SPENDING THINKING TIME WISELY: ACCELERATING MCTS WITH VIRTUAL EXPANSIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

One of the most important AI research questions is to trade off computation versus performance, since "perfect rational" exists in theory but it is impossible to achieve in practice. Recently, Monte-Carlo tree search (MCTS) has attracted considerable attention due to the significant improvement of performance in varieties of challenging domains. However, the expensive time cost during search severely restricts its scope for applications. This paper proposes the Virtual MCTS (V-MCTS), a variant of MCTS that mimics the human behavior that spends adequate amounts of time to think about different questions. Inspired by this, we propose a strategy that converges to the ground truth MCTS search results with much less computation. We give theoretical bounds of the the proposed method and evaluate the performance in 9×9 Go board games and Atari games. Experiments show that our method can achieve similar performances as the original search algorithm while requiring less than 50% number of search times on average. We believe that this approach is a viable alternative for tasks with limited time and resources.

1 INTRODUCTION

When artificial intelligence was first studied in the 1950s, researchers seek to answer the question of what is the solution to the question if the agent were "perfect rational". The term "perfect rational" here refers to the decision made with an infinite amount of computations. However, without taking into consideration the practical computation time, one can only solve small-scale problems, since classical search algorithms usually exhibit exponential running time. Recent AI researches no longer seek to achieve "perfect rational", but instead carefully trade-off computation versus the level of rationality. People have developed computational models like "bounded optimality" to model these settings (Russell & Subramanian, 1994). The increasing level of rationality under the same computational budget has given us a lot of AI successes nowadays. Notable algorithms include the Monte-Carlo sampling algorithms, the variational inference algorithms, and using neural networks as universal function approximators (Coulom, 2006; Chaslot et al., 2008; Gelly & Silver, 2011; Silver et al., 2016; Hoffman et al., 2013).

More recently, MCTS-based RL algorithms have achieved a lot of success, mainly in board games. The most notable achievement is that AlphaGO beats Hui Fan in 2015 (Silver et al., 2016). This is the first time that a computer program beats a human professional player. After that, AlphaGo beats two top-ranking human players, Lee Sedol in 2016 and Jie Ke in 2017, the latter of which rank first worldwide at the time. Later, the MCTS-based RL algorithms are further extended to other board games, as well as the Atari video games (Schrittwieser et al., 2020). EfficientZero (Ye et al., 2021) greatly improves the sample efficiency of MCTS-based RL algorithms, shielding light on its future applications in real-world applications like robotics and self-driving.

Despite the impressive performance of MCTS-based RL algorithms, they require massive computations to train and evaluate. For example, Schrittwieser et al. (2020) used 1000 TPUs trained for 12 hours to learn the game of GO, and for a single Atari game, it needs 40 TPUs to train 12 hours. Compared to previous algorithms on the Atari games benchmark, it needs around two orders of magnitude more compute. This prohibitively large computational requirement has slowed down both the further development of MCTS-based RL algorithms, as well as practical use.

Under the hood, MCTS-based RL algorithms are model-based methods, that imagine what the futures look like when doing different future action sequences, just like what humans would do. How-

ever, this imaging process for the current method is not computationally efficient. For example, AlphaGo needs to look ahead 1600 game states to place a single stone. On the contrary, top human professional players can only think through around 100-200 game states per minute (Silver et al., 2016). Besides being computationally inefficient, the current MCTS algorithm deals with easy cases and hard ones with the same computational budget. On the other hand, human knows to use their time when it is most needed.

In this paper, we aim to design new algorithms that save the computational time of the MCTS-based RL methods. More specifically, we are interested in pushing the Pareto front of the rationality level - computation curve. Empirical results show that our method can achieve comparable performance while requiring less than 50% simulations to search on average.

2 Related Work

2.1 Multi-armed Bandit Problem

Reinforcement learning algorithms are always brought into the exploration and exploitation dilemma. Multi-armed bandit (MAB) problem (Berry & Fristedt, 1985; Auer et al., 2002; Lattimore & Szepesvári, 2020) is one of the most extensively studied but fundamental instances. The K-armed MAB problem is a sequential game with a collection of K unidentified but independent reward distributions, each associated with the corresponding arms. For each round, the learner pulls an arm and receives a reward sampled from the corresponding distributions. In general, the optimal policy of the learner for the MAB problem is to maximize the cumulative rewards obtained from the sequential decisions.

In the cases where the cost of pulling arms is little, the learner is allowed to trial and error for enough times until convergence. A series of upper confidence bound (UCB) algorithms (Auer et al., 2002; Bubeck & Cesa-Bianchi, 2012) are proposed to resolve the stochastic MAB problem and analyzed for the theoretical bound. However, when there exist costs for each trial, pure exploration methods are presented to make the best use of the given limited resources and finite trials (Bubeck et al., 2011; Lattimore & Szepesvári, 2020). To do further planning, Kocsis & Szepesvári (2006) proposed the UCT to adapt the UCB algorithms to the tree structures, which is the basis of those MCTS-based methods.

2.2 Reinforcement Learning with MCTS

Model-free learning (Mnih et al., 2013; Schulman et al., 2017; Haarnoja et al., 2018) and model-based learning (Clavera et al., 2018; Hafner et al., 2019) are two types of popular reinforcement learning algorithms. Model-based reinforcement learning (MBRL) is aimed to plan or train with the returned information of the next step from the environment or the learned model with the unavailable environment in contrast to the model-free one, which only utilizes the current step rewards. Therefore, MBRL is able to plan with the model so as to do a better search or plan. Recently, MCTS-based methods (Silver et al., 2016; Schrittwieser et al., 2020) have become increasingly popular and achieve some super-human performance for the strong ability of search. However, search is quite consuming, which obstacles the MCTS-based algorithms in the real-time domains.

3 Background

The AlphaGo series of work (Silver et al., 2016; 2017) are all MCTS-based reinforcement learning algorithms. Those algorithms assume the environment transition dynamics are known or learn the environment dynamics. Based on the dynamics, they use the Monte-Carlo tree search (MCTS) as the policy improvement operator. I.e. taking in the current policy, MCTS returns a better policy with the search algorithm. The systematic search allows the MCTS-based RL algorithm to quickly improves the policy, and perform much better in the setting where a lot of reasoning is required. MCTS is the core component in the algorithms like AlphaGo.

3.1 MCTS

In this part, we give a brief introduction to the MCTS method implemented in reinforcement learning applications. MCTS takes in the current MDP state and runs a search algorithm guided by the current policy function. It outputs an improved policy of the current state. The improved policy is later to select an action in the environment. More concretely, MCTS includes four stages per search iteration: selection with UCT, expand the unvisited node, evaluate the new node, backpropagation along the search path to update the Q-values $Q(s, a)$ and visit counts $N(s, a)$.

The selection step selects which node to visit next in the search tree. The MCTS algorithm iteratively builds a search tree under a selection rule named UCT (Kocsis & Szepesvári, 2006), as shown below:

$$a^k = \arg\max_{a \in \mathcal{A}} Q(s, a) + P(s, a) \frac{\sqrt{\sum_b N(s, b)}}{1 + N(s, a)} \left(c_1 + \log \left(\frac{\sum_b N(s, b) + c_2 + 1}{c_2} \right) \right) \quad (1)$$

where k is the index of the iterative step, $\mathcal{A}$ is the action set, $Q(s, a)$ is the estimated Q-value of given state s with action a, $P(s, a)$ is the policy prior obtained from the neural networks and $N(s, a)$ is how many times the tree selects the action a from the state s. The expansion step expands the selected node and updates the search tree. The evaluation step evaluates the new node with the value network. The final backpropagation step propagates the newly computed value to the root search node, obtaining more accurate Q values along the search path.

The output of MCTS is the visit count of each action of the root node. After the N search iterations, the normalized visit count distribution of the root is the output $\pi(s) = \pi_N(s)$, where $\pi_k(s, a) = (N(s, a))/\sum_{b \in \mathcal{A}} N(s, b) = N(s, a)/k, a \in \mathcal{A}$

3.2 Computation Requirement

Most of the computations in MCTS-based RL are the MCTS procedure. For each action taken by the MCTS, it needs N times neural network evaluations, where N is the number of search iterations in MCTS. Traditional RL algorithms, such as PPO () or DQN (), only need a single neural network evaluation per action. Thus, MCTS-based RL is roughly N times computationally more expansive than traditional RL algorithms.

In practice, training a single Atari game needs 12 hours of computation time on 40 TPUs (Schrittwieser et al., 2020). The computation need is roughly two orders of magnitude more than traditional RL algorithms (Schulman et al., 2017), although the final performance of MuZero is much better.

4 Method

In this paper, we propose an algorithm named virtual-MCTS to reduce the computation cost of the original MCTS-based RL algorithms. More concretely, we aim to push the front of the Pareto curve of the performance–computation trade-off.

Intuitively, human knows when to make a quick decision and when to make a slow decision in different circumstances. It gives humans the ability to overcome more difficult decision-making problems without wasting a lot of time on easy ones. This situation-aware behavior is absent in the current MCTS algorithm. We propose an MCTS algorithm variant that can behave like a human.

The virtual-MCTS is consists of two components, the virtual expansion to estimate the final visit count based on the current partial tree; the termination rule that decides when to terminate based on the hardness of the current scenario.

4.1 Termination Rule

We propose to terminate the MCTS early based on the current tree statistics. Intuitively, during the MCTS tree expansion process, if we find that recent searches do not further change the root visitation distribution, then we no longer need to search further. With this intuition in mind, we propose a simple modification to the MCTS search algorithm. Let π_k denote the root visitation

distribution at MCTS expansion iteration k. We propose to terminate when:

$$||\pi_k - \pi_{k/2}||_1 < \epsilon$$

where ϵ is a tolerance hyper-parameter. Note that, in the MCTS-based RL algorithm, not only the best arm matters but also the other arms as well. It is because MCTS is used in the exploration process, and we need to make sure proper exploration happens at the non-best arms.

This seems to be a heuristic rule, without any guarantees whether π_k will be close to π_N, where N is the original MCTS search iteration. However, we show that under certain conditions, a bound on $||\pi_k - \pi_{k/2}||_1$ implies a bound on $||\pi_k - \pi_N||_1$.

First of all, we list some notations: k is the index of the current search iteration and N is the number of total search simulations, $\hat{Q}_t(s,a)$ is the predicted q value outputs at the t-th iteration, $N_k(s,a)$ denotes the total visit counts of the action a from the state s after the k-th iteration, the Q value at k-th iteration is $Q_k(s,a) = \sum_{t=1}^{k} \hat{Q}_t(s,a)/N_k(s,a)$. The Lemma here gives a brief bound for the ranges of Q values at different iterations.

Lemma 1 *$\forall a \in \mathcal{A}$, given that $\hat{Q}_t(s,a) \in [-1,1]$ and the $Q_k(s,a) = \frac{\sum_{t=1}^{N_k(s,a)} \hat{Q}_t(s,a)}{N_k(s,a)}$, then at iteration $1 \leq k_1 < k_2 \leq N$, $Q_{k_2}(s,a) - Q_{k_1}(s,a) \leq (1 - \frac{N_{k_1}(s,a)}{N_{k_2}(s,a)})(Q_{k_1}(s,a) + 1)$.*

Intuitively, if the current policy candidate $\hat{\pi}_k$ is close enough to the previous one $\hat{\pi}_{k/2}$ under some conditions, then the future changes of Q values with $N - k$ further searches will be bounded in a small range. Furthermore, we can measure the distance between the oracle policy distribution π_N and our current policy candidate $\hat{\pi}_k$. If the policy candidates $\hat{\pi}_k$ and $\hat{\pi}_k/2$ are close enough, so are $\hat{\pi}_k$ and the oracle policy $\hat{\pi}_N$. As mentioned in the previous section, $\hat{\pi}_N$ is equal to π_N. Therefore we conclude that the current policy candidate $\hat{\pi}_k$ is a near-oracle policy if the termination rule $\left|\left|\tilde{\pi}_k(s) - \tilde{\pi}_{k/2}(s)\right|\right|_1 < \epsilon$ is satisfied.

Lemma 2 *$\forall a \in \mathcal{A}$, given that $r \in (0,1]$, if $\exists k \in [rN, N], \left|\left|\tilde{\pi}_k(s) - \tilde{\pi}_{k/2}(s)\right|\right|_1 < \epsilon$, then $||\tilde{\pi}_k(s) - \pi_N(s)||_1 < \epsilon + 1 - r$.*

Algorithm 1 Iteration of Search in MCTS

Current k-th iteration step:
Given: $\mathcal{A}, Q_k(s,a), P(s,a), N_k(s,a)$
$s \leftarrow s_{\text{root}}$
repeat do search
    $a^* \leftarrow UCT(Q, P, N)$
    $s \leftarrow$ next state(s, a^*)
until $N_k(s, a^*) = 0$
Predict the value $\hat{Q}(s,a)$ and $P(s,a)$
for s along the search path **do**
    $Q_{k+1}(s,a) = \frac{N_k(s,a) \cdot Q_k(s,a) + \hat{Q}(s,a)}{N_k(s,a)+1}$
    $N_{k+1}(s,a) = N_k(s,a) + 1$
end for
Return $Q_{k+1}(s,a), N_{k+1}(s,a)$

Algorithm 2 Iteration of Search in MCTS with Virtual Expansion

Current k-th iteration step:
Given: $\mathcal{A}, Q_k(s,a), P(s,a), N_k(s,a), \hat{N}_k(s,a)$

if Not init $\hat{N}_k(s,a)$ **then**
    Init: $\hat{N}_k(s,a) \leftarrow N_k(s,a)$
end if

$s \leftarrow s_{\text{root}}$
$a^* \leftarrow UCT(Q, P, \hat{N})$
$\hat{N}_k(s,a) \leftarrow \hat{N}_k(s,a) + 1$

Return $\hat{N}_k(s,a)$

4.2 Virtual Expansion in MCTS

In the derivation above, we assume π_i and π_j are directly comparable. Here π_i and π_j denotes two root node visit count distributions at iteration i and j respectively. However, because the tree is expanded with the UCT algorithm, they are not directly comparable. UCT is an algorithm that maintains the upper bound of the node values in the search tree. As the number of visits increases, the upper bound would be tighter and the latter visits will be more focused on the most promising part of the tree. Thus earlier visit count distribution (iteration number small) will exhibit more exploratory distribution, while latter visit count distribution (larger iteration number) will be more exploitative on promising part. Our earlier attempt without considering this effect all fails, see Section **??** for ablations.

To be able to compare π_i and π_j properly, we propose a method called virtual expanded MCTS. In a nutshell, it aligns two distributions by doing virtual UCT expansions until a common node number N. When the tree is expanded at iteration i, it has $N - i$ iterations to go. A normal expansion would require evaluating neural network $N - i$ times to get a more accurate $Q(s, a)$ estimate for each of the arms at the root node. Our proposed virtual expansion still expands $N - i$ times according to the UCT algorithm, but it ignores the $N - i$ neural network evaluations and simply assumes that each arm's $Q(s, a)$ does not change. We denote the virtual expanded distribution from π_i as $\hat{\pi}_i$. By doing virtual expansions to both π_i and π_j, we effectively remove the different levels of exploration/exploitation in them.

The comparisons between the MCTS and the one with virtual expansion are illustrated in Algorithm 1, 2. Here we display the complete one-step iteration of MCTS with or without virtual expansion. The time-consuming computations are highlighted in Algorithm 1. Line 4 to 7 in Algorithm 1 target at searching with UCT to reach an unvisited state for exploration. Then it will evaluate the state and backpropagate along the search path to fit a better estimation of Q values. After a total of N iterations, the visit count distribution of the root node is considered as the final policy distribution $\pi = \pi_N$. However, in the MCTS with virtual expansions, listed in Algorithm 2, it will only search one step from the root node and select the action based on the current estimations without changing any properties of the search tree. Furthermore, the virtual visited counts $\hat{N}_k(s, a)$ will change after virtual visits to balance the exploitation and the exploration issue. Then the policy candidate after virtual expansions becomes $\tilde{\pi}_k(s, a) = \hat{N}_k(s, a)/N$ instead of $N_k(s, a)/k$. In extreme cases that $k = N$, we have $\tilde{\pi}_N(s, a) = \pi_N(s, a)$. In this way, the final visit count distribution obtained through the virtual expansions keeps the same as the oracle one, given the condition that the Q value estimations $Q_k(s, a)$ change little after the next $N - k$ iterations.

4.3 V-MCTS Algorithm Summary

The procedure of MCTS with the termination rule is listed as Algorithm 3. Compared with the original implementation of MCTS, the line 5-10 are the pseudo code on the termination rule. In each iteration, we will do some calculations with little cost to judge whether the condition of termination is satisfied. If it is, then the search process will be terminated and return the current policy candidate. Thus, it can skip the next $N - k$ model predictions from neural networks in the evaluation part highlighted in line 5. In this way, we can approximate the oracle distribution π_N by $\hat{\pi}_k$ while reducing the budget of N simulations to k. Noticed that $k \geq rN$, and r are a hyperparameter of the minimum budget we define. Then we can reduce the tree size by $1/r$ times at most. However, r cannot be set to a tiny value as the minimum distance of distributions ϵ is bounded by r. Otherwise, the rule of termination will never be satisfied.

Algorithm 3 Virtual MCTS

1: Given budget N, state s, conservativeness r, error ϵ
2: **for** $k \in N$ **do**
3: Selection with UCT
4: Expansion for the new node
5: Evaluation with Neural Networks
6: Backpropagation for updating Q and visit counts
7: $\pi_k(s, a) \leftarrow N_k(s, a)/n$
8: Virtual expand $N - k$ nodes and update $\hat{N}(s, a)$
9: $\tilde{\pi}_k(s) \leftarrow \hat{N}_k(s, a)/N$
10: **if** $k \geq rN \wedge \left|\left|\tilde{\pi}_k(s) - \tilde{\pi}_{k/2}(s)\right|\right|_1 < \epsilon$ **then**
11: $\pi(s) \leftarrow \tilde{\pi}_k(s)$
12: **Break**
13: **end if**
14: $\pi(s) \leftarrow \pi_k(s)$
15: **end for**
16: **Return** $\pi(s)$

In conclusion, this rule tells the MCTS to terminate if the policy candidates have converged. Then the Q values after an extra $N - k$ iterations can be bounded in a small range by the current Q values. On such occasions, the virtual expansion method will have similar effects as the real expansion of MCTS to generate a near-oracle policy, which can save the time cost of the left $N - k$ simulations. Furthermore, we will do some ablations in the next section to further investigate the effects of the two hyperparameters r, ϵ. Finally, we name our method Virtual MCTS (V-MCTS), a variant of MCTS with a termination rule based on virtual expansion.

5 EXPERIMENTS

In this section, the goal of the experiments is to prove the effectiveness and efficiency of our proposed algorithm. We compare the performance as well as the cost of the budget of the MCTS-based methods with or without the termination rule. Specifically, we will evaluate the board game Go 9×9, and some Atari games. In addition, we do some ablations to further examine the effectiveness of the virtual expansion and figure out how sensitive our method is to the hyperparameters. Finally, we try to understand the adaptive mechanism including some visualizations and performance analysis.

5.1 SETUP

Recently, Ye et al. (2021) proposed EfficientZero, a variant of MuZero (Schrittwieser et al., 2020) with three extra components to improve the sample efficiency, which only requires 8 GPUs in training and can be more affordable. Here we choose the EfficientZero as our benchmark for the board game Go 9×9, a challenging planning problem, and some visually complex games on Atari.

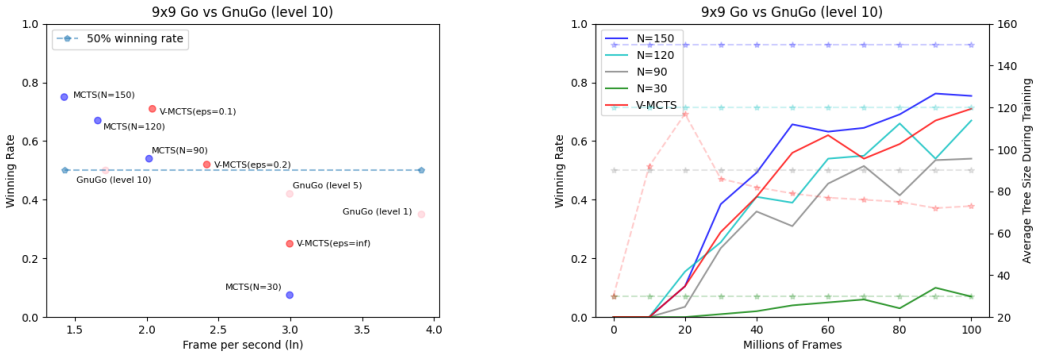
As for the Go 9×9, we choose the Chinese rules during training and evaluation. The environment of Go is build based on an open-source codebase, GymGo (Huang, 2021) and we evaluate the performance of the agent against GNU v3.8 at level 10 (Bump et al., 2005) for 200 games, which include 100 pieces as the black player and 100 pieces as the white one with different seeds. As for the Atari games, we choose 5 games with 100k environment steps, which follows the setting of EfficientZero. We evaluate all these games for 32 distinct seeds.

5.2 COMPARATIVE EVALUATION

We compare our method to the original version of EfficientZero, on Go 9×9 and some Atari games. Figure 1a illustrates the comparisons on Go among the different algorithms or models against the GnuGo (level 10) agent concerning the training speed and the winning rate. Here, we train the baseline method with different static budgets N, noted as the blue points. Besides, we also train the V-MCTS with hyperparameters $r = 0.2, \epsilon = 0.1$. And then we evaluate the trained model with different ϵ to display the tradeoff between performance and the time cost, noted as the red points. The pink points are the GnuGo with different levels.

The result shows that the performance of V-MCTS is comparable to the MCTS of maximum budget ($N = 150$) while it requires less time to search. Furthermore, it is notable that V-MCTS achieves a 72% winning rate against the GnuGo level 10. Meanwhile, the time cost of our method for a one-step move is 0.12s while the GnuGo engine is 0.18s. For the data points with the winning rate higher than 50%, the V-MCTS is significantly better than the original MCTS considering both the winning rate and the time cost. Consequently, we are in the belief that such termination rule of MCTS can keep the strong performance while saving lots of budgets, which means our method is effective enough and more time-efficient.

The training curve presented in Figure 1b illustrates that the budget of search times is quite significant to the MCTS method. However, the performance of V-MCTS is better than that of MCTS with $N = 120$ while maintaining an average tree size of less than 80. Furthermore, it is interesting to find that the tree size varies over training procedures. In the beginning, the outputs of the value network are close to zero because the agent usually draws in self-play and receives the zero reward signal, which means the little changes of $Q_k(s, a)$ during the search. The visit count distribution of the root after virtual expansions is similar to that after real expansions. Therefore, in this stage, the termination condition is easy to meet. Afterward, as the model is trained better and receives more diverse reward signals during self-play, the probability of searching some valuable states becomes much larger. The value $Q_k(s, a)$ varies considerably in each iteration of the search, which results in the changes of distributions of policy candidates. Thus, it is much more difficult for the search process to terminate and leads to a larger number of tree sizes. Finally, with more training steps, the prediction of the policy network is more accurate and then gives a stronger prior heuristic knowledge $P(s, a)$ before search. For those actions with higher prior knowledge, the changes of $Q_k(s, a)$ have less impact on the UCT scores. The procedure of the virtual expansion is similar to that of the real expansion. However, the changes of $Q_k(s, a)$ are still possible during the search process. Consequently, the average number of tree size keeps in a reasonable range, which is larger than that



(a) Evaluations of Performance

(b) Wining Rates and Tree Size on Training Stage

Figure 1: Performance of Virtual MCTS on Go 9×9 against GnuGo (level 10). (a) Evaluating the speed of search and winning rate of MCTS, V-MCTS as well as the GnuGO at different levels. The termination rule is able to reduce the search cost while keeping comparable performance and it outperforms the GnuGo at level 10 in the aspect of speed and winning rate. (b) Evaluating the winning rate as well as the average tree size in different training phases. The solid lines and dashed lines display the winning probability and the tree size respectively. The termination rule can make the tree size adaptive in training with a little loss of performance.

Table 1: Results from Atari games. Original scores over total 32 seeds on 5 environments.

	MCTS ($N = 10$)	**MCTS** ($N = 30$)	**MCTS** ($N = 50$)	**Ours**	**Size Avg.**
Pong	2.1	17.2	20.8	**19.94**	13
Breakout	309	347.4375	411.1	**389.2**	16
Seaquest	625.6	930.6	1737.5	**1340.1**	15
Hero	7310	**7499.1**	9715	7465.0	15
Qbert	6035.2	7792.9	15465.6	**10880.5**	17

in the innocent beginning stage and the minimum budget rN. Our method can determine whether to continue searching or not.

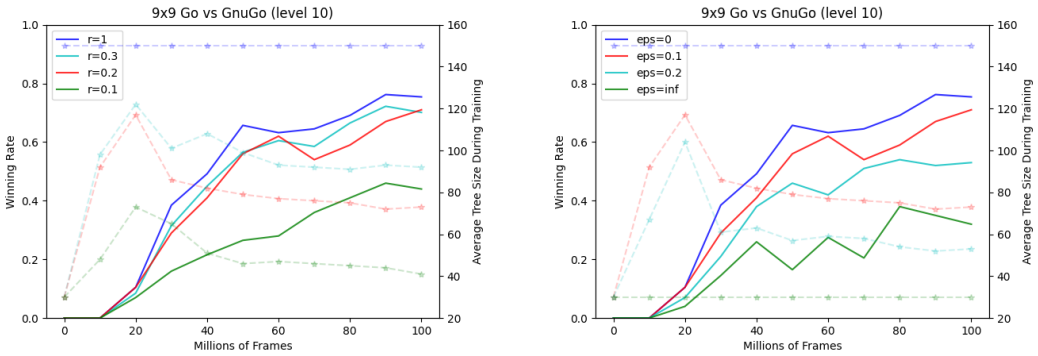
Apart from the results of Go, we also evaluate our method on some visually complex games. Since the search space of Atari games is much smaller than that of Go, here we study how the proposed method impacts the performance under less necessity of search. Besides, to understand the behavior of V-MCTS in the games which requires different levels of exploration, we choose some Atari games of distinct difficulty. We follow the setting of EffcientZero, 100k Atari benchmark, which contains only 400k frames data. The results are shown in Table 1. Generally, we find that our method works in Atari games. The tree size during a search is adaptive and the performance of V-MCTS is still comparable to the MCTS with full search trails. It has better performance than the MCTS($N = 30$) while requiring much fewer searches, which proves the effectiveness and the efficiency of the termination rule in the tree search method. The Hero game is not an outlier considering the similar performance between $N = 50$ and $N = 30$. Besides, we find that the number of search times decelerates more than that on Go. To sum up, Virtual MCTS shapes better policy candidates close to the oracles through the virtual expansion with less cost of the budget. The performance can keep sound and the savings of search cost is more substantial in the environment with less action space.

5.3 Ablation Study

The results in the previous section suggest that our method reduces the response time of MCTS while keeping considerable performance on challenging tasks such as the Go and Atari games. In this section, we try to figure out which component of our method contributes to the performance and how the hyperparameters affect it.

Table 2: Ablation results of different expansion methods on Go 9×9 against GnuGo (level 10)

Algorithm	Size Avg.	Winning Rate
Original expansion	30	16%
Greedy expansion	30	5%
Virtual expansion	30	32%



(a) Ablation of minimum budget $N_0 = rN$

(b) Ablation of minimum distance ϵ

Figure 2: Sensitivity of the termination rules to the hyperparameter r, ϵ on Go 9×9. The solid lines and dashed lines display the winning probability and the average tree size respectively.

Virtual Expansion In Section 4.2, we introduce the virtual expansion and discuss the difference between the MCTS with and without virtual expansion. We compare the MCTS with virtual expansion and another two expansion methods. Here we will introduce the two methods briefly. One is the original expansion, which does nothing once termination. It will sample an action directly from $\pi_k(s,a) = N_k(s,a)/k$. Another is the greedy expansion, which will spend the left $N - k$ simulations in searching the current best action greedily, indicating that $\hat{\pi}_k(s,a) = (N_k(s,a) + (N-k)\mathbb{I}_{a=\arg\max N_k(s,a)})/N$. Besides, we turn off the termination rules and stop the search process after N_0 iterations, where $N_0 = rN$ and $r = 0.2, N = 150$.

We compare the winning rate against the GnuGo engine and the results are listed as Table 2 shows. All of them only search for the given minimum tree size, and the virtual expansion method can still achieve a 32% winning rate, which is much better than the others. Notably, greedy expansion does not work. It is over exploitation and results in severe exploration issues. Consequently, the virtual expansion method can generate a better policy distribution because it can balance the exploration and exploitation problem through UCT with further virtual simulations.

Termination Rule Since the virtual expansion provides a better choice of policy distributions, it is significant to explore a better termination rule to keep the sound performance while decreasing the tree size as much as possible. As mentioned in Section 4.1, the termination rule sets two hyperparameters r, ϵ to determine the termination rule. Then we will do ablations for the different values of r and ϵ respectively. The default values of r, ϵ are set to $0.2, 0.1$ in all experiments here.

Figure 2 compares the winning rate as well as the average tree size across the training stage. Firstly, Figure 2a gives the results of different minimum search times r. The winning probability is not sensitive to r because the values are similar when $r \geq 0.2$. But the average tree size is sensitive to r because V-MCTS is supposed to search for at least rN times. In addition, there is a drop between the performance between $r = 0.1$ and $r = 0.2$. Therefore, it is reasonable to choose $r = 0.2$ to balance the speed and the performance.

Besides, the comparisons of different minimum distance ϵ are shown in Figure 2b. A larger ϵ will result in a smaller tree size for $\left|\left|\tilde{\pi}_k(s) - \tilde{\pi}_{k/2}(s)\right|\right|_1 < \epsilon$ is easier to reach. In practical, we find that the performance is highly correlated with ϵ. In terms of the winning probability, a smaller ϵ outperforms a larger one. However, the better performance is at cost of a larger response time.

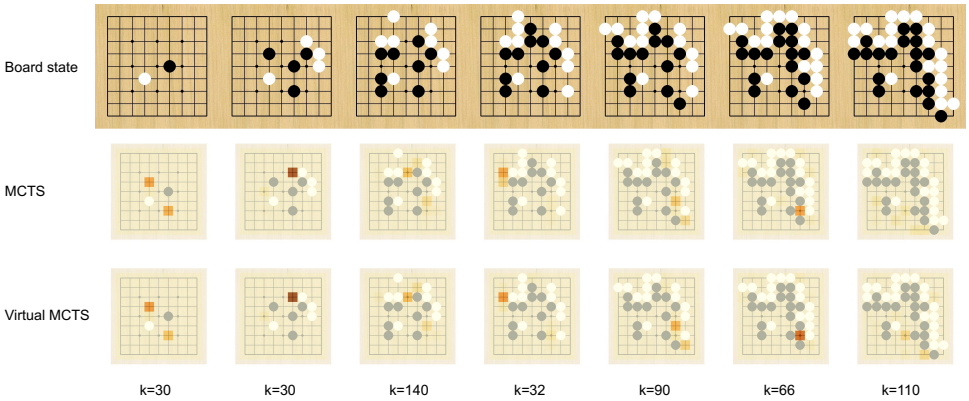


Figure 3: Heatmap of policy distributions from the MCTS ($N = 150$) and the V-MCTS ($r = 0.2, \epsilon = 0.1$). A darker red color represents a larger visit counts of the corresponding action. The V-MCTS will terminate with different search times k according to the situations and generate a near-oracle policy distribution.

Notably, $\epsilon = 0.1$ can reach comparable results to the $\epsilon = 0$ but reduce the tree size in half. Therefore, it is a good choice to set $r = 0.2, \epsilon = 0.1$. We suggest selecting an appropriate minimum distance to balance the performance and the response time.

5.4 Visualization

In this section, we will do some visualizations to better understand the behavior of Virtual MCTS.

Specifically, we choose some stages on one game of Go against the GnuGo engine and visualize the heatmap of policy distribution, as Figure 3 shows. Here we set the komi to 6.5 as most papers do, and our player is the black one in this figure. The board states are shown in the first row, and the next two rows are the heatmap visualization for MCTS ($N = 150$) and V-MCTS with $r = 0.2, \epsilon = 0.1$. The darker the color is on the grid, the more the corresponding action is visited during the search. In general, $\hat{\pi}_k$ is close to the π_N at distinct stages, which indicates that our termination rule is reasonable and effective. The less valuable actions there are, the earlier stage the V-MCTS will terminate on. For example, the termination occurs earlier in the states of columns 1, 2, 4 where there are fewer hot points in the heatmap of the oracle distribution. But it is the opposite when the situation is more complicated, especially in the closing stages of the game. Notably, the termination step k is not related to the number of Go pieces. Therefore, V-MCTS makes adaptive terminations according to the situations of the current state, to save computations while maintaining comparable performances. And the policy candidate obtained after virtual expansion can be close to the oracle one at distinct states of a game.

6 Discussion

In this paper, we propose a novel method named V-MCTS to accelerating the MCTS to determine the termination of searches. It can keep similar performances while reducing half of the time to search adaptively. We are in the belief that this work can be one step toward applying the MCTS-based methods to some real-time domains.

7 REPRODUCIBILITY STATEMENT

The main implementations of our proposed method are in Algorithm 2 and 3. In addition, the settings of the experiments and hyper-parameters we choose are in Appendix A.1. The proof of the lemma is in Appendix A.2. More significantly, the details of the design of training procedures for Go are around Appendix A.1.2. Besides, we will release the codebase if this paper is accepted.

REFERENCES

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

Donald A Berry and Bert Fristedt. Bandit problems: sequential allocation of experiments (monographs on statistics and applied probability). *London: Chapman and Hall*, 5(71-87):7–7, 1985.

Sébastien Bubeck and Nicolo Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *arXiv preprint arXiv:1204.5721*, 2012.

Sébastien Bubeck, Rémi Munos, and Gilles Stoltz. Pure exploration in finitely-armed and continuous-armed bandits. *Theoretical Computer Science*, 412(19):1832–1852, 2011.

Daniel Bump, Man Lung Li, Wayne Iba, and et al. Gnugo, 2005. URL `http://www.gnu.org/software/gnugo/gnugo.html`.

Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai. *AIIDE*, 8:216–217, 2008.

Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel. Model-based reinforcement learning via meta-policy optimization. In *Conference on Robot Learning*, pp. 617–629. PMLR, 2018.

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

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

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14(5), 2013.

Eddie Huang. Gymgo. `https://github.com/aigagror/GymGo`, 2021.

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

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

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Stuart J Russell and Devika Subramanian. Provably bounded-optimal agents. *Journal of Artificial Intelligence Research*, 2:575–609, 1994.

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

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering atari games with limited data. In *NeurIPS*, 2021.

A Appendix

A.1 Experimental setup

A.1.1 Models and Hyper-parameters

Model Design As for the architecture of the networks, we follow the implementation of EfficientZero (Ye et al., 2021) in Atari games.

Since the board game Go is harder than the Atari games, we add more residual blocks (two times blocks) and remove the downsampling part because there is no need to do downsampling for Go states.

Besides, considering that the reward is sparse in Go (only the final value) and sufficient data, we only add the self-consistency component in EfficientZero to give more temporal supervision during training.

Hyper-parameters In each case, we train EfficientZero for unrolled $K = 5$ steps and mini batches of size 256. Besides, the model is trained for 100k batches with 100M frames of data in board games while 100k batches with 400k frames in Atari games. We stack 8 frames in board games without frameskip while stacking 4 frames in Atari games with a frameskip of 4. During both training and evaluation, EfficientZero chooses 150 simulations of budget for each search in board games while 50 simulations of budget in Atari games. Other hyper-parameters are listed as follows:

	Go 9×9	**Atari**
Maximum number of tree size	150	50
Observation down-sampling	No	96×96
Total frames	100M	400k
Replay buffer size	2M	100k
Max frames per episode	163	108k
Cost of training time	24h	8h
Komi of Go	6.5	-
Frame stack	8	4
Frame skip	1	4
Training steps	100k	100k
Mini batch	256	256
Learning rate	0.05	0.2
Policy loss coefficient	1	1
Reward loss coefficient	0	1
Value loss coefficient	1	0.25
Consistency loss coefficient	2	0.5
ϵ	0.1	0.1
r	0.2	0.2

Table 3: Hyper-parameters of V-MCTS in board games and Atari games

A.1.2 TRAINING DETAILS OF GO

The detailed implementations of Atari games are discussed in EfficientZero (Ye et al., 2021). However, it is nontrivial to adapt to the board games. We follow the designs of AlphaZero and we use the Chinese rules as most of work do. The input states of the go board are encoded into a $17 \times 9 \times 9$ array which stacks the historical 8 frames and uses the last channels to identify the current player, 0 for black and 1 for white. Notably, the one historical frame consists of two planes, where the first one plane represents the stones of the current player and the second one represents the stones of the opponent. Besides, if there is a stone on board, then the state of the corresponding position in the frame will be set to 1, otherwise to 0. All the other implementations follows the Alpha-series paper (Silver et al., 2016; 2017; Schrittwieser et al., 2020).

The value network will always predict the Q-value of the black player. Therefore, we should do some changes in the MCTS with the two-player game. A significant change is that during backpropagation of MCTS, the value should be updated with the negative value from the child node, which is different from that of one-player games. Because the child node is the opponent, the higher value of the opponent indicates a worse value of the current player.

A.2 PROOF

Lemma 1. *$\forall a \in \mathcal{A}$, given that $\hat{Q}_t(s,a) \in [-1,1]$ and the $Q_k(s,a) = \frac{\sum_{t=1}^{N_k(s,a)} \hat{Q}_t(s,a)}{N_k(s,a)}$, then at iteration $1 \leq k_1 < k_2 \leq N$, $Q_{k_2}(s,a) - Q_{k_1}(s,a) \leq (1 - \frac{N_{k_1}(s,a)}{N_{k_2}(s,a)})(Q_{k_1}(s,a) + 1)$.*

proof. Since $Q_k(s,a) = \frac{\sum_{t=1}^{k} \hat{Q}_t(s,a)}{N_k(s,a)}$, we have

$$Q_{k_2}(s,a) = \frac{N_{k_1}(s,a)Q_{k_1}(s,a) + \sum_{t=N_{k_1}(s,a)}^{N_{k_2}(s,a)} \hat{Q}_t(s,a)}{N_{k_2}(s,a)} \tag{2}$$

Because $\hat{Q}_t(s,a) \in [-1,1]$, then

$$Q_{k_2}(s,a) \leq \frac{N_{k_1}(s,a)}{N_{k_2}(s,a)} Q_{k_1}(s,a) + \frac{N_{k_2}(s,a) - N_{k_1}(s,a)}{N_{k_2}} \tag{3}$$

Therefore

$$Q_{k_2}(s,a) - Q_{k_1}(s,a) \leq (1 - \frac{N_{k_1}(s,a)}{N_{k_2}(s,a)})(Q_{k_1}(s,a) + 1) \tag{4}$$

Lemma 2. *Given that $r \in (0,1]$, if $\exists k \in [rN, N], \left|\left|\tilde{\pi}_k(s) - \tilde{\pi}_{k/2}(s)\right|\right|_1 < \epsilon$, then $||\tilde{\pi}_k(s) - \pi_N(s)||_1 < \epsilon + 1 - r$*

proof. Since the $\tilde{\pi}_k(s,a)$ is a distribution that equals to $\frac{\hat{N}_k(s,a)}{N}$. Therefore,

$$||\tilde{\pi}_N(s) - \tilde{\pi}_k(s)||_1 \leq \left|\left|\tilde{\pi}_N(s) - \tilde{\pi}_{N/2}(s)\right|\right|_1 + ||\tilde{\pi}_N(N/2) - \tilde{\pi}_k(s)||_1 \tag{5}$$

Then

$$||\tilde{\pi}_N(s) - \tilde{\pi}_k(s)||_1 \leq \sum_{t=0}^{M} \left|\left|\tilde{\pi}_{\frac{N}{2^t}}(s) - \tilde{\pi}_{\frac{N}{2^{t+1}}}(s)\right|\right|_1 + \left|\left|\tilde{\pi}_k(s) - \tilde{\pi}_{\frac{N}{2^{M+1}}}(s)\right|\right|_1 \tag{6}$$

, where $\frac{N}{2^{M+2}} \leq k \leq \frac{N}{2^{M+1}}$. In addition, $\left|\left|\tilde{\pi}_k(s) - \tilde{\pi}_{\frac{N}{2^{M+1}}}(s)\right|\right|_1 \leq \left|\left|\tilde{\pi}_k(s) - \tilde{\pi}_{k/2}(s)\right|\right|_1 = \epsilon$. Besides, $k \geq rN$, then we have

$$\begin{aligned} ||\tilde{\pi}_N(s) - \tilde{\pi}_k(s)||_1 &\leq \sum_{t=0}^{M} \left|\left|\tilde{\pi}_{\frac{N}{2^t}}(s) - \tilde{\pi}_{\frac{N}{2^{t+1}}}(s)\right|\right|_1 + \epsilon \\ &\leq 1 + \epsilon - \frac{1}{2^M} \\ &\leq \epsilon + 1 - r \end{aligned} \tag{7}$$