Smooth UCT Search in Computer Poker

Johannes Heinrich
University College London
London, UK
j.heinrich@cs.ucl.ac.uk

David Silver
Google DeepMind
London, UK
davidsilver@google.com

Abstract

Self-play Monte Carlo Tree Search (MCTS) has been successful in many perfect-information two-player games. Although these methods have been extended to imperfect-information games, so far they have not achieved the same level of practical success or theoretical convergence guarantees as competing methods. In this paper we introduce Smooth UCT, a variant of the established Upper Confidence Bounds Applied to Trees (UCT) algorithm. Smooth UCT agents mix in their average policy during self-play and the resulting planning process resembles game-theoretic fictitious play. When applied to Kuhn and Leduc poker, Smooth UCT approached a Nash equilibrium, whereas UCT diverged. In addition, Smooth UCT outperformed UCT in Limit Texas Hold'em and won 3 silver medals in the 2014 Annual Computer Poker Competition.

1 Introduction

MCTS [Coulom, 2007] is a simulation-based search algorithm that has been incredibly successful in perfect-information domains [Browne *et al.*, 2012; Gelly *et al.*, 2012]. Its success is often attributed to prioritising the most promising regions of the search space by sampling trajectories of the game selectively. Furthermore, by planning online it can focus its search effort on a relevant subset of states.

Applications of MCTS to imperfect-information games have faced major difficulties. Common MCTS variants lack convergence guarantees and have failed to converge in practice [Ponsen *et al.*, 2011; Lisý, 2014]. Furthermore, solving a subgame of an imperfect-information game, by using online search, can produce exploitable strategies [Burch *et al.*, 2014; Ganzfried and Sandholm, 2015]. Online Outcome Sampling [Lisý *et al.*, 2015], a variant of Monte Carlo counterfactual regret minimization (MCCFR) [Lanctot *et al.*, 2009], is the first MCTS approach that addresses both of these problems in a theoretically sound manner. It is guaranteed to converge to a Nash equilibrium in all two-player zero-sum games, using either full-game or online search. In computer poker, Ponsen *et al.* [2011] compared the performance of Outcome Sampling to UCT [Kocsis and Szepesvári, 2006], a popular MCTS variant. They concluded that UCT quickly finds a good but suboptimal policy, while Outcome Sampling initially learns more slowly but converges to the optimal policy over time. In this paper, we address the question whether the inability of UCT to converge to a Nash equilibrium can be overcome while retaining UCT's fast initial learning rate. We focus on the full-game MCTS setting, which is an important step towards developing sound variants of online MCTS in imperfect-information games.

In particular, we introduce Smooth UCT, which combines the notion of fictitious play [Brown, 1949] with MCTS. Fictitious players perform the best response to other players' average behaviour. We introduce this idea into MCTS by letting agents mix in their average strategy with their usual utility-maximizing actions. Intuitively, apart from mimicking fictitious play, this might have further potential benefits, e.g. breaking correlations and stabilising self-play learning due to more smoothly changing agent behaviour.

We evaluated Smooth UCT in two sets of experiments. Firstly, we compared to UCT and Outcome Sampling in Kuhn Poker and Leduc Hold'em. Confirming the observations of [Ponsen *et al.*, 2011], both UCT-based methods initially learned faster than Outcome Sampling but UCT later suffered divergent behaviour and failure to converge to a Nash equilibrium. Smooth UCT, on the other hand, continued to approach a Nash equilibrium, but was eventually overtaken by Outcome Sampling. Secondly, we evaluated Smooth UCT and UCT in Limit Texas Hold'em (LHE). This poker variant has been a major driver of advances in computational game theory and is one of the primary games of the Annual Computer Poker Competition (ACPC) [Bard *et al.*, 2013]. Smooth UCT outperformed UCT in two- and three-player LHE and achieved 3 silver medals in the 2014 ACPC. The most impressive aspect of this result is that it was achieved using a fraction of the computational resources relative to other competitors.

2 Background

2.1 Extensive-Form Games

Extensive-form games are a model of sequential interaction involving multiple agents. The representation is based on a game tree and consists of the following components: $\mathcal{N} =$

$\{1, ..., n\}$ denotes the set of players. $\mathcal{S}$ is a set of states corresponding to nodes in a finite rooted game tree. For each state node $s \in \mathcal{S}$ the edges to its successor states define a set of actions $\mathcal{A}(s)$ available to a player or chance in state s. The player function $P : \mathcal{S} \to \mathcal{N} \cup \{c\}$, with c denoting chance, determines who is to act at a given state. Chance is considered to be a particular player that follows a fixed randomized strategy that determines the distribution of chance events at chance nodes. For each player i there is a corresponding set of information states $\mathcal{U}^i$ and an information function $I^i : \mathcal{S} \to \mathcal{U}^i$ that determines which states are indistinguishable for the player by mapping them on the same information state $u \in \mathcal{U}^i$. Finally, $R : \mathcal{S} \to \mathbb{R}^n$ maps terminal states to a vector whose components correspond to each player's payoff.

A game has the property of perfect recall if each player's current information state u^i_k implies knowledge of the sequence of their information states and actions, $u^i_1, a^i_1, u^i_2, a^i_2, ..., u^i_k$, that led to this information state. If a player only knows some strict subset of this sequence then the game is said to have imperfect recall.

An abstraction of an extensive-form game aggregates some of the players' information states and therefore does not let the players distinguish between the aggregated states. Formally, an abstraction of an extensive-form game Γ is a collection of surjective functions, $f^i_A : \mathcal{U}^i \to \tilde{\mathcal{U}}^i$, $i \in \mathcal{N}$, that map the information states of Γ to alternative information state spaces $\tilde{\mathcal{U}}^i$. An abstraction can significantly lower the size of a game while partially retaining its strategic structure.

A player's behavioural strategy, $\pi^i(u) \in \Delta\left(\mathcal{A}(u)\right), \forall u \in \mathcal{U}^i$, determines a probability distribution over actions given an information state, and Π^i is the set of all behavioural strategies of player i. A strategy profile $\pi = (\pi^1, ..., \pi^n)$ is a collection of strategies for all players. π^{-i} refers to all strategies in π except π^i. $R^i(\pi)$ is the expected payoff of player i if all players follow the strategy profile π. The set of best responses of player i to their opponents' strategies π^{-i} is $b^i(\pi^{-i}) = \arg\max_{\pi^i \in \Pi^i} R^i(\pi^i, \pi^{-i})$. For $\epsilon > 0$, $b^i_\epsilon(\pi^{-i}) = \{\pi^i \in \Pi^i : R^i(\pi^i, \pi^{-i}) \geq R^i(b^i(\pi^{-i}), \pi^{-i}) - \epsilon\}$ defines the set of ϵ-best responses to the strategy profile π^{-i}.

Definition 1. A Nash equilibrium of an extensive-form game is a strategy profile π such that $\pi^i \in b^i(\pi^{-i})$ for all $i \in N$. An ϵ-Nash equilibrium is a strategy profile π such that $\pi^i \in b^i_\epsilon(\pi^{-i})$ for all $i \in N$.

2.2 MCTS

MCTS [Coulom, 2007; Browne *et al.*, 2012] is a simulation-based search algorithm. It is able to plan in high-dimensional environments by sampling episodes through Monte Carlo simulation. These simulations are guided by an action selection mechanism that explores the most promising regions of the state space, resulting in asymmetric search trees that provide high quality value estimates in practice. MCTS produces optimal policies in some Markovian environments, e.g. partially observable Markov decision processes (POMDPs) [Silver and Veness, 2010] and perfect-information two-player zero-sum games [Kocsis and Szepesvári, 2006].

A MCTS algorithm requires the following components: a) A black box simulator that, given a state and action, samples a successor state and reward. b) A learning algorithm that uses simulated trajectories and outcomes to update statistics in the search tree's visited nodes. c) A tree policy to define an action selection mechanism that chooses actions based on a node's statistics. d) A rollout policy that determines the default behaviour for states that are out of the scope of the search tree. For a specified amount of planning time, MCTS repeats the following. It starts each Monte Carlo simulation at the root node and follows its tree policy until either reaching a terminal state or the boundary of the search tree. Leaving the scope of the search tree, the rollout policy is used to play out the simulation until reaching a terminal state. In this case, we expand our tree by a state node where we have left the tree. This approach selectively grows the tree in areas that are frequently encountered in simulations. After reaching a terminal state, the rewards are propagated back so that each visited node can update its statistics.

Common MCTS keeps track of the following node values. $N(s)$ is the number of visits by a Monte Carlo simulation to node s. $N(s, a)$ counts the number of times action a has been chosen at node s. $Q(s, a)$ is the estimated value of choosing action a at node s. The action value estimates are usually updated by Monte Carlo evaluation, e.g. $Q(s, a) = \frac{1}{N(s,a)} \sum_{k=1}^{N(s,a)} G(s, a, k)$, where $G(s, a, k)$ is the cumulative discounted reward achieved after visiting state s and taking action a in the k-th simulation that encountered s.

[Kocsis and Szepesvári, 2006] suggested using the bandit-based algorithm UCB [Auer *et al.*, 2002] to recursively select actions in the search tree. The resulting MCTS method, UCT, selects greedily amongst action values that have been enhanced by an exploration bonus,

$$\pi_{tree}(s) = \arg\max_{a \in \mathcal{A}(s)} Q(s, a) + c\sqrt{\frac{\log N(s)}{N(s, a)}}, \qquad (1)$$

breaking ties uniformly at random. The exploration bonus parameter c adjusts the balance between exploration and exploitation. For suitable c, the probability of choosing a suboptimal action converges to 0 [Kocsis and Szepesvári, 2006].

3 Self-Play MCTS in Extensive-Form Games

This section develops Smooth UCT, a self-play MCTS algorithm for planning in extensive-form games. We first extend the work of [Auger, 2011] and [Cowling *et al.*, 2012] to a general MCTS algorithm for extensive-form games, and then introduce Smooth UCT as a particular instantiation of this family of methods.

The asymmetry of information in an extensive-form game with imperfect information does not allow for a single collective search tree. We therefore describe an algorithm that uses a separate search tree for each player. For each player we grow a tree T^i over their information states $\mathcal{U}^i$. $T^i(u^i)$ is the node in player i's tree that represents their information state u^i. In a game with perfect recall, a player i's sequence of previous information states and actions, $u^i_1, a^i_1, u^i_2, a^i_2, ..., u^i_k$, is incorporated in the information state u^i_k and therefore T^i is a proper tree. The imperfect-recall abstractions that we used in our LHE experiments yield recombining trees.

Algorithm 1 describes MCTS that has been adapted to the multi-player imperfect-information setting of extensive-form games. The game mechanics are sampled from transition and reward simulators $\mathcal{G}$ and $\mathcal{R}$. The transition simulator takes a state and action as inputs and generates a sample of a successor state $s_{t+1} \sim \mathcal{G}(s_t, a_t^i)$, where the action a_t^i belongs to the player i who makes decisions at state s_t. The reward simulator generates all players' payoffs at terminal states, i.e. $\boldsymbol{r}_T \sim \mathcal{R}(s_T)$. The information function, $I^i(s)$, determines the acting player i's information state. OUT-OF-TREE keeps track of which player has left the scope of their search tree in the current episode.

This algorithm can be specified by the action selection and node updating functions, SELECT and UPDATE. These functions are responsible to sample from and update the tree policy. In this work, we focus on UCT-based methods.

Algorithm 1 Self-play MCTS in extensive-form games

```
function SEARCH(Γ)
    while within computational budget do
        s_0 ~ Γ
        SIMULATE(s_0)
    end while
    return π_tree
end function
function ROLLOUT(s)
    a ~ π_rollout(s)
    s' ~ G(s, a)
    return SIMULATE(s')
end function
function SIMULATE(s)
    if ISTERMINAL(s) then
        return r ~ R(s)
    end if
    i = PLAYER(s)
    if OUT-OF-TREE(i) then
        return ROLLOUT(s)
    end if
    u^i = I^i(s)
    if u^i ∉ T^i then
        EXPANDTREE(T^i, u^i)
        a ~ π_rollout(s)
        OUT-OF-TREE(i) ← true
    else
        a = SELECT(u^i)
    end if
    s' ~ G(s, a)
    r ← SIMULATE(s')
    UPDATE(u^i, a, r^i)
    return r
end function
```

3.1 Extensive-Form UCT

Extensive-form UCT uses UCB to select from and update the tree policy in algorithm 1. It can be seen as a multi-agent version of Partially Observable UCT (PO-UCT) [Silver and Veness, 2010], that searches in trees spanned over information states of a perfect-recall extensive-form game instead of histories of observations and actions of a POMDP.

3.2 Extensive-Form Smooth UCT

Smooth UCT is a MCTS algorithm that selects actions from Smooth UCB, a variant of the bandit algorithm UCB. Smooth UCB is inspired by fictitious play [Brown, 1949; Leslie and Collins, 2006] which is a game-theoretic model of learning in games. In fictitious play, players repeatedly play a game and at each iteration choose a best response to their opponents' average strategies. The average strategy profile of fictitious players converges to a Nash equilibrium in certain classes of games, e.g. two-player zero-sum and potential games.

By counting how often each action has been taken, UCB already keeps track of an average strategy π. It can be readily extracted by setting $\pi(a) = \frac{N(a)}{N}, \forall a \in \mathcal{A}$. However, UCB does not use this strategy and therefore potentially ignores some useful information while planning.

The basic idea of Smooth UCB is to mix in the average strategy when selecting actions in order to induce the other agents to respond to it. This resembles the idea of fictitious play, where agents are supposed to best respond to the average strategy. The average strategy might have further beneficial properties. Firstly, it is a stochastic strategy and it changes ever more slowly over time. This can decrease correlation between the players' actions and thus help to stabilise the self-play process. Furthermore, a more smoothly changing strategy can be relied upon by other agents and is easier to adapt to than an erratically changing greedy policy like UCB.

Smooth UCB requires the same information as UCB but explicitly makes use of the average strategy via the action counts. In particular, it mixes between UCB and the average strategy with probability η_k, where η_k is an iteration k-adapted sequence with $\eta_k \to \gamma > 0$ as $k \to \infty$. In this work, we use

$$\eta_k = \max\left(\gamma, \eta\left(1 + d\sqrt{N_k}\right)^{-1}\right), \tag{2}$$

where γ, η and d are constants that parameterise the schedule and N_k is the total number of visits to the respective node.

As UCT is obtained from applying UCB at each information state, we similarly define Smooth UCT as a MCTS algorithm that uses Smooth UCB at each information state. Algorithm 2 instantiates extensive-form MCTS with a Smooth UCB tree policy. For a constant $\eta_k = 1$, we obtain UCT as a special case. Compared to UCT, the UPDATE operation is left unchanged. Furthermore, for a cheaply determined η_k the SELECT procedure has little overhead compared to UCT.

4 Experiments

We evaluated Smooth UCT in the Kuhn, Leduc and Limit Texas Hold'em poker games.

Kuhn poker [Kuhn, 1950] and Leduc Hold'em [Southey *et al.*, 2005] are small imperfect-information two-player zero-sum games, for which the quality of an approximate Nash equilibrium can be readily evaluated. Both games feature typical elements of poker, e.g. hiding information via a balanced stochastic strategy. Kuhn poker consists of only one betting

Algorithm 2 Smooth UCT

SEARCH(Γ), SIMULATE(s) and ROLLOUT(s) as in algorithm 1
function SELECT(u^i)
  $z \sim U[0,1]$
  if $z < \eta_t(u^i)$ **then**
    return $\arg\max_a Q(u^i,a) + c\sqrt{\frac{\log N(u^i)}{N(u^i,a)}}$
  else
    $\forall a \in A(u^i) : p(a) \leftarrow \frac{N(u^i,a)}{N(u^i)}$
    return $a \sim p$
  end if
end function
function UPDATE(u^i, a, r^i)
  $N(u^i) \leftarrow N(u^i) + 1$
  $N(u^i,a) \leftarrow N(u^i,a) + 1$
  $Q(u^i,a) \leftarrow Q(u^i,a) + \frac{r^i - Q(u^i,a)}{N(u^i,a)}$
end function

round and does not include public community cards. A parameterization of its Nash equilibrium solutions is known in closed form. Leduc Hold'em is a poker variant that is similar to Texas Hold'em. However, with two betting rounds, a limit of two raises per round and only 6 cards in the deck it is a much smaller game. Texas Hold'em is a poker game that is not only popular with humans but it is also a focus of research in computational game theory [Sandholm, 2010]. The Limit variant restricts betting to fixed increments, which puts it just within reach of current approaches. Recently Bowling *et al.* [2015] solved two-player LHE after more than 10 years of effort since it had been identified as a fruitful and challenging research topic [Billings *et al.*, 2002].

4.1 Kuhn Poker

[Ponsen *et al.*, 2011] tested UCT against Outcome Sampling in Kuhn poker. We conducted a similar experiment, comparing UCT to Smooth UCT. Learning performance was measured in terms of the average policies' mean squared errors with respect to the closest Nash equilibrium determined from the known parameterization of equilibria. Smooth UCT's mixing parameter schedule (2) was manually calibrated and set to $\gamma = 0.1$, $\eta = 0.9$ and $d = 0.001$. We calibrated the exploration parameters of UCT and Smooth UCT by training 4 times for 10 million episodes each with parameter settings of $c = 0.25k$, $k = 1, ..., 10$. The best average final performance of UCT and Smooth UCT was achieved with 2 and 1.75 respectively. In each main experiment, each algorithm trained for 20 million episodes. The results, shown in figure 1, were averaged over 50 repeated runs of the experiment.

The results demonstrate that Smooth UCT approached a Nash equilibrium, whereas UCT exhibited divergent performance. Furthermore, after 600 simulated episodes Smooth UCT performed strictly better.

4.2 Leduc Hold'em

We also compared Smooth UCT to Outcome Sampling and UCT in Leduc Hold'em. Due to not knowing the Nash equilibria in closed form, we used the exploitability of a strategy profile to measure learning performance. In a two-player zero-sum game, the exploitability of a strategy profile, π, is defined as $\delta = R^1\left(b^1(\pi^2), \pi^2\right) + R^2\left(\pi^1, b^2(\pi^1)\right)$. An exploitability of δ yields at least a δ-Nash equilibrium.

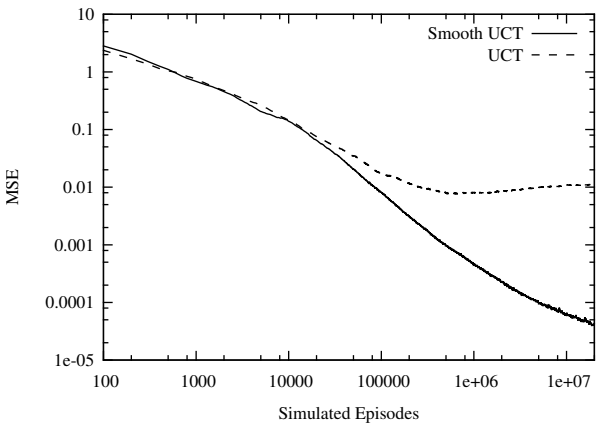

Figure 1: Learning curves in Kuhn poker.

Smooth UCT's mixing parameter schedule (2) was manually calibrated and set to $\gamma = 0.1$, $\eta = 0.9$ and $d = 0.002$. Smooth UCT and UCT trained 5 times for 500 million episodes each with exploration parameter settings of $c = 14 + 2k$, $k = 0, ..., 4$. We report the best average performance which was achieved with $c = 20$ and $c = 18$ for UCT and Smooth UCT respectively.

We used both Parallel and Alternating Outcome Sampling [Lanctot, 2013]. Both variants trained for 500 million episodes with exploration parameter settings of $\epsilon = 0.4 + 0.1k$, $k = 0, ..., 4$. We report the best results which were achieved with $\epsilon = 0.5$ for both variants.

The results in figure 2 show that both UCT-based methods initially learned faster than Outcome Sampling. However, UCT diverged rather quickly and never reached a level of low exploitability. Smooth UCT, on the other hand, learned just as fast as UCT but continued to approach a Nash equilibrium. Only after about 85 million simulated episodes and at an exploitability of 0.036 Alternating Outcome Sampling achieved better performance. Parallel Outcome Sampling overtook Smooth UCT after about 170 million episodes and at an exploitability of 0.028.

The results suggest that Smooth UCT is an interesting candidate for online MCTS. In online MCTS the computational budget for local replanning is usually very limited and being able to learn fast initially is therefore a very beneficial property. However, for a longer-time full-game search Outcome Sampling might yield better performance than Smooth UCT.

4.3 Limit Texas Hold'em

Finally, we consider LHE with two and three players. The two-player game tree contains about 10^{18} nodes. In order to reduce the game tree to a tractable size, various abstraction techniques have been proposed in the literature [Billings *et al.*, 2003; Johanson *et al.*, 2013]. The most common approach is to group information states according to strategic

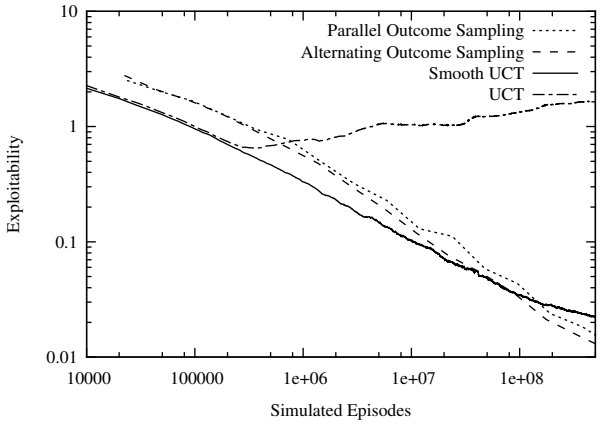

Figure 2: Learning curves in Leduc Hold'em.

Game	Preflop	Flop	Turn	River
two-player LHE	169	1000	500	200
three-player LHE	169	1000	100	20

Table 1: $\mathbb{E}[\text{HS}^2]$ discretisation grids used in experiments.

similarity of the corresponding cards and leave the players' action sequences unabstracted.

A player i's information state can be described as a tuple $u_t^i = (\bar{\boldsymbol{a}}_t, \boldsymbol{c}_t, \boldsymbol{p}_t^i)$, where $\bar{\boldsymbol{a}}_t$ is the sequence of all players' actions except chance, $\boldsymbol{p}_t^i$ are the two private cards held by player i and $\boldsymbol{c}_t$ is the sequence of community cards publicly revealed by chance by time t. Card-bucketing [Billings *et al.*, 2003] maps combinations of cards, $(\boldsymbol{c}, \boldsymbol{p})$, to an abstraction bucket identified by a number $n \in \mathbb{N}$. Thus it defines an abstraction that maps the information states to alternative information states that have the form of tuples $(\bar{\boldsymbol{a}}, n)$, where $\bar{\boldsymbol{a}}$ is the unabstracted sequence of all players' actions except chance and $n \in \mathbb{N}$ identifies the abstraction bucket.

In this work, we use a common bucketing metric called expected hand strength squared [Zinkevich *et al.*, 2007]. At a final betting round the hand strength of a combination of cards, $(\boldsymbol{c}, \boldsymbol{p})$, is defined as its winning percentage against all combinations of c with any other possible private holding. On any betting round, $\mathbb{E}[\text{HS}^2]$ denotes the expected value of the squared hand strength on the final betting round. We have discretised the resulting $\mathbb{E}[\text{HS}^2]$ values with an equidistant grid over their range, $[0, 1]$. Table 1 shows the grid sizes that we used in our experiments. We used an imperfect recall abstraction that does not let players remember their $\mathbb{E}[\text{HS}^2]$ values of previous betting rounds.

In LHE, there is an upper bound on the possible final pot size given the betting that has occurred so far in the episode. This is because betting is capped at each betting round. To avoid potentially unnecessary exploration we dynamically update the exploration parameter during a simulated episode at the beginning of each betting round and set it to

$$c = \min\left(C,\ \text{potsize} + k * \text{remaining betting potential}\right) \quad (3)$$

where C and $k \in [0, 1]$ are constant parameters and the remaining betting potential is the maximum possible amount that players can add to the pot in the remainder of the episode.

Annual Computer Poker Competition

We submitted SmooCT, an agent trained with an early version of the Smooth UCT algorithm [Heinrich and Silver, 2014], to the 2014 ACPC, where it achieved three silver medals in the LHE competitions. The Smooth UCT agent presented in this paper fixes several issues present in the SmooCT agent submitted to the competition: We use a more principled mixing parameter schedule (2), we fixed a bug in SmooCT's implementation of preflop abstractions and we do not use any refinement of abstraction granularity in often-visited subtrees.

Two-player

We trained strategies for two-player LHE by UCT and Smooth UCT. Both methods performed a simulation-based search in the full game. For evaluation, we extracted a greedy policy profile that at each information state takes the action with the highest estimated value. Learning performance was measured in milli-big-blinds won per hand, mb/h, in play against benchmark opponents. To reduce variance, we averaged the results of symmetric play that permuted the positions of players, reusing the same random card seeds. We had access to the ACPC's benchmark server, which enabled us to evaluate against the contestants of the 2014 competition.

Based on the experiments in Kuhn and Leduc poker, we selected a parameterization for Smooth UCT's mixing parameter schedule (2) and set it to $\gamma = 0.1$, $\eta = 0.9$ and $d = 20000^{-1}$. In the exploration schedule (3) we set $k = 0.5$ and $C = 24$, which corresponds to half of the maximum potsize achievable in two-player LHE. This results in exploration parameter values of $c \in [10, 24]$, which is centred around 17, a value that has been reported in a previous calibration of UCT in LHE by [Ponsen *et al.*, 2011].

UCT and Smooth UCT planned for 14 days each, generating about 62.1 and 61.7 billion simulated episodes respectively; note that Smooth UCT had almost no computational overhead compared to UCT. We trained on a modern desktop PC, using a single thread and less than 200 MB of RAM. Each greedy strategy profiles' uncompressed size was 8.1 MB.

During training, snapshots of the strategies were taken at regular time intervals in order to measure learning performance over time. In particular, we evaluated each snapshot by symmetric play for 2.5 million games against SmooCT, the silver-medal contestant of the 2014 ACPC. In addition, we compared UCT's and Smooth UCT's snapshots by symmetric play against each other. The results in figure 3 show that UCT performed slightly better for a training time of under 72 hours. After 72 hours Smooth UCT outperformed UCT and was able to widen the gap over time.

Table 2 presents an extensive evaluation of greedy policies that were obtained from UCT and Smooth UCT. The table includes results against all but one contestants of the 2014 ACPC, in the order of their ranking in the competition. We had to omit the one contestant because it was broken on the benchmark server. The table also includes matches between Smooth UCT and UCT. Smooth UCT performed better than UCT against all but one of the top-7 benchmark agents. Performance against the weaker 6 ACPC contestants was more

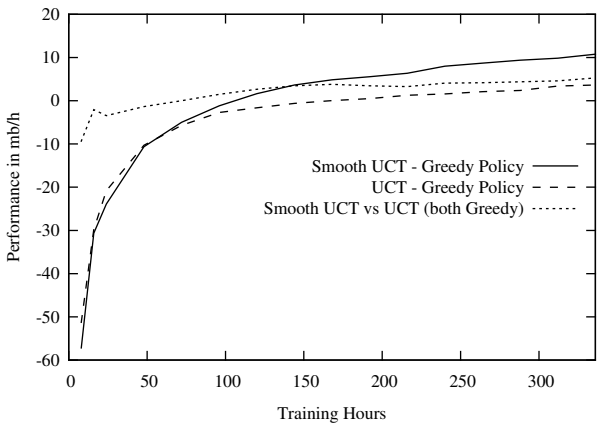

Figure 3: Learning curves in two-player Limit Texas Hold'em. The estimated standard error at each point of the curves is less than 1 mb/h.

even, with Smooth UCT performing better in just half of the match-ups. Finally, Smooth UCT won in the match-up against UCT and achieved a higher average performance.

Smooth UCT lost against all but 2 of the top-7 contestants of the 2014 ACPC. This might be partly due to the much bigger and sophisticated abstractions and resources used by most of these agents. However, Smooth UCT achieved strong average performance against the whole field of agents. This suggests that despite being a self-play approach without prior knowledge, using tiny resources and a small abstraction, Smooth UCT can train highly competitive policies that perform well in an ecosystem of variable player types. However, given the losses against the top agents, it is unclear whether Smooth UCT is able to efficiently compute good approximate Nash equilibria in larger abstractions.

Three-player

Next we performed a similar evaluation in three-player LHE. Smooth UCT used the same mixing parameter schedule as in two-player LHE. The exploration schedule (3) was set to $k = 0.5$ and $C = 36$, which corresponds to half of the maximum potsize achievable in three-player LHE. UCT and Smooth UCT planned for 10 days each, generating about 50.9 and 49.4 billion simulated episodes respectively. Once again we trained on a modern desktop PC, using a single thread and less than 3.6 GB of RAM. Each final greedy strategy profiles' uncompressed size was about 435 MB.

The 2014 ACPC featured two three-player LHE competitions that were won by Hyperborean_tbr and Hyperborean_iro. In both competitions SmooCT and KEmpfer finished second and third out of 5 respectively.

Table 3 presents our three-player results. Smooth UCT outperformed UCT in all but 3 match-ups and achieved a higher average performance overall.

5 Conclusion

We have introduced Smooth UCT, a MCTS algorithm for extensive-form games with imperfect information. In two small poker games, it was able to learn as fast as UCT but approached a Nash equilibrium whereas UCT diverged. Furthermore, in two- and three-player LHE, a game of real-world scale, Smooth UCT outperformed UCT and achieved three silver medals in the 2014 ACPC. The results suggest that highly competitive strategies can be learned by a full-game simulation-based search with Smooth UCT.

In addition, in Leduc Hold'em Smooth UCT initially learned much faster than Outcome Sampling and was only outperformed after 85 million simulated episodes. This suggests Smooth UCT is a promising step toward the development of sound and efficient online MCTS algorithms for imperfect-information games.

Match-up	Smooth UCT	UCT
escabeche	-23.49 ± 3.2	-30.26 ± 3.2
SmooCT	10.78 ± 0.8	3.64 ± 0.9
Hyperborean	-24.81 ± 4.2	-25.03 ± 4.3
Feste	28.45 ± 4.0	20.02 ± 4.1
Cleverpiggy	-25.22 ± 4.0	-30.29 ± 4.0
ProPokerTools	-18.30 ± 4.0	-19.84 ± 3.9
652	-20.76 ± 4.0	-19.49 ± 4.0
Slugathorus	93.08 ± 5.7	93.13 ± 5.8
Lucifer	139.23 ± 4.7	138.62 ± 4.7
PokerStar	167.65 ± 4.9	173.19 ± 5.0
HITSZ_CS_14	284.72 ± 4.5	281.55 ± 4.5
chump9	431.11 ± 7.7	435.26 ± 7.8
chump4	849.30 ± 8.5	789.28 ± 8.7
Smooth UCT	0	-5.28 ± 0.6
UCT	5.28 ± 0.6	0
average	51.38	48.33
average*	135.50	128.89

Table 2: Two-player limit Texas Hold'em winnings in mb/h and their standard errors. The average results are reported with and without including chump4 and chump9.

Match-up	Smooth UCT	UCT
Hyperborean_iro, KEmpfer	27.1 ± 8	11.4 ± 8
Hyperborean_tbr, KEmpfer	8.8 ± 9	2.8 ± 9
Hyperborean_tbr, SmooCT	-17.1 ± 8	-31.5 ± 9
Hyperborean_tbr, HITSZ_CS_14	69.9 ± 10	74.7 ± 10
SmooCT, KEmpfer	42.3 ± 8	50.4 ± 8
SmooCT, HITSZ_CS_14	133.5 ± 9	125.8 ± 9
KEmpfer, HITSZ_CS_14	172.0 ± 9	194.6 ± 9
2x SmooCT	6.2 ± 1	-5.8 ± 1
2x Smooth UCT	0	-8.2 ± 1
2x UCT	7.5 ± 1	0
average	50.0	46.0

Table 3: Three-player limit Texas Hold'em winnings in mb/h and their standard errors.

Acknowledgments

This research was supported by the UK Centre for Doctoral Training in Financial Computing and Google DeepMind.

References

[Auer *et al.*, 2002] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[Auger, 2011] David Auger. Multiple tree for partially observable monte-carlo tree search. In *Applications of Evolutionary Computation*, pages 53–62. Springer, 2011.

[Bard *et al.*, 2013] Nolan Bard, John Hawkin, Jonathan Rubin, and Martin Zinkevich. The annual computer poker competition. *AI Magazine*, 34(2):112, 2013.

[Billings *et al.*, 2002] Darse Billings, Aaron Davidson, Jonathan Schaeffer, and Duane Szafron. The challenge of poker. *Artificial Intelligence*, 134(1):201–240, 2002.

[Billings *et al.*, 2003] Darse Billings, Neil Burch, Aaron Davidson, Robert Holte, Jonathan Schaeffer, Terence Schauenberg, and Duane Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In *IJCAI*, pages 661–668, 2003.

[Bowling *et al.*, 2015] Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. Heads-up limit holdem poker is solved. *Science*, 347(6218):145–149, 2015.

[Brown, 1949] George W Brown. Some notes on computation of games solutions. Technical report, DTIC Document, 1949.

[Browne *et al.*, 2012] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, 2012.

[Burch *et al.*, 2014] Neil Burch, Michael Johanson, and Michael Bowling. Solving imperfect information games using decomposition. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[Coulom, 2007] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and games*, pages 72–83. Springer, 2007.

[Cowling *et al.*, 2012] Peter I Cowling, Edward J Powley, and Daniel Whitehouse. Information set monte carlo tree search. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(2):120–143, 2012.

[Ganzfried and Sandholm, 2015] Sam Ganzfried and Tuomas Sandholm. Endgame solving in large imperfect-information games. In *Proceedings of the Fourteenth International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2015.

[Gelly *et al.*, 2012] Sylvain Gelly, Levente Kocsis, Marc Schoenauer, Michèle Sebag, David Silver, Csaba Szepesvári, and Olivier Teytaud. The grand challenge of computer go: Monte carlo tree search and extensions. *Communications of the ACM*, 55(3):106–113, 2012.

[Heinrich and Silver, 2014] Johannes Heinrich and David Silver. Self-play monte-carlo tree search in computer poker. In *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[Johanson *et al.*, 2013] Michael Johanson, Neil Burch, Richard Valenzano, and Michael Bowling. Evaluating state-space abstractions in extensive-form games. In *Proceedings of the Twelfth International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2013.

[Kocsis and Szepesvári, 2006] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, pages 282–293. Springer, 2006.

[Kuhn, 1950] Harold W Kuhn. A simplified two-person poker. *Contributions to the Theory of Games*, 1:97–103, 1950.

[Lanctot *et al.*, 2009] Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael H Bowling. Monte carlo sampling for regret minimization in extensive games. In *NIPS*, pages 1078–1086, 2009.

[Lanctot, 2013] Marc Lanctot. *Monte Carlo sampling and regret minimization for equilibrium computation and decision-making in large extensive form games*. Ph.D. thesis, University of Alberta, 2013.

[Leslie and Collins, 2006] David S Leslie and Edmund J Collins. Generalised weakened fictitious play. *Games and Economic Behavior*, 56(2):285–298, 2006.

[Lisý *et al.*, 2015] Viliam Lisý, Marc Lanctot, and Michael Bowling. Online monte carlo counterfactual regret minimization for search in imperfect information games. In *Proceedings of the Fourteenth International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2015.

[Lisý, 2014] Viliam Lisý. Alternative selection functions for information set monte carlo tree search. *Acta Polytechnica: Journal of Advanced Engineering*, 54(5):333–340, 2014.

[Ponsen *et al.*, 2011] Marc Ponsen, Steven de Jong, and Marc Lanctot. Computing approximate nash equilibria and robust best-responses using sampling. *Journal of Artificial Intelligence Research*, 42(1):575–605, 2011.

[Sandholm, 2010] Tuomas Sandholm. The state of solving large incomplete-information games, and application to poker. *AI Magazine*, 31(4):13–32, 2010.

[Silver and Veness, 2010] David Silver and Joel Veness. Monte-carlo planning in large pomdps. In *Advances in Neural Information Processing Systems*, pages 2164–2172, 2010.

[Southey *et al.*, 2005] Finnegan Southey, Michael Bowling, Bryce Larson, Carmelo Piccione, Neil Burch, Darse Billings, and Chris Rayner. Bayes bluff: Opponent modelling in poker. In *In Proceedings of the Twenty-First Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, 2005.

[Zinkevich *et al.*, 2007] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Advances in neural information processing systems*, pages 1729–1736, 2007.