Search-based Reinforcement Learning through Bandit Linear Optimization

Milan Peelman[*] , **Antoon Bronselaer** , **Guy De Tré**
Ghent University
milan.peelman@ugent.be

Abstract

The development of AlphaZero was a breakthrough in search-based reinforcement learning, by employing a given world model in a Monte-Carlo tree search (MCTS) algorithm to incrementally learn both an action policy and a value estimation. When extending this paradigm to the setting of simultaneous move games we find that the selection strategy of AlphaZero has theoretical shortcomings, including that convergence to a Nash equilibrium is not guaranteed. By analyzing these shortcomings, we find that the selection strategy corresponds to an approximated version of bandit linear optimization using Tsallis entropy regularization with α parameter set to zero, which is equivalent to log-barrier regularization. This observation allows us to refine the search method used by AlphaZero to obtain an algorithm that has theoretically optimal regret as well as superior empirical performance on our evaluation benchmark.

1 Introduction

Monte-Carlo tree search commonly uses UCT (Upper Confidence Bounds for Trees) [Kocsis and Szepesvári, 2006] or some variant thereof as the method for action selection in the search tree. One of these variants, PUCT (Polynomial Upper Confidence Bounds for Trees) [Rosin, 2011], is used in several search-based reinforcement learning (RL) algorithms, e.g. AlphaZero [Silver *et al.*, 2017], MuZero [Schrittwieser *et al.*, 2019] and recently Player of Games [Schmid *et al.*, 2021]. A possible explanation for the choice for this variant in AlphaZero and subsequent algorithms is due to the fact that the PUCT action selection formula is parametrized by a prior policy, which allows for a learned prior action distribution to be utilized to guide the search procedure.

In [Grill *et al.*, 2020] they develop an MCTS implementation inspired by ideas from MPO, a model-free policy-optimization algorithm [Abdolmaleki *et al.*, 2018]. A key insight was that the empirical visit distribution of actions in PUCT approximates the solution of a regularized policy-optimization objective. Consequently, a method was derived to compute the exact solution to this regularized policy-optimization objective, allowing their implementation to work significantly better than the standard PUCT implementation when the amount of search iterations is low, since this setting suffers the most from the approximative nature of PUCT. Another disadvantage of PUCT arises in the domain of simultaneous move games. The straightforward extension of PUCT to this domain suffers from the same theoretical issues as Decoupled UCT (DUCT) [Tak *et al.*, 2014], i.e. asymptotic convergence to a Nash equilibrium is not guaranteed, despite Decoupled (P)UCT performing favorably in practice compared to some algorithms that *do* converge to a Nash equilibrium [Tak *et al.*, 2014]. In this work we seek to expand on the work in [Grill *et al.*, 2020] by deriving a search-based reinforcement learning algorithm that converges to a Nash equilibrium in simultaneous move games, while having strong empirical performance at the same time.

Contributions. We analyze the implicit regularized policy-optimization objective from PUCT and find that this is equivalent to log-barrier regularization, which is a special case of Tsallis entropy [Tsallis, 1988] regularization, i.e. with the α parameter set to zero. This raises the question whether there are other values of α that would result in improved performance. Previous literature on bandit algorithms indicates that the answer to this question is affirmative. Concretely, setting α to 0.5, combined with an appropriate learning rate, achieves the theoretically optimal regret bound of $\mathcal{O}(\sqrt{KT})$, with K being the size of the action space and T the amount of search iterations. Other values of α fail to obtain this regret bound. We find that in a certain sense, regularization with $\alpha = 0.5$ combined with an appropriate learning rate corresponds to a hybrid of UCT and PUCT and we experimentally verify that this results in an increased rate of improvement for our reinforcement learning agents, both in the setting with few search iterations and beyond.

2 Background

2.1 Two-Player Zero-Sum Markov Games

We consider a finite-horizon discounted two-player zero-sum Markov game, which can be described by a tuple $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathcal{P}, r, \gamma)$: a state space $\mathcal{X}$, an action space $\mathcal{A}$, a transition probability function $\mathcal{P} : \mathcal{X} \times \mathcal{A} \times \mathcal{A} \to \Delta(\mathcal{X})$, a reward function $r : \mathcal{X} \times \mathcal{A} \times \mathcal{A} \to [0, 1]$ and a discount

[*]Contact Author

factor $\gamma \in (0, 1]$. A policy or strategy π is defined as a point on the n-dimensional simplex $\mathcal{S}$, where n is equal to the size of the action space $\mathcal{A}$. We will use the terms "policy" and "strategy" interchangeably, with the former being more common in RL literature and the latter being more common in game-playing literature. The RL problem consists in finding a policy which maximizes the discounted cumulative reward $\mathbb{E}_{\pi_\theta}\left[\sum_{t\geq 0} \gamma^t r_t\right]$.To scale the method to large environments, we assume that the learned policy π_θ is parameterized by a neural network θ.

2.2 Regret and Nash Equilibria

In this work we focus on simultaneous move games, as opposed to sequential move games such as chess and Go, or games where the player plays against a non-adversarial environment, e.g. the Atari57 suite of games. The reason for our particular interest in simultaneous move games is due to the fact that the action selection strategy of AlphaZero, i.e. PUCT, does not guarantee convergence to a Nash equilibrium, i.e. a pair of policies where neither player gains by unilaterally changing its own policy. One can show this by constructing a counterexample analogously to the one in [Shafiei *et al.*, 2009] for UCT.

To remedy this we construct an algorithm that is Hannan consistent, or equivalently: an algorithm with no external regret. We now formally define the notion of external regret used in this work.

Definition 1 (External Regret)**.** *The external regret for playing a sequence of strategies $\pi_1, ..., \pi_T$ and a sequence of cost vectors $\mathbf{g}_1, ..., \mathbf{g}_T$, is defined as*

$$R_T = \sum_{t=1}^{T} \mathbf{g}_t^\top \pi_t - \min_{\pi \in \Delta_n} \sum_{t=1}^{T} \mathbf{g}_t^\top \pi$$

where Δ_n is the n-dimensional simplex: $\Delta_n = \{\pi \in \mathbb{R}^n, \sum_i \pi_i = 1, \pi_i \geq 0\}$

Note that the cost vectors g_t are the negation of the obtained rewards. An algorithm is said to be Hannan consistent if its external regret is sublinear as a function of T, i.e. $R_T = o(T)$, which implies that on average the algorithm performs as well as the best fixed strategy in hindsight. If both players use a Hannan consistent algorithm in self-play in a zero-sum normal form game, the average strategies chosen converge to a Nash equilibrium of the game [Blum and Mansour, 2007]. Common Hannan-consistent algorithms for simultaneous move games are Exp3 [Auer *et al.*, 2003] and Regret Matching (RM) [Hart and Mas-Colell, 2001].

Though there is one caveat, since normal form games, or matrix games, entail only a limited subset of simultaneous move games. More generally, we are interested in results for zero-sum extensive form games with simultaneous moves, or stacked matrix games (SMG). In these games we have the additional requirement that the selection strategy needs to satisfy the *guaranteed exploration* condition, which ensures that the whole game tree can be covered. Moreover, one has to be careful in the way the value of child nodes are estimated, since one can construct an estimation method that fails to converge to a Nash equilibrium, despite using a Hannan consistent algorithm. Both of these results are discussed extensively in [Kovarík and Lisý, 2018].

2.3 Bandit Linear Optimization

Bandit linear optimization [Abernethy *et al.*, 2008; Hazan, 2019] can be thought of as online linear optimization in the bandit setting. The bandit setting implies that only partial information is available to each agent with regard to the valuation of strategies. More concretely, an agent observes only the scalar value $\mathbf{g}_t^\top \pi_t$ in Definition 1, as opposed to the full cost vector $\mathbf{g}_t$. Thus, it is not possible to determine the reward for counterfactual strategies.

Just like solving a zero-sum matrix game is equivalent to solving a linear optimization problem, the case where the entries of the matrix, i.e. the Q-values, have to be estimated through sampling corresponds to a bandit linear optimization problem. In this work we utilize a notion of performing MCTS that is inspired by ideas from bandit linear optimization literature. In particular, while MCTS typically results in an *empirical visit distribution*, i.e. the distribution of visit counts for each action by the search, we instead store the *average strategy* throughout the search, where the individual strategies can be mixed, i.e. comprising multiple actions. The implications on the MCTS implementation are further detailed in Section 4.

3 Methodology

3.1 PUCT and the Logarithmic Barrier

The PUCT action selection strategy, used in AlphaZero and originally defined in [Rosin, 2011], selects the best action a^* as follows:

$$a^* \triangleq \arg\max_a \left[Q(x,a) + c \cdot \pi_\theta(a|x) \cdot \frac{\sqrt{\sum_b n(x,b)}}{1 + n(x,a)} \right], \tag{1}$$

where c is a numerical constant, $n(x,a)$ is the number of times that action a has been selected from state x during the search, and $Q(x,a)$ is an estimate of the Q-value for state-action pair (x,a) computed from search statistics.

In Proposition 1 in [Grill *et al.*, 2020] it is shown that this can be rewritten as

$$a^* = \arg\max_a \left[\frac{\partial}{\partial n_a} \left(\boldsymbol{q}^\intercal \hat{\boldsymbol{\pi}} - \lambda_N D_{\text{KL}} \left(\boldsymbol{\pi_\theta} \| \hat{\boldsymbol{\pi}} \right) \right) \right], \tag{2}$$

where D_{KL} denotes the *reversed* KL-divergence since π_θ is the target distribution, and where the learning rate λ_N is defined as

$$\lambda_N \triangleq c \cdot \frac{\sqrt{\sum_b n_b}}{|\mathcal{A}| + \sum_b n_b}, \tag{3}$$

and where the empirical visit distribution $\hat{\pi}$ is defined as

$$\hat{\pi}(a|x) \triangleq \frac{1 + n(x,a)}{|\mathcal{A}| + \sum_b n(x,b)}. \tag{4}$$

In other words: a^* is the action maximizing the partial derivative with respect to its visit count and the resulting strategy at time step t is the one-hot vector $\mathbb{1}_t(a^*)$. The empirical visit distribution $\hat{\pi}$ is consequently updated by incrementing n_{a^*}.

Additionally they note that, if we maximize the function over the simplex $\mathcal{S}$, then instead of obtaining a discrete approximate solution, we obtain the optimal mixed strategy

$$\bar{\boldsymbol{\pi}} \triangleq \arg\max_{\mathbf{y}\in\mathcal{S}} \left[\boldsymbol{q}^{\mathsf{T}}\mathbf{y} - \lambda_N D_{\mathrm{KL}}\left(\boldsymbol{\pi_\theta} \| \mathbf{y}\right)\right]. \tag{5}$$

We can rewrite this as

$$\bar{\boldsymbol{\pi}} = \arg\max_{\mathbf{y}\in\mathcal{S}} \left[\boldsymbol{q}^{\mathsf{T}}\mathbf{y} - \lambda_N\left(\boldsymbol{\pi_\theta}\log\boldsymbol{\pi_\theta} - \boldsymbol{\pi_\theta}\log\mathbf{y}\right)\right],$$

and because π_θ is a constant vector we can simplify the expression as

$$\bar{\boldsymbol{\pi}} = \arg\max_{\mathbf{y}\in\mathcal{S}} \left[\boldsymbol{q}^{\mathsf{T}}\mathbf{y} + \lambda_N\boldsymbol{\pi_\theta}\log\mathbf{y}\right]. \tag{6}$$

Thus, we obtain an optimization problem that is regularized by a log-barrier, which is a regularization method that was first proposed in the bandit setting in [Foster *et al.*, 2016]. Note that it is the approximative nature of PUCT that prevents Hannan consistency, which causes its failure to converge to a Nash equilibrium. Sampling from $\bar{\pi}$ on the other hand, does lead to a Hannan consistent action selection algorithm, but we will see in Section 3.2 that we can utilize the Tsallis entropy to generalize to a whole family of Hannan consistent algorithms.

3.2 Generalization With Tsallis Entropy

We now give the definition of the (negative) Tsallis entropy $H_\alpha(x)$, originally described by Tsallis [1988]:

$$H_\alpha(x) \triangleq \frac{1}{1-\alpha}\left(1 - \sum_i x_i^\alpha\right), \tag{7}$$

where x is a point on a simplex. Regularization using a log-barrier happens to be a special case of regularization based on the Tsallis entropy, i.e. the case where α is zero. It is straightforward to see that if $\alpha = 0$, then $H_\alpha(x)$ as defined above becomes meaningless, since the same value is obtained regardless of the values x_i. Therefore the Tsallis entropy for $\alpha = 0$ is defined as

$$H_0(x) \triangleq -\sum_i \log\left(x_i\right). \tag{8}$$

In the Appendix we show how this definition originates from the $H_\alpha(x)$ entropy formula.

A natural question to ask at this point is whether setting α to zero, like AlphaZero[1] approximately does through the use of PUCT, is indeed the optimal choice. It turns out that, combined with an appropriate learning rate, the case where α is set to 0.5 actually has the best theoretical guarantees on the regret, i.e. it obtains the optimal adversarial asymptotic regret bound of $\mathcal{O}(\sqrt{KT})$ [Audibert and Bubeck, 2009; Abernethy *et al.*, 2015]. Comparatively, setting α to 0 yields a bound of $\mathcal{O}(\sqrt{KT\log T})$ [Wei and Luo, 2018] and setting α to 1 yields a bound of $\mathcal{O}(\sqrt{KT\log K})$ [Seldin and Lugosi, 2017].

[1]To the best knowledge of the authors the naming of AlphaZero is not related to setting this α parameter to zero and this matching is thus a mere coincidence.

3.3 Combining PUCT and UCT

We have established that the reversed KL-divergence is equivalent to the f-divergence with $f(x) = -\log(x)$, i.e. the log-barrier. The f-divergence that results from a Tsallis entropy regularizer with $\alpha = 0.5$ can be easily computed from Equation (7) and is also equivalent to the squared Hellinger distance

$$f(x) = 2 - 2\sqrt{x}. \tag{9}$$

When we replace the KL-divergence in Equation (5) with this f-divergence, we obtain the following formula after simplifying:

$$\bar{\boldsymbol{\pi}}_{0.5} = \arg\max_{\mathbf{y}\in\mathcal{S}} \left[\boldsymbol{q}^{\mathsf{T}}\mathbf{y} + \lambda_N^* \sum_b \sqrt{\boldsymbol{\pi_{\theta}}_b \cdot \mathbf{y}_b}\right], \tag{10}$$

where the subscript on $\bar{\pi}$ denotes the value of α and λ_N^* denotes the learning rate for the f-divergence from Equation (9), such that the optimal regret bound is obtained. In Lemma 1 we determine λ_N^* and subsequently, through proving Proposition 1, we show how $\bar{\pi}_{0.5}$ resembles a combination of PUCT and UCT.

Lemma 1. $\lambda_N^* = \lambda_N$

Proof. This follows directly from Theorem 3 in [Zimmert and Seldin, 2018] and setting α to 0.5. □

Proposition 1. $\bar{\pi}_{0.5}$ *is the strategy that results from combining the f-divergence from UCT with the learning rate λ from PUCT.*

Proof. First we define the action selection formula for UCT:

$$a_{UCT}^* \triangleq \arg\max_a \left[Q(x,a) + c\cdot\sqrt{\boldsymbol{\pi_\theta}\cdot\frac{\log\sum_b n(x,b)}{1+n(x,a)}}\right]. \tag{11}$$

Note that π_θ can be assumed to be the uniform distribution when there is no better estimate for the prior. Now we rewrite a_{UCT}^* such that we can determine its underlying f-divergence and learning rate:

$$\begin{aligned} a_{UCT}^* &= \arg\max_a \\ &\left[Q(x,a) + c\cdot\sqrt{\log\sum_b n(x,b)}\cdot\sqrt{\frac{\boldsymbol{\pi_\theta}}{1+n(x,a)}}\right] \\ &= \arg\max_a \\ &\left[Q(x,a) + c\cdot\sqrt{\frac{\log\sum_b n(x,b)}{|\mathcal{A}|+\sum_b n(x,b)}}\cdot\sqrt{\frac{\boldsymbol{\pi_\theta}}{\frac{1+n(x,a)}{|\mathcal{A}|+\sum_b n(x,b)}}}\right], \end{aligned} \tag{12}$$

which then becomes

$$a_{UCT}^* = \arg\max_a \left[Q(x,a) - \lambda_N^{UCT}\cdot f'\left(\frac{\bar{\boldsymbol{\pi}}}{\boldsymbol{\pi_\theta}}\right)\right], \tag{13}$$

with $\lambda_N^{UCT} \triangleq c\cdot\sqrt{\frac{\log\sum_b n(x,b)}{|\mathcal{A}|+\sum_b n(x,b)}}$ and $f'(x) = -\frac{1}{\sqrt{x}}$.

Then we rewrite a^*_{UCT} using Lemma 2 in [Grill *et al.*, 2020]:

$$a^*_{UCT} = \arg\max_a \left[\frac{\partial}{\partial n_a} \left(\boldsymbol{q}^\top \hat{\boldsymbol{\pi}} - \lambda_N^{UCT} \cdot D_f \left(\hat{\boldsymbol{\pi}} || \boldsymbol{\pi_\theta} \right) \right) \right], \tag{14}$$

with D_f being the f-divergence from Equation (9). In combination with Lemma 1 this concludes the proof. □

4 Implementation

4.1 Separation of $\hat{\pi}$ and $\bar{\pi}$

In each MCTS iteration we compute $\bar{\pi}$ for both players as

$$\bar{\boldsymbol{\pi}} \triangleq \arg\max_{\mathbf{y}\in\mathcal{S}} \left[\boldsymbol{q}^\top \mathbf{y} - \lambda_N D_f \left(\boldsymbol{\pi_\theta} \| \mathbf{y} \right) \right]. \tag{15}$$

However, $\bar{\pi}$ depends on the Q-values $\mathbf{q}$. In sequential MCTS, $\mathbf{q}$ is readily available from the child nodes, but in simultaneous MCTS this typically gets computed as $\mathbf{q} = Q\pi_{opp}$, with Q being the matrix of running mean Q-values for action pairs and π_{opp} being the current best estimate of the opponent's strategy. In a standard Decoupled (P)UCT implementation we set π_{opp} to be equal to $\hat{\pi}_{opp}$, the empirical visit distribution of the opponent. However, due to the previously mentioned inaccuracy of $\hat{\pi}$, we instead leverage the $\bar{\pi}$ solutions from previous MCTS iterations. A naive approach would be to set π_{opp} to the computed $\bar{\pi}_{opp}$ from the last iteration, but this is problematic because $\bar{\pi}$ itself does not converge to a Nash equilibrium, whereas its average does, as mentioned in Section 2.2. Therefore, it is beneficial to use a (weighted) average of the $\bar{\pi}$ solutions from all the previous iterations.

Note that storing a weighted average of $\bar{\pi}$ solutions implies that an extra variable has to be kept inside of the nodes next to the already present variable storing the empirical visit distribution $\hat{\pi}$. With the approach described above, the latter variable is only needed for computing the running mean of the Q-values obtained from the search.

4.2 Efficient Computation of $\bar{\pi}$

In each MCTS iteration a convex optimization problem as in Equation (15) needs to be solved to compute the policy $\bar{\pi}$. It turns out that computing $\bar{\pi}$ can be done in a computationally efficient way. Analogously to Proposition 4 in [Grill *et al.*, 2020], where the $\alpha = 0$ case is studied, one can derive for the Tsallis entropy $\alpha = 0.5$ case:

$$\exists \alpha \in \mathbb{R} : q + \lambda_N \cdot \sqrt{\frac{\pi_\theta}{\bar{\pi}}} = \alpha \mathbb{1},$$

with $\mathbb{1}$ being the the vector such that $\forall a : \mathbb{1}_a = 1$. Therefore

$$\bar{\pi} = \left(\frac{\lambda_N \cdot \sqrt{\pi_\theta}}{\alpha - q} \right)^2,$$

with α set such that $\sum_b \bar{\pi}_b = 1$ and $\forall b : \bar{\pi}_b \geq 0$. Note that the α parameter here is different from the Tsallis-α.

We find the following bounds on α, using a similar derivation as [Grill *et al.*, 2020]:

$$\alpha_{min} \triangleq \max_{b\in\mathcal{A}} (q[b] + \lambda_N \cdot \sqrt{\pi_\theta[b]})$$
$$\alpha_{max} \triangleq \max_{b\in\mathcal{A}} q[b] + \lambda_N$$

As $\sum_b \bar{\pi}_\alpha[b]$ is strictly decreasing on $\alpha \in (\alpha_{min}, \alpha_{max})$, we can then compute $\bar{\pi}$ easily using dichotomic search over $(\alpha_{min}, \alpha_{max})$. Alternatively, it is also possible to utilize Newton's method to determine $\bar{\pi}$ as done in [Zimmert and Seldin, 2018].

4.3 Weighing of $\bar{\pi}$ Solutions

We said in Section 2.2 that the average of the strategies, i.e. the average of $\bar{\pi}$ solutions, converges to a Nash equilibrium. However, we found empirically that logarithmically increasing the weight of the $\bar{\pi}$ solutions in the average leads to faster convergence due to quicker decay of the impact of old solutions. We show that we are still guaranteed convergence to a Nash equilibrium by proving Proposition 2, which states that, asymptotically, there is no difference between an unweighted average and a log-weighted average.

Proposition 2.

$$\lim_{n\to\infty} \frac{\sum_{i=1}^n \log(i)\bar{\pi}_i}{\sum_{i=1}^n \log(i)} = \lim_{n\to\infty} \frac{\sum_{i=1}^n \bar{\pi}_i}{n} \tag{16}$$

Proof. First we rewrite the left-hand side as

$$\lim_{n\to\infty} \frac{\sum_{i=1}^n \log(i)\bar{\pi}_i}{\log n!}.$$

Using Stirling's approximation one has

$$\lim_{n\to\infty} \frac{\sum_{i=1}^n \log(i)\bar{\pi}_i}{\log n!} = \lim_{n\to\infty} \frac{\sum_{i=1}^n \log(i)\bar{\pi}_i}{n \log n} = \lim_{n\to\infty} \frac{\sum_{i=1}^n \frac{\log i}{\log n}\bar{\pi}_i}{n}.$$

We can now rewrite (16) as

$$\lim_{n\to\infty} \sum_{i=1}^n \frac{\log i}{\log n}\bar{\pi}_i = \lim_{n\to\infty} \sum_{i=1}^n \bar{\pi}_i$$

and using dot product notation one has

$$\lim_{n\to\infty} \left(\frac{\log 1}{\log n}, ..., \frac{\log n}{\log n} \right) \cdot \bar{\pi} = (1, ..., 1) \cdot \bar{\pi}$$

which is implied by

$$\lim_{n\to\infty} \left(\frac{\log 1}{\log n}, ..., \frac{\log n}{\log n} \right) = (1, ..., 1). \tag{17}$$

To show that (17) holds, we first note that the dot product of both sides is equal to n by using Stirling's approximation. This, combined with the norm of the right-hand side being equal to $\sqrt{n}$, implies that the norm of the left-hand side must be $\geq \sqrt{n}$, and equal to $\sqrt{n}$ only if both vectors are identical. Since the absolute values of the elements of the vector on the left-hand side are all ≤ 1, the norm must be $\leq \sqrt{n}$, implying that both vectors are equal.

□

4.4 Imputation of Q-values

To get a better Q-value estimation of the unvisited nodes, we set the Q-value of the unvisited nodes to be the mean Q-value of the visited nodes each time a tree node is selected, similar to the implementation of ELF OpenGo [Tian *et al.*, 2019].

Parameter	Value
Pod radius	400
Checkpoint radius	600
Pod mass	1
Friction coefficient	0.15
Shield timer	4
Shield mass multiplier	10
Thrust	[0, 200]
Angle	[-18, 18]

Table 1: Parameter values for Pod Pursuit

5 Experiments

5.1 Test Domain

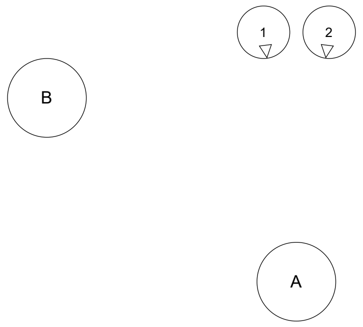
As test domain we use an adaptation of the bot programming game *Mad Pod Racing* (MPR) - formerly known as *Coders Strike Back* (CSB) - on the CodinGame competitive programming platform. This is the platform's most popular game, with supposedly over 100,000 participants. The top entries are based on a variety of reinforcement learning algorithms, such as AlphaZero, DQN [2] [Mnih *et al.*, 2013] and A2C [Mnih *et al.*, 2016].

MPR/CSB is a zero-sum two-player simultaneous move game with a continuous action and state space in which each agent controls two Star Wars-themed pods, which participate in a race against the pods of the opponent. However, for testing our algorithm we will instead use a simplified version of the game which we call *Pod Pursuit*. Here each player controls one pod, with one player focusing strictly on preventing the other player from completing the race. This leads to an asymmetric game that has some high-level similarity to pursuit-evasion games. These types of games are commonly studied in simultaneous move settings [Bosansky *et al.*, 2016]. Additionally, pursuit-evasion games tend to suit UCT well due to a high frequency of the states requiring pure strategies [Bosansky *et al.*, 2016]. Thus making it more challenging to compete against UCT with an algorithm seeking a (possibly mixed) Nash equilibrium strategy.

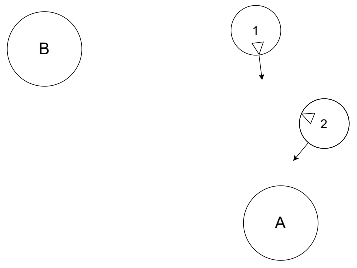
Another reason why we deem this game as a good test domain for our algorithm is because it is a game that is highly suited for reinforcement learning, or stated alternatively: it is very difficult to obtain comparable results with alternative methods. On top of that, the game is simple enough to enable successful training on high-end consumer hardware, yet playing optimally is (highly) non-trivial. Lastly, we note that the CodinGame platform has been used before to conduct reinforcement learning research, e.g. [Kowalski and Miernik, 2020].

5.2 Game Description

Pod Pursuit [3] is a fully observable asymmetric zero-sum two-player simultaneous move game featuring two pods. Pod 1's task is to hit one or more checkpoints in a given order, with a limited amount of turns for each checkpoint. Pod 2's task is to prevent Pod 1 from completing its task by blocking its path. The pods move continuously in a minimalist 2D physics engine, though the actions and friction are applied in discrete turns. Each turn the players determine the next action to apply to their pod, where each action consists out of both a thrust and an angle. As in the original game, both pods also have to ability to *shield* instead of thrust, which causes their mass to increase tenfold for one turn, at the cost of not being able to apply thrust for some turns. While the ability to shield may seem like a gimmick at first, it has profound consequences on the game. For example, we conjecture that the ability to shield results in any game state being a theoretical win for Pod 1, assuming the absence of a turn limit. Otherwise, some positions would be trivial wins for Pod 2. An illustration of the game is provided in Figure 1. In Figure 1a we can see how both pods start out in an idle state, both facing the first checkpoint from an equal distance, separated by a distance of 1000. In Figure 1b we can see how the pods have left their starting position and how Pod 2 is now in a position where it can prevent Pod 1 from easily capturing the next checkpoint. In Table 1 we show an overview of the relevant parameters for the physics engine. We choose to leave the parameters unit-less in general, but the interaction between the different quantities should happen as if they were SI units with the exception of the angle, which is in degrees instead of radians. The game features two types of collisions: pod-pod collisions and pod-checkpoint collisions. Pod-pod collisions are elastic and have a minimum impulse of 120. Pod-checkpoint collisions happen when the center of the pod is within the radius of the checkpoint and do not result in a physical effect.



(a) An initial configuration of the pods, both facing checkpoint A.

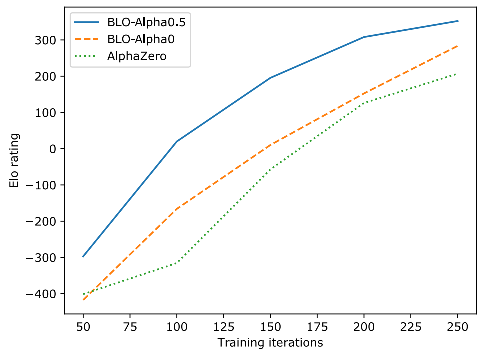


(b) Pod 2 managed to get in front of Pod 1 and may prevent Pod 1 from reaching checkpoint A.

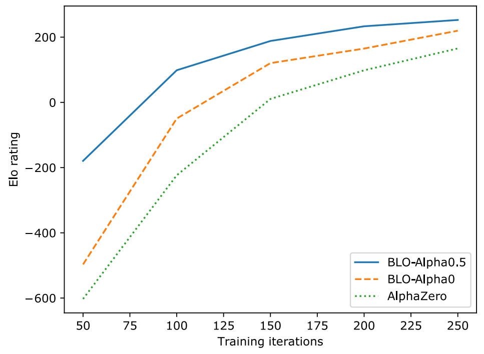
Figure 1: Depictions of two game states. The heading of the pods is indicated with a triangle and the velocity vector with an arrow.

[2]A write-up of a DQN implementation can be found here: https://github.com/pb4git/Nash-DQN-CSB-Article

[3]Implementation at: https://github.com/mpeelm/Pod-Pursuit



(a) 50 search iterations per turn



(b) 200 search iterations per turn

Figure 2: Training BLO-Alpha0.5, BLO-Alpha0 and AlphaZero for 250 iterations, where in each iteration the network parameters are updated based on the result of 4000 games, resulting in a total of one million games played for each algorithm. Every model is trained three times independently and the shown data points are the mean performance. The Elo rating is computed in a tournament setting that includes all 45 models.

5.3 Algorithm Comparison

In our experiments we are interested in the comparison of three algorithms. Firstly, we test a baseline straightforward extension of AlphaZero for simultaneous move games, employing Decoupled PUCT as search method. As in the AlphaZero paper, we make use of Dirichlet noise in the root node. The parameters for the noise are $\epsilon = 0.25$ and $\alpha = 1$. Secondly, we test BLO-Alpha0, which replaces AlphaZero's PUCT with the search detailed in Section 4, but with the Tsallis-α set to zero. And lastly, we test BLO-Alpha0.5, which is identical to BLO-Alpha0, except for using regularization based on the Tsallis entropy with $\alpha = 0.5$.

Each algorithm uses the outcomes and computed policies from self-play games to update an MLP with three hidden layers with dimension 88 and ReLU as activation function. The optimizer used is SGD with momentum (0.9) and a constant learning rate of 0.1. The inputs of the network are the (scaled) relative positions between the pods and the next two checkpoints (12 inputs), as well as the (scaled) velocity vectors of the pods (4 inputs) and the one-hot encoded shield timers (8 inputs). This yields a total of 24 inputs. The network has both a value-head and a policy-head and the latter has 7 actions that discretize the action space into actions with thrust in $\{0, 200\}$ and angles in $\{-18, 0, 18\}$, plus a shield action with angle 0. Lastly, we use a discount factor γ of 0.99 and the constant c in the definition of λ_N is set to 1.

In Figure 2a we can see the (relative) Elo rating of the three algorithms when each algorithm gets 50 search iterations per turn. We can observe that BLO-Alpha0.5 does significantly better than the other two algorithms, while BLO-Alpha0 still yields a modest improvement over AlphaZero, which is expected due to the results obtained for low simulation budgets in [Grill *et al.*, 2020]. However, when we look at Figure 2b, in which 200 iterations are used per turn, we can see that the BLO algorithms still perform better than AlphaZero's PUCT, which shows that even for higher simulation budgets it is worthwhile to use a more precise search algorithm.

For the BLO algorithms we opted not to use Dirichlet noise. While Dirichlet noise can be beneficial during the early training, later on it actually tends to slow down improvement. We attempted to run AlphaZero without Dirichlet noise as well, but found that this hurts AlphaZero's ability to improve severely at 50 search iterations per turn. Increasing the amount of search iterations has been observed to overcome this, and thus allows AlphaZero to learn without Dirichlet noise, but we found that even in this setting, the presence of Dirichlet noise is beneficial to learning. Taking this into consideration we don't deem the asymmetry of the presence of Dirichlet noise among the algorithms as something that hurts the validity of our experiments.

6 Conclusion

We have shown that the PUCT algorithm used by AlphaZero and MuZero corresponds to an MCTS using log-barrier regularization. This insight has allowed us to use the Tsallis entropy to generalize to different kinds of divergences. Previous work on bandit algorithms has shown that Tsallis entropy with $\alpha = 0.5$ obtains the theoretically optimal regret bound. After experimentally testing the impact of the value of α on the rate of improvement for a search-based reinforcement learning algorithm, we found that setting α to 0.5 yields significant improvements compared to the log-barrier regularization used by PUCT. Additionally, we developed our algorithm with an ideology rooted in bandit linear optimization, most notably leading to the maintaining of the average strategy in the MCTS nodes instead of the usual empirical visit counts. We tested our algorithm on a simultaneous move game since these kinds of games have an especially strong theoretical need for an alternative to PUCT due to the failure of PUCT to converge to a Nash equilibrium, but we expect that our findings will translate well to other domains, such as sequential move games and non-adversarial games.

Acknowledgments

This research received funding from the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme. We also want to thank Roman Ring and Nicolas Vercheval for helpful discussions.

References

[Abdolmaleki *et al.*, 2018] Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Rémi Munos, Nicolas Heess, and Martin A. Riedmiller. Maximum a posteriori policy optimisation. *CoRR*, abs/1806.06920, 2018.

[Abernethy *et al.*, 2008] Jacob Abernethy, Elad Hazan, and Alexander Rakhlin. Competing in the dark: An efficient algorithm for bandit linear optimization. volume 3, pages 263–274, 01 2008.

[Abernethy *et al.*, 2015] Jacob D. Abernethy, Chansoo Lee, and Ambuj Tewari. Fighting bandits with a new kind of smoothness. In *NIPS*, 2015.

[Audibert and Bubeck, 2009] Jean-Yves Audibert and Sébastien Bubeck. Minimax Policies for Adversarial and Stochastic Bandits. In *Proceedings of the 22nd Annual Conference on Learning Theory (COLT)*, January 2009. Edition: Proceedings of the 22nd Annual Conference on Learning Theory (COLT).

[Auer *et al.*, 2003] Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM J. Comput.*, 32(1):48–77, January 2003.

[Blum and Mansour, 2007] Avrim Blum and Yishay Mansour. Learning, regret minimization, and equilibria. *Algorithmic Game Theory*, 01 2007.

[Bosansky *et al.*, 2016] Branislav Bosansky, Viliam Lisy, Marc Lanctot, Jiri Cermak, and Mark Winands. Algorithms for computing strategies in two-player simultaneous move games. *Artificial Intelligence*, 237, 04 2016.

[Foster *et al.*, 2016] Dylan J. Foster, Zhiyuan Li, Thodoris Lykouris, Karthik Sridharan, and Éva Tardos. Fast convergence of common learning algorithms in games. *CoRR*, abs/1606.06244, 2016.

[Grill *et al.*, 2020] Jean-Bastien Grill, Florent Altché, Yunhao Tang, Thomas Hubert, Michal Valko, Ioannis Antonoglou, and Rémi Munos. Monte-carlo tree search as regularized policy optimization. *CoRR*, abs/2007.12509, 2020.

[Hart and Mas-Colell, 2001] Sergiu Hart and Andreu Mas-Colell. A reinforcement procedure leading to correlated equilibrium. *Economic Essays*, 01 2001.

[Hazan, 2019] Elad Hazan. Introduction to online convex optimization. *CoRR*, abs/1909.05207, 2019.

[Kocsis and Szepesvári, 2006] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. volume 2006, pages 282–293, 09 2006.

[Kovarík and Lisý, 2018] Vojtech Kovarík and Viliam Lisý. Analysis of hannan consistent selection for monte carlo tree search in simultaneous move games. *CoRR*, abs/1804.09045, 2018.

[Kowalski and Miernik, 2020] Jakub Kowalski and Radoslaw Miernik. Evolutionary approach to collectible card game arena deckbuilding using active genes. *CoRR*, abs/2001.01326, 2020.

[Mnih *et al.*, 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[Mnih *et al.*, 2016] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

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

[Schmid *et al.*, 2021] Martin Schmid, Matej Moravcik, Neil Burch, Rudolf Kadlec, Joshua Davidson, Kevin Waugh, Nolan Bard, Finbarr Timbers, Marc Lanctot, Zach Holland, Elnaz Davoodi, Alden Christianson, and Michael Bowling. Player of games. *CoRR*, abs/2112.03178, 2021.

[Schrittwieser *et al.*, 2019] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *CoRR*, abs/1911.08265, 2019.

[Seldin and Lugosi, 2017] Yevgeny Seldin and Gábor Lugosi. An improved parametrization and analysis of the EXP3++ algorithm for stochastic and adversarial bandits. *CoRR*, abs/1702.06103, 2017.

[Shafiei *et al.*, 2009] Mohammad Shafiei, Nathan R Sturtevant, and Jonathan Schaeffer. Comparing uct versus cfr in simultaneous games. 2009.

[Silver *et al.*, 2017] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017.

[Tak *et al.*, 2014] Mandy Tak, Marc Lanctot, and Mark Winands. Monte carlo tree search variants for simultaneous move games. 08 2014.

[Tian *et al.*, 2019] Yuandong Tian, Jerry Ma, Qucheng Gong, Shubho Sengupta, Zhuoyuan Chen, James Pinkerton, and C. Lawrence Zitnick. ELF opengo: An analysis and open reimplementation of alphazero. *CoRR*, abs/1902.04522, 2019.

[Tsallis, 1988] Constantino Tsallis. Possible generalization of boltzmann-gibbs statistics. *Journal of Statistical Physics*, 52:479–487, 07 1988.

[Wei and Luo, 2018] Chen-Yu Wei and Haipeng Luo. More adaptive algorithms for adversarial bandits. *CoRR*, abs/1801.03265, 2018.

[Zimmert and Seldin, 2018] Julian Zimmert and Yevgeny Seldin. An optimal algorithm for stochastic and adversarial bandits. *CoRR*, abs/1807.07623, 2018.