudo-code of VG-UCT using Eq. (12) is described in Algorithm 1, where the improvements to UCT (with progressive widening) are highlighted as blue.

4 Experiments

In this section, we compare the performance of VG-UCT with those of representative set of other online planning algorithms for continuous action spaces: (1) UCT with progressive widening (UCT), (2) Kernel Regression UCT (KR-UCT) (Yee, Lisy, and Bowling 2016), (3) Grad-MPC (Camacho and Alba 2013), a simple gradient-based MPC that randomly initializes actions $a_{0:T}$ and then updates them via value-gradient ascent using Eq. (12), (4) Cross-Entropy Method (CEM) (Weinstein and Littman 2013), (5) Uniform Random Shooting (Uniform-RS) (Nagabandi et al. 2018), (6) Hierarchical Open-Loop Optimistic Planning (HOLOP) (Weinstein and Littman 2012). For UCT and VG-UCT, the new actions are sampled uniformly from the entire action space when performing progressive widening, unless otherwise noted. Detailed experimental settings are provided in Appendix D.

Illustrative 2D Example

We first show how the combination of UCT search (global but coarse-grained) and gradient-based optimization (local but fine-grained) can effectively work together in a smooth dynamical system. We use an illustrative example shown in Figure 1. This is a three-step planning problem, where the initial state is $s_0 = (1, 1)$, the action space is $\mathcal{A} = [0, 2] \times [0, 2] \subset \mathbb{R}^2$, and $\rho(s, a, \xi) = s + a + \xi$ where $\xi \sim \mathcal{N}(0, (0.03)^2 I)$. As described in Figure 1, in order to achieve the maximal reward, the agent should reach the goal at $(5, 5)$ colored in red very precisely, while avoiding the negative reward regions colored in blue.

As can be seen in Figure 1, the VG-UCT agent was the only one that was able to arrive at the goal very accurately

[1]Here, the central difference is also applicable, but it requires more queries to the simulator than the forward difference. In Eq. (12), $\textsc{return}(s_t, \{a_{t:T}\}, \{\xi_{t:T}\})$ is obtained during UCT rollout, and forward difference can reuse it. We tested the the central difference too, but the results were almost identical to those of forward difference under the same number of simulation (but twice the wall clock time).

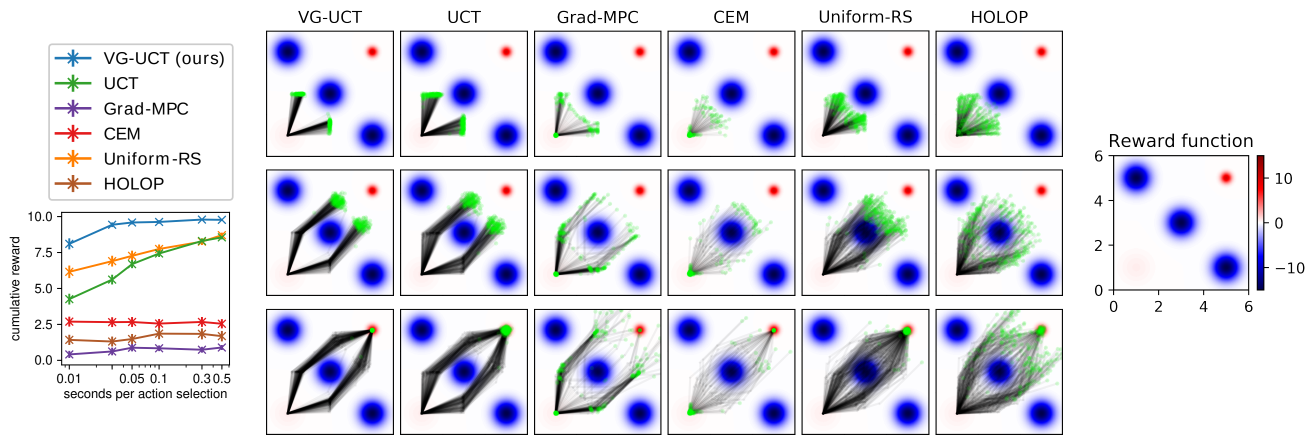

Figure 1: An illustrative example that requires planning to explore beyond blue regions and choose actions precisely. Each row shows the trajectory of the algorithms for each time step under the search time limit of 0.1 second. Each green dot represents the current position of each agent executed with different random seed, and each black line represents its trajectory. In this example, VG-UCT agent was most accurate in arriving at the goal location within the search time limit.

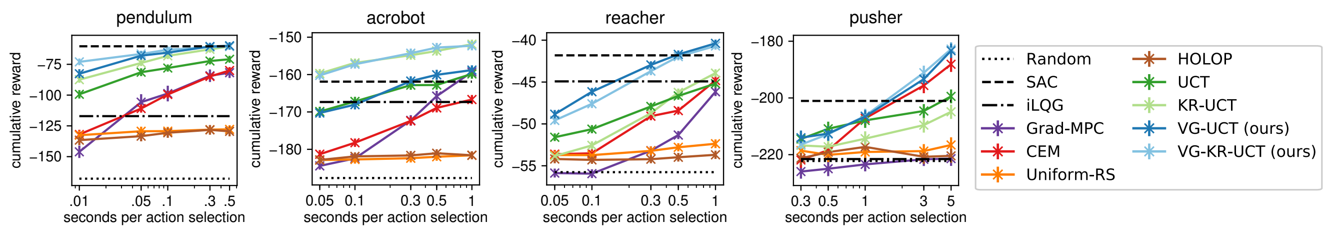

Figure 2: The results of four stochastic continuous control benchmarks. For each figure, the x-axis denotes the search time for action selection (seconds), and the y-axis represents the sum of rewards during 200 time steps for Pendulum and Acrobot, 50 steps for Reacher-v2, and 100 steps for Pusher-v2. The errorbars indicate $1.96 \times$ (standard error).

within the search time limit. UCT can only explore the (coarsely) discretized actions, thus it is very difficult to reach the goal precisely within short search time limit. Similarly, Uniform-RS had difficulty to reach the goal exactly since it uses sequences of randomly sampled actions. CEM often suffers from bad local minima if the sampled trajectories at the first iteration are not close enough to the goal. HOLOP generates plan by recursively splitting up the $(T \times m)$-dimensional solution space for $a_{0:T}$. In order to get sufficiently high resolution for the discretized space, it required exponentially many simulations with respect to $T \times m$. On the other hand, Grad-MPC failed to overcome the bad local optima. If the initial sequence of actions $a_{0:T}$ was not good enough, it tended to stay at s_0 or not improved at all due to lack of gradient signal almost everywhere. VG-UCT shows the best performance by choosing a roughly good sequence of actions through UCT and fine-tuning it through value gradients.

Continuous Control Benchmark Tasks

Realistic experiments were conducted on four *stochastic* variants of continuous control tasks: Pendulum (continuous action version), Acrobot (continuous action version), Reacher and Pusher from the OpenAI Gym environment (Brockman et al. 2016; Todorov, Erez, and Tassa 2012), ranging from low-dimensional control problem ($n = 2$, $m = 1$) to high-dimensional control problem ($n = 22$, $m = 7$). The reward functions of Pendulum and Acrobot were defined in the form of

$$r(s) = \exp(-\|\mathbf{x} - \mathbf{x}_{\text{goal}}\|^2/\sigma) - 1$$

as in (Deisenroth and Rasmussen 2011), where $\mathbf{x}$ is the current position of the tip, and $\mathbf{x}_{\text{goal}}$ is the position of the upright tip. In all tasks, we injected Gaussian white noise to the action and the state, before and after the deterministic forward simulation, in order to make the transition stochastic. Other settings regarding the dynamics remained the same as in OpenAI Gym.

In Figure 2, we summarize experimental results of our methods (VG-UCT, VG-KR-UCT) and the baseline algorithms [2]. VG-KR-UCT is our extended version of KR-UCT

[2]Though the main focus of the paper is *online planning* using a small amount of search time with a black-box simulator, we also report the result of Soft-Actor Critic (SAC) (Haarnoja et al. 2018) trained for a very long time, i.e. 10 million steps for relative performance comparison. We also report the results of a simple iLQG given 10 seconds of search time per decision, but note that it requires Jacibian/Hessian of the dynamics. Estimating Jacobian/Hessian via finite differences requires $O(n+m)/O(n^2 + nm + m^2)$ queries to the black-box simulator.

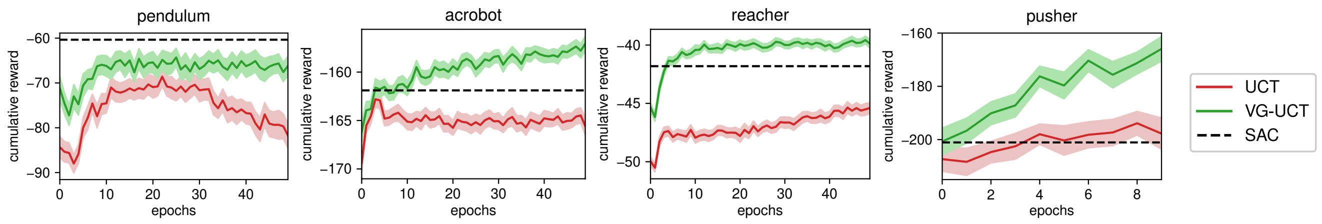

Figure 3: The result of reinforcement learning via MCTS on four stochastic continuous control benchmarks. The x-axis denotes training epochs that correspond to 5 episodes, and the y-axis represents the sum of rewards during 200 time steps for Pendulum and Acrobot, 50 steps for Reacher, and 100 steps for Pusher. The shades in the plots indicate $1.96\times$(standard error). The search times for MCTS planners are given 0.05s for Pendulum, Acrobot and Reacher, and 1s for Pusher. The dotted line show the performance of SAC trained with 10 million steps.

that integrates our core idea on updating action particles via value-gradient ascent (Eq. (12)), and its pseudo-code is provided in Appendix E. As can be seen in the figure, VG-UCT (and VG-KR-UCT) outperforms other baselines, which highlights the effectiveness of VG-UCT that combines global exploration and local fine-tuning. Other algorithms such as UCT, Uniform-RS, and HOLOP suffer from crude discretization and thus their control from planning was not accurate enough to obtain large rewards. In addition, CEM, HOLOP, and Uniform-RS are *open-loop* planning algorithms, which can lead to sub-optimal action in stochastic dynamics even if an infinite amount of time is given (Weinstein and Littman 2012). The performance of Grad-MPC is heavily affected by the initial location of the actions. Using KR-UCT (without value gradients) helped improve the performance of MCTS in some tasks via information sharing among similar actions and guiding action generation via kernel density estimation, but improvement was marginal since it could not fundamentally resolve coarse-grained discretization of the action space.

MCTS as a Policy Improvement Operator

As suggested by Silver et al. (2017), MCTS can be used as a powerful policy improvement operator for reinforcement learning. It is clear that trajectories $\tau = \{s_0, a_0, r_0, s_1, a_1, r_1, \dots\}$, which are generated from the MCTS search guided by some policy π (i.e. used for sampling actions in intermediate nodes and rollouts), will have higher rewards compared to those generated by the direct execution of π. This idea can be adopted in VG-UCT to further improve its performance.

Specifically, we iteratively employ following procedures to improve the quality of new action particle proposals in MCTS:

1. Using the current stochastic policy π_θ parameterized by θ as the proposal distribution for generating new action particles and for rollout in MCTS, collect trajectories $\tau^{(0)}, \tau^{(1)}, \dots$, where

$$\tau^{(i)} = \{s_0^{(i)}, a_0^{(i)}, s_1^{(i)}, a_1^{(i)}, \dots\}$$

is the i-th trajectory collected from the environment.

2. Update the policy parameter θ by maximizing the log-likelihood

$$\arg\max_\theta \sum_i \sum_{t=0}^{|\tau^{(i)}|} \log \pi_\theta(a_t^{(i)} | s_t^{(i)}),$$

which corresponds to the behavioral cloning of the MCTS-improved policy.

In the above scheme, the performance improvement rate of the policy π_θ would reflect the performance of search.

For this set of experiments, we use Gaussian policies $\mathcal{N}\big(a|\mu_\theta(s), \mathrm{diag}(\sigma_\theta^2)\big)$ with state-independent covariance for VG-UCT and UCT. The state-dependent parameterized mean μ_θ is represented as a neural network with two hidden layers, where each hidden layer uses 64 tanh activation units. We use the learned policy network as both an action sampler and a rollout policy for MCTS.

In Figure. 3, we present the learning curves of MCTS algorithms when they were adopted as policy improvement operator to the policy network that guides search. VG-UCT significantly outperforms UCT in all four stochastic continuous control tasks. These are mainly due to the coarse action selection made by UCT, which turns out to have a negative impact on policy improvement, compared to VG-UCT.

5 Conclusion

We presented VG-UCT, an online MCTS algorithm for continuous action spaces, which combines traditional MCTS with local fine-tuning of action particles via value-gradient ascent. Our approach circumvents estimating Jacobian of the dynamics and directly approximate the value-gradient using a finite-difference method, which only requires black-box simulator. Our experimental results show that VG-UCT outperforms the existing MCTS methods and strong baselines for continuous action spaces over various tasks.

As for future work, we can use both policy and value networks to further improve the performance of VG-UCT. By training a value function network, we would obtain an actor-critic-like method that can yield value-gradients without calls to the black-box simulator. This can potentially reduce the variance of value-gradients induced by the stochastic nature of rollout policy and transition while improving the search efficiency by lowering the number of queries to the black-box simulator, which is the main bottleneck of search.

Acknowledgments

This work was supported by the National Research Foundation (NRF) of Korea (NRF-2019M3F2A1072238 and NRF-2019R1A2C1087634), the Information Technology Research Center (ITRC) program of Korea (IITP-2019-2016-0-00464), and the Ministry of Science and Information communication Technology (MSIT) of Korea (IITP No. 2019-0-00075-001 and IITP No. 2017-0-01779 XAI).

References

Arrington, R.; Langley, C.; and Bogaerts, S. 2016. Using domain knowledge to improve monte-carlo tree search performance in parameterized poker squares. In *AAAI Conference on Artificial Intelligence*.

Asadi, K.; Misra, D.; and Littman, M. 2018. Lipschitz continuity in model-based reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*.

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine Learning* 47(2).

Auger, D.; Couëtoux, A.; and Teytaud, O. 2013. Continuous upper confidence trees with polynomial exploration - consistency. In Blockeel, H.; Kersting, K.; Nijssen, S.; and Železný, F., eds., *Machine Learning and Knowledge Discovery in Databases*.

Balla, R.-K., and Fern, A. 2009. UCT for tactical assault planning in real-time strategy games. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym.

Browne, C.; Powley, E.; Whitehouse, D.; Lucas, S.; Cowling, P. I.; 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* 4:1–49.

Bubeck, S.; Stoltz, G.; Szepesvári, C.; and Munos, R. 2009. Online optimization in X-armed bandits. In *NIPS*.

Camacho, E. F., and Alba, C. B. 2013. *Model predictive control*. Springer Science & Business Media.

Chaslot, G. M. J.-B.; Winands, M. H. M.; van den Herik, H. J.; Uiterwijk, J. W. H. M.; and Bouzy, B. 2008. Progressive strategies for Monte-Carlo tree search. *New Math. Nat. Comput.* 4(3).

Couëtoux, A.; Hoock, J.-B.; Sokolovska, N.; Teytaud, O.; and Bonnard, N. 2011a. Continuous upper confidence trees. In *Learning and Intelligent Optimization*.

Couëtoux, A.; Milone, M.; Brendel, M.; Doghmen, H.; Sebag, M.; and Teytaud, O. 2011b. Continuous rapid action value estimates. In *Asian Conference on Machine Learning*.

Coulom, R. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In *Proceedings of the 5th International Conference on Computers and Games*.

Coulom, R. 2007. Computing elo ratings of move patterns in the game of Go. In *Computer Games Workshop*.

Deisenroth, M. P., and Rasmussen, C. E. 2011. Pilco: A model-based and data-efficient approach to policy search. In *In Proceedings of the International Conference on Machine Learning*.

Enzenberger, M.; Muller, M.; Arneson, B.; and Segal, R. 2010. Fuegoan open-source framework for board games and Go engine based on Monte Carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games* 2(4):259–270.

Gelly, S., and Silver, D. 2011. Monte-Carlo tree search and rapid action value estimation in computer Go. *Artif. Intell.* 175(11).

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*.

Heess, N.; Wayne, G.; Silver, D.; Lillicrap, T.; Erez, T.; and Tassa, Y. 2015. Learning continuous control policies by stochastic value gradients. In *NIPS*.

Jiang, N.; Kulesza, A.; Singh, S.; and Lewis, R. 2015. The dependence of effective planning horizon on model accuracy. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*.

Kocsis, L., and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In *Proceedings of the Seventeenth European Conference on Machine Learning*.

Lee, K.; Kim, S.-A.; Choi, J.; and Lee, S.-W. 2018. Deep reinforcement learning in continuous action spaces: a case study in the game of simulated curling. In *Proceedings of the 35th International Conference on Machine Learning*.

Mansley, C. R.; Weinstein, A.; and Littman, M. L. 2011. Sample-based planning for continuous action Markov decision processes. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling*.

Nagabandi, A.; Kahn, G.; Fearing, R. S.; and Levine, S. 2018. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *ICRA*.

Pepels, T.; Winands, M. H. M.; and Lanctot, M. 2014. Real-time monte carlo tree search in ms pac-man. *IEEE Transactions on Computational Intelligence and AI in Games*.

Rubinstein, R. Y., and Kroese, D. P. 2004. *The Cross Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning*. Springer-Verlag.

Sabharwal, A.; Samulowitz, H.; and Reddy, C. 2012. Guiding combinatorial optimization with UCT. In *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems*.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 484–489.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; Chen, Y.; Lillicrap, T.; Hui, F.; Sifre, L.; van den Driessche, G.; Graepel, T.; and Hassabis, D. 2017. Mastering the game of Go without human knowledge. *Nature* 550:354–359.

Todorov, E.; Erez, T.; and Tassa, Y. 2012. Mujoco: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Weinstein, A., and Littman, M. 2012. Bandit-based planning and learning in continuous-action markov decision processes. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling*.

Weinstein, A., and Littman, M. 2013. Open-loop planning in large-scale stochastic domains. In *AAAI Conference on Artificial Intelligence*.

Yee, T.; Lisy, V.; and Bowling, M. 2016. Monte Carlo tree search in continuous action spaces with execution uncertainty. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*.

Appendix A Stochastic Computation Graph of MDP

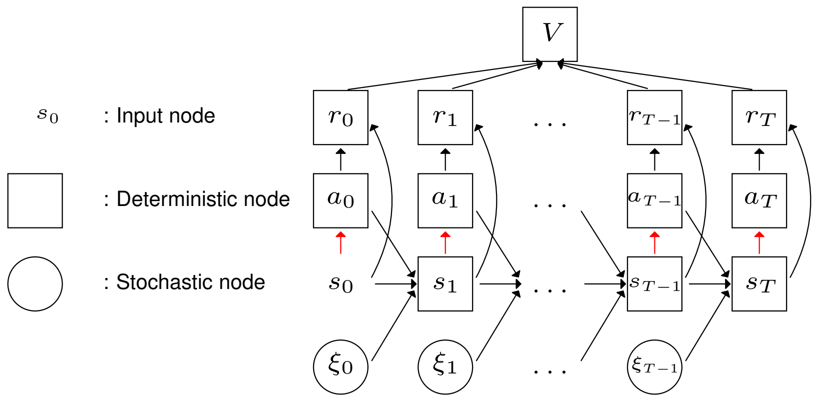

Figure 4: Stochastic computation graph of MDP and the objective function value for the finite-horizon planning, where $V = \mathbb{E}_\xi\left[\sum_{t=0}^{T} r_t \mid s_0, \pi_{0:T}\right]$, $r_t = r(s_t, a_t)$, $s_{t+1} = \rho(s_t, a_t, \xi_t)$, $\xi_t \sim p(\xi)$, and $a_t = \pi_t(s_t)$.

Appendix B Proof of Theorem 1

We first introduce some definitions and assumptions to use, mostly from (Asadi, Misra, and Littman 2018).

Definition 1. *Given two metric spaces (M_1, d_1) and (M_2, d_2) consisting of a space and a distance metric, a function $f : M_1 \to M_2$ is Lipschitz continuous (or simply Lipschitz) if the Lipschitz constant, defined as*

$$K_{d_1,d_2}(f) := \sup_{s_1\in M_1, s_2\in M_1} \frac{d_2(f(s_1), f(s_2))}{d_1(s_1, s_2)}$$

is finite.

Definition 2. *(Wasserstein metric)*
(Primal definition) *Given a metric space (M, d) and the set $\mathcal{P}(M)$ of all probability measures on M, the Wasserstein metric between two probability distributions μ_1 and μ_2 in $\mathcal{P}(M)$ is defined as*

$$W(\mu_1, \mu_2) := \inf_{j\in\Lambda} \iint j(s_1, s_2) d(s_1, s_2) ds_2 ds_1$$

where Λ denotes the collection of all joint distributions j on $M \times M$ with marginals μ_1 and μ_2.
(Dual definition) *Wasserstein distance is linked to Lipschitz continuity using duality:*

$$W(\mu_1, \mu_2) = \sup_{f: K_{d,d_{\mathbb{R}}}(f)\le 1} \int \big(f(s)\mu_1(s) - f(s)\mu_2(s)\big) ds$$

Definition 3. ***(Lipschitz Model Class) (Asadi, Misra, and Littman 2018)*** *Given a metric state space $(\mathcal{S}, d_{\mathcal{S}})$ and an action space $\mathcal{A}$, let Φ_q as a collection of functions $\Phi_q = \{\phi : \mathcal{S} \to \mathcal{S}\}$ distributed according to $q(\phi|a)$ where $a \in \mathcal{A}$. We say that Φ_q is a Lipschitz model class if there exists a finite L_s such that*

$$\sup_{\phi\in\Phi_g} \sup_{s_1\in\mathcal{S}, s_2\in\mathcal{S}} \frac{d_{\mathcal{S}}\big(\phi(s_1), \phi(s_2)\big)}{d_{\mathcal{S}}(s_1, s_2)} \le L_s$$

We will refer to L_s as a Lipschitz constant of Φ_q.

Assumption 1. *Transition function $\mathcal{P}(\cdot|s, a)$, induced by a Lipschitz model class Φ_q with Lipschitz constant L_s, satisfies $\mathcal{P}(s'|s, a) = \int q(\phi|a)\mathbb{1}(\phi(s) = s')d\phi$. Also, for all $s \in S$, $a_1 \in \mathcal{A}$ and $a_2 \in \mathcal{A}$, the following inequality holds:*

$$\frac{W\big(\mathcal{P}(\cdot|s, a_1), \mathcal{P}(\cdot|s, a_2)\big)}{d\big(a_1, a_2\big)} \le L_a \tag{13}$$

Also, reward function $r(s, a)$ is L_r-Lipschitz continuous: for all $(s_1, a_1) \in \mathcal{S} \times \mathcal{A}$ and $(s_2, a_2) \in \mathcal{S} \times \mathcal{A}$,

$$\frac{|r(s_1, a_1) - r(s_2, a_2)|}{d\big((s_1, a_1), (s_2, a_2)\big)} \le L_r$$

Finally, for all t, policy π_t is L_π-Lipschitz continuous: for all $s1 \in \mathcal{S}$ and $s_2 \in \mathcal{S}$,

$$\frac{d(\pi_t(s_1), \pi_t(s_2))}{d(s_1, s_2)} \leq L_\pi$$

Definition 4. $\mu_t(s|\pi_{0:t-1}) = \Pr(s_t = s|s_0, \pi_{0:t-1})$ *and* $d(\pi_1, \pi_2) = \sup_{s \in \mathcal{S}} d_\mathcal{A}(\pi_1(s), \pi_2(s))$.

Lemma 1. *For all $t \geq 1$, the difference of two state distributions at time step t is bounded by policy perturbation:*

$$W(\mu_t(\cdot|\pi_{0:t-1}), \mu_t(\cdot|\hat{\pi}_{0:t-1})) \leq L_a \sum_{i=0}^{t-1} L_s^{t-i-1} d(\pi_i, \hat{\pi}_i)$$

Proof. We provide the proof by induction. When $t = 1$,

$$\begin{aligned}
W(\mu_1(\cdot|\pi_0), \mu_1(\cdot|\hat{\pi}_1)) &= W(p(\cdot|s_0, \pi_0), p(\cdot|s_0, \hat{\pi}_0)) \\
&= W(\mathcal{P}(\cdot|s_0, \pi_0(s_0)), \mathcal{P}(\cdot|s_0, \hat{\pi}_0(s_0))) \\
&\leq L_a d(\pi_0, \hat{\pi}_0)
\end{aligned}$$

We now prove the inductive step. Assume that $W(\mu_t(\cdot|\pi_{0:t-1}), \mu_t(\cdot|\hat{\pi}_{0:t-1})) \leq L_a \sum_{i=0}^{t-1} L_s^{t-i-1} d(\pi_i, \hat{\pi}_i)$. Then,

$$\begin{aligned}
&W(\mu_{t+1}(\cdot|\pi_{0:t}), \mu_{t+1}(\cdot|\hat{\pi}_{0:t})) \\
&\leq \underbrace{W(\mu_{t+1}(\cdot|\pi_{0:t}), \mu_{t+1}(\cdot|\pi_{0:t-1}\hat{\pi}_t))}_{\text{(a)}} + \underbrace{W(\mu_{t+1}(\cdot|\pi_{0:t-1}\hat{\pi}_t), \mu_{t+1}(\cdot|\hat{\pi}_{0:t}))}_{\text{(b)}}
\end{aligned}$$

Here, (a) is bounded by:

$$\begin{aligned}
\text{(a)} &= W(\mu_{t+1}(\cdot|\pi_{0:t}), \mu_{t+1}(\cdot|\pi_{0:t-1}, \hat{\pi}_t)) \\
&= \sup_f \int \Big(\mu_{t+1}(s|\pi_{0:t-1}, \pi_t) - \mu_{t+1}(s|\pi_{0:t-1}, \hat{\pi}_t)\Big) f(s) ds \\
&= \sup_f \iint \mu_t(s_0|\pi_{0:t-1}) \Big(p(s|s_0, \pi_t) - p(s|s_0, \hat{\pi}_t)\Big) ds_0 f(s) ds \\
&\leq \int \mu_t(s_0|\pi_{0:t-1}) \sup_f \int \Big(p(s|s_0, \pi_t) - p(s|s_0, \hat{\pi}_t)\Big) f(s) ds ds_0 \\
&= \int \mu_t(s_0|\pi_{0:t-1}) W(\mathcal{P}(\cdot|s_0, \pi_t(s_0)), \mathcal{P}(\cdot|s_0, \hat{\pi}_t(s_0))) ds_0 \\
&\leq \int \mu_t(s_0|\pi_{0:t-1}) L_a d(\pi_t(s_0), \hat{\pi}_t(s_0)) ds_0 \\
&\leq L_a \int \mu_t(s_0|\pi_{0:t-1}) d(\pi_t, \hat{\pi}_t) ds_0 \\
&= L_a d(\pi_t, \hat{\pi}_t)
\end{aligned}$$

Also, (b) is bounded by:

$$\begin{aligned}
\text{(b)} &= W(\mu_{t+1}(\cdot|\pi_{0:t-1}, \hat{\pi}_t), \mu_{t+1}(\cdot|\hat{\pi}_{0:t})) \\
&= \sup_{f: K_{d_\mathcal{S}, \mathbb{R}}(f) \leq 1} \int \Big(\mu_{t+1}(s|\pi_{0:t-1}, \hat{\pi}_t) - \mu_{t+1}(s|\hat{\pi}_{0:t-1}, \hat{\pi}_t)\Big) f(s) ds \\
&= \sup_{f: K_{d_\mathcal{S}, \mathbb{R}}(f) \leq 1} \iint \Big(\mu_t(s_0|\pi_{0:t-1}) - \mu_t(s_0|\hat{\pi}_{0:t-1})\Big) p(s|s_0, \hat{\pi}_t) ds_0 f(s) ds \\
&= \sup_{f: K_{d_\mathcal{S}, \mathbb{R}}(f) \leq 1} \iint \Big(\mu_t(s_0|\pi_{0:t-1}) - \mu_t(s_0|\hat{\pi}_{0:t-1})\Big) \mathcal{P}(s|s_0, \hat{\pi}_t(s_0)) f(s) ds ds_0 \\
&= \sup_{f: K_{d_\mathcal{S}, \mathbb{R}}(f) \leq 1} \iint \Big(\mu_t(s_0|\pi_{0:t-1}) - \mu_t(s_0|\hat{\pi}_{0:t-1})\Big) \int q(\phi|\hat{\pi}_t(s_0)) \mathbb{1}(\phi(s_0) = s) f(s) d\phi ds ds_0 \\
&= \sup_{f: K_{d_\mathcal{S}, \mathbb{R}}(f) \leq 1} \iint \Big(\mu_t(s_0|\pi_{0:t-1}) - \mu_t(s_0|\hat{\pi}_{0:t-1})\Big) q(\phi|\hat{\pi}_t(s_0)) f(\phi(s_0)) d\phi ds_0
\end{aligned}$$

$$
\begin{aligned}
&\quad f(\phi(s_0)) \text{ is composition of } L_s\text{-Lipschitz and 1-Lipschitz} \Rightarrow L_s\text{-Lipschitz.}\\
&= L_s \sup_{f:K_{d_\mathcal{S},\mathbb{R}}(f)\le 1} \iint \Big(\mu_t(s_0|\pi_{0:t-1}) - \mu_t(s_0|\hat{\pi}_{0:t-1})\Big) q(\phi|\hat{\pi}_t(s_0)) \frac{f(\phi(s_0))}{L_s} d\phi ds_0\\
&\le L_s \sup_{g:K_{d_\mathcal{S},\mathbb{R}}(f)\le 1} \iint \Big(\mu_t(s_0|\pi_{0:t-1}) - \mu_t(s_0|\hat{\pi}_{0:t-1})\Big) q(\phi|\hat{\pi}_t(s_0)) g(s_0) d\phi ds_0\\
&= L_s \sup_{g:K_{d_\mathcal{S},\mathbb{R}}(f)\le 1} \int \Big(\mu_t(s_0|\pi_{0:t-1}) - \mu_t(s_0|\hat{\pi}_{0:t-1})\Big) \left(\int q(\phi|\hat{\pi}_t(s_0)) d\phi\right) g(s_0) ds_0\\
&= L_s \sup_{g:K_{d_\mathcal{S},\mathbb{R}}(f)\le 1} \int \Big(\mu_t(s_0|\pi_{0:t-1}) - \mu_t(s_0|\hat{\pi}_{0:t-1})\Big) g(s_0) ds_0\\
&= L_s W(\mu_t(\cdot|\pi_{0:t-1}), \mu_t(\cdot|\hat{\pi}_{0:t-1}))
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
&W(\mu_{t+1}(\cdot|\pi_{0:t}), \mu_{t+1}(\cdot|\hat{\pi}_{0:t}))\\
&\le \text{(a)} + \text{(b)}\\
&\le L_a d(\pi_t, \hat{\pi}_t) + L_s W(\mu_t(\cdot|\pi_{0:t-1}), \mu_t(\cdot|\hat{\pi}_{0:t-1}))\\
&\le L_a d(\pi_t, \hat{\pi}_t) + L_s \left(L_a \sum_{i=0}^{t-1} L_s^{t-i-1} d(\pi_i, \hat{\pi}_i)\right) && \text{(inductive hypothesis)}\\
&= L_a \left(d(\pi_t, \hat{\pi}_t) + \sum_{i=0}^{t-1} L_s^{t-i} d(\pi_i, \hat{\pi}_i)\right)\\
&= L_a \left(\sum_{i=0}^{t} L_s^{t-i} d(\pi_i, \hat{\pi}_i)\right)
\end{aligned}
$$

which concludes the proof. □

Theorem 1. *Assume transition function is induced by a Lipschitz model class (Asadi, Misra, and Littman 2018) Φ_g, and reward function is Lipschitz continuous. If two Lipschitz policies $\pi_{0:T}$ and $\hat{\pi}_{0:T}$ are ϵ-close (i.e. $\sup_{t,s} d(\pi_t(s), \hat{\pi}_t(s)) \le \epsilon$), then for all s_0, the gap between two value functions are bounded by ϵ:*

$$|V(s_0, \pi_{0:T}) - V(s_0, \hat{\pi}_{0:T})| \le L_v \epsilon$$

where $L_v = L_r \left(1 + T + \frac{L_a(1+L_\pi)}{1-L_s}\left(T - \frac{L_s(1-L_s^T)}{1-L_s}\right)\right)$, L_s, L_a are Lipschitz constant for transition function, L_r is Lipschitz constant for reward function, and L_π is Lipschitz constant for policies.

Proof. We first define a function $f(s) = \frac{r(s,\hat{\pi}(s))}{L_r(1+L_\pi)}$. It can be shown that $K_{d_\mathcal{S},\mathbb{R}}(f) = 1$ as follows:

$$
\begin{aligned}
|f(s) - f(\hat{s})| &= \frac{1}{L_r(1+L_\pi)} |r(s, \hat{\pi}(s)) - r(\hat{s}, \hat{\pi}(\hat{s}))|\\
&\le \frac{1}{L_r(1+L_\pi)} L_r |d(s, \hat{s}) + d(\hat{\pi}(s), \hat{\pi}(\hat{s}))|\\
&\le \frac{1}{L_r(1+L_\pi)} L_r |d(s, \hat{s}) + L_\pi d(s, \hat{s})|\\
&= d(s, \hat{s})
\end{aligned}
$$

Then,

$$
\begin{aligned}
&V(s_0, \pi_{0:T}) - V(s_0, \hat{\pi}_{0:T})\\
&= \sum_{t=0}^{T} \left(\int \mu_t(s|\pi_{0:t-1}) r(s, \pi_t(s)) - \mu_t(s|\hat{\pi}_{0:t-1}) r(s, \hat{\pi}_t(s)) ds\right)\\
&= \sum_{t=0}^{T} \int \mu_t(s|\pi_{0:t-1}) \underbrace{\Big(r(s, \pi_t(s)) - r(s, \hat{\pi}_t(s))\Big)}_{\le L_r d(\pi_t, \hat{\pi}_t)} + \Big(\mu_t(s|\pi_{0:t-1}) - \mu_t(s|\hat{\pi}_{0:t-1})\Big) r(s, \hat{\pi}_t(s)) ds
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_{t=0}^{T} \int \mu_t(s|\pi_{0:t-1}) \underbrace{\Big(r(s,\pi_t(s)) - r(s,\hat{\pi}_t(s))\Big)}_{\le L_r d(\pi_t,\hat{\pi}_t)} + \Big(\mu_t(s|\pi_{0:t-1}) - \mu_t(s|\hat{\pi}_{0:t-1})\Big) r(s,\hat{\pi}_t(s)) ds \\
&\le \sum_{t=0}^{T} \left(L_r d(\pi_t,\hat{\pi}_t) + \int \Big(\mu_t(s|\pi_{0:t-1}) - \mu_t(s|\hat{\pi}_{0:t-1})\Big) r(s,\hat{\pi}_t(s)) ds \right) \\
&= \sum_{t=0}^{T} \left(L_r d(\pi_t,\hat{\pi}_t) + L_r(1+L_\pi) \int \Big(\mu_t(s|\pi_{0:t-1}) - \mu_t(s|\hat{\pi}_{0:t-1})\Big) f(s) ds \right) \\
&\le \sum_{t=0}^{T} \left(L_r d(\pi_t,\hat{\pi}_t) + L_r(1+L_\pi) \sup_{f: K_{d_{\mathcal{S}},\mathbb{R}}(f) \le 1} \int \Big(\mu_t(s|\pi_{0:t-1}) - \mu_t(s|\hat{\pi}_{0:t-1})\Big) f(s) ds \right) \\
&= \sum_{t=0}^{T} \Big(L_r d(\pi_t,\hat{\pi}_t) + L_r(1+L_\pi) W(\mu_t(\cdot|\pi_{0:t-1}), \mu_t(\cdot|\hat{\pi}_{0:t-1})) \Big) \\
&= L_r d(\pi_t,\hat{\pi}_t) + \sum_{t=1}^{T} \Big(L_r d(\pi_t,\hat{\pi}_t) + L_r(1+L_\pi) W(\mu_t(\cdot|\pi_{0:t-1}), \mu_t(\cdot|\hat{\pi}_{0:t-1})) \Big) \\
&\le L_r d(\pi_t,\hat{\pi}_t) + \sum_{t=1}^{T} \left(L_r d(\pi_t,\hat{\pi}_t) + L_r(1+L_\pi) \left(L_a \sum_{i=0}^{t-1} L_s^{t-i-1} d(\pi_i,\hat{\pi}_i) \right) \right) \qquad \text{(Lemma 1)} \\
&\le L_r \epsilon + \sum_{t=1}^{T} \left(L_r + L_a L_r (1+L_\pi) \sum_{i=0}^{t-1} L_s^i \right) \epsilon \\
&= L_r \epsilon + \sum_{t=1}^{T} \left(L_r + L_a L_r (1+L_\pi) \frac{1-L_s^t}{1-L_s} \right) \epsilon \\
&= L_r \left(1 + T + \frac{L_a(1+L_\pi)}{1-L_s} \left(T - \frac{L_s(1-L_s^T)}{1-L_s} \right) \right) \epsilon
\end{aligned}
$$

□

Appendix C Convergence Analysis - Finite Tree and Deterministic Transition Case

Careful readers can notice that the gradient update of VG-UCT may sometimes negatively affect the parent node in terms of rewards depending on the node selection in the subtree. Fortunately, we can further show that under some assumptions detailed below, VG-UCT asymptotically improves the best sequence of actions toward the maximally achievable rewards within Δ-bounded region.

In order to show this, we start from some definitions and well-known facts regarding gradient-ascent.

Definition 5. *A differentiable function $f(x)$ is said to be μ-strongly concave if for any x and y,*

$$f(y) \le f(x) + \langle \nabla f(x), y - x \rangle - \frac{1}{2}\mu \|y - x\|_2^2 \tag{14}$$

Definition 6. *A differentiable and concave function $f(x)$ is said to be L-smooth if for any x and y,*

$$f(y) \ge f(x) + \langle \nabla f(x), y - x \rangle - \frac{1}{2}L \|y - x\|_2^2 \tag{15}$$

Lemma 2. *If f is L-smooth, then*

$$\|\nabla f(x)\|_2^2 \le -2L(f(x) - f(x^*)) \tag{16}$$

where x^* is the point that maximizes $f(x)$.

Proof. In Eq. (15), by inserting $y = x + \frac{1}{L}\nabla f(x)$,

$$
\begin{aligned}
f\left(x + \frac{1}{L}\nabla f(x)\right) &\ge f(x) + \frac{1}{L}\langle \nabla f(x), \nabla f(x) \rangle - \frac{1}{2}L \|\frac{1}{L}\nabla f(x)\|_2^2 \\
&= f(x) + \frac{1}{2L}\|\nabla f(x)\|_2^2
\end{aligned}
$$

$$\Leftrightarrow 2L\left(f\left(x - \frac{1}{L}\nabla f(x)\right) - f(x) \right) \ge \|\nabla f(x)\|_2^2$$

Therefore, since x^* is the point that maximizes $f(x)$, we obtain

$$\|\nabla f(x)\|_2^2 \leq 2L\left(f\left(x - \frac{1}{L}\nabla f(x)\right) - f(x)\right) \leq 2L(f(x^*) - f(x)) = -2L(f(x) - f(x^*))$$

□

Lemma 3. *Let f be μ-strongly concave and L-smooth function. Then, for any x^n, $\eta_n = \frac{1}{\mu(n+\alpha)}$, and $\alpha > 1$ such that $\eta_n \leq \frac{1}{L}$, the gradient update $x^{n+1} = x^n + \eta_n \nabla f(x^n)$ makes the point closer to the optima:*

$$\|x^{n+1} - x^*\|_2^2 \leq \frac{n+\alpha-1}{n+\alpha}\|x^n - x^*\|_2^2$$

Proof. By definition 14 with $x \to x^n$, $y \to x^*$,

$$2\langle \nabla f(x^n), x^n - x^* \rangle \leq 2(f(x^n) - f(x^*)) - \mu\|x^n - x^*\|_2^2 \tag{17}$$

$$\begin{aligned}
\|x^{n+1} - x^*\| &= \|x^n - x^* + \eta_n \nabla f(x^n)\|_2^2 \\
&= \|x^n - x^*\|_2^2 + 2\eta_n\langle \nabla f(x^n), x^n - x^*\rangle + \eta_n^2\|\nabla f(x^n)\|_2^2 && \text{(Eq. (17))} \\
&\leq \|x^n - x^*\|_2^2 + 2\eta_n(f(x^n) - f(x^*)) - \mu\eta_n\|x^n - x^*\|_2^2 + \eta_n^2\|\nabla f(x^n)\|_2^2 \\
&= (1 - \mu\eta_n)\|x^n - x^*\|_2^2 + 2\eta_n(f(x^n) - f(x^*)) + \eta_n^2\|\nabla f(x^n)\|_2^2 \\
&\leq (1 - \mu\eta_n)\|x^n - x^*\|_2^2 + 2\eta_n(f(x^n) - f(x^*)) - 2L\eta_n^2(f(x^n) - f(x^*)) && \text{(Lemma 2)} \\
&= (1 - \mu\eta_n)\|x^n - x^*\|_2^2 + 2\eta_n(1 - L\eta_n)(f(x^n) - f(x^*)) \\
&\leq (1 - \mu\eta_n)\|x^n - x^*\|_2^2 && (\because \eta_n \leq 1/L) \\
&= \frac{n+\alpha-1}{n+\alpha}\|x^n - x^*\|_2^2 && (\because \eta_n = 1/(\mu(n+\alpha)))
\end{aligned}$$

□

Then, we make definitions and assumptions about VG-UCT and the underlying environment.

Assumption 2. *A finitely but fully (i.e. up to planning horizon) expanded search tree is given. Here, progressive-widening does not occur. Transition dynamics of the environment is deterministic, i.e. $s_{t+1} = \rho(s_t, a_t)$.*

Definition 7. *We define the followings:*

1. *For any sequence of actions, $a^1_{0:T}$, $V(s_0, a_{0:T})$ means:*

$$V(s_0, a_{0:T}) := \sum_{t=0}^{T} r(s_t, a_t) \text{ s.t. } s_{t+1} = \rho(s_t, a_t)$$

2. *For any two sequences of chance nodes selected by VG-UCT, $\nu^1_{0:T}$ and $\nu^2_{0:T}$, $\nu^1_{0:T} < \nu^2_{0:T}$ means:*

$$V(s_0, \nu^1_{0:T}) := \sum_{t=0}^{T} r(s^1_t, \text{ACTION}(\nu^1_t)) < \sum_{t=0}^{T} r(s^2_t, \text{ACTION}(\nu^2_t)) = V(s_0, \nu^2_{0:T})$$

where $s^1_{t+1} = \rho(s^1_t, \text{ACTION}(\nu^1_t))$, and $s^2_{t+1} = \rho(s^2_t, \text{ACTION}(\nu^2_t))$.

3. *Let S be the set of all possible sequence of chance nodes that can be selected by VG-UCT.*
4. *Let $\nu^*_{0:T}$ be the action sequence that can be selected by VG-UCT, whose value is the highest, i.e., $\nu^*_{0:T} := \arg\sup_{\nu_{0:T} \in S} V(s_0, \nu_{0:T})$.*
5. *Finally, we define δ_Δ as:*

$$\delta_\Delta := \inf_{\nu_{0:T} \in S \setminus \{\nu^*_{0:T}\}} \inf_{\forall t, a^*_t : d(\text{INIT-ACTION}(\nu^*_t), a^*_t) \leq \Delta} \inf_{\forall t, a_t : d(\text{INIT-ACTION}(\nu_t), a_t) \leq \Delta} |V(s_0, a^*_{0:T}) - V(s_0, a_{0:T})|$$

which means the value gap between the best possible value and the second best value.

Assumption 3. *We assume the followings:*

1. $\delta_\Delta > 0$.

2. *For any two sequences of chance node, $\nu^1_{0:T}$ and $\nu^2_{0:T}$, that can be selected by VG-UCT, the order of values functions are preserved with respect to Δ-perturbation of actions, i.e. if $\nu^1_{0:T} < \nu^2_{0:T}$, then the following also holds:*

$$\sup_{\forall t, a^1_t : d(\text{INIT-ACTION}(\nu^1_t), a^1_t) \leq \Delta} V(s_0, a^1_{0:T}) < \inf_{\forall t, a^2_t : d(\text{INIT-ACTION}(\nu^2_t), a^2_t) \leq \Delta} V(s_0, a^2_{0:T})$$

3. *The value function $V(s_0, a_{0:T})$ is differentiable, μ-strongly concave, and L-smooth function within the Δ-bounded region for all $a_{0:T} \in \{\text{ACTION}(\nu_{0:T}) : \nu_{0:T} \in S\}$.*
4. *The step-size sequence for the value-gradient ascent at n-th simulation is $\eta_n = \frac{1}{\mu(n+\alpha)}$ where $\alpha > 1$ such that $\eta_n \leq \frac{1}{L}$.*
5. *For all $a_{0:T}$ within Δ-bounded region, the norm of gradient is upper bounded by a constant b: $\|\partial V(s_0, a_{0:T}) / \partial a_{0:T}\| \leq b$.*

We now provide the main theorem as follows.

Theorem 2. *Under Assumptions 2 and 3, a sequence of actions selected by VG-UCT, $a_{0:T}$, converges to the one that receives the maximally achievable rewards within Δ-bounded region, with probability 1 as the number of simulations goes to infinity:*

$$a_{0:T} \to \operatorname*{arg\,sup}_{\forall t, a^*_t : d(\text{INIT-ACTION}(\nu^*_t), a^*_t) \leq \Delta} V(s_0, a^*_{0:T})$$

Proof. Let n be the number of simulations performed by VG-UCT. Now, let y_n be the gap between the maximally achievable value and the current value with respect to actions stored in $\nu^*_{0:T}$ at n-th simulation:

$$y_n := \left\| \text{ACTION}(\nu^*_{0:T})^n - \operatorname*{arg\,sup}_{\forall t, a^*_t : d(\text{INIT-ACTION}(\nu^*_t), a^*_t) \leq \Delta} V(s_0, a^*_{0:T}) \right\|_2^2$$

By Assumption 3-5, the amount of the worst possible update (by subtree's node selection) is bounded by:

$$\begin{aligned} \left\| \text{ACTION}(\nu^*_{0:T})^n - \eta_n g - \operatorname*{arg\,sup}_{\forall t, a^*_t : d(\text{INIT-ACTION}(\nu^*_t), a^*_t) \leq \Delta} V(s_0, a^*_{0:T}) \right\|_2^2 &\leq y_n + 2\eta_n \|g\| \sqrt{y_n} + \eta_n^2 \|g\|^2 \\ &\leq y_n + 4\Delta b \eta_n + b^2 \eta_n^2 \\ &\leq y_n + \underbrace{(4\Delta b + b^2)}_{:=B} \eta_n \\ &= y_n + B\eta_n \end{aligned} \tag{18}$$

Then, from Lemma 3 and Eq. (18), we can consider the following sequence:

$$y_{n+1} = \begin{cases} \frac{n+\alpha-1}{n+\alpha} y_n & \text{case (A): if the best } \nu^*_{0:T} \text{ is selected by VG-UCT.} \\ y_n + \frac{B}{\mu(n+\alpha)} & \text{case (B): if a suboptimal } \nu_{0:T} \neq \nu^*_{0:T} \text{ is selected by VG-UCT.} \end{cases}$$

which corresponds to the sequence of the worst possible update of $\nu^*_{0:T}$.

For sufficiently large n, case (B) becomes increasingly rarely (definitely sublinearly with respect to n) chosen due to $\delta_\Delta > 0$ and the property of UCT, which concludes that $y_n \to 0$ as $n \to \infty$. □

Note that if $\Delta = 0$, VG-UCT reduces to UCT, and Assumption 3 can always be satisfied. As $\Delta > 0$ increases, VG-UCT starts to have more advantages over UCT by Theorem 2, but Assumption 3 is increasingly difficult to be satisfied, thus the result of Theorem 2 can be invalidated accordingly.

Appendix D Experimental Setup

We used a computing server with Intel(R) Xeon(R) CPU E5-2660 v3 and 96GB RAM for all the experiments. For the experiments, we did not parallelize Eq. (12). We performed value-gradient ascent with probability $1/4$ at each simulation, instead of always performing it at every simulation, as a trade-off between gradient-update and more simulations.

D.1 Detailed Reward Description of Illustrative 2D Example

The reward is 0.5 for $s_0 = (1, 1)$, 10 for $s_g = (5, 5)$, and their values decreases exponentially in the squared distance to these locations. Similarly, the negative reward of -15 are given for locations $s^1_- = (1, 5)$, $s^2_- = (3, 3)$, $s^3_- = (5, 1)$ with exponential decay (but with relatively larger bandwidth).

$$r(s) = 0.5 \exp\left(-\frac{\|s - s_0\|^2}{0.5}\right) + 10 \exp\left(-\frac{\|s - s_g\|^2}{0.05}\right) - \sum_{i=1}^{3} 15 \exp\left(-\frac{\|s - s^i_-\|^2}{0.3}\right)$$

dimension	state (n)	observation (o)	action (m)
Pendulum	2	3	1
Acrobot	4	6	1
Reacher-v2	8	11	2
Pusher-v2	22	23	7

Table 1: Dimesion of target environments

D.2 Experimental Setup

Let $\bar{s}' = f(s, a)$ be an underlying deterministic transition function of each task. We made the transition be stochastic by adding Gaussian white-noises $\xi_a \sim \mathcal{N}(0, \varepsilon_a^2 I_m)$ and $\xi_s \sim \mathcal{N}(0, \varepsilon_s^2 I_n)$. With $\xi = (\xi_a, \xi_s)$, the stochastic transition used for black-box simulation was defined as $\rho(s, a, \xi) = f(s, a + \xi_a) + \xi_s$.

The followings are the experimental setup for each task, where PW stands for progressive-widening, Δ is the maximum amount of action perturbation. For the step-size η of VG-UCT, we tested $\{0.001, 0.01, 0.05\}$ and 0.01 was chosen. For Δ, we did not perform a parameter search and just selected a simple value.

$(\varepsilon_a, \varepsilon_s)$	(0.1, 0.05)
UCB constant c	0.5
PW constant (α, β)	(0.5, 0.5)
search time	$\{0.01, 0.05, 0.1, 0.3, 0.5\}$
# of runs	300
planning horizon	20
KR bandwidth σ	0.5
step-size η	0.01
Δ	0.5

Table 2: Pendulum

$(\varepsilon_a, \varepsilon_s)$	(0.2, 0.1)
UCB constant c	1
PW constant (α, β)	(0.5, 0.5)
search time	$\{0.05, 0.1, 0.3, 0.5, 1\}$
# of runs	300
planning horizon	20
KR bandwidth σ	0.5
step-size η	0.01
Δ	0.5

Table 3: Acrobot

$(\varepsilon_a, \varepsilon_s)$	(0.5, 0.05)
UCB constant c	1
PW constant (α, β)	(0.5, 0.5)
search time	$\{0.05, 0.1, 0.3, 0.5, 1\}$
# of runs	300
planning horizon	15
KR bandwidth σ	0.5
step-size η	0.01
Δ	0.5

Table 4: Reacher

$(\varepsilon_a, \varepsilon_s)$	(1, 0.05)
UCB constant c	1
PW constant (α, β)	(0.5, 0.5)
search time	$\{0.3, 0.5, 1, 3, 5\}$
# of runs	500
planning horizon	15
KR bandwidth σ	0.5
step-size η	0.01
Δ	1

Table 5: Pusher

For the experiments of *MCTS as a Policy Improvement Operator*, the learning rate and the number of iterations per epoch were chosen to be 0.005 and 100, respectively, with the size of minibatches being 200.

The results in Figure 2 were obtained by averaging 300 runs for {Pendulum, Acrobot, Reacher} and 500 runs for Pusher. For Figure 3, the results were obtained by averaging 300 runs for {Pendulum, Acrobot, Reacher} and 100 runs for Pusher.

Appendix E Pseudo-code of KR-VG-UCT

In this section, we provide the pseudo-code of VG-KR-UCT, which is our extended version of KR-UCT, and it integrates our core idea on updating action particles via value-gradient ascent. Here, $K(a, b)$ is a kernel function, where we use a Gaussian kernel: $K(a, b) \propto \exp\left(-\frac{\|a-b\|^2}{2\sigma^2}\right)$. For brevity, we use a notation $K(\nu_1^C, \nu_2^C)$ to denote $K(\text{ACTION}(\nu_1^C), \text{ACTION}(\nu_2^C))$, and $K(\nu^C, a)$ for $K(\text{ACTION}(\nu^C), a)$. KR-UCT parts are highlighted as red.

Algorithm 2 Value-Gradient Kernel-Regression UCT (VG-KR-UCT)

```
function SEARCH($s_0$)
  Create a root node $\nu^D$.
  repeat
    SIMULATE($s_0$, $\nu^D$, 0)
  until TIMEOUT()
  $\nu^C \leftarrow \arg\max_{\nu^C \in \text{CHILDREN}(\nu^D)} Q(\nu^D, \nu^C)$
  $a^* \leftarrow \text{ACTION}(\nu^C)$
  return $a^*$
end function

function SIMULATE($s$, $\nu^D$, $t$)
  if $t$ = (planning horizon $T$) then
    return $[0, \emptyset, \emptyset]$
  end if
  $[\nu^C, a, \text{rollout}] \leftarrow$SELECTACTION($s$, $\nu^D$, $c$)
  $[\nu^{D\prime}, \xi] \leftarrow$SAMPLENOISE($\nu^D$, $\nu^C$)
  $[r, s'] \leftarrow \mathcal{G}(s, a, \xi)$
  if rollout then
    $[R', \{a_{t+1:T}\}, \{\xi_{t+1:T}\}] \leftarrow$ROLLOUT($s'$, $t+1$)
  else
    $[R', \{a_{t+1:T}\}, \{\xi_{t+1:T}\}] \leftarrow$SIMULATE($s'$, $\nu^{D\prime}$, $t+1$)
  end if
  $R \leftarrow r + R'$
  $N(\nu^D) \leftarrow N(\nu^D) + 1$
  $N(\nu^D, \nu^C) \leftarrow N(\nu^D, \nu^C) + 1$
  $Q(\nu^D, \nu^C) \leftarrow Q(\nu^D, \nu^C) + \frac{R - Q(\nu^D, \nu^C)}{N(\nu^D, \nu^C)}$
  // After simulation, perform value-gradient ascent.
  for $j = 1, \ldots, m$ do
    $\tilde{a}^j \leftarrow a + \epsilon e_j$
    $[g]_j = \frac{\text{RETURN}(s, \{\tilde{a}^j, a_{t+1:T}\}, \{\xi, \xi_{t+1:T}\}) - R}{\epsilon}$ (Eq. (12))
  end for
  $\text{ACTION}(\nu^C) \leftarrow \text{ACTION}(\nu^C) + \eta g$ // $a$ remains same.
  $\text{ACTION}(\nu^C)$ is clipped within:
      $\{\hat{a} \in \mathcal{A} : \|\hat{a} - \text{INIT-ACTION}(\nu^C)\| \le \Delta\}$
  return $[R, \{a, a_{t+1:T}\}, \{\xi, \xi_{t+1:T}\}]$
end function

function NEWACTION($W$, $\nu^D$, $a^*$)
  // $a \approx \arg\min_{a : K(a, a^*) > \tau} W(a)$
  $A \leftarrow \emptyset$
  Sample $k$ actions from $\mathcal{N}(a^*, \sigma^2 I)$ and add them to $A$
  return $\arg\min_{a \in A} \sum_{\nu^C \in \text{CHILDREN}(\nu^D)} K(\nu^C, a) N(\nu^D, \nu^C)$
end function

function SELECTACTION($s$, $\nu^D$, $c$)
  for $\nu_1 \in \text{CHILDREN}(\nu^D)$ do
    $W(\nu_1) := \sum_{\nu_2 \in \text{CHILDREN}(\nu^D)} K(\nu_1, \nu_2) N(\nu^D, \nu^C)$
    $V(\nu_1) := \frac{\sum_{\nu_2 \in \text{CHILDREN}(\nu^D)} K(\nu_1, \nu_2) N(\nu^D, \nu_2) Q(\nu^D, \nu_2)}{\sum_{\nu_2 \in \text{CHILDREN}(\nu^D)} K(\nu_1, \nu_2) N(\nu^D, \nu_2)}$
  end for
  $W(\nu^D) := \sum_{\nu_1 \in \text{CHILDREN}(\nu^D)} W(\nu_1)$
  $\nu^* \leftarrow \arg\max_{\nu^C \in \text{CHILDREN}(\nu^D)} V(\nu^C) + c\sqrt{\frac{\log W(\nu^D)}{W(\nu^C)}}$
  $a^* \leftarrow \text{ACTION}(\nu^*)$
  if $\lfloor N(\nu^D)^\alpha \rfloor \ge |\text{CHILDREN}(\nu^D)|$ then
    Create a new chance node $\nu^C$
    Add $\nu^C$ to $\text{CHILDREN}(\nu^D)$
    $a \leftarrow$NEWACTION($W$, $\nu^D$, $a^*$)
    $\text{ACTION}(\nu^C) \leftarrow a$, $\text{INIT-ACTION}(\nu^C) \leftarrow a$
    $N(\nu^D, \nu^C) \leftarrow 0$, $Q(\nu^D, \nu^C) \leftarrow 0$
    rollout $\leftarrow$ true
  else
    $\nu^C \leftarrow \nu^*$, $a \leftarrow a^*$
    rollout $\leftarrow$ false
  end if
  return $[\nu^C, a, \text{rollout}]$
end function

function SAMPLENOISE($\nu^D$, $\nu^C$)
  if $\lfloor N(\nu^D, \nu^C) \rfloor^\beta \ge |\text{CHILDREN}(\nu^C)|$ then
    Create a new decision node $\nu^{D\prime}$
    Add $\nu^{D\prime}$ to $\text{CHILDREN}(\nu^C)$
    $\text{NOISE}(\nu^{D\prime}) \leftarrow p(\xi)$ and $N(\nu^{D\prime}) \leftarrow 0$
  else
    $\nu^{D\prime} \leftarrow$ (least visited node in $\text{CHILDREN}(\nu^C)$)
  end if
  $\xi \leftarrow \text{NOISE}(\nu^{D\prime})$
  return $[\nu^{D\prime}, \xi]$
end function

function ROLLOUT($s$, $t$)
  if $t$ = (planning horizon $T$) then
    return $[0, \emptyset, \emptyset]$
  end if
  $a \sim \pi_{\text{rollout}}(\cdot|s)$ and $\xi \sim p(\xi)$
  $[r, s'] \leftarrow \mathcal{G}(s, a, \xi)$
  $[R', \{a_{t+1:T}\}, \{\xi_{t+1:T}\}] \leftarrow$ROLLOUT($s'$, $t+1$)
  $R \leftarrow r + R'$
  return $[R, \{a, a_{t+1:T}\}, \{\xi, \xi_{t+1:T}\}]$
end function
```