Multiagent Gumbel MuZero: Efficient Planning in Combinatorial Action Spaces

Xiaotian Hao[1], Jianye Hao[1,2*], Chenjun Xiao[2], Kai Li[2], Dong Li[2], Yan Zheng[1]

[1]College of Intelligence and Computing, Tianjin University
[2]Noah's Ark Lab, Huawei
{xiaotianhao, jianye.hao, yanzheng}@tju.edu.cn, {xiaochenjun, likai210, lidong106}@huawei.com

Abstract

AlphaZero and *MuZero* have achieved state-of-the-art (SOTA) performance in a wide range of domains, including board games and robotics, with discrete and continuous action spaces. However, to obtain an improved policy, they often require an excessively large number of simulations, especially for domains with large action spaces. As the simulation budget decreases, their performance drops significantly. In addition, many important real-world applications have combinatorial (or exponential) action spaces, making it infeasible to search directly over all possible actions. In this paper, we extend *AlphaZero* and *MuZero* to learn and plan in more complex multiagent (MA) Markov decision processes, where the action spaces increase exponentially with the number of agents. Our new algorithms, *MA Gumbel AlphaZero* and *MA Gumbel MuZero*, respectively without and with model learning, achieve superior performance on cooperative multiagent control problems, while reducing the number of environmental interactions by up to an order of magnitude compared to model-free approaches. In particular, we significantly improve prior performance when planning with much fewer simulation budgets. The code and appendix are available at https://github.com/tjuHaoXiaotian/MA-MuZero.

Introduction

AlphaZero family algorithms have achieved great success in many applications, such as playing board games like Go and chess (Silver et al. 2016, 2017, 2018; Schrittwieser et al. 2020), making planning decisions in robotics and autonomous vehicles (Lenz, Kessler, and Knoll 2016), predicting the structure of large protein complexes (Bryant et al. 2022), and discovering faster matrix multiplication algorithms (AlphaTensor) (Fawzi et al. 2022) and sorting algorithms (AlphaDev) (Mankowitz et al. 2023), etc.

However, these algorithms usually require a large number of simulations before making a decision, which is computationally expensive. For example, in the board games Go, chess, and Shogi, the search is run for 800 simulations per move to pick an action (Schrittwieser et al. 2020). AlphaTensor also uses 800 simulations for each MCTS (Fawzi et al. 2022). Hamrick et al. (2020); Grill et al. (2020) find that *AlphaZero* performs poorly when the simulation budgets are low. Thus, we cannot reduce the computational cost by simply decreasing the simulation budget.

In the real world, many important applications have combinatorial action spaces(Kemmerling, Lütticke, and Schmitt 2023; Ausiello et al. 2012). These problems include multiagent control, power grid management, combinatorial auctions, resource allocation problems like job scheduling, routing problems like capacitated vehicle routing, chip design, portfolio optimization, and optimizing large language models like ChatGPT (Ouyang et al. 2022), etc.

Applying *AlphaZero* to such problems is challenging as larger action spaces require much more simulations to keep a good performance. To improve the sample efficiency, various methods have been proposed. Instead of recomputing the Q-values at the root node from scratch through simulations, Hamrick et al. (2020) integrate a model-free Q-learning agent which learns a global Q-function to give a better initialization of the Q-values in the tree, thus reducing the simulation numbers. Grill et al. (2020) show that *AlphaZero*'s search heuristics is an approximated solution to a regularized policy optimization problem and propose to directly compute the exact solution instead of using more simulations to approximate. Danihelka et al. (2022) exploit the idea of Grill et al. (2020) further and propose *Gumbel MuZero*, employing the Gumbel top-k trick to sample k actions at the root node to further reduce the simulation cost.

While these methods could mitigate the performance drop due to the low simulation budgets, they still cannot be directly applied to problems with combinatorial action spaces where fully enumerating all possible actions is infeasible. Hubert et al. (2021) propose *Sampled MuZero*, which extends *MuZero* to more complex action spaces such as high-dimensional continuous ones. Rather than enumerating all possible actions, its idea is to sample a small subset of actions with replacement and do policy evaluation and improvement with respect to the sampled actions. However, using sampling with replacement can be inefficient because it may resample high-probability actions over and over again, which are less informative. Besides, like *MuZero*, it has to use more simulations to approximate the exact solution shown in (Grill et al. 2020). When using a smaller number of simulations, it even cannot guarantee a policy improvement (Danihelka et al. 2022).

In this paper, combining the strengths of both *Sam-*

*Corresponding author: Jianye Hao (jianye.hao@tju.edu.cn)

pled MuZero and *Gumbel MuZero*, we propose *MA Gumbel MuZero*, which extends *MuZero* to combinatorial action spaces. Specifically, we consider the challenging cooperative multiagent control problems, where a potentially large number of agents jointly optimize a global reward function, which leads to exponential growth in the action space. Key contributions of this work are summarized below:

- We extend *MuZero* to combinatorial action control problems and propose two novel algorithms: *MA Gumbel AlphaZero* and *MA Gumbel MuZero*.
- We adopt the idea of stochastic beam search to efficiently sample top-k joint actions without replacement from the combinatorial action space and devise a well-matched sample-based policy improvement operator.
- Compared with *Sampled MuZeo*, ❶ we use sampling without replacement instead of sampling with replacement during both the selection and expansion phases of MCTS, which is more sample efficient. ❷ We use the exact solution to the regularized policy optimization as the improved policy instead of using the visit count distribution to approximate. A full comparison of the *MuZero* family algorithms is summarized in Table 1.
- We delicately design a world model for accurately modeling state transitions in multiagent domains.
- Extensive experiments validate that our method reliably outperforms the prior algorithms especially when using much fewer simulation budgets.

Preliminaries and Related Work

Notations. For more compact notation, we use bold fonts to denote vector quantities and omit the dependency of quantities on state s when the context is clear. For example, we use $\mathbf{q} \in \mathbb{R}^{|\mathcal{A}|}$ and $\boldsymbol{\pi} \in \mathbb{R}^{|\mathcal{A}|}$ to denote the vectors of $Q_\pi(s,\cdot)$ and policy $\pi(\cdot|s)$, i.e., $\mathbf{q}[a] = Q(s,a)$ and $\boldsymbol{\pi}[a] = \pi(a|s)$.

Monte-Carlo Tree Search in *MuZero*

Monte-Carlo Tree Search (MCTS) (Browne et al. 2012) is a famous heuristic search algorithm, which uses a resettable simulator or an environment model to explore possible future states and actions, with the aim of finding a better action for the current state. Each search procedure iteratively performs 3 steps: selection, expansion, and backpropagation.

Selection. MCTS starts traversing the search tree from the current state s^0 (root node) and selects the most promising action according to the pUCT formula (Silver et al. 2018):

$$a^k = \underset{a}{\operatorname{argmax}} \left[Q(s^k,a) + c \cdot \pi_\theta(a|s^k) \frac{\sqrt{\sum_b N(s^k,b)}}{1+N(s^k,a)} \right] \quad (1)$$

where $Q(s^k,a)$ is the currently estimated value of taking action a from state s^k and $\pi_\theta(a|s^k)$ is the prior probability of selecting a in current policy. $N(s^k,a)$ is the number of times of taking a from s^k and $N(s^k,b)$ denotes that of a's siblings. c is a numerical constantwhich control the relative importance of the Q-value and the exploration term. After selecting a^0, the search moves from node s^0 to node s^1. This selection procedure is repeated for $T-1$ times until a new action a^{T-1} that has not been explored is selected at s^{T-1}.

Expansion. The unexplored a^{T-1} is executed in the simulator (or world model), resulting in a reward r_{T-1} and a new state s^T. The new state s^T is appended to the tree, and its value $V(s^T)$ is estimated via a state-value function $V_\theta(s)$.

Backpropagation. The reward r_{T-1} and the value $V(s^T)$ of the new leaf node are used to update the values of all its parent states along the search path. For each state $s^k, k \in \{T-1,...,0\}$, we get a $T-k$-step estimate of the cumulative discounted reward bootstrapping from $V(s^T)$:

$$G^k = \sum_{j=k}^{T-1} \gamma^{j-k} r_j + \gamma^{T-k} V(s^T) \quad (2)$$

γ is the discount factor. For $k \in \{T-1,...,0\}$, we update:

$$\begin{aligned} Q(s^k,a^k) &:= \frac{N(s^k,a^k)\cdot Q(s^k,a^k) + G^k}{N(s^k,a^k)+1} \\ N(s^k,a^k) &:= N(s^k,a^k)+1 \end{aligned} \quad (3)$$

Get the improved policy. We refer to every 3-step (selection, expansion, and backpropagation) as one simulation. MCTS repeats the search process for N_{sim} simulations. N_{sim} is a parameter depending on the computational budget. Then we get the visit count distribution under the root node s^0:

$$\hat{\pi}(a|s^0) \triangleq \frac{N(s^0,a)^{1/\tau}}{\sum_b N(s^0,b)^{1/\tau}} \quad (4)$$

where τ is the temperature parameter. $\hat{\pi}(a|s^0)$ will generally reflect how good each action is and can be considered as the improved policy. During training, one action is sampled from $\hat{\pi}(a|s^0)$ to execute in the environment and $\pi_\theta(a|s^0)$ is updated towards $\hat{\pi}(a|s^0)$ by minimizing the KL-divergence loss $D_{\text{KL}}(\hat{\pi}(\cdot|s^0), \pi_\theta(\cdot|s^0))$.

Sampled MuZero

Hubert et al. (2021) propose *Sampled MuZero*, extending *MuZero* to domains with continuous action spaces. It makes 2 changes. ❶ Instead of planning over the original large action space, it uses the Monte Carlo method to sample k actions with replacement according to $\pi_\theta(a|s^k)$ as candidate actions to search and expand. ❷ During MCTS, it replaces $\pi_\theta(a|s^k)$ in Eq. 1 with $\hat{\beta}(a|s^k)$, where $\hat{\beta}(a|s^k)$ is the empirical distribution of the k samples, and only selects actions within the k samples. For all other parts, it keeps the same with *MuZero*. As k increases, the performance will gradually approach the one considering all actions. However, *Sampled MuZero* still has the same problems as *MuZero*.

Problems of *MuZero*'s Action Selection

Grill et al. (2020) provide deeper analysis about the policy update of *MuZero* and find its action selection criteria can be interpreted as approximating the solution to the following regularized policy optimization problem:

$$\hat{\boldsymbol{\pi}} \approx \bar{\boldsymbol{\pi}} \triangleq \underset{\mathbf{y}\in\mathcal{S}}{\arg\max} \left[\mathbf{q}^\top \mathbf{y} - \lambda \operatorname{KL}(\boldsymbol{\pi}_\theta, \mathbf{y}) \right], \quad (5)$$

where $\boldsymbol{\pi}_{\boldsymbol{\theta}}$ and $\mathbf{q}$ denote the vectors of the prior policy and Q-values used in Eq. 1, $\mathcal{S}$ is the $|\mathcal{A}|$-dimensional simplex, and $\operatorname{KL}(\boldsymbol{\pi}_\theta, \mathbf{y}) \triangleq \sum_a \boldsymbol{\pi}_\theta[a] \log \frac{\boldsymbol{\pi}_\theta[a]}{\mathbf{y}[a]}$.

The problem is that *MuZero* does not use the search Q-values to compute the improved policy, but instead uses the indirect visit count distribution $\hat{\pi}$ to approximate. This degrades the performance especially when the low simulation budgets are low. The reasons are: ❶ when a new high-value leaf is discovered, many additional simulations are needed before this information can be reflected in $\hat{\pi}$; ❷ By definition (Eq. 4), $\hat{\pi}$ is the ratio of two integers and has limited expressiveness when N_{sim} is low; ❸ Due to the deterministic action selection (Eq. 1), it may require a large number of simulations before some actions can be sampled. Thus, approximating Eq. 5 with $\hat{\pi}$ may require prohibitively large numbers of simulations, especially for domains with large-scale action spaces. As evidenced by (Hamrick et al. 2020; Grill et al. 2020; Danihelka et al. 2022), the performance of *MuZero* drops significantly as N_{sim} decreases.

Gumbel MuZero

With this insight, *MuZero-ALL* (Grill et al. 2020) directly use the exact solution of Eq. 5 as the policy target and use it to guide the tree search. Similarly, *Muesli* (Hessel et al. 2021) and *Gumbel MuZero* (Danihelka et al. 2022) use the exact solution to the following policy optimization:

$$\boldsymbol{\pi}_{\text{MPO}} \triangleq \arg\max_{\mathbf{y}\in\mathcal{S}} \left[\mathbf{q}^\top \mathbf{y} - \lambda\, \text{KL}\left(\mathbf{y}, \boldsymbol{\pi}_\theta\right)\right] \tag{6}$$

replacing $\text{KL}\left(\boldsymbol{\pi}_\theta, \mathbf{y}\right)$ in Eq. 5 with $\text{KL}\left(\mathbf{y}, \boldsymbol{\pi}_\theta\right)$, which is also adopted by the MPO algorithm (Maximum a Posteriori policy Optimization) (Abdolmaleki et al. 2018). We denote it as $\boldsymbol{\pi}_{\text{MPO}}$ for convenience. The exact solution to Eq. 6 is:

$$\pi_{\text{MPO}}(a|s) = \text{softmax}\Big(\log(\boldsymbol{\pi}_\theta) + \frac{\mathbf{q}}{\lambda})\Big)\,[a] \tag{7}$$

Appendix B.1 gives a detailed derivation. *Gumbel MuZero* makes 5 critical changes to *MuZero* from 3 aspects:

(1) Search: select actions inner MCTS. ❶ Sample actions to search at the root node. As *MuZero* selects actions according to Eq. 1 deterministically, some actions may never be selected especially when $N_{\text{sim}} < |\mathcal{A}|$. To help explore different actions, *Gumbel MuZero* randomly samples k actions without replacement according to π_θ as the candidate actions to search by using the Gumbel-Top-k trick (Vieira 2014).

Gumbel-Top-k Trick. Sampling k actions without replacement from the categorical distribution $\pi_\theta(\cdot|s)$ is equivalent to taking top-k actions by:

$$\mathcal{A}^k_{\text{top}} \triangleq \{a_1, \ldots, a_k\} = \arg\operatorname*{top}_k \left(\log(\boldsymbol{\pi}_\theta) + \boldsymbol{g}_0\right) \tag{8}$$

$\log(\boldsymbol{\pi}_\theta) \in \mathbb{R}^{|\mathcal{A}|}$ is the log-probability vector of $\pi_\theta(\cdot|s)$ and $\boldsymbol{g}_0 \in \mathbb{R}^{|\mathcal{A}|}$ is a vector of independent Gumbel noises: $\boldsymbol{g}_0[a] \sim \text{Gumbel}(0), a \in \mathcal{A}$ i.i.d. *Gumbel MuZero* only searchs over the sampled top-k actions $\mathcal{A}^k_{\text{top}}$ at the root node.

❷ Select actions at the root node according to:

$$a^{\text{root}} \triangleq \arg\max_{a\in\mathcal{A}^k_{\text{top}}} \left(\boldsymbol{g}_0[a] + \log(\boldsymbol{\pi}_\theta[a]) + \sigma\left(\mathbf{q}[a]\right)\right) \tag{9}$$

where $\sigma(\mathbf{q}) = \left(c_{\text{visit}} + \max_b N(b)\right) c_{\text{scale}}\, \mathbf{q}$ is a monotonically increasing transformation function, controlling the scale[1] of $Q(s,a)$. $c_{\text{visit}} > 0$ and $c_{\text{scale}} > 0$ are constants. $\max_b N(b)$ is the visit count of the most visited action. Eq. 9 guarantees a policy improvement when $\mathbf{q}$ values are correctly evaluated, because for $\forall \boldsymbol{g}_0 \sim \text{Gumbel}(0)$, we have

$$\mathbf{q}[a^{\text{root}}] \geq \mathbf{q}[\arg\max_{a\in\mathcal{A}^k_{\text{top}}} \left(\boldsymbol{g}_0[a] + \log(\boldsymbol{\pi}_\theta[a])\right)] \tag{10}$$

By the Gumbel-Max trick (Vieira 2014), sampling a from $\pi_\theta(\cdot|s)$ is equivalent to $\arg\max\left(\boldsymbol{g}_0[a] + \log(\boldsymbol{\pi}_\theta[a])\right)$. Thus, $\mathbb{E}_{\boldsymbol{g}_0}[\mathbf{q}[a^{\text{root}}]] \geq \mathbb{E}_{\boldsymbol{g}_0}[\text{argmax}(\boldsymbol{g}_0[a] + \log(\boldsymbol{\pi}_\theta[a]))] = \mathbb{E}_{\pi_\theta}[\mathbf{q}[\mathbf{a}]]$. To minimize simple regret, *Gumbel MuZero* applies the *Sequential Halving* algorithm (Karnin, Koren, and Somekh 2013) at the root node. The *Sequential Halving* algorithm evenly divides the N_{sim} simulation budget into $\log_2(k)$ search phases. At each search phase $j \in \{0, \ldots, \log_2(k)\}$, it picks the top-$\frac{k}{2^j}$ remaining actions according to Eq. 9 as considered actions and visits these actions evenly often.

❸ Select actions at non-root nodes. *Gumbel MuZero* does not sample actions at non-root nodes. It sets λ in Eq. 7 to $\frac{1}{(c_{\text{visit}} + \max_b N(b)) c_{\text{scale}}}$ and designs a deterministic action selection with the smallest mean-squared-error between the improved policy $\pi_{\text{MPO}}(\cdot|s)$ and the normalized visit counts.

$$\pi_{\text{MPO}}(\cdot|s) = \text{softmax}(\log(\boldsymbol{\pi}_\theta) + \sigma(\mathbf{q}))) \tag{11}$$

$$a^{\text{non-root}} \triangleq \arg\max_a \left(\pi^{\text{MPO}}(a) - \frac{N(a)}{1 + \sum_b N(b)}\right) \tag{12}$$

(2) Act: select actions in the environment. ❹ *Gumbel MuZero* selects the singular action resulting from the Sequential Halving search procedure according to Eq. 9.

(3) Learn: outer-policy improvement. ❺ *Gumbel MuZero* updates the policy network π_θ towards the improved policy π_{MPO} (Eq. 11) at the root node by minimizing the KL-divergence $D_{\text{KL}}(\pi_{\text{MPO}}(\cdot|s^0), \pi_\theta(\cdot|s^0))$, which guarantees a policy improvement when $\mathbf{q}$ are correctly evaluated.

Gumbel MuZero matches the SOTA on Go, chess, and Atari, and its performance is less affected by a small number of simulations (Danihelka et al. 2022). Since the simulation budget depends on the number of actions, these improvements are critical in tasks with large numbers of actions, e.g., multiagent control problems with exponential action spaces.

Problem Description

Our target is to extend *MuZero* to more complex Multiagent Markov Decision Processes (MMDPs) (Boutilier 1996; Bernstein et al. 2002). An MMDP can be defined as a tuple $\langle \mathcal{N}, \mathcal{S}, \mathcal{A}, P, r \rangle$, where $\mathcal{N} = \{1, 2, \cdots, n\}$ is the set of n agents, $\mathcal{S}$ is the state space, $\mathcal{A} = A_1 \times \cdots \times A_n$ is the joint action space, $P\left(s, \vec{a}, s'\right) \in [0,1]$ denotes the probability of transiting to state s' when applying the joint action $\vec{a} = \langle a_1, \cdots a_n \rangle$ at s, and $r\left(s, \vec{a}, s'\right) \in \mathbb{R}$ denotes the immediate global reward after taking action $\vec{a}$ at state s and ending at state s'. The joint policy $\vec{\pi} : S \to \Delta(\mathcal{A})$ maps the global states to probability distributions over the joint actions. We define the state value

[1] As the search progresses, this transformation progressively increases the scale for $\mathbf{q}$ and reduces the effect of the prior policy.

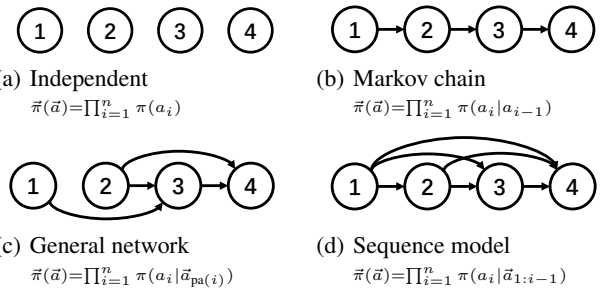


Figure 1: Different ways to factorize a joint policy $\vec{\pi}(\vec{a}|s)$. (d) has the strongest expressiveness.

function, the Q-value function, and the advantage function as $V^{\vec{\pi}}(s) \triangleq \mathbb{E}_{\vec{\pi}}\left[\sum_{k=0}^{T-1}\gamma^k r\left(s_k,\vec{a}_k,s_{k+1}\right)|\, s_0=s\right]$, $Q^{\vec{\pi}}(s,\vec{a}) \triangleq \mathbb{E}_{\vec{\pi}}\left[\sum_{k=0}^{T-1}\gamma^k r\left(s_k,\vec{a}_k,s_{k+1}\right)|\, s_0=s,\vec{a}_0=\vec{a}\right]$, and $A^{\vec{\pi}}(s,\vec{a}) \triangleq Q^{\vec{\pi}}(s,\vec{a}) - V^{\vec{\pi}}(s)$ respectively. $\gamma \in [0,1]$ is the discount factor. The target of solving an MMDP is to find a joint policy $\vec{\pi}$ that can maximize the expected discounted cumulative reward $V^{\vec{\pi}}(s)$.

Literature commonly factorizes the joint policy $\vec{\pi}$ as $\vec{\pi}(\vec{a}|s)=\pi_1(a_1|s)\times\ldots\times\pi_n(a_n|s)$, where $\pi_i: S\to\Delta(A_i)$ is the individual policy of agent i. As shown in Fig. 1, there are multiple ways to factorize $\vec{\pi}(\vec{a})$ into local policies depending on the problem (s is omitted), including (a) independent factorization, factorization according to (b) Markov chain, (c) general network, or (d) sequence model (like the Transformer Decoder in ChatGPT). All these can be represented by a probabilistic directed acyclic graphical model. Each node in the graph represents an agent. $\text{pa}(i)$ indicates the parents of node i (if any). Any directed acyclic graph has at least one *topological* order (Kahn 1962). To simplify notations, we assume $[1,\ldots,n]$ is a valid *topological* order (reindexing the agents can achieve this) if some agents have to make decisions based on previous agents' actions.

Challenges of extending *MuZero* to MMDP. ❶ As the joint action space grows exponentially with the increase of the agent number, the available simulation budget is usually much less than the size of the action space, i.e., $N_{\text{sim}} \ll |\mathcal{A}|$. Therefore, it's computationally prohibitive to directly run MCTS over the full joint action space. ❷ As it might only be feasible to search over a small subset of actions, which actions should be selected, and how to efficiently pick out such actions? ❸ How to compute an improved policy based on the evaluations of the selected subset of actions and how to update the policy network? ❹ It is more difficult to build an accurate world model in MMDP.

Multiagent Gumbel MuZero

Notations. For more compact notation, we omit the dependency of $\pi(a|s)$, $Q(s,a)$, $V(s)$ and $N(s,a)$ on state s when the context is clear. Here, we list the symbols we will use:

- $\vec{a}_{1:i}=\langle a_1,\ldots,a_i\rangle$ denotes the actions selected by agents $\{1,\ldots,i\}$. Specially, $\vec{a}_{1:0}=\langle\rangle$ and $\vec{a}_{1:n}=\vec{a}$.
- $\vec{\pi}_{1:i}$ is the partial joint policy of agents $\{1,\ldots,i\}$.

Algorithm 1: Stochastically sample top-k joint actions without replacement from the joint policy $\vec{\pi}$.

Input: n agents' policies $\pi_1,\ldots,\pi_n$; sample number k.
Initialize QUEUE empty.
Add $(\vec{a}_{1:0}=\langle\rangle, \phi(\vec{a}_{1:0})=0, G_{\phi(\vec{a}_{1:0})}=0)$ to QUEUE.
for each agent $i=1,\ldots,n$ **do**
  Initialize EXPANSIONS empty.
  for $(\vec{a}_{1:i-1}, \phi(\vec{a}_{1:i-1}), G_{\phi(\vec{a}_{1:i-1})}) \in$ QUEUE **do**
    $Z \leftarrow -\infty$.
    // Compute the log-probability $\phi(\vec{a}_{1:i})$ for action $\vec{a}_{1:i}$.
    for each action $a_i=1,\ldots,|\mathcal{A}_i|$ **do**
      $\vec{a}_{1:i} \leftarrow \vec{a}_{1:i-1}+\langle a_i\rangle$.
      // $\vec{a}_{\text{par}(i)}$ is the actions selected by i's parents (if i has).
      $\phi(\vec{a}_{1:i}) \leftarrow \phi(\vec{a}_{1:i-1}) + \log(\pi_i(a_i|\vec{a}_{\text{par}(i)}))$.
      $G_{\phi(\vec{a}_{1:i})} \sim \text{Gumbel}(\phi(\vec{a}_{1:i}))$.
      $Z \leftarrow \max(Z, G_{\phi(\vec{a}_{1:i})})$.
    end for
    // Sample $\tilde{G}_{\phi(\vec{a}_{1:i})}$ based on its maximum $G_{\phi(\vec{a}_{1:i-1})}$.
    for each action $a_i=1,\ldots,|\mathcal{A}_i|$ **do**
      $\tilde{G}_{\phi(\vec{a}_{1:i})} = -\log(\exp(-G_{\phi(\vec{a}_{1:i-1})}) - \exp(-Z)+\exp(-G_{\phi(\vec{a}_{1:i})}))$.
      Add $(\vec{a}_{1:i}, \phi(\vec{a}_{1:i}), \tilde{G}_{\phi(\vec{a}_{1:i})})$ to EXPANSIONS.
    end for
  end for
  QUEUE $\leftarrow$ take top-k of EXPANSIONS according to their $\tilde{G}_{\phi(\vec{a}_{1:i})}$s.
end for
return QUEUE. // i.e., $[\ldots,(\vec{a}_{1:n}, \phi(\vec{a}_{1:n}), \tilde{G}_{\phi(\vec{a}_{1:n})})^k]$.

- $\phi(a_i)=\log(\pi_i(a_i))$ and $\phi(\vec{a}_{1:i})=\log(\vec{\pi}_{1:i}(\vec{a}_{1:i}))$ are the log-probabilities of $\pi_i(a_i)$ and $\vec{\pi}_{1:i}(\vec{a}_{1:i})$ respectively.
- $G_{\phi(\vec{a}_{1:i})}=\phi(\vec{a}_{1:i})+G_0 \sim \text{Gumbel}(\phi(\vec{a}_{1:i}))$ is the perturbed log-probability, which can be obtained by adding independent Gumbel noises $G_0\sim\text{Gumbel}(0)$ to the log-probability (Kool, Van Hoof, and Welling 2019).
- $\pi_i(a_i|\vec{a}_{1:i-1})$ is the conditional probability of selecting a_i for agent i given $\vec{a}_{1:i-1}$ if i has any parents $\text{pa}(i)$.

Sample Joint Actions without Replacement from $\vec{\pi}$

As it's computationally prohibitive to directly run MCTS over the full joint action space, following *Gumbel MuZero*, our target is to stochastically sample top-k joint actions without replacement according to $\vec{\pi}(\vec{a})$, i.e., $\vec{\pi}_{1:n}(\vec{a}_{1:n})$, as the preferred actions to search and expand for both root nodes and non-root nodes. However, directly applying the Gumbel-top-k trick, i.e., $\mathcal{A}^k_{\text{top}}=\arg\text{top}_k\left(G_{\phi(\vec{a}_{1:n})}\right)$, is impracticable as it is inefficient to enumerate all actions and then pick out the top-k. To efficiently sample from the exponentially large space, we apply the *Stochastic Beam Search* algorithm (Kool, Van Hoof, and Welling 2019, 2020) to our setting. For completeness, the algorithm is summarized in Alg. 1. The key idea is that the perturbed log-probability of a partial joint action $\vec{a}_{1:i-1}$ is the maximum of perturbed log-probabilities of its possible successors $\vec{a}_{1:i}$, i.e.,

$$G_{\phi(\vec{a}_{1:i-1})} = \max_{a_i\in\mathcal{A}_i} G_{\phi(\vec{a}_{1:i})} \tag{13}$$

Algorithm	Support combinatorial action space?	Improved policy: visit count distribution?	Improved policy: exact solution?	Sample actions: Sample w/o replacement?	Sample actions: Sample w/ replacement?	Sample actions: Sample actions at root node?	Sample actions: Sample actions at non-root node?	Sample-based policy improvement?
MuZero	×	✓	×	N/A	N/A	×	×	×
Gumbel MuZero	×	×	✓	✓	×	✓	×	×
Sampled MuZero	✓	✓	×	×	✓	✓	✓	✓
Multiagent Gumbel MuZero	✓	×	✓	✓	×	✓	✓	✓

Table 1: Comparing the algorithmic properties of *MuZero* family methods in combinatorial action control problems.

This relation is the key to convert the bottom-up sampling procedure to a top-down sampling one, with the benefit that we do not have to enumerate all actions in $\mathcal{A}$. Take factorizing $\vec{\pi}(\vec{a})$ in an auto-regressive way for example, for two successive actions $\vec{a}_{1:i-1}$ and $\vec{a}_{1:i}$, we have $\phi(\vec{a}_{1:i-1}) = \log \sum_{a_i \in \mathcal{A}_i} \exp(\phi(\vec{a}_{1:i}))$ because

$$\begin{aligned}
\phi(\vec{a}_{1:i-1}) &= \log(\vec{\pi}_{1:i-1}(\vec{a}_{1:i-1}) \cdot 1) \\
&= \log\left(\vec{\pi}_{1:i-1}(\vec{a}_{1:i-1}) \cdot \left(\sum_{a_i \in \mathcal{A}_i} \pi_i(a_i|\vec{a}_{1:i-1})\right)\right) \\
&= \log\left(\sum_{a_i \in \mathcal{A}_i} [\vec{\pi}_{1:i-1}(\vec{a}_{1:i-1})\pi_i(a_i|\vec{a}_{1:i-1})]\right) \\
&= \log \sum_{a_i \in \mathcal{A}_i} \vec{\pi}_{1:i}(\vec{a}_{1:i}) = \log \sum_{a_i \in \mathcal{A}_i} \exp(\log(\vec{\pi}_{1:i}(\vec{a}_{1:i}))) \\
&= \log \sum_{a_i \in \mathcal{A}_i} \exp(\phi(\vec{a}_{1:i}))
\end{aligned} \tag{14}$$

According to the Gumbel Max Trick (Maddison, Tarlow, and Minka 2014), for any set $\mathcal{A}_i$ it holds that

$$\max_{a_i \in \mathcal{A}_i} G_{\phi(\vec{a}_{1:i})} \sim \text{Gumbel}\left(\log \sum_{a_i \in \mathcal{A}_i} \exp(\phi(\vec{a}_{1:i}))\right) \tag{15}$$

Combining Eq. 14 with Eq. 15, we get Eq. 13, i.e.,

$$\max_{a_i \in \mathcal{A}_i} G_{\phi(\vec{a}_{1:i})} = G_{\phi(\vec{a}_{1:i-1})} \sim \text{Gumbel}\left(\phi(\vec{a}_{1:i-1})\right). \tag{16}$$

This property is applicable to all the policy factorization formats shown in Fig. 1. As our target is to get $\mathcal{A}^k_{\text{top}} = \arg\text{top}_k\big(G_{\phi(\vec{a}_{1:n})}\big)$, with Eq. 13, we know the top-k of $G_{\phi(\vec{a}_{1:n})}$ must belong to the successors of the top-k of $G_{\phi(\vec{a}_{1:n-1})}$. Therefore, we can convert the bottom-up sampling procedure to a top-down sampling one, i.e., we can sample $G_{\phi(\vec{a}_{1:i-1})}$ first and then sample $\tilde{G}_{\phi(\vec{a}_{1:i})} \sim \text{Gumbel}(\phi(\vec{a}_{1:i}))$ conditionally on that their maximum equal to $G_{\phi(\vec{a}_{1:i-1})}$ (line 16-20 of Alg. 1). Refer to (Kool, Van Hoof, and Welling 2019, 2020) for more details.

Planning at the root node. For the root node, at the beginning of each MCTS, we stochastically sample top-k joint actions $\mathcal{A}^k_{\text{top}}$ according to Alg. 1 as the candidate actions to search. Then we follow *Gumbel MuZero* and use the *Sequential Halving* algorithm to evenly divide the simulation budget N_{sim} into $\log_2(k)$ search phases. At each search phase $j \in \{0, \ldots, \log_2(k)\}$, it only takes the $(\text{top-}k, \text{top-}\frac{k}{2}, \ldots, \text{top-1})[j]$ actions according to Eq. 9 as considered ones and visits the considered actions evenly often, with the target of minimizing simple regret. See Fig. 1 of (Danihelka et al. 2022) for more details.

Sample-based Policy Improvement Operator

Compute the improved policy. After a number of simulations, we get $Q(s, \vec{a})$ and $N(s, \vec{a})$ for all visited actions and an estimated V-value $V(s)$ for each state.

Assumption 1 *Following (Danihelka et al. 2022), for all unvisited actions, we assume that their Q-values can be approximated by the state value $V(s)$.*

Now we have $Q(\vec{a}) = \begin{cases} Q(\vec{a}) & \text{if } N(\vec{a}) > 0 \\ V, & \text{otherwise} \end{cases}$. We normalize $Q(\vec{a})$ and V to $[0, 1]$ by their min and max values, i.e., $\hat{Q}(\vec{a}) = \text{norm}(Q(\vec{a}))$ and $\hat{V} = \text{norm}(V)$, where $\text{norm}(x) \triangleq \frac{x - \min(V, \min_{\vec{a}} Q(\vec{a}))}{\max(V, \max_{\vec{a}} Q(\vec{a})) - \min(V, \min_{\vec{a}} Q(\vec{a}))}$. Then we compute the normalized advantages as $\hat{\text{adv}}(\vec{a}) = \hat{Q}(\vec{a}) - \hat{V}$, whose scales are within $[-1, 1]$. According to Eq. 11, we get the improved policy $\vec{\pi}_{\text{MPO}}(\cdot)$ as:

$$\vec{\pi}_{\text{MPO}}(\cdot) = \text{softmax}\Big(\log(\vec{\pi}_\theta(\cdot)) + \sigma(\hat{\text{adv}}(\cdot))\Big) \tag{17}$$

where $\sigma(x) = (c_{\text{visit}} + \max_{\vec{b}} N(\vec{b}))c_{\text{scale}} \cdot x$ as mentioned in Eq. 9. We can replace $\hat{Q}$ with $\hat{\text{adv}}$ because offsetting the inputs of softmax by a constant value does not change the output. After constructing the new improved policy $\vec{\pi}_{\text{MPO}}$, we can distill it to the policy network $\vec{\pi}_\theta$ via minimizing the cross-entropy between $\vec{\pi}_{\text{MPO}}(\cdot|s)$ and $\vec{\pi}_\theta(\cdot|s)$:

$$L_{\text{MPO}}(\vec{\pi}_\theta) = -\sum_{\vec{a} \in \mathcal{A}} \vec{\pi}_{\text{MPO}}(\vec{a}) \log(\pi_\theta(\vec{a})) \tag{18}$$

Sample-based policy improvement. As computing the exact $L_{\text{MPO}}(\vec{\pi})$ needs to enumerate all joint actions to obtain the full $\vec{\pi}_{\text{MPO}}(\cdot)$ distribution, and then apply the cross entropy, it is infeasible for exponentially large action spaces. Therefore, we propose to compute a stochastic estimate of $L_{\text{MPO}}(\vec{\pi})$ based on the sampled top-k actions, where

$k \ll |\mathcal{A}|$. We define $\mathcal{A}^k_{\text{top}} \triangleq \langle \vec{a}_1, \dots, \vec{a}_k \rangle$ as the ordered set of sampled top-k actions via Alg. 1, where the actions are sorted *in descending order* by their perturbed log-probabilities $\tilde{G}_{\phi(\vec{a})}$s.

Our target is using the top-k samples $\mathcal{A}^k_{\text{top}}$ to estimate Eq.18, i.e., $L_{\text{MPO}}(\vec{\pi}_\theta) = -\mathbb{E}_{\vec{a}\sim\vec{\pi}_{\text{MPO}}(\vec{a})} \log(\vec{\pi}_\theta(\vec{a}))$. However, the top-$k$ samples are not sampled from $\vec{\pi}_{\text{MPO}}(\vec{a})$ but sampled from $\vec{\pi}_\theta$ without replacement using the Gumbel-Top-k trick. Thus, we cannot simply use the *Monte Carlo* method to estimate $L_{\text{MPO}}(\vec{\pi}_\theta)$. Instead, we need to use importance weights to correct for the changed sampling probabilities. Derived from priority sampling (Duffield, Lund, and Thorup 2007; Vieira 2017), we get an unbiased estimator:

$$\begin{aligned} L_{\text{MPO}}(\vec{\pi}_\theta) &= -\mathbb{E}_{\vec{a}\sim\vec{\pi}_{\text{MPO}}(\vec{a})} \log(\vec{\pi}_\theta(\vec{a})) \\ &\approx -\sum_{\vec{a}_j \in \mathcal{A}^k_{\text{top}}} \frac{\vec{\pi}_{\text{MPO}}(\vec{a}_j)}{q_{\theta,\kappa}(\vec{a}_j)} \log(\vec{\pi}_\theta(\vec{a}_j)) \end{aligned} \tag{19}$$

where $q_{\theta,\kappa}(\vec{a}_j)$ is the probability of $\vec{a}_j \in \mathcal{A}^k_{\text{top}}$. If we sample $k+1$ actions from $\vec{\pi}_\theta$ via Alg. 1 and denote κ as the $(k+1)$-th largest perturbed log-probabilities $\tilde{G}_{\phi(\vec{a}_{k+1})}$, we can consider κ as the *empirical threshold* for the Gumbel-Top-k sampling (since $\vec{a} \in \mathcal{A}^k_{\text{top}}$ if $\tilde{G}_{\phi(\vec{a})} > \kappa$). We define $q_{\theta,\kappa}(\vec{a}_j) = P(\tilde{G}_{\phi(\vec{a}_j)} > \kappa)$, which equals to 1 minus the Gumbel cumulative distribution function value, i.e., $1 - P(\tilde{G}_{\phi(\vec{a}_j)} \le \kappa) = 1 - \exp(-\exp(\phi(\vec{a}_j) - \kappa))$. Due to the space limit, please see Appendix B.2 for more details.

To compute $\vec{\pi}_{\text{MPO}}(\vec{a}_k)$, we have to estimate the normalization factor z of softmax in Eq. 17. As we use V to approximate the Q-values for the unsampled actions, their advantages are 0. Thus,

$$\begin{aligned} z &= \sum_{\vec{a}\in\mathcal{A}} \vec{\pi}_\theta(\vec{a}) \exp\Big(\sigma(\hat{\text{adv}}(\vec{a}))\Big) \\ &= \sum_{\vec{a}\in\mathcal{A}^k_{\text{top}}} \vec{\pi}_\theta(\vec{a}) \exp\Big(\sigma(\hat{\text{adv}}(\vec{a}))\Big) + \sum_{\vec{a}\in\mathcal{A}\setminus\mathcal{A}^k_{\text{top}}} \vec{\pi}_\theta(\vec{a}) \\ &= \sum_{\vec{a}\in\mathcal{A}^k_{\text{top}}} \pi_\theta(\vec{a}) \exp\Big(\sigma(\hat{\text{adv}}(\vec{a}))\Big) + 1 - \sum_{\vec{a}\in\mathcal{A}^k_{\text{top}}} \vec{\pi}_\theta(\vec{a}) \\ &= 1 + \sum_{a\in\mathcal{A}^k_{\text{top}}} \pi_\theta(\vec{a}) \left(\exp\Big(\sigma(\hat{\text{adv}}(\vec{a}))\Big) - 1\right) \end{aligned} \tag{20}$$

As $k \ll |\mathcal{A}|$, z can be efficiently computed. With z, for $\forall \vec{a} \in \mathcal{A}^k_{\text{top}}, \vec{\pi}_{\text{MPO}}(\vec{a}) = \frac{\vec{\pi}_\theta(\vec{a}) \exp\left(\sigma(\hat{\text{adv}}(\vec{a}))\right)}{z}$ and we can estimate $L_{\text{MPO}}(\vec{\pi}_\theta)$ according to Eq. 19 efficiently.

Planning at non-root nodes. At the time of expanding a new non-root node, we stochastically sample top-k joint actions $\mathcal{A}^k_{\text{top}}$ via Alg. 1 as the candidate actions to search. Then we directly select actions according to the locally improved policy $\pi^{\text{local}}_{\text{MPO}}(\mathcal{A}^k_{\text{top}})$ at the non-root nodes:

$$\pi^{\text{local}}_{\text{MPO}}(\mathcal{A}^k_{\text{top}}) = \text{softmax}\Big(\log(\vec{\pi}_\theta(\mathcal{A}^k_{\text{top}})) + \sigma(\hat{\text{adv}}(\mathcal{A}^k_{\text{top}}))\Big) \tag{21}$$

i.e., only apply softmax on the sampled top-k actions to get a locally improved policy. Due to the space limit, details of the world model design are shown in Appendix B.3.

a_1 \ a_2	A	B	C
A	**8**	-12	-12
B	-12	6	0
C	-12	0	6

When a_1=A

a_2 \ a_3	A	B
A	0	0
B	**23**	12

When a_1=B

a_2 \ a_3	A	B
A	17	20
B	0	17

Table 2: (Left 1) payoffs of a 2-player matrix game. Each player $i \in \{1, 2\}$ has 3 actions $\{A, B, C\}$. (Right 2) payoffs of a 3-player matrix game. Each player $i \in \{1, 2, 3\}$ has 2 actions $\{A, B\}$. The complete payoff matrix is split into two submatrices depending on player 1's action a_1. The boldface indicates the payoffs of the optimal actions.

Empirical Results

In all experiments, we use the independent format shown in Fig.1(a) to factorize the joint policy $\vec{\pi}(\vec{a})$. Each agent i's individual policy $\pi_i(a_i|o_i)$ takes its own observation as input.

Single State Cooperative Matrix Game

First, we use two simple matrix games to verify the benefit of using sampling without replacement (Algo. 1) instead of sampling with replacement (Monte Carlo sampling) when doing policy evaluation and improvement.

Setup. Table 2 shows the payoffs of a 2-player 3-action matrix game and the payoffs of a 3-player 2-action matrix game. These two games are challenging in the MARL literature (Son et al. 2019; Huang et al. 2022) as algorithms can easily get stuck in suboptimal actions due to the large miss-coordination penalty. We compare the following 3 methods.

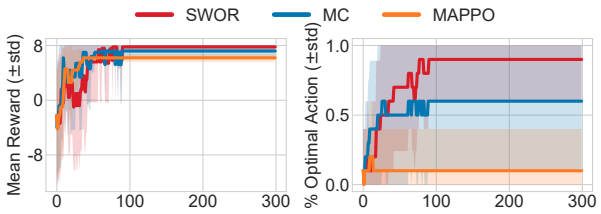
- **Sample without Replacement (SWOR)**. At each iteration, we sample k joint actions without replacement according to Algo. 1 and use Eq. 19 to improve the policy.
- **Monte Carlo (MC)**. At each iteration, MC samples k joint actions according to the joint policy $\vec{\pi}$ with replacement and uses the same loss function Eq. 18 to improve the policy but with the following Monte Carlo estimate:

$$L_{\text{MPO}}(\vec{\pi}_\theta) \approx -\frac{1}{k}\sum_{j=1}^{k} \frac{\vec{\pi}_{\text{MPO}}(\vec{a}_j)}{\vec{\pi}_\theta(\vec{a}_j)} \log(\vec{\pi}_\theta(\vec{a}_j))$$

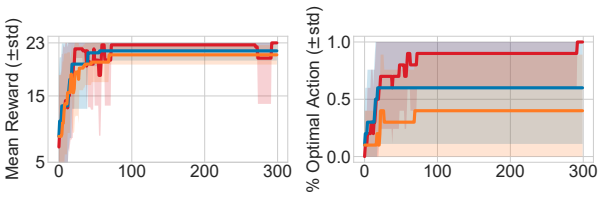
  Replace Eq. 20 with $z \approx \frac{1}{k}\sum_{j=1}^{k} \exp\Big(\sigma(\hat{\text{adv}}(\vec{a}_j))\Big)$.
- **MAPPO** (Yu et al. 2022), which extends PPO (Schulman et al. 2017) to cooperative multiagent settings. MAPPO also samples k actions according to $\vec{\pi}$ per iteration and updates the policy with the MAPPO loss.

We set $k = 4$ and keep all parameters the same for these three methods. Each experiment is repeated for 10 times.

Results. Fig. 2 shows the learning curves of the three methods. The left two figures show the average test rewards and the right two figures show the percentages of optimal actions. Using sampling without replacement performs significantly better than sampling with replacement and MAPPO. Our method can find globally better actions. Only changing the sampling method can make a significant improvement.



(a) 2-player 3-action matrix game results.



(b) 3-player 2-action matrix game results.

Figure 2: Learning curves of the average test reward and the percent of optimal actions in the matrix games. The mean and the standard deviation are shown. Best viewed in color.

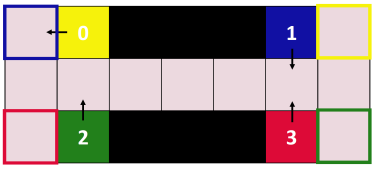


Figure 3: The starting state of Switch4. The 4 agents need to reach their corresponding destinations. Each colored block (yellow, blue, green, or red) with a number denotes an agent. Black blocks are barriers. Blocks with colored boundaries denote the destinations for the corresponding colored agents.

Multiagent Switch

As shown in Fig 3, Switch4 in MA-Gym (Koul 2019) is a hard coordination task that 4 agents need to reach their corresponding home by passing through the one-agent wide narrow corridor. Each agent has 5 actions, resulting in a joint action space size of 625. At each step, all agents get a step penalty of -0.5. Having a collision with other agents results in a penalty of -1. Once an agent reaches the goal, the team will get a reward of +5. The target of the agents is learning to reach their goals as soon as possible without having a collision with each other. The optimal solution requires 17 steps to reach the goals. We compare the following 2 methods.

- ***MA Sampled AlphaZero***: the multiagent version of *Sampled AlphaZero* Hubert et al. (2021).
- ***MA Gumbel AlphaZero***: our method with a perfect environment model. We keep all designs (e.g., model architectures and hyperparameters) the same with *MA Sampled AlphaZero* except that shown in Table 1.

The learning curves under different simulation budgets are shown in Figure 4 and Figure 5. We set the sample number $k = \text{clamp}(n_{\text{sim}}/2, \text{min=2}, \text{max=16})$. Results are averaged over 5 random seeds. We see that under all simulation

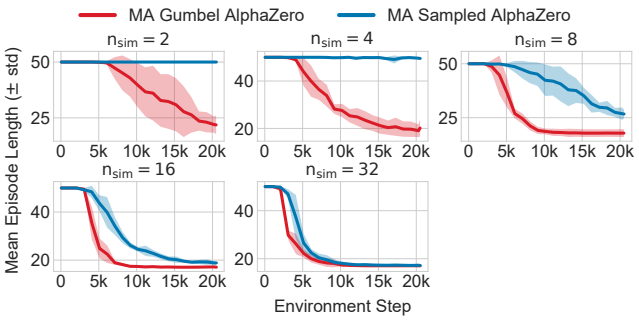


Figure 4: The average episode lengths (smaller is better) for the two methods when training with $n_{\text{sim}} \in \{2, 4, 8, 16, 32\}$.

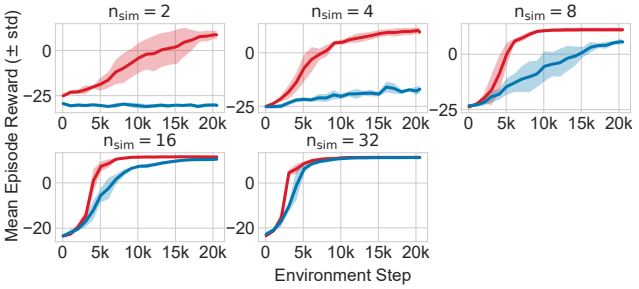


Figure 5: The average episode rewards (larger is better) for the two methods when training with $n_{\text{sim}} \in \{2, 4, 8, 16, 32\}$.

numbers, *MA Gumbel AlphaZero* (using sampling without replacement and Eq. 17 as policy target) can find significantly better solutions than *MA Sampled AlphaZero* (using sampling with replacement and the visit count distribution as policy target). *MA Gumbel AlphaZero* can even achieve good performance with only 2 simulations, which shows that the method is more robust to the simulation budget. The full experimental results including *MuZero*-based methods, model-free baselines, and experiments on more complex StarCraft Multiagent Challenge benchmark (Samvelyan et al. 2019) are shown in Appendix C.

Conclusion

In this paper, we propose *MA Gumbel MuZero*, which extends *MuZero* to learning and planning in discrete combinatorial action spaces. *MA Gumbel MuZero* adopts the idea of stochastic beam search to efficiently sample top-k joint actions without replacement from the combinatorial action space. For both the root and non-root nodes, we sample top-k actions without replacement as the candidate actions to search and expand, which is more efficient than *Sampled MuZero*'s sampling with replacement. We devise a well-matched sample-based policy improvement operator. Empirical results show that our method reliably outperforms the prior algorithms in multiple cooperative multiagent control problems and its performance is less affected by a small number of simulations. For future work, we will apply the algorithms to more practical problems, e.g., combinatorial optimization and fine-tuning Large Language Models (LLMs).

Acknowledgments

This work is supported by the National Natural Science Foundation of China (Grant No. 92370132), the National Key R&D Program of China (Grant No. 2022ZD0116402), and the National Natural Science Foundation of China (Grant No. 62106172).

References

Abdolmaleki, A.; Springenberg, J. T.; Tassa, Y.; Munos, R.; Heess, N.; and Riedmiller, M. 2018. Maximum a Posteriori Policy Optimisation. In *International Conference on Learning Representations*.

Ausiello, G.; Crescenzi, P.; Gambosi, G.; Kann, V.; Marchetti-Spaccamela, A.; and Protasi, M. 2012. *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Springer Science & Business Media.

Bernstein, D. S.; Givan, R.; Immerman, N.; and Zilberstein, S. 2002. The complexity of decentralized control of Markov decision processes. *Mathematics of operations research*, 27(4): 819–840.

Boutilier, C. 1996. Planning, learning and coordination in multiagent decision processes. In *TARK*, volume 96, 195–210. Citeseer.

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

Bryant, P.; Pozzati, G.; Zhu, W.; Shenoy, A.; Kundrotas, P.; and Elofsson, A. 2022. Predicting the structure of large protein complexes using AlphaFold and Monte Carlo tree search. *Nature communications*, 13(1): 6028.

Danihelka, I.; Guez, A.; Schrittwieser, J.; and Silver, D. 2022. Policy improvement by planning with Gumbel. In *International Conference on Learning Representations*.

Duffield, N.; Lund, C.; and Thorup, M. 2007. Priority sampling for estimation of arbitrary subset sums. *Journal of the ACM (JACM)*, 54(6): 32–es.

Fawzi, A.; Balog, M.; Huang, A.; Hubert, T.; Romera-Paredes, B.; Barekatain, M.; Novikov, A.; R Ruiz, F. J.; Schrittwieser, J.; Swirszcz, G.; et al. 2022. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930): 47–53.

Grill, J.-B.; Altché, F.; Tang, Y.; Hubert, T.; Valko, M.; Antonoglou, I.; and Munos, R. 2020. Monte-Carlo tree search as regularized policy optimization. In *International Conference on Machine Learning*, 3769–3778. PMLR.

Hamrick, J. B.; Bapst, V.; Sanchez-Gonzalez, A.; Pfaff, T.; Weber, T.; Buesing, L.; and Battaglia, P. W. 2020. Combining Q-Learning and Search with Amortized Value Estimates. In *International Conference on Learning Representations*.

Hessel, M.; Danihelka, I.; Viola, F.; Guez, A.; Schmitt, S.; Sifre, L.; Weber, T.; Silver, D.; and Van Hasselt, H. 2021. Muesli: Combining improvements in policy optimization. In *International conference on machine learning*, 4214–4226. PMLR.

Huang, W.; Li, K.; Shao, K.; Zhou, T.; Taylor, M.; Luo, J.; Wang, D.; Mao, H.; Hao, J.; Wang, J.; et al. 2022. Multiagent q-learning with sub-team coordination. *Advances in Neural Information Processing Systems*, 35: 29427–29439.

Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Barekatain, M.; Schmitt, S.; and Silver, D. 2021. Learning and planning in complex action spaces. In *International Conference on Machine Learning*, 4476–4486. PMLR.

Kahn, A. B. 1962. Topological sorting of large networks. *Communications of the ACM*, 5(11): 558–562.

Karnin, Z.; Koren, T.; and Somekh, O. 2013. Almost optimal exploration in multi-armed bandits. In *International Conference on Machine Learning*, 1238–1246. PMLR.

Kemmerling, M.; Lütticke, D.; and Schmitt, R. H. 2023. Beyond Games: A Systematic Review of Neural Monte Carlo Tree Search Applications. *arXiv preprint arXiv:2303.08060*.

Kool, W.; Van Hoof, H.; and Welling, M. 2019. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. In *International Conference on Machine Learning*, 3499–3508. PMLR.

Kool, W.; Van Hoof, H.; and Welling, M. 2020. Ancestral gumbel-top-k sampling for sampling without replacement. *The Journal of Machine Learning Research*, 21(1): 1726–1761.

Koul, A. 2019. ma-gym: Collection of multi-agent environments based on OpenAI gym. https://github.com/koulanurag/ma-gym. Accessed: 2019-08-16.

Lenz, D.; Kessler, T.; and Knoll, A. 2016. Tactical cooperative planning for autonomous highway driving using Monte-Carlo Tree Search. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, 447–453. IEEE.

Maddison, C. J.; Tarlow, D.; and Minka, T. 2014. A* sampling. *Advances in neural information processing systems*, 27.

Mankowitz, D. J.; Michi, A.; Zhernov, A.; Gelmi, M.; Selvi, M.; Paduraru, C.; Leurent, E.; Iqbal, S.; Lespiau, J.-B.; Ahern, A.; et al. 2023. Faster sorting algorithms discovered using deep reinforcement learning. *Nature*, 618(7964): 257–263.

Ouyang, L.; Wu, J.; Jiang, X.; Almeida, D.; Wainwright, C.; Mishkin, P.; Zhang, C.; Agarwal, S.; Slama, K.; Ray, A.; et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35: 27730–27744.

Samvelyan, M.; Rashid, T.; de Witt, C. S.; Farquhar, G.; Nardelli, N.; Rudner, T. G. J.; Hung, C.-M.; Torr, P. H. S.; Foerster, J.; and Whiteson, S. 2019. The StarCraft Multi-Agent Challenge. *arXiv preprint arXiv:1902.04043*.

Schrittwieser, J.; Antonoglou, I.; Hubert, T.; Simonyan, K.; Sifre, L.; Schmitt, S.; Guez, A.; Lockhart, E.; Hassabis, D.; Graepel, T.; et al. 2020. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839): 604–609.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

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

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419): 1140–1144.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the game of go without human knowledge. *nature*, 550(7676): 354–359.

Son, K.; Kim, D.; Kang, W. J.; Hostallero, D. E.; and Yi, Y. 2019. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International conference on machine learning*, 5887–5896. PMLR.

Vieira, T. 2014. Gumbel-max trick and weighted reservoir sampling. https://timvieira.github.io/blog/post/2014/08/01/gumbel-max-trick-andweighted-reservoir-sampling. Accessed: 2014-08-01.

Vieira, T. 2017. Estimating means in a finite universe. https://timvieira.github.io/blog/post/2017/07/03/estimating-means-in-a-finite-universe/. Accessed: 2017-07-03.

Yu, C.; Velu, A.; Vinitsky, E.; Gao, J.; Wang, Y.; Bayen, A.; and Wu, Y. 2022. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in Neural Information Processing Systems*, 35: 24611–24624.