Regret-based Traces-Exploration Abstractions for Game Solving

Jacopo P. Gargano
DEIB, Politecnico di Milano
Milan, Italy
jacopopio.gargano@mail.polimi.it

Nicola Gatti
DEIB, Politecnico di Milano
Milan, Italy
nicola.gatti@polimi.it

Abstract

Despite Artificial Intelligence's vision being on general intelligence, development of game-theoretical methods to find optimal strategies has mainly been focused on domain-specific tasks and recreational games, allowing for little domain independence. The complexity of most real-world strategic scenarios, including governance and military, mainly due to the huge number of states and actions, has not allowed the proposed algorithms to scale. In spite of these challenges, game realizations observed through actual play or simulations are readily accessible. As such, we focus our research on developing a domain-independent model-free abstraction framework, able to find approximate mixed strategy Nash Equilibria in any extensive-form game in a simulation-based fashion. We propose Regret-based Traces-Exploration Counterfactual Regret Minimization (ReTrE), a game-theoretical framework leveraging deep neural networks and confidence-based exploration techniques to approximate the behavior of Counterfactual Regret Minimization (CFR), an optimal regret minimization algorithm, in the full game. We show that ReTrE achieves comparable performance with CFR in terms of exploitability when dealing with games small enough to be analyzed by both. We analyze different configurations for the proposed framework, whose flexibility and scalability allow suitability for a variety of domains. Therefore, the practical use of the proposed framework in large games is possible and performance is likely to be in line with what CFR could theoretically achieve, allowing to find competitive suboptimal strategies.

Keywords: Algorithmic Game Theory, Confidence-based Exploration, Equilibrium Computation

1 Introduction

Most real-world strategic scenarios, including theoretical economics, political science, and military are suitable for classical game theoretical analysis [17]. However, their complexity, mainly due to the huge or infinite number of states and actions, has not allowed the proposed algorithms to scale. These scenarios are expressed through large and infinite games, respectively infeasible and impossible to fully represent in practice, as their states and actions belong to a huge or continuous space. Obtaining exact information about these games is possible by collecting game realizations and corresponding payoffs for the players involved, according to their preferences.

In order to cope with complexity in decision making for large extensive-form games, the concept of *abstracting games* was developed by Billings et al. [6], providing a way to lower game complexity while retaining most of the relevant information. Abstracting generally consists in building a smaller version of the game tree with a reduced number of states and actions. Marvin Minsky medal recipients Brown and Sandholm developed *Libratus* [9], the most competitive two-players Texas hold'em poker AI, by leveraging abstractions, and, a few years later, extended it to *Pluribus* [10] (up to six-players). Abstractions are designed for games with signals, in which the game tree is the same in terms of rules and available actions, regardless of the information defining states. Despite the remarkable contributions in the field of abstractions, there are still several open problems to tackle, such as domain independence and model-freedom. A general approach for solving these games, namely finding optimal strategies for the players, would allow significant breakthroughs in the several applications. The aim of game-theoretical analysis is to find strategies for the agents allowing them to reach an *equilibrium* (usually, a *Nash Equilibrium* [19]), in which each player does not benefit from deviating from their strategy, keeping the strategies of all the other players fixed.

What makes playing equilibrium strategies so important to agents is knowing the worst-case utility they will receive before playing. In fact, as long as an agent sticks to the equilibrium strategy, its opponents will not benefit from deviating from the equilibrium strategy itself, as doing so would result in gaining a lower utility. Moreover, according to Nash's Existence Theorem [18], every game with a finite number of players in which each player can choose from finitely many strategies has at least one Nash Equilibrium. As such, the goal of this work is to develop a model-free abstraction method, able to find approximate mixed strategy Nash Equilibria in any extensive-form game in a simulation-based fashion, that is, starting from observations, obtained via actual play or simulations. We focus our inquiry on two-player zero-sum extensive-form games, in which each player's gain or loss of utility is exactly balanced by the losses or gains of the utility of the other.

Original Contributions. We propose ReTrE, a domain-independent pre-play game-theoretical framework leveraging deep neural networks and confidence-based exploration techniques to approximate the behavior of Counterfactual Regret Minimization (CFR), an optimal regret minimization algorithm, in the full game. ReTrE obviates the need for abstraction by using deep neural networks to find suboptimal competitive strategies in large games and to focus on the most exploitable parts of the game so as to approach equilibria. We evaluate ReTrE's performance through Leduc Poker by measuring its exploitability in the full game. We analyze different configurations of ReTrE, tweaking some parameters to simulate different scenarios and to find the best performing version. We observe ReTrE achieves comparable performance with respect to CFR, though being an approximation of it. Thanks to its flexibility and scalability, ReTrE results in a suitable framework for a variety of domains.

2 Notation and Background

2.1 Game Theory

We focus our inquiry on *sequential games*, in which players play in succession taking turns. Sequential, or *extensive-form*, games are represented through a *game tree*, that is a triple (S, E, s_0) where (S, E) is an oriented graph – S the set of vertices, or states, E the set of edges – and $s_0 \in S$ is the root of the tree, namely a vertex such that there is a unique path from s_0 to s, $\forall s \in S \setminus \{s_0\}$.

An imperfect-information extensive-form game Γ is a tuple $(N, A, S, V, Z, H, \chi, \rho, \theta, U)$, where: N is the set of players; A is the set of actions, and $A_h \subseteq A$ is the set of available actions at information set h; S is the set of states; $V \subseteq S$ is the set of nonterminal nodes; $Z \subseteq S$ is the set of terminal nodes ($Z \cap V = \emptyset$ and $Z \cup V = S$); $H = \{H_1, ..., H_n\}$ is the collection of information sets of all players, and H_i is an information partition of V_i such that decision nodes within the same information set $h \in H_i$ are indistinguishable by player i; $\chi : V \rightarrow 2^A$ is a function assigning to each nonterminal node a set of possible actions; $\rho : V \rightarrow N$ is a function assigning to each nonterminal node the player $i \in N$ taking an action at that node; $\theta : V \times A \rightarrow S$ is a function mapping a nonterminal node and an action to the following state; $U = \{u_1, ..., u_n\}$ is the collection of utility functions of all players.

When a game is finite but large or infinite, it is not possible to build an explicit representation of it. In order to obtain exact information on the game, game realizations in the form of *traces* and corresponding payoffs for the players are collected. In this setting, payoffs are available as the output of an *oracle*, which can be intended as a simulator, rather than specified analytically or through a payoff matrix, which is the classical approach [28].

A *simulation-based game* is a tuple (N, Σ, O), where N is the set of players, Σ is the set of strategies, and O is an oracle producing a possibly noisy sample from the joint payoff function of players, given a joint strategy. A *trace* of a game is an array $\tau = (s_1, a_1, ..., s_m, a_m, z)$, where $s_j \in V$ are the traversed states, $a_j \in A$ are the undertaken actions, $j \in [1, m]$, s_m and a_m such that $z = \theta(s_m, a_m) \in Z$, namely the history of the trace reaches a terminal node.

A *behavioral strategy* is a function $\sigma_i : H_i \rightarrow \Delta^{|A_{H_i}|}$, $i \in N$, that associates to each information set $h \in H_i$ a probability distribution over the available actions A_h at that information set h. A *strategy*, or *policy*, σ, is a vector of $|N|$ behavioral strategies σ_i, one for each player in N. Given player i and the opponents' strategy σ_{-i}, the player's *best response* to σ_{-i} is a strategy $BR(\sigma_{-i}) \in \Sigma$ such that $u_i(BR(\sigma_{-i}), \sigma_{-i}) = \max_{\sigma'_i \in \Sigma_i} u_i(\sigma'_i, \sigma_{-i})$.

Given a game, a strategy σ^* is a *Nash Equilibrium* (NE) if and only if, $\forall i \in N$, $\forall \sigma_i \in \Sigma_i$ it holds $u_i(\sigma^*) \geq u_i(\sigma_i, \sigma^*_{-i})$, where $u_i(\sigma)$ is the expected utility of player i if strategy σ is played, Σ_i is the set of strategies of player i, σ^*_{-i} is a strategy containing the strategies of all players except that of player i, and $(\sigma_i, \sigma^*_{-i})$ is the strategy obtained by the combination of σ_i and σ^*_{-i}.

Let, for any $\sigma \in \Sigma$, $\delta_i(\sigma) = u_i\big(BR(\sigma_{-i}), \sigma_{-i}\big) - u_i(\sigma)$ and $\varepsilon = \max_{i \in N} \delta_i(\sigma)$. Then, given a NE σ^*, and exploitability $e(\sigma_i) = u_i\big(\sigma^*_i, BR(\sigma^*_i)\big) - u_i\big(\sigma_i, BR(\sigma_i)\big)$, an *$\varepsilon$-approximate Nash Equilibrium (ε-NE)* is a NE where no player has exploitability $e(\sigma_i) > \varepsilon$. When it holds $\delta_i(\sigma) = 0, \forall i \in N$, then σ is a NE.

2.2 Equilibria

Algorithmic Game Theory is a field of study that aims to analyze strategic conditions and design algorithms able to find strategies allowing agents to reach an equilibrium.

Once a strategy is found, its performance is evaluated through its exploitability. The *exploitability* $e(\sigma_i)$ of a strategy σ_i in a game is how much worse σ_i performs versus $BR(\sigma_i)$ compared to how a NE strategy σ^*_i does against $BR(\sigma^*_i)$. More formally, $e(\sigma_i) = u_i\big(\sigma^*_i, BR(\sigma^*_i)\big) - u_i\big(\sigma_i, BR(\sigma_i)\big)$.

A commonly used metric for poker AI evaluation is NashConv, an exploitability evaluation metric defined over a strategy σ as: $\text{NashConv}(\sigma) = \sum_{i \in N} \max_{\sigma'_i \in \Sigma_i} u_i(\sigma'_i, \sigma_{-i})$. It represents, in total, how much each player gains by deviating to their best response unilaterally. It can be interpreted as a distance from a NE.

We define the *reach* of an information set to capture how likely an information set is to be visited during play. The *reach* $\pi^\sigma(h) = \prod_{h' \cdot a \sqsubset h} \sigma_{i'}(h', a)$ of an information set h is the probability h is reached if all players play according to σ, where i' is the player playing at information set h'. All players contribute to the reach: we define π^σ_i as the *agent reach* and π^σ_{-i} as the *external* or *counterfactual reach*, that is,

the probability of reaching h with strategy σ except that we treat i's actions to reach the state as having probability 1. Specifically, $\pi^\sigma(h) = \pi_i^\sigma(h) + \pi_{-i}^\sigma(h)$, for any player $i \in N$.

The *counterfactual value* of information set h, for player i, if all players play according to strategy σ, is given by $v_i^\sigma(h) = \sum_{z \in Z} \pi_{-i}^\sigma(h) \cdot \pi^\sigma(h, z) \cdot u_i(z)$.

Finally, player i's *regret* of not having taken action a at information set h and having instead followed strategy σ is $r_i(h, a) = v_i^\sigma(h, a) - v_i^\sigma(h)$, where v_i is the counterfactual value. Player i's *cumulative counterfactual regret* of not having taken action a at information set h at time T is $R_i^T(h, a) = \sum_{t=1}^T r_i^t(h, a)$. In two-player zero-sum games, where each player's gain or loss of utility is exactly balanced by the losses or gains of the utility of the other players, if both players' average total regret satisfies $\frac{R_i^T}{T} \leq \varepsilon$, then their average strategies $(\bar{\sigma}_1^T, \bar{\sigma}_2^T)$ form a 2ε-NE [29].

2.3 Counterfactual Regret Minimization (CFR)

The framework we will adopt throughout this inquiry is that of regret minimization. The most successful family of algorithms for imperfect-information games have been variants of Counterfactual Regret Minimization (CFR), first introduced by Zinkevich et al. [30].

Counterfactual Regret Minimization (CFR) is an iterative policy improvement algorithm that computes a new strategy σ^t on each iteration t. The average of these strategies converges to a Nash Equilibrium as $t \to \infty$. On each iteration t, CFR traverses the entire game tree and updates the regrets for every information set in the game according to strategy σ^t. These regrets define a new strategy σ^{t+1}. The strategy σ^{T+1} at time $T+1$ is obtained through regret-matching:

$$\sigma_i^{T+1}(h, a) = \begin{cases} \frac{R_i^{T,+}(h,a)}{\sum_{a \in A_h} R_i^{T,+}(h,a)} & \text{if } \sum_{a \in A_h} R_i^{T,+}(h, a) > 0 \\ \frac{1}{|A_h|} & \text{otherwise} \end{cases} . \quad (1)$$

For each information set, Equation 1 is used to compute action probabilities in proportion to the positive cumulative regrets [20]. For each action, CFR then produces the next state in the game, and computes utilities of each actions recursively. Regrets are computed from the returned values, and the value of playing for the current state is finally computed and returned. It is the *average* strategy that converges to a Nash Equilibrium, and not the *final* strategy.

The initial policy is set to uniform random. The average policy $\bar{\sigma}_i^T$ is:

$$\bar{\sigma}_i^T(h, a) = \frac{\sum_{t=1}^T \pi_i^{\sigma,t}(h) \cdot \sigma_i^t(h, a)}{\sum_{t=1}^T \pi_i^{\sigma,t}(h)}. \quad (2)$$

Since CFR only needs to store values at each information set, its space requirement is $O(|H|)$. However, CFR requires a complete traversal of the game tree on each iteration, which prohibits its use in many large games. The convergence bound of CFR is $O(\frac{1}{\sqrt{T}})$.

2.4 Abstractions

The complexity of decision making is positively correlated to the number of states and actions. This is why when analyzing large games, in order to lower game complexity while trying to retain all relevant information, abstractions are used.

A game *abstraction* is a smaller version of the game with the purpose of capturing the most essential properties of the real domain, such that the solution of the abstracted game provides a useful approximation of an optimal strategy for the underlying real game.

Abstractions are distinguished in three categories: *information* abstractions, *action* abstractions, and *simulation-based* abstractions. *Information abstraction* is an abstraction method such that the agents cannot distinguish some of the states that they could distinguish in the actual game [22]. In *action abstractions* the number of available actions to each player is less than in the original game. In *simulation-based abstraction* the abstracted version of the game is built starting from game realizations.

3 Related Work

3.1 Regret Minimization

Besides CFR, other regret minimization algorithms were developed contributing with major improvements.

In CFR+ any action with negative regret $r_i(h, a) < 0$ is set to 0 regret, that is $r_i(h, a) = max\big(r_i(h, a), 0\big)$. Then, CFR+ uses a weighted average strategy where iteration T is weighted by T rather than using a uniformly-weighted average strategy as in CFR. CFR+ is worse in exploitability compared to CFR, however it typically converges much faster than CFR.

Linear CFR (LCFR) is identical to CFR, except on iteration t the updates to the regrets and average strategies are given weight t.

Monte Carlo CFR (MCCFR) is a variant of CFR in which certain player actions or chance outcomes are sampled [16]. Combined with abstractions, MCCFR has produced state-of-the-art poker AIs (e.g., *Libratus* [9]). The key to the approach behind MCCFR is to avoid traversing the entire game tree on each iteration, while still having the immediate counterfactual regrets be unchanged in expectation, restricting the terminal histories considered at each iteration.

The most relevant work to our research is DeepCFR [7], a variant of CFR that obviates the need for abstraction by instead using deep neural networks to approximate the behavior of CFR in the full game. It avoids calculating and accumulating regrets at each information set, by generalizing across similar ones using function approximation via deep neural networks. DeepCFR achieves strong performance in large poker games compared to domain specific abstraction techniques without relying on advanced domain knowledge. Variants of DeepCFR with improved performance are SingleDeepCFR [25] and DREAM [26].

3.2 Leveraging Abstractions

The majority of research carried out on abstractions is on information abstraction. Early works were initiated by Shi et al.[23] and by Billings et al. [6] leveraging *linear programming* and *bucketing*. More specific information abstractions were introduced, including *automated* [11], *expectation-based* [12], *potential-aware* [13], and *extensive-form game* abstractions [15].

Action abstractions were first analyzed by Hawkin et al. [14] studying the choice of the value of parameters of an action. The first substantial contribution to the field of action abstraction was made by Brown et al. [8] providing the first action abstraction algorithm with convergence guarantees for extensive-form games. Basak [5] introduced the idea of abstracting games by clustering strategies and then solving them by finding and solving suitable subgames.

A theoretical contribution in the field of simulation-based abstractions was given by Tuyls et al. [27] by deriving guarantees on the quality of all equilibria learned from finite samples providing theoretical bounds for empirical game-theoretical analysis of complex multi-agent interactions. They show that a NE of the empirical game is an approximate-NE of the true underlying game and they provide insights on the number of data samples required to obtain a close enough approximation. Furthermore, Areyan et al. [3] study simulation-based games starting from game traces and approximate game utilities, generating an abstracted version of the game through which they are able to learn all *pure* equilibria of a game. The most interesting contribution in the specific field of simulation-based games was made by Areyan et al. [2], designing algorithms that uniformly approximate simulation-based games with finite sample guarantees, achieving the same performance as previous work with far fewer samples.

Still, the most advanced techniques do not rely on abstractions only. The major contributions were developed by Brown and Sandholm presenting *Libratus* [9] and *Pluribus* [10]. Despite the implementation of the former being limited to two-player heads-up no-limit poker, the authors claim that their game-theoretic approach is application-independent. The latter is an enhanced version of its predecessor able to play six-player heads-up no-limit poker.

4 ReTrE Counterfactual Regret Minimization

Regret-based Traces-Exploration Counterfactual Regret Minimization (ReTrE) is a domain independent pre-play framework for large games, built on top of DeepCFR, leveraging neural networks to find suboptimal competitive strategies. In addition to DeepCFR, it explores the original game focusing on its most exploitable parts.

The main components of ReTrE are the Policy Network (PN), a neural network approximating the output strategy of CFR in the full game, the Exploration Dictionary (ED), a data structure holding information about the information sets to guide exploration of the original game, and the Exploration Network (EN), a neural network approximating the ED when memory resources are not enough, as it is the case of large games.

4.1 Overview

We hereby provide an overview of ReTrE by describing the main components and processes from a high level perspective, deepening the focus right after.

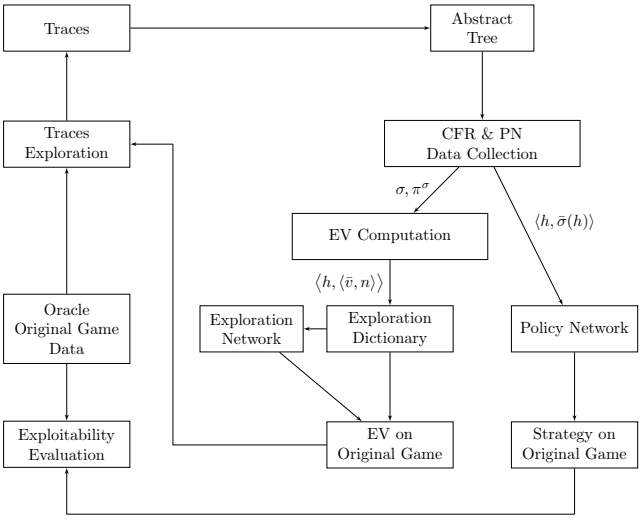

Figure 1. Overview of ReTrE.

With reference to Figure 1, initially, a set of traces is available, obtained either offline – as a subset of all the available ones – or online – either through an oracle or in a Reinforcement Learning like fashion.

The collected traces make up a smaller game tree, whose size must be smaller than 10^{12} nodes to be analyzed through tabular regret minimization techniques.

We run T iterations of CFR for each player obtaining the average strategy for the two players, which converges to a NE as $T \to \infty$ [30]. While running CFR, we collect samples to train the PN and the information sets' value and external reach are stored for Exploration Value computation.

Once CFR is over, for every information set h, for every available action $a \in A_h$, we compute the Exploration Value (EV) – the value guiding the traces exploration phase – through the information sets' value and reach obtained through CFR. Then we update the ED entry for the child information set reached taking action a from information set h with the couple $\langle \bar{v}, n \rangle$, where $\bar{v}$ is the average EV and n is the number of times the child information set was included in the abstract game tree throughout ReTrE iterations. Once CFR is over and the ED is updated, we train the EN using the ED.

The ED and the EN are then used in the traces exploration phase, in which the new set of traces is obtained by choosing those traces maximizing an upper confidence bound of the EV.

The new traces are then used to generate a new abstract game tree focused on the parts of the original game that are more exploitable by the opponent compared to the previous ones. The process described so far is repeated until computational resources are available.

Finally, at the last iteration of ReTrE, we train the PN with the collected samples to obtain the ultimate PN defining an artificial agent able to play an ε-NE strategy.

4.2 Policy Network

The ultimate objective of game solving consists in finding a competitive strategy for a game. To approximate the average strategy of the original game, we resort to the PN, introduced in DeepCFR [7], and whose architecture is shown in Figure 2. Given an information set h, the network $PN : H \rightarrow \Delta^{|A_{H_i}|}$ predicts the average strategy over the available actions A_h. The learning problem we define consists in the supervised learning of the probability distribution over the actions available at a certain information set. Therefore, the network training samples are in the form $\langle h, \bar{\sigma}_h \rangle$. Information sets are represented by the private information of the player they belong to, by the public information and by the history of the game until then. The training samples are collected throughout CFR traversals of the abstract tree.

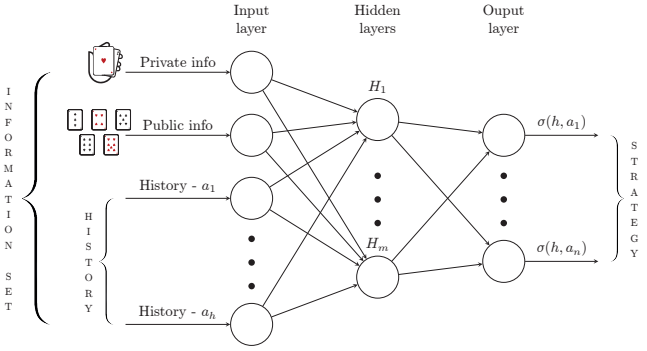

Figure 2. Policy Network.

Since the learning problem consists in estimating a probability distribution, we minimize the *Kullback-Leibler divergence*, or *relative entropy*: a measure of how one probability distribution is different from another. For probability distributions P and Q of a continuous random variable, the Kullback–Leibler divergence is defined as $D_{KL}(P||Q) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} dx$.

4.3 Exploration Dictionary and Network

The ED is a data structure holding information for each information set, until memory allows. The information stored in the ED is the couple $\langle h, \langle \bar{v}, n \rangle \rangle$, where: h is an information set; $\bar{v}$ is the average EV; n is the number of times that information set h was visited throughout ReTrE iterations.

The EV $\bar{v}$ is what guides the traces exploration phase together with n. We use a measure of the counterfactual value of information sets for the EV, however, other measures may be used, such as the regret or the advantage [7]. The EV that we use is the following:

$$\hat{v}_i(h, a) = \frac{\sum_{t=1}^{T} \left(t \cdot \pi_{-i}^{\sigma,t}(h) \cdot \left(v_i^{\sigma,t}(\theta(h, a)) - v_i^{\sigma,t}(h) \right) \right)}{\sum_{t=1}^{T} \left(t \cdot \pi_{-i}^{\sigma,t}(h) \right)}, \quad (3)$$

where T is the number of CFR iterations. This measure is very similar to the regret, and it only considers the opponents' strategy. This is fundamental, as humbleness is for learning. In fact, if the player willing to learn an optimal strategy used an EV that is biased on their own strategy, they would not be completely able to explore new possibilities in the original game, possibly lowering exploitability.

This component has limited size, so when it reaches its maximum capacity we remove least recently visited information sets, namely the ones with lower n, to make space for the newly observed ones, which, as $T \rightarrow \infty$, are the ones where the focus shall be.

Since the size of the ED is limited, we resort to the EN to estimate the EV. The EN is a neural network $EN : H \rightarrow \mathbb{R}$ that takes as input an information set $h \in H$ and outputs the estimated EV for it. Since the learning problem consists in estimating a real positive number, we train the EN by minimizing the Mean Squared Error.

When the ED maximum capacity is reached, some information sets need to be discarded together with their information. While the EV can be estimated through the EN, we use the following methodology for n. First, if n needs to be estimated, it means that the information set was either never considered in the past or it was previously discarded. Then, n cannot be greater than the minimum value stored in the ED, namely $n \leq \hat{n} = \min_{h' \in ED} n_{h'}$, otherwise it would be in the ED. Therefore, n assumes values in $[0, \hat{n}]$. The estimated n must be $\hat{n}$, otherwise $n \in [0, \hat{n}]$ could be not valid for future cases.

Therefore, a sound estimation of n is $\hat{n} = \min_{h' \in ED} n_{h'}$.

4.4 The Algorithms

ReTrE (Algorithm 1) is the core algorithm of the framework. It takes as parameters: the Exploration Parameter (EP), that is an integer k, defining the maximum number of actions to explore for each information set; the players of the game; the number of iterations it is run for ($\hat{T}$) and those of CFR (T). At each iteration, ReTrE fetches new traces focusing the search on those parts of the original game tree that maximize the chosen EV. Once the EV is computed or estimated through the EN, the ED is updated in ComputeEV (Algorithm 2). The

EN is trained at each ReTrE iteration, and the PN is trained only at the end, as there is no need to train it beforehand since the search becomes more focused on the interesting parts of the tree as $\tau \to \hat{T}$. Note that π^σ at Line 8 is obtained through CFR. The output of ReTrE is the PN for both players, which is indeed the only component needed to play.

Algorithm 1 ReTrE

```
1:  function ReTrE(k, N, T, T̂):
2:      Initialize PN, ED, EN for each player i ∈ N
3:      for ReTrE iteration τ = 1 to T̂ do
4:          traces ← GetTracesUCB(ED, EN, τ, k)
5:          for CFR iteration t = 1 to T do
6:              for all i ∈ N do
7:                  CFR(i, t, τ)
8:          ComputeEV(τ, π^σ, ED, EN)
9:          Train EN through ED
10:     Train PN
```

Algorithm 2 ReTrE - EV Computation and Collection

```
1:  function ComputeEV(τ, π^σ, ED, EN):
2:      for all information sets h ∈ H do
3:          i ← ρ(h)
4:          for all a ∈ A_h do
5:              h' ← θ(h, a)
6:              v̂_i ← compute EV using Eq. 3
7:              if τ = 1 then
8:                  v̄ ← v̂_i
9:                  n ← 1
10:             else
11:                 if h' ∈ ED_i then
12:                     v̄, n ← get ⟨v̄, n⟩ of h' from ED_i
13:                 else
14:                     v̄ ← predict v̄(h') through EN_i
15:                     n ← estimate n
16:                 v̄ ← (v̄·n + v̂_i)/(n+1)      ▷ compute the average EV
17:                 n ← n + 1
18:             Store ⟨h', ⟨v̄, n⟩⟩              ▷ store information in ED_i
```

Differently from DeepCFR, the traces exploration phase leverages an Upper Confidence Bound (UCB) like approach to guarantee exploration and exploitation.

GetTracesUCB (Algorithm 3) gathers the traces to run CFR on by calling TraverseUCB (Algorithm 4), which guides the search phase. The latter is a recursive algorithm which starts from a state s and explores the original game. In our implementation, the original game is explored, however, similarly, an oracle or a set of data could be queried with the relative parameters to perform exploration.

TraverseUCB first builds a set of tuples $\langle action, state \rangle$, namely the possible parts of the original game tree to explore,

Algorithm 3 ReTrE - Traces Gathering

```
1:  function GetTracesUCB(ED, EN, τ, k):
2:      traces ← empty list
3:      s_0 ← get initial state
4:      trace ← empty list
5:      trace.add(s_0)
6:      for all s' ∈ θ(s_0, RA) do
7:          trace.add(⟨RA, s'⟩)
8:          TraverseUCB(s', traces, trace, ED, EN, τ, k)
9:          trace.remove(⟨RA, s'⟩)
10:     return traces
```

and computes the EV accordingly. Then, it explores the k most promising ones.

For state s:

- If s is terminal, then the trace can be inserted in the set of traces (Lines 3-5).
- Instead, if s is chance, we consider all of its outcomes as extensions to the current trace (Lines 6-8). *RA* stands for Random Action.
- Otherwise, we retrieve the EV and the number of times the information set associated to s was visited throughout ReTrE.
  - If the information set is in the *ED*, then we retrieve its information directly from it (Lines 15-16).
  - Else, we estimate the EV through the EN, and n accordingly (Lines 17-19).

Then, we compute the UCB value associated with EV and n (Lines 20-21). The UCB we use in our implementation is that of UCB1 [4]:

$$ucb = \bar{v} + \sqrt{\frac{2\log(\tau)}{n}}. \tag{4}$$

Finally, we sort the state's children on *ucb* descending (Line 22). We continue the trace generation traversing only on the first k children, k being the EP determining the size of the abstract game tree (Lines 23-28). When $k = |A_h|$, the whole game tree is explored.

4.5 Convergence

Our framework employs two main approaches: strategy computation through regret minimization and exploitability minimization through upper confidence bound exploration. The former is achieved through CFR, converging to a Nash equilibrium in any two-player zero-sum game with a theoretical convergence bound of $O(\frac{1}{\sqrt{T}})$, where T is the number of CFR iterations [30]. ReTrE limits the number of actions CFR considers at each information set to the EP. Therefore, the convergence of CFR to a NE is approximately respected. On the other hand, the regret of UCB1, and consequently that of ReTrE, is proportional to $O(log(\hat{T}))$, where $\hat{T}$ is the number of ReTrE iterations [4].

Algorithm 4 ReTrE - Traces Exploration

function TraverseUCB(s, $traces$, $trace$, ED, EN, τ, k):
  $actions\_states \leftarrow$ empty list
  if s is terminal **then**
    $traces$.insert($trace$)
    return
  else if s is chance **then**
    for all $s' \in \theta(s, RA)$ **do**
      $actions\_states$.insert$(\langle RA, s' \rangle)$
  else
    $children \leftarrow$ empty list
    for all $a \in A_s$ **do** ▷ A_s are the legal actions at s
      $s' \leftarrow \theta(s, a)$
      $h' \leftarrow$ information set corresponding to s'
      $i \leftarrow \rho(s)$
      if $h' \in ED_i$ **then**
        $\bar{v}, n \leftarrow$ get $\langle \bar{v}, n \rangle$ of h' from ED_i
      else
        $\bar{v} \leftarrow$ predict $\bar{v}(h')$ through EN_i
        $n \leftarrow$ estimate n
      $ucb_{s'} \leftarrow \bar{v} + \sqrt{\frac{2\log(\tau)}{n}}$
      $children$.insert$(\langle a, s', ucb_{s'} \rangle)$
    Sort $children$ on ucb descending
    $k \leftarrow min(k, |children|)$
    $actions\_states \leftarrow children[:k]$
  for all $a, \hat{s} \in actions\_states$ **do**
    $trace$.add$(\langle a, \hat{s} \rangle)$
    TraverseUCB($\hat{s}$, $traces$, $trace$, ED, EN, τ, k)
    $trace$.remove$(\langle a, \hat{s} \rangle)$

5 Experimental Evaluation

We run experiments on classical Leduc Poker [24], measuring ReTrE's performance through strategy exploitability evaluation using NashConv and running head-to-head simulations. Despite Leduc Poker being a small game compared to classical Texas hold'em Poker and to real-world strategic scenarios, it is fundamental to evaluate suboptimal algorithms (ReTrE) on games small enough to run an optimal algorithm (CFR) and compare performances.

In its current implementation, the information ReTrE encodes is that of information sets, to be fed to the neural networks, namely the PN and the EN. We encode an information set uniquely by considering the available information to a player at that information set, including their private information (their private card), the public information (the public card, if any) and the history of actions taken by all players until then.

For instance, this is how a Leduc information set is encoded:

$$\langle Jack\clubsuit, King\spadesuit, CALL, RAISE, CALL, -, CALL, -, -, - \rangle$$

where $Jack\clubsuit$ is the player's private card; $King\spadesuit$ is the public card; '$-$' is needed for encoding standardization with other information sets, as the maximum length of the history of a Leduc Poker information set is 8.

Best Configuration. ReTrE is a highly configurable framework, scalable to the requirements imposed by computational resources. In fact, there are several parameters that can be adjusted: the Exploration Dictionary size, the Exploration Value, the upper confidence bound, and the Exploration Parameter k.

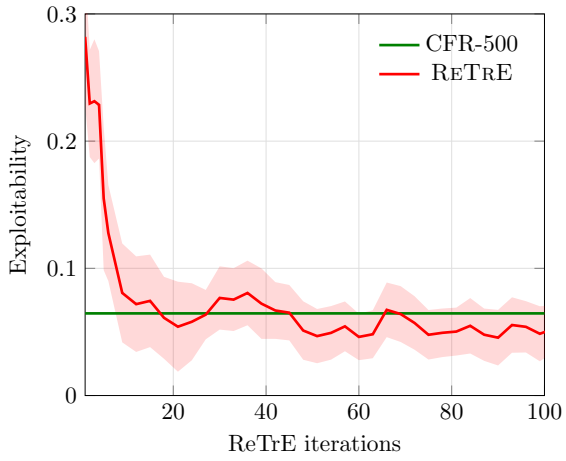

Figure 3. Exploitability Evaluation for CFR-500 and ReTrE's best configuration.

Figure 3 compares the performance of CFR with the best version of ReTrE achieved in Leduc Poker in terms of exploitability, specifically with $k = 2$, EV from Equation 3, UCB1, and ED size = ∞. Whilst the performance of baseline CFR-500, namely CFR run for 500 iterations, does not depend on the number of ReTrE iterations, ReTrE's performance does. In particular, just after less than 10 iterations ReTrE shows good performance compared to CFR-500, and, after an exploration phase happening later on in the iterations, it shows lower exploitability than CFR-500, resulting in being closer to a NE. This result is due to the opportunity that ReTrE has to focus on the most exploitable parts of the tree, discarding the less interesting ones, mastering its strategy accordingly.

Upper Confidence Bound. We compare two different UCBs with not having a bound at all on the EV for the Traces Exploration phase. Figure 4 shows the results obtained. In particular, we consider UCB1 as in Equation 4, UCBt = $\sqrt{\frac{2\log(\tau)}{n \cdot (\tau - 1)}}$ and no UCB. The figure shows that the best performance is achieved by UCB1. UCBt's performance is close to that of UCB1, as expected, considering their similarity. However, convergence is more stable for UCB1. Not

having an UCB results in mediocre performance, as the algorithm is not able to abandon its belief to explore unexplored paths reducing exploitability.

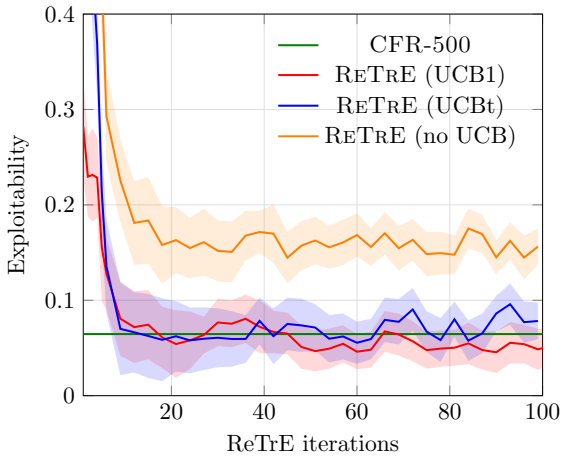

Figure 4. Exploitability Evaluation for different UCBs.

Exploration Dictionary Size. ReTrE is a framework thought and designed for large games. Therefore, we simulate its behavior in large games by limiting the size of the ED. We first consider infinite capacity, which would allow full tree traversal, and consequently the use of tabular CFR algorithms. Then, we limit the ED size to 10% and to 1% of all the information sets. Figure 5 shows the performance of limited ED size ReTrE. Low capacity does not influence the exploitability of ReTrE overall, whereas very low capacity does not show desirable performance.

This result, however, is to be analyzed further as it does not properly simulate the behavior ReTrE would have in a large game. In fact, when an information set is not present in the ED, ReTrE leverages the EN to estimate its EV. The EN is a neural network which is trained in a deep learning fashion: information sets are provided as is and their relevant features are extracted automatically when training. Deep learning needs many training samples to have solid performances. This is why it is hard to simulate the behavior ReTrE would have in large games using a small game.

Exploration Value. We compare different possibilities for the EV. In particular, we show performance for ReTrE with the EV of Equation 3, ReTrE-v with the EV being the information set value, and ReTrE-r where the EV is the cumulative regret. Figure 6 shows the EV of Equation 3 outperforms the other two.

Head-to-head Simulations. Finally, we run AI vs AI simulations to evaluate actual performance during play. First, we let CFR-500, ReTrE and ReTrE-jr (limited ED) play against CFR-500. Figure 7 shows the value for the first player playing

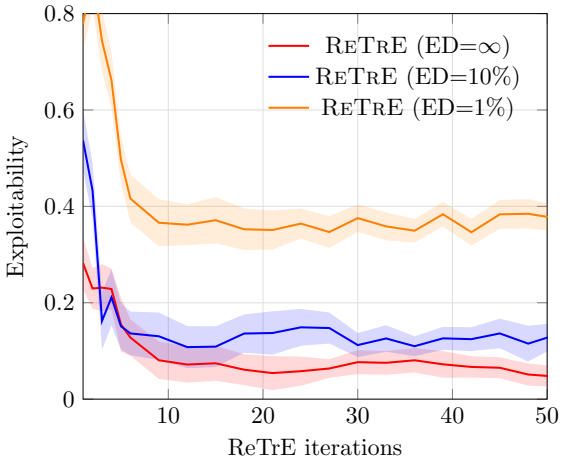

Figure 5. Exploitability Evaluation for different ED sizes.

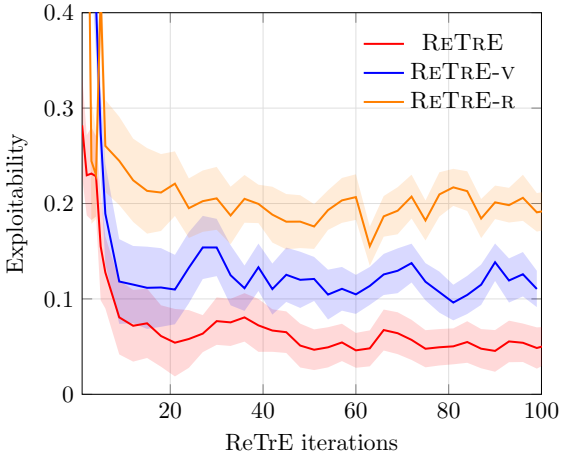

Figure 6. Exploitability Evaluation for different EVs.

Table 1. AI vs AI results. Row is player 1, Column is player 2. Value is shown for player 1.

	CFR-500	*ReTrE*	*ReTrE-jr*
CFR-500	−0.085	0.136	0.252
ReTrE	−0.101	−0.113	0.173
ReTrE-jr	−0.294	−0.213	−0.174

against CFR-500. Despite ReTrE achieving lower exploitability than CFR-500, it still loses against it. This is due to CFR-500 being able to exploit ReTrE's vulnerabilities and not viceversa. However, ReTrE is closer to a NE. More detailed results are provided in Table 1.

Results. Overall, ReTrE achieves comparable performance with CFR in terms of exploitability when dealing with games small enough to be analyzed by both. We compare different configurations to find the most promising one. We

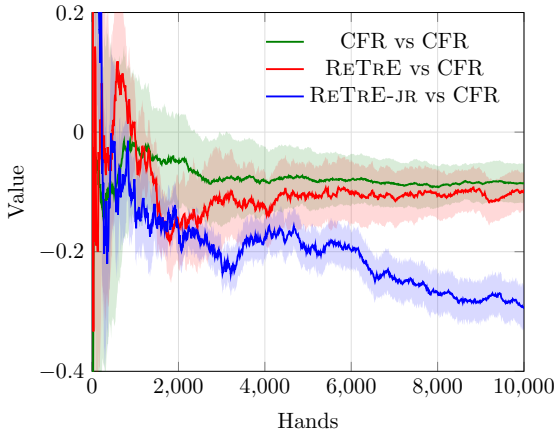

Figure 7. AI vs CFR value for AI.

conclude that the practical use of the proposed framework in large games is possible and performance is likely to be in line with what CFR could theoretically achieve, allowing to find competitive suboptimal strategies.

6 Conclusions

In this work, we focused on the challenge of analyzing large and infinite games so as to solve them by finding approximate mixed strategy Nash Equilibria. It is infeasible to represent these games through a game tree and traverse through them because of their complexity. This is why in practice *abstractions* are used to lower complexity, allowing to find suboptimal strategies close enough to optimal ones.

We introduced ReTrE, a domain-independent model-free abstraction framework, able to find approximate mixed strategy Nash Equilibria in any extensive-form game in a simulation-based fashion, that is, starting from observations. ReTrE obviates the need for abstraction by leveraging deep neural networks to approximate the behavior of CFR, an optimal regret minimization algorithm, in the full game.

We evaluated ReTrE's performance by measuring exploitability on games small enough to be analyzed by CFR also. Despite being an approximation of it, we showed ReTrE performs considerably well compared to CFR. As such, considering the promising results achieved, the practical use of the proposed framework in large games is possible, allowing to find competitive suboptimal strategies.

Interestingly, despite having lower exploitability, we observed that, in the case of Leduc Poker, ReTrE loses against CFR if agents are *static*. However, in practice, agents are *dynamic*, as they can change their strategy. Therefore, in the long run, a dynamic agent would shift away from their strategy to exploit the opponent's vulnerabilities, earning back what it previously lost.

Finally, besides being a domain-independent framework, ReTrE is scalable to specific computing requirements by adjusting the Exploration Dictionary size and the Exploration Parameter k.

7 Future Work

ReTrE opens up several promising directions of research that can be explored next.

First, the performance of ReTrE in large games shall be evaluated through practical experimentation: more complex recreational games, such as heads-up no-limit Texas hold'em Poker and Contract Bridge, can be experimented with initially. Then, the focus may be shifted towards real-world applications, such as car racing or cybersecurity scenarios.

Comparing ReTrE with the leading abstraction algorithms, both domain dependent and independent, would allow for further inquiries and deeper analyses on the advantages of each method. Furthermore, several other regret minimization algorithms and variants of CFR can be used to enhance ReTrE, including LCFR and MCCFR, the latter showing significant performance when dealing with large games.

Particularly, capturing the essence of information sets through enhanced, perhaps domain-specific, embeddings would allow better performance. In fact, it would be interesting to consider the potential of information sets, building upon the research of Gilpin et al. [13], by integrating it in a suitable embedding.

ReTrE leverages UCB1, which is a widely chosen possibility for upper confidence bounds. Using other stochastic measures for the bound could provide better performance in practice. Another possibility is to use Bayesian methodologies for the learning purpose (e.g., Thompson Sampling [1]), exploiting prior knowledge on the Exploration Value of information sets. Moreover, other exploration methods, such as Monte Carlo search [21], could be applied during the traces exploration phase.

Finally, ReTrE is a pre-play only framework, namely, it outputs a suboptimal strategy for an agent to stick with for the whole game. However, it can be integrated with strategy refinement algorithms, such as depth-limited search, to exploit the current state of the game and allow for better performance.

References

[1] Shipra Agrawal and Navin Goyal. 2012. Analysis of thompson sampling for the multi-armed bandit problem. In *Conference on learning theory*. 39–1.

[2] Enrique Areyan Viqueira, Cyrus Cousins, and Amy Greenwald. 2020. Improved Algorithms for Learning Equilibria in Simulation-Based Games. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. 79–87.

[3] Enrique Areyan Viqueira, Cyrus Cousins, Eli Upfal, and Amy Greenwald. 2019. Learning Equilibria of Simulation-Based Games. arXiv:1905.13379 [cs.GT]

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

[5] Anjon Basak. 2016. Abstraction using analysis of subgames. In *Thirtieth AAAI Conference on Artificial Intelligence*.

[6] D. Billings, N. Burch, A. Davidson, R. Holte, J. Schaeffer, T. Schauenberg, and D. Szafron. 2003. Approximating Game-theoretic Optimal Strategies for Full-scale Poker. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence* (Acapulco, Mexico) *(IJCAI'03)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 661–668. http://dl.acm.org/citation.cfm?id=1630659.1630756

[7] Noam Brown, Adam Lerer, Sam Gross, and Tuomas Sandholm. 2019. Deep counterfactual regret minimization. In *International Conference on Machine Learning*. 793–802.

[8] Noam Brown and Tuomas Sandholm. 2014. Regret transfer and parameter optimization. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*.

[9] Noam Brown and Tuomas Sandholm. 2018. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science* 359, 6374 (2018), 418–424.

[10] Noam Brown and Tuomas Sandholm. 2019. Superhuman AI for multiplayer poker. *Science* 365, 6456 (2019), 885–890.

[11] Andrew Gilpin and Tuomas Sandholm. 2006. A competitive Texas Hold'em poker player via automated abstraction and real-time equilibrium computation. In *Proceedings of the National Conference on Artificial Intelligence*, Vol. 21. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 1007.

[12] Andrew Gilpin and Tuomas Sandholm. 2007. Better automated abstraction techniques for imperfect information games, with application to Texas Hold'em poker. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*. ACM, 192.

[13] Andrew Gilpin, Tuomas Sandholm, and Troels Bjerre Sørensen. 2007. Potential-aware Automated Abstraction of Sequential Games, and Holistic Equilibrium Analysis of Texas Hold'Em Poker. In *Proceedings of the 22Nd National Conference on Artificial Intelligence - Volume 1* (Vancouver, British Columbia, Canada) *(AAAI'07)*. AAAI Press, 50–57. http://dl.acm.org/citation.cfm?id=1619645.1619655

[14] John Alexander Hawkin, Robert Holte, and Duane Szafron. 2011. Automated action abstraction of imperfect information extensive-form games. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*.

[15] Christian Kroer and Tuomas Sandholm. 2014. Extensive-form Game Abstraction with Bounds. In *Proceedings of the Fifteenth ACM Conference on Economics and Computation* (Palo Alto, California, USA) *(EC '14)*. ACM, New York, NY, USA, 621–638. https://doi.org/10.1145/2600057.2602905

[16] Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael Bowling. 2009. Monte Carlo sampling for regret minimization in extensive games. In *Advances in neural information processing systems*. 1078–1086.

[17] Michael Maschler, Eilon Solan, and Shmuel Zamir. 2013. Game Theory. *Cambridge University Press*.

[18] John Nash. 1951. Non-cooperative games. *Annals of mathematics* (1951), 286–295.

[19] John F Nash et al. 1950. Equilibrium points in n-person games. *Proceedings of the national academy of sciences* 36, 1 (1950), 48–49.

[20] Todd W Neller and Marc Lanctot. 2013. An introduction to counterfactual regret minimization. In *Proceedings of Model AI Assignments, The Fourth Symposium on Educational Advances in Artificial Intelligence (EAAI-2013)*, Vol. 11.

[21] Stuart Russell and Peter Norvig. 2009. *Artificial Intelligence: A Modern Approach* (3rd ed.). Prentice Hall Press, USA.

[22] Tuomas Sandholm. 2010. The State of Solving Large Incomplete-Information Games, and Application to Poker. *AI Magazine* 31, 4 (Sep. 2010), 13–32. https://doi.org/10.1609/aimag.v31i4.2311

[23] Jiefu Shi and Michael L Littman. 2000. Abstraction methods for game theoretic poker. In *International Conference on Computers and Games*. Springer, 333–345.

[24] Finnegan Southey, Michael P Bowling, Bryce Larson, Carmelo Piccione, Neil Burch, Darse Billings, and Chris Rayner. 2012. Bayes' bluff: Opponent modelling in poker. *arXiv preprint arXiv:1207.1411* (2012).

[25] Eric Steinberger. 2019. Single deep counterfactual regret minimization. *arXiv preprint arXiv:1901.07621* (2019).

[26] Eric Steinberger, Adam Lerer, and Noam Brown. 2020. DREAM: Deep Regret minimization with Advantage baselines and Model-free learning. *arXiv preprint arXiv:2006.10410* (2020).

[27] Karl Tuyls, Julien Perolat, Marc Lanctot, Joel Z Leibo, and Thore Graepel. 2018. A Generalised Method for Empirical Game Theoretic Analysis. arXiv:1803.06376 [cs.GT]

[28] Yevgeniy Vorobeychik and Michael P Wellman. 2008. Stochastic search methods for Nash equilibrium approximation in simulation-based games. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*. International Foundation for Autonomous Agents and Multiagent Systems, 1055–1062.

[29] Kevin Waugh. 2009. Abstraction in large extensive games. (2009).

[30] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. 2008. Regret minimization in games with incomplete information. In *Advances in neural information processing systems*. 1729–1736.