Solving Imperfect Information Games Using Decomposition

Neil Burch, Michael Johanson and Michael Bowling
Computing Science Department, University of Alberta
{nburch,johanson,bowling}@ualberta.ca
January 10, 2014

Abstract

In this work, we present two tools for decomposing an imperfect information game into subgames that can be solved separately, while retaining optimality guarantees on the full-game solution. Using decomposition allows us to make better use of information available at run-time, to overcome memory or disk limitations at run-time, and to make a time/space trade-off to overcome memory or disk limitations while solving a game. We present a method for solving a subgame which guarantees performance in the whole game, in contrast to existing methods which may have unbounded error. We also present a game solving algorithm, CFR-D, which can produce a Nash equilibrium for a game larger than available storage.

1 Introduction

A game solving algorithm considers a game, such as checkers or poker, and produces or approximates an optimal strategy (i.e., a Nash equilibrium) for playing the game. Finding such a strategy is called solving a game. Perfect information games such as checkers, where game states are entirely public, have historically been more tractable to solve than imperfect information games such as poker, where some information about the game state is hidden from one or more players. The primary reason is that perfect information games can easily be split into simpler subgames that can be solved independently, producing strategy fragments that can be combined to form an optimal strategy for the entire game. At any decision point where a player is about to act, we can consider the subgames following from each possible action without needing to consider the subgames following the other actions, the history of actions that led up to this decision, or any other unreachable parts of the game tree. Reasoning about subgames independently allows for time and memory efficient algorithms like depth-first iterative-deepening [11], and allowed an optimal strategy to be computed for the game of checkers [12].

The corresponding application of decomposition to imperfect information games does not work. Consider the game of rock-paper-scissors, where rock beats scissors,
scissors beats paper, and paper beats rock. We usually think of this as a game where players simultaneously make their actions, but we can describe it as a game where the first player picks rock, paper, or scissors without showing the other player, then the second player picks an action. The correct choice for the second player depends on the first player’s strategy. If the first player always picks rock, the second player should play paper. If the first player randomly picks between rock or scissors, the second player should play rock. Player one’s strategy, however, will depend on the values of the subgames following each action, which depend on player two’s strategy, which in turn depended on player one’s strategy. By adding hidden information, these decisions can no longer be analysed independently in the same way they can be for perfect information games. Any fixed assumptions about the initial player one choice — even if the initial choice is correct for some Nash equilibrium — can result in a player two strategy that is suboptimal in the full game, by being exploitable by a different initial choice for player one.

This inability to decompose imperfect information games is unfortunate, as it is a very useful technique in perfect information games, as we will explain with two applications. First, decomposition can allow for a very large savings in the memory required to solve a game. In a depth $D$ game tree with a branching factor of $B$, there are $S = \frac{B^D-1}{B-1}$ states. When we split the game into subgames at depth $D/2$, we do not end up with two pieces of size $S/2$ (i.e., a top half and a bottom half), but $1 + B^{D/2}$ pieces of size $\frac{B^{D/2}-1}{B-1} \approx \sqrt{S}$: a single “trunk” spanning from the start of the game to the split depth, and $B^{D/2}$ subgames. If we need to consider only the trunk and a single subgame at a time, then we use an amount of storage on the order of $\sqrt{S}$. The subgame pieces can also be further split, so that in perfect information games, many algorithms (including some game solving algorithms) use only $O(D)$ memory. The independence from the branching factor $B$ means that we are effectively not limited by space, and with sufficient time, can solve any perfect information game. For example, checkers has around $5 \times 10^{20}$ states and has been solved [12].

A second useful application of decomposing perfect information games is that we do not always need to store a complete strategy. If we have time available at run-time, we do not need to store the subgame strategies. Instead, we can recompute these parts of the strategy at run-time as needed. Because the latter portion of a game requires the majority of the space, this represents a very large space savings. The checkers solution uses this approach: without it, the strategy would have been too large to store on disk. In imperfect information games, there are currently no methods for recomputing subgame strategies with a guarantee that a recovered subgame strategy is also part of an optimal strategy for the whole game.

Since there are no current algorithms for imperfect information games that allow for decomposing the game or recovering subgame strategies, state-of-the-art algorithms are all limited to problems where the complete strategy fits in available space. As a result, 2-Player Limit Texas Hold’em Poker, with $3.194 \times 10^{14}$ decision points, is smaller than checkers but has not been solved despite research interest in the game. Computing an optimal strategy for this game would require hundreds of terabytes using a state-of-the-art game solving algorithm. In addition, this poker game is among the simplest played by humans. The 2-Player No-Limit game played in the Annual Computer Poker
Competition is far larger, with $6.311 \times 10^{164}$ game states and $6.376 \times 10^{161}$ decision points [6].

Computer agents for these domains must use some technique to manage the large game size which makes an optimal strategy computation infeasible. Many poker agents in the Annual Computer Poker Competition are optimal strategies for simpler “abstract” poker games, which are smaller and simpler versions of the original problem which ignore some information [9]. The quality of the solution is strongly correlated with the size of the abstraction, so there is a very strong incentive to use the largest possible abstraction. Even though decomposition techniques have historically offered no guarantees of correctness in imperfect information games, the temptation of using decomposition to (unsafely) use a larger abstraction has resulted in its application. Both PS-Opti [1] and GS1 [5] were strong computer poker agents for their time, and both chose to split the game, in an unsafe fashion, in order to use a larger abstraction. Waugh et al. used a decomposition approach called “strategy grafting” [13] to create the 2-player limit Hyperborean2009-Eqm agent, which used an abstraction 28 times larger than the Hyperborean2008 agent. Although it was stronger against its opponents in one-on-one games, it was also suboptimal for 440.823 milli-big-blinds per game, far more than the 2008 agent’s suboptimality of 266.797. More recently, the Tartanian6 [2] agent uses decomposition at run-time to re-solve the latter part of the game with a much larger abstraction. The method used has promising experimental results, but no theoretical guarantees.

In this paper we present, for the first time, two methods which safely use decomposition in imperfect information games. We give a definition of subgames which is useful for imperfect information games, and a method for re-solving these subgames which is guaranteed to not increase the exploitability (i.e., suboptimality) of a strategy for the whole game. We also give a general method called CFR-D for computing an error-bounded approximation of a Nash equilibrium through decomposing and independently analyzing subgames of an imperfect information game. Finally, we give experimental results comparing our new methods to existing techniques, showing that the prior lack of theoretical bounds can lead to significant error in practice.

For a motivating example, consider the space used by various situations described in table 1. 2-Player Limit Texas Hold’em would currently require 522TB of memory to compute an optimal strategy, using poker-specific techniques to eliminate redundant chance events. Once it was solved, it would still require hundreds of terabytes of disk simply to save the strategy. Using CFR-D, and recovering subgame strategies at run-time, we would require less than 16GB. The trade-off is time: a very rough internal guess suggests we might need at least 100,000 core years of computer time to generate a reasonable approximation of an optimal strategy and validate it. While this is not currently feasible for us, improvements in CPU speed, the increasing number of cores on machines, further algorithmic advances, and simply improving our implementation suggest the CPU time will arrive before half a petabyte of memory. This sort of space-time tradeoff was not previously feasible without losing any guarantee of optimality.
An extensive-form game is a model of the interaction of one or more agents in some problem domain. It contains a set of players $P$ including a “chance player” $P_c$ whose actions represent stochastic events such as dice rolls and card deals. $H$ is a set of all possible game states, represented by the history of actions taken from the initial game state $\varnothing$. For any history $h \in H$, $P(h) \mapsto P \cup \{leaf\}$ gives the player that is about to act or $leaf$ if the game is over, and $A(h)$ gives the set of legal actions. $H_p$ is the set of all states $h$ such that $P(h) = p$. The state $h \cdot a$ is said to be a child of the state $h$, $h$ is the parent of $h \cdot a$, and we will say $h_i$ is an ancestor of $h_j$ or $h_i \sqsubseteq h_j$ if $h_i$ is the parent of $h_j$ or $h_i \sqsubseteq h_k$ and $h_k$ is the parent of $h_j$. $h \sqsubseteq j$ if $h \sqsubseteq j$ or $h = j$. Conversely, $h[S]$ is the longest history $j \in S$ for which $j \sqsubseteq h$. We will let $Z$ be the set of all $leaf$ states. For every $z \in Z$, $u_p(z) \mapsto \mathbb{R}$ gives the payoff for player $p$ if the game ends in state $z$.

For each player $p$, hidden information is described by information sets, which are a partition $I_p$ of $H_p$. For any information set $I \in I_p$, any two states $h, j \in I$ are indistinguishable to player $p$. A behaviour strategy $\sigma_p \in \Sigma_p$ is a function $\sigma_p(I, a) \mapsto \mathbb{R}$ which assigns a probability distribution over valid actions to every information set $I \in I_p$. We will say $\sigma(h, a) = \sigma(I(h), a)$ where $I(h)$ is the information set which contains $h$, since a player cannot act differently depending on information they did not observe. $Z(I) = \{z \in Z \text{ s.t. } z \sqsubseteq h \in I\}$ is the set of all terminal states $z$ reachable from some state in information set $I$. We can also consider the leaves reachable from $I$ after some action $a$, stated as $Z(I, a) = \{z \in Z \text{ s.t. } z \sqsubseteq h \cdot a, h \in I\}$.

In games with perfect recall, any two histories $h$ and $j$ in an information set $I \in I_p$ must have passed through the same sequence of player $p$ information sets, and made the same action at those information sets. Informally speaking, perfect recall means that a player does not forget their own actions, or any information about chance or opponent actions that they have observed. Having a unique history of information sets also lets us say an information set $J$ is a child of information set $I$, or $I \sqsubset J$, if for any $h \in J$, $h[I]$ is the longest strict subsequence of $h$ where $P(h[I]) = P(h)$.

A strategy profile $\sigma \in \Sigma$ is a tuple of strategies, one for each player. Given a strategy profile $\sigma$ we can construct a new profile $\sigma_{(\sigma_p')}$, which is identical except that

| Method                  | Space for Leduc | Space for Rhode | Space for Texas |
|------------------------|-----------------|-----------------|----------------|
| CFR (solving)          | 40.7KB          | 4.34GB          | 522TB          |
| (using)                | 20.8KB          | 2.17GB          | 261TB          |
| SR (solving)           | 40.7KB          | 4.34GB          | 522TB          |
| (using)                | 9.84KB          | 15.5MB          | 14.1GB         |
| CFR-D (solving)        | 6.56KB          | 19.9MB          | 11.4GB         |
| (using)                | 9.84KB          | 15.5MB          | 14.1GB         |

Table 1: Total space used for finding and using a strategy in three limit Hold’em poker variants. The three cases for each game are a strategy generated by CFR, a strategy generated by CFR with subgame strategies recovered at runtime (SR), and a strategy generated by CFR-D with strategies recovered at run-time.
player $p$’s strategy has been replaced by $\sigma_p'$. Given $\sigma$, it is also useful to refer to certain products of probabilities. For any $h \in H$ and $\sigma \in \Sigma$, $\pi_\sigma(h) = \prod_{(j,a) \in h} \sigma_{P(j)}(j,a)$ gives the joint probability of reaching $h$ if all players follow $\sigma$. We also use $\pi_p(h)$ to refer to the product of only the terms where player $p$ acts, and $\pi_{-p}(h)$ to refer to the product of terms where any player but $p$ acts. We use $\pi(j, h)$ to refer to the product of terms from $j$ to $h$, rather than from $\emptyset$ to $h$. Finally, it is useful to speak of replacing portions of a strategy with another strategy: $\sigma[S \leftrightarrow \sigma']$ is the strategy that is equal to $\sigma$ everywhere except at information sets in $S$, where it is equal to $\sigma'$.

Given a strategy profile $\sigma$, the expected utility $u^\sigma_p$ to player $p$ if all players follow $\sigma$ is $\sum_{z} \pi(z) u_p(z)$. The expected utility $u^\sigma_p(I, a)$ of taking an action at an information set is $\sum_{z \in Z(I, a)} \pi^\sigma(z) u_p(z)$. In this paper, we will frequently use a variant of expected value called counterfactual value: $v^\sigma_p(I, a) = \sum_{z \in Z(I, a)} \pi^\sigma_p(z) \pi^\sigma_r(z[I] : a, z) u_p(z)$. Informally, the counterfactual value of $I$ for player $p$ is the expected value of $I$ if $p$ had always played to reach $I$.

A best response $BR_p(\sigma) = \arg\max_{\sigma_p' \in \Sigma_p} u^{\sigma [P_p \leftarrow \sigma_p']}$ is a strategy for $p$ which maximises $p$’s value if all other player strategies remain fixed. A Nash equilibrium is a strategy profile where all strategies are simultaneously best responses to each other, and an $\epsilon$-Nash equilibrium approximation is a profile where the expected value for each player is within $\epsilon$ of the value of a best response strategy. In two player constant-sum games, the expected utility of any Nash equilibrium is a game-specific constant, called the game value. In a two-player zero-sum game, we use the term exploitability to refer to a strategy profile’s average loss to a best response across its component strategies. A Nash equilibrium has an exploitability of zero. All of the work in this paper assumes two player, zero-sum, perfect recall games.

A counterfactual best response $CBR_p(\sigma)$ is a strategy for $p$ which maximises $v^\sigma_p(I)$ at all information sets $I \in I_p$. $CBR_p$ is necessarily a best response, but $BR_p$ may not be a counterfactual best response, because $BR_p$ may choose to make an action which does not maximise $v_p(I)$ if $\pi_p(I) = 0$. The well known recursive bottom-up technique of constructing a best response generates a counterfactual best response.

### 3 Decomposition into Subgames

In this paper, we will use an imperfect-information extension of a perfect information subgame. A subgame, in a perfect information game, is a tree. It is rooted at some arbitrary state and contains all states which can be reached from the root state. The state-rooted subgame definition is sometimes used directly in imperfect information games, but the definition is largely characterised by a lack of useful properties. The reason that a state-rooted subgame is not useful in an imperfect information game is that the tree cuts across information set boundaries: for any state $s$ in the tree, there is generally at least one state $t \in I(s)$ which is not in the tree. We will use an imperfect information extension of the concept of a subgame.

First, it is also convenient to extend the concept of an information set. $I(h)$ is defined in terms of the states which player $p = P(h)$ can not distinguish. We would
also like to partition states where player \( p \) acts into those which player \( p' \neq p \) can not distinguish. \( I_{p'}(h) \) is the information set for player \( p' \) containing \( h \). These augmented information sets could be given as part of the definition of the game, or they can be constructed from the standard information sets. To construct the augmented information sets automatically, we use the ancestor information sets. Let \( H_{p'}(h) \) be the sequence of player \( p' \) information sets reached by player \( p' \) on the path to \( h \), and the actions taken by player \( p' \). Then for two states \( h \) and \( j \), \( I_{p'}(h) = I_{p'}(j) \iff H_{p'}(h) = H_{p'}(j) \).

For any states \( h, j \), we will say \( \text{grouped}(h, j) \) if there exists a player \( p \) such that \( I_{p}(h) = I_{p}(j) \) or if there exists a state \( k \) such that \( \text{grouped}(h, k) \) and \( \text{grouped}(k, j) \). A subgame can now be defined as a \( \text{grouped} \) set of root states, and all states reachable from any of these initial states. A group-rooted subgame no longer crosses information set boundaries: it contains all states in the information set of any state in the subgame, for all players.

We will use the game of rock-paper-scissors as a running example. In rock-paper-scissors, two players simultaneously choose rock, paper, or scissors. They then reveal their choice, with rock beating scissors, scissors beating paper, and paper beating rock. The simultaneous moves in rock-paper-scissors can be modeled using an extensive form game where one player goes first, without revealing their action, then the second player acts. The extensive form game is shown on the left side of figure 1. The dashed box indicates the information set \( I_2 = \{R, P, S\} \) which tells us player two does not know player one’s action.

On the right side of figure 1, we have decomposed the game into two parts: a trunk and a single subgame. The trunk contains the single state \( \emptyset \), and the subgame contains the three states \( R, P, \) and \( S \). In the subgame, there is one player two information set \( I_2 = \{R, P, S\} \) and three augmented player one information sets \( I_R^1 = \{R\}, I_P^1 = \{P\}, \) and \( I_S^1 = \{S\} \). Ignoring the degenerate case where the subgame or trunk is empty, this is the only possible decomposition. While the rock-paper-scissors example is very simple (there is only a single subgame, player two never acts in the trunk, and player one never acts in the subgame) it will be sufficient to demonstrate how a naive method of decomposition fails, and motivate the techniques in this paper.

One contribution of the paper is a method for recovering a strategy in a subgame, with the property that the combined trunk and subgame strategy satisfy an exploitability guarantee in the whole game. We require a strategy in the trunk and counterfactual values for the information sets at the root of the subgame. This method also works
in the case where the trunk strategy is part of a Nash equilibrium: if we know the
counterfactual values of all of the subgames, the method can be used to re-generate a
complete $\epsilon$-Nash equilibrium in a piecewise fashion, one subgame at a time.

Using our rock-paper-scissors example, assume we have an equilibrium policy in
the trunk (player one picks between $R$, $P$, and $S$ uniformly at random) and some
counterfactual values for $I_1^R$, $I_1^P$, $I_1^S$, and $I_2^{R,P,S}$ (all zero.) We have no policy in the
subgame (player two picking an action) and would like to generate a policy so we have
a complete equilibrium profile for the whole game. The recovery method will correctly
come up with the player two policy which picks between $R$, $P$, and $S$ uniformly at
random.

The second contribution of this paper is CFR-D, a low-memory algorithm which
generates an approximation of a Nash equilibrium. Instead of storing information about
the entire game CFR-D only stores information about the trunk. As described in table 1,
this can result in very large space savings. CFR-D is described in section 5.

4 Subgame Strategy Recovery

In this section, we will present a method of recovering a strategy in a subgame, given a
trunk strategy and some information about the root of the subgame. The novel property
of this method is a bound on the exploitability of the combined trunk and subgame
strategy in the whole game. The recovery problem might arise in various situations.
For example, we might wish to move a Nash equilibrium strategy from some large
machine to one with very limited memory. If we can recover the strategy in a subgame,
we can throw away parts of the original strategy until the remaining portion is small
enough, reconstructing and discarding the subgames as needed. In some other case, we
may have an existing strategy, and we wish to find a better subgame strategy, with a
guarantee that the new strategy does at least as well as the existing strategy in the worst
case.

First, we note that it is not sufficient to simply re-solve the subgame with the as-
sumption that the trunk policy is fixed. If the trunk is part of a Nash equilibrium, the
simple method of re-solving a subgame will produce a strategy which achieves the
game value when played against an opponent using the given fixed trunk strategy, but
is unlikely to recover a Nash equilibrium strategy.

Consider the rock-paper-scissors example. In the trunk, player one picks uniformly
between $R$, $P$, and $S$. In the subgame, player one has only one possible (vacuous)
policy: they take no actions. To find an equilibrium, player two must pick a strategy
which is a best response to the empty player one policy, given the probability of $\frac{1}{3}$ for
$R$, $P$, and $S$ induced by the trunk strategy.

If we consider player two choosing rock, paper, or scissors, they have an expected
utility of $\frac{1}{2}(0 - 1 + 1)$, $\frac{1}{2}(1 + 0 - 1)$, and $\frac{1}{2}(-1 + 1 + 0)$ respectively. All of these
values are 0, so player two can pick an arbitrary policy. For example, player two might
choose to always play rock. Always playing rock achieves the game value of 0 against
the combined trunk and subgame strategy for player one, but gets a value of $-1$ if
player one switched to playing paper. A player two strategy of always playing rock is
not part of any Nash equilibrium.
The problems that arise with re-solving the subgame are related to the counterfactual values at the root of the subgame. In rock-paper-scissors, the player one counterfactual values for $R$, $P$, and $S$ are all 0 in an equilibrium profile. When player two always played rock in the example above, the player one counterfactual values for $R$, $P$, and $S$ were 0, 1, and $-1$ respectively. Because the counterfactual value for $P$ was higher than the equilibrium value of 0, player one had an incentive to switch to playing paper. That is, they could change their policy to convert the larger “what-if” counterfactual value of paper into a higher expected utility.

In rock-paper-scissors, the player two equilibrium policy of choosing to play rock, paper, or scissors uniformly at random is the unique strategy which gives player one a counterfactual value of 0 for $R$, $P$, and $S$. If we knew the counterfactual values in an equilibrium profile were 0, and we then found a player two strategy which did not let player one have higher counterfactual values, we would have correctly recovered the player two equilibrium strategy.

This can be generalised beyond equilibrium play. If the recovered subgame strategy does not allow a best response opponent to have counterfactual values which are larger than the counterfactual values for a best response to our original strategy, then our worst case performance is no worse when using the recovered strategy.

More formally, assume we have a player $p$, an opponent $o$, a strategy $\sigma_p$, a strategy $\sigma_p' = \sigma_p, [S \leftarrow \sigma_p^g]$ constructed from $\sigma_p$ using a recovered player $p$ subgame $S$ strategy $\sigma_p^g$, and $v_o^{<\sigma_p',CBR(\sigma_p)>}(I) \leq v_o^{<\sigma_p,CBR(\sigma_p)>}(I)$ for all information sets $I$ at the root of subgame $S$. Because the expected utility $u_o$ can be broken down into the utility in the trunk and a linear combination of subgame counterfactual values $v_o(I)$ with non-negative weights, $u_o^{<\sigma_p',CBR(\sigma_p)>} \leq u_o^{<\sigma_p,CBR(\sigma_p)>} \leq u_o^{<\sigma_p,CBR(\sigma_p)>}$.

We propose using the game shown in figure 2 to recover a strategy in a subgame. From here on, we will assume, without loss of generality, that we are recovering a strategy for player $p_1$ (by providing a guarantee on the best case performance of player $p_2$.) We will distinguish the recovery game from the original game by using ‘ to distinguish states, utilities, or strategies for the recovery game. The basic construction is that each state $r$ at the root of the original subgame turns into a three states: $p_2$ choice node $\tilde{r}$, a terminal state $\tilde{r} \cdot T$, and a state $\tilde{r} \cdot F$ which is identical to $r$. All other states in the original subgame are directly copied into the recovery game. Along with the original game, we must also be given a $p_1$ strategy $\sigma_1$, from which we construct a strategy profile $\sigma = <\sigma_1, CBR(\sigma_1) >$.

There is an initial chance node which leads to states $\tilde{r} \in \tilde{R}$, corresponding to the probability of reaching state $r \in R$ in the original game. Each state $\tilde{r} \in \tilde{R}$ occurs with probability $\pi_{\tilde{Z}_2}(r)/k$, where the constant $k = \sum_{r \in R} \pi_{\tilde{Z}_2}(r)$ is used to ensure that the probabilities sum to 1. $\tilde{R}$ is partitioned into information sets $\tilde{I}_2^R$ that are identical to the (augmented) information sets $I_2^R$ which partition states $R$.

At each $\tilde{r} \in \tilde{R}$, $p_2$ has a binary choice of $F$ or $T$. After $T$, the game ends. After $F$, the game is the same as the original subgame. All leaf utilities are multiplied by $k$ to undo the effects of normalising the initial chance event. So, if $\tilde{z}$ corresponds to a leaf $z$ in the original subgame, $\tilde{u}_2(\tilde{z}) = ku_2(z)$. If $\tilde{z}$ is a terminal state after a $T$ action, $\tilde{u}_2(\tilde{z}) = u_2(\tilde{r} \cdot T) = ku_2(I(r))/\sum_{h \in I(r)} \pi_{\tilde{Z}_2}(h)$. This means that for any $I \in \tilde{I}_2^R$,
\( \tilde{u}_2(I, T) = v_2^\sigma(I) \), the counterfactual value of \( I \).

No further construction is needed. If we solve the proposed game to get a new strategy profile \( \sigma' \), we can directly use \( \sigma' \) in the original subgame of the full game. To see that \( \sigma' \) achieves the goal of not increasing the counterfactual values for \( p_2 \), consider \( \tilde{u}_2(I) \) for \( I \in \mathcal{I}_S^R \) in an equilibrium profile for the recovery game. \( p_2 \) can always pick \( T \) at the initial choice to get the original counterfactual values, so \( \tilde{u}_2(I) \geq v_2^\sigma(I) \). Because \( p_2 \) was already playing a counterfactual best response in \( \sigma =< \sigma_1, CBR(\sigma_1) > \), \( \tilde{u}_2(I) \leq v_2^\sigma(I) \) in an equilibrium. So, in a solution \( \sigma' \) to the recovery game, \( \tilde{u}_2'(I) = v_2^\sigma(I) \), and \( \tilde{u}_2^{< \sigma_1, CBR(\sigma_1)>}(I \cdot F) \leq v_2^\sigma(I) \). By the construction of the recovery game, this implies that \( v_2^{< \sigma_1, CBR(\sigma_1)>}(I) \leq v_2^{< \sigma_1, CBR(\sigma_1)>}(I) \).

If we recover the strategy for both players at a subgame, the regret of the complete recovered strategy is increased by no more than \( (|I_R| - 1)\epsilon_S + \epsilon_R \), where \( \epsilon_R \) is the regret bound of the subgame strategy in the recovery subgame, \( \epsilon_S \) is the regret bound of the original subgame strategy in the full game, and \(|I_R|\) is the number of information sets for both players at the root of a subgame. A proof is given in Section 8.

\section{Generating a Trunk Strategy using CFR-D}

CFR-D is part of the family of counterfactual regret minimisation (CFR) algorithms, which are all efficient methods for finding an approximation of a Nash equilibrium in very large games. CFR is an iterated self play algorithm, where the average policy across all iterations approaches a Nash equilibrium [15]. It has independent regret minimisation problems being simultaneously updated at every information set, at each iteration. Each minimisation problem at an information set \( I \in \mathcal{I}_p \) uses immediate counterfactual regret, which is just external regret over counterfactual values:

\[
\max_{a \in A(I)} \sum_t v_{p(I)}^\sigma(I, a) - \sum_{a'} \sigma(I, a')v_{p(I)}^\sigma(I, a').
\]

For more details, see the original CFR paper.

The desired end result, minimising regret across the space of entire strategies, is an emergent property of the CFR algorithm. Zinkevich et al. show that the immediate counterfactual regrets place an upper bound on the regret across all strategies, and
another short argument shows that an $\epsilon$-regret strategy profile is a $2\epsilon$-Nash equilib-rium. These proofs of convergence to a Nash equilibrium are given in the original CFR paper [15].

Using separate regret minimisation problems at each information set makes CFR a very flexible framework. First, any single regret minimisation problem at an information set $I$ only uses the counterfactual values of the actions. The action probabilities of the strategy profile outside $I$ are otherwise irrelevant. Second, while the strategy profile outside $I$ is generated by the other minimisation problems in CFR, the source does not matter. Any sequence of strategy profiles will do, as long as they have low regret.

A CFR variant, CFR-BR [7] uses these properties. The game is split up into a trunk and a number of subgames. Instead of updating and using CFR’s usual regret minimisation problems for both players in the subtree, one player generates a strategy as usual to minimise regret, while the other player uses a best response to the regret player’s current strategy. After updating regrets for the regret player in the subgame, and computing counterfactual values for both players at the root of the subgame, the best response strategy for the current iteration is discarded.

As Johanson et al. [7] show, a player using a best response has non-positive cumulative regret after any number of iterations, and the average strategy for both players is a Nash equilibrium approximation. Because the best response is re-calculated at every iteration, CFR-BR does not need to store the regrets or average policy at information sets in a subgame that are controlled by the best response player.

CFR-D takes the idea of CFR-BR one step further. Instead of having one player use a best response strategy in each subgame, which is discarded at each iteration after computing counterfactual values, we have both players use a best response in a subgame. That is, in each iteration for each subgame, CFR-D finds a Nash equilibrium for the subgame, computes the counterfactual values at the root of the subgame, and then discards the subgame strategy. Using the counterfactual values from the subgames, CFR-D updates regrets in the trunk in the usual CFR fashion.

Like CFR-BR, CFR-D does not need to store any values for an information set in a subgame controlled by a best response player, but in CFR-D this means no values are stored for information sets in a subgame for either player. CFR-D only stores values for information sets in the trunk, giving it memory requirements which are sub-linear in the number of information sets. The correctness of CFR-D immediately follows from the correctness proof in the CFR-BR paper: there are simply more information sets which are guaranteed to have non-positive regret, as both players are playing a mutual best response in the subgames.

The benefit of CFR-D is the reduced memory requirements. Depending on the sizes of the trunk and subgames, and the number of subgames, treating the subgames independently could lead to a substantial reduction in space, as shown in table[1]. There are two costs to the reduced space. The first is that the subgame strategies must be recovered at run-time. The second cost is increased CPU time to solve the game. At each iteration, CFR-BR must find a Nash equilibrium for a number of subgames. CFR variants require $O(1/\epsilon^2)$ iterations to have an error less than $\epsilon$, and this bound applies to the number of trunk iterations required for CFR-D. If we use CFR to solve the subgames, each of the subgames will also require $O(1/\epsilon^2)$ iterations at each trunk
ALGORITHM 1: CFR-D

Input: Number of time steps $T$, and an extensive-form game partitioned into a trunk and subgames

Output: action probabilities $probs_{I,a}$ in the trunk, and counterfactual values $cfv_{p,I}$ at the root of subgames

$\text{regret} = 0; \;\text{probs} = 0; \;\text{cfv} = 0;$

for each time step $t$ from 1 to $T$
  for each information set $I$ in the trunk and action $a$
    $\sigma_{I,a} = \max(0, r_{I,a}) / \sum_{a'} \max(0, r_{I,a'})$;
    $probs_{I,a} = (probs_{I,a} \ast (T - 1) + \sigma_{I,a} \ast \pi^p_{P(I)}(I)) / T$;
  end
  for each subgame $S$
    $\tilde{\sigma} = \text{SOLVE}(S)$;
    for each player $p$ and information set $I_{S,p}$ at root of $S$
      $v_{I_{S,p}} = \sum_{z \in Z(I)} \pi^p_{SG \leftarrow \tilde{\sigma}}(z) \pi^p_{I}(z) u_p(z)$;
      $cfv_{I_{S,p}} = (cfv_{I_{S,p}} \ast (T - 1) + v_{I_{S,p}}) / T$;
    end
  end
  for each information set $I$ in the trunk and action $a$
    $v_I = \sum_{a} \sigma_{I,a} \ast \sum_{J \supset I,a} v_J$;
    $\text{regret}_{I,a} = \text{regret}_{I,a} + \sum_{J \supset I,a} v_J - v_I$;
  end
end

iteration, so CFR-D ends up doing $O(1/\epsilon^4)$ work.

If the counterfactual regret at an information set $I$ at the root of a subgame is bounded by $\epsilon_S$ at each time step, then at time $T$ the accumulated full counterfactual regret $R^{T}_{\text{full}}(I) \leq T \epsilon_S$. Following Zinkevich et al.’s argument in Appendix A.1 [15], the average regret over the whole game will be bounded by $N_T A \sqrt{T}/T + N_S \epsilon_S$, where $N_T$ is the number of information sets in the trunk, and $N_S$ is the number of information sets which are at the root of a subgame, and $A$ is the maximum number of available actions at an information set in the trunk.

5.1 Solving a Subgame in CFR-D

As part of the CFR-D algorithm, we need to solve a subgame given a trunk strategy $\sigma$. With a fixed policy $\sigma$ in the trunk, solving a subgame means finding a strategy profile $\tilde{\sigma}$ for the subgame such that for either player $p$, $\sigma_{p[SG \leftarrow \tilde{\sigma}]}$ is a best response to $\sigma_{p[SG \leftarrow \tilde{\sigma}]}$ within the restricted space of strategies that play like $\sigma$ outside of the subgame. For CFR-D, the solution $\tilde{\sigma}$ must also be a pair of mutual counterfactual best responses. Without this constraint, $\tilde{\sigma}$ might have positive counterfactual regret in the subgame.

To see why there can be positive counterfactual regret with an unconstrained solution, consider the case when $\pi^p_I(J) = 0$ for some $I$ at the root of a subgame. For any information set $J \in I_p$ which is a descendant of $I$, the normal utility $u^p_{\pi^p_{SG \leftarrow \tilde{\sigma}]}(J,a) = 0$
regardless of the policy $\sigma(J)$ because $\pi_p(I)$ is part of each term in that sum. At the same time, the counterfactual value $v^p_{\sigma_p} (J, a)$ is not multiplied by $\pi_p(I)$ and can have a non-zero value. If every possible policy under $I$ is a part of a Nash equilibrium because the values are uniformly 0, an arbitrarily chosen Nash equilibrium is unlikely to achieve the best counterfactual value, leading us to have some regret for playing $\sigma_p$ in the subgame.

There are at least three possible methods for finding an equilibrium which is also a pair of mutual counterfactual best responses. First, we could generate any Nash equilibrium, and then fix it with a post-processing step which computes the best response using counterfactual values whenever $\pi_p(I)$ is 0. Second, we could try directly adding the constraint to some other solution method, like a sequence form linear program [10] or iterated smoothing [4]. Finally, we could simply use some CFR variant to solve the subgame, as they naturally produce strategies which are counterfactual best responses. Note that because CFR-D is a solution method, we could also use CFR-D itself as a solution method for the subgames. Using CFR-D recursively on a game with sufficient structure, the memory requirements to find the top-level trunk strategy could be linear in the depth of the game, although the running time would grow geometrically with depth.

6 Experimental Results

We have three main claims to demonstrate. First, if we have a strategy, we can reduce the space usage by keeping only summary information about the subgames, and then recover any subgame strategy with arbitrarily small error. Second, that we can decompose a game, only use space for the trunk and a single subgame, and generate an arbitrarily good approximation of a Nash equilibrium using CFR-D. Finally, we show that we can use the subgame re-solving technique to reduce the exploitability of an existing strategy. All timing results were generated on a 2.67GHz Intel Xeon X5650 based machine running Linux.

6.1 Recovering Strategies in Subgames

In order to show that re-solving a subgame introduces at most an arbitrarily small exploitability, we use the game of Leduc Hold’em, which has become a testbed for research on imperfect information games [14, 3]. The game involves a deck of 6 cards (2 suits and 3 ranks) and two rounds of betting, with at most 1 bet and 1 raise per round. Each player starts by paying one chip, with bets and raises costing 2 chips in the first round and 4 chips in the second round. The game is complicated enough to show interesting behaviours, but with only 936 information sets it is small enough that a wide range of experiments can easily be run and evaluated.

In this experiment, the trunk used was the first round of betting, and there were five subgames corresponding to the five different betting sequences where no player folds in the first round. When recovering subgame strategies, we used the Public Chance Sampling (PCS) variant of CFR [8] for solving the recovery games.
To demonstrate the practicality of recovering subgame strategies, we started with an almost exact Nash equilibrium (exploitable by less than $2.5 \times 10^{-11}$ chips per hand,) computed the counterfactual values of every hand in each subgame for both players, and discarded the strategy in all subgames. These steps correspond to a real scenario where we pre-compute and store a Nash equilibrium in an offline fashion. At run-time, we then recovered a strategy for every subgame, using the subgame recovery game constructed from the counterfactual values and trunk strategy. We could then measure the exploitability of the trunk strategy combined with all of the newly recovered subgame strategies.

Figure 3 shows the exploitability when using a different number of CFR iterations to solve the recovery games. The $O(1/\sqrt{T})$ error bound for CFR in the recovery games very clearly translates into the expected $O(1/\sqrt{T})$ error in the overall exploitability of the re-constructed strategy. The top plot shows the exploitability plotted against the number of iterations, and the bottom plot shows the exploitability of the same runs plotted against the time to solve one subgame.

For comparison, the “unsafe recovery technique” line on the plots in figure 3 shows the performance of a system for approximating undominated subgame solutions [2]. Not only is there no theoretical bound on exploitability, the real world behaviour is not ideal. Instead of approaching a Nash equilibrium (0 exploitability), the exploitability of the recovered strategy approaches a value of around 0.080 chips/hand.

6.2 Solving Games with Decomposition

To demonstrate CFR-D, we split Leduc Hold’em across the two rounds, in the same fashion as the strategy recovery experiments. Our implementation of CFR-D used CFR for solving subgames and strategy recovery games. All the reported results use 200,000 iterations for each of the recovery subgames (~ 0.8 seconds per subgame.) Each line of figure 4 plots the exploitability for different numbers of subgame iterations, ranging from 100 to 12,800 iterations. There are results for 500, 2,000, 8,000, and 32,000 trunk iterations.

Looking from left to right, each of the lines show the decrease in exploitability as the quality of subgame solutions increases. The different lines compare exploitability across an increasing number of CFR-D iterations in the trunk.

Given the error bound for CFR variants is $O(\sqrt{T})$, one might expect exploitability results to be a straight line on a log-log plot. In these experiments, CFR-D is using CFR for the trunk, subgames, and the recovery games, so the exploitability is a sum of trunk, subgame, and subgame recovery errors. For each of the lines on the graph, trunk and subgame recovery error are constant non-zero values. Only subgame error decreases as the number of subgame iteration increases, so each line is approaching the non-zero trunk and recovery error, which shows up as a plateau on a log-log plot.

Using 32,000 trunk iterations, 12,800 subgame iterations, and 200,000 recovery game iterations, we get an exploitability of 0.0075 chips/hand. Using a single CPU core, generating that strategy with CFR-D took approximately 114 minutes. More than half of the final exploitability comes from the recovery: using the saved trunk strategy and counterfactual values with 6,400,000 recovery iterations reduces the exploitability to 0.0028 chips/hand. In comparison, CFR produces a 0.0028 chips/hand exploitable
Figure 3: exploitability after subgame recovery
strategy in 23 seconds: the reduction in space for CFR-D comes at the cost of extra time required to solve the subgames at each iteration.

6.3 Re-solving Subgames

Generating a new subgame strategy at run-time can also be used to improve the exploitability of a strategy. For example, in the paper describing their new subgame solving technique, Ganzfried et al. report positive results in experiments where subgames are re-solved using a much larger abstract game than the original solution [2]. Our new subgame re-solving method adds a theoretical guarantee to the ongoing research in this area.

In figure 5, we demonstrate re-solving subgames with a Leduc Hold’em strategy generated using an abstraction. In the original strategy, the players can not tell the difference between a Queen on the board or a King on the board if they hold a Jack, nor can they tell the difference between a Jack on the board or a Queen on the board if they hold a King. While this gives the player perfect knowledge about the strength of the hand (against a uniform random opponent card), they have lost valuable knowledge and the resulting strategy is exploitable for 0.382 chips/hand in the full game. To generate the counterfactual values needed for our method, we simply do a best response computation within the subgame: the standard recursive best response algorithm naturally produces counterfactual values.

The top plot shows the expected value of the abstract strategy with re-solving, when played in the full game against the original abstract strategy. With little recovery effort, both re-solving techniques see some improvement against the original strategy. With more effort, the unsafe re-solving technique has a small edge of around 0.004 chips/hand over our re-solving technique.

If we measure the exploitability of the re-solved strategy instead of the performance against the strategy we used for re-solving, we get the results in the bottom plot.
Figure 5: plots of re-solved abstract strategy in Leduc Hold’em showing performance against the original strategy (top) and exploitability (bottom)
Within 200 iterations, our new re-solving method finds a strategy which has decreased the exploitability from 0.382 chips/hand to 0.331 chips/hand. By 2,000 iterations, the exploitability falls to somewhere between 0.230 to 0.290 chips/hand, where it remains. The unsafe method, after 3,125 iterations, stays in a range of around 0.382 to 0.418 chips/hand. In the most generous case, the unsafe method produces a strategy which can be exploited for almost 0.100 chips/hand more. In the worst case, the difference is almost 0.200 chips/hand. Note that the unsafe method’s apparent convergence in figure 5 to the original exploitability is a coincidence: in other situations the re-solved strategy can be less exploitable, or significantly more exploitable than the original strategy.

If we are confident about our knowledge of the opponent’s strategy, we might want to use the existing unsafe method of re-solving. In any other circumstance, the difference in exploitability means that the safe method could be ahead by a much larger margin. Our re-solving technique provides a robust strategy: it comes with a guarantee that the re-solved strategy does no worse than the original strategy (minus the user-chosen error determined by the solution quality of the subgame recovery problem.)

Note that the re-solving process does not require us to have used abstraction: it is more general. In any circumstance where we know, or suspect, that we have a sub-optimal strategy in some part of the game, we can use our re-solving technique on the relevant subgame.

7 Conclusions

In perfect information games, decomposing the problem into independently handled subgames is a simple and effective method which is used to greatly reduce the space and time requirements of algorithms. The incomplete knowledge in imperfect information games has previously meant that decomposition into parts leads to the loss of any theoretical guarantees on solution quality. We give a method which uses counterfactual values to recover a strategy. Our method combines the space savings of decomposition with a guarantee that the worst case performance is no worse than the strategy used to generate the counterfactual values. Previous methods have had one or the other properties, but not both, and we demonstrate that unsafe methods can be significantly exploitable in practice. We also present CFR-D, an algorithm which uses decomposition to solve games using asymptotically less space than existing algorithms.

8 Proofs

Theorem 1 gives a proof of the upper bound on exploitability of the recovered strategy. The context for this section is as follows. Strategy profile $\sigma$ is an approximation of a Nash equilibrium for the whole game. The induced recovery game strategy profile $\sigma^F$ is the strategy where for all information sets in the subtrees under the $F$ action, $\sigma^F$ takes the same action as $\sigma$, and at the $p_2$ information sets where $F$ or $T$ is chosen, $p_2$ always picks $F$. As in Section 4 we will be considering the process from the point of view of recovering a strategy for $p_1$. 

Note that because $\pi^p(h) = \pi^p(j)$ for all $h, j \in I$, $u^p(I, a) = \sum_{Z(I)} \pi^a(z)u^p(z) = \pi^p(I)u^p(I, a)$.

**Lemma 1** For any $p_2$ strategy $\rho$ in the original game and $p_1$ strategy $\hat{\rho}$ in the recovery game, if we let $\hat{\sigma} = (\sigma_1[S_{G-e}\hat{\rho}], \rho)$, then for any $I \in \mathcal{I}_2^R$, $u^2_2(I) = \pi^p_2(I)\hat{u}_{2\hat{\rho}}(\hat{\rho})(I)$.

**Proof**

\[
u^2_2(I) = \sum_{z \in \mathcal{Z}(U)} \pi^2_2(z[I])\pi^2_2(z[I])\pi^2_2(z[I], z) v_2(z)
= \pi^p_2(I) \sum_{z} \pi^p_2(z[I])/k \pi^p_2(z[I], z) v_2(z) k = \pi^p_2(I) \hat{u}_{2\hat{\rho}}(\hat{\rho})(I)
\]

\[\square\]

**Lemma 2** If $\tilde{\sigma}$ is an $\epsilon_R$-Nash equilibrium in the recovery game, $0 \leq c_I \leq 1$, and $u^2_2([\sigma_1, BR(\sigma_1)])(I) \leq \epsilon_S + u^2_2(I)$ for all $I$, then

\[
\sum_{I \in \mathcal{I}_2^R} c_I \tilde{u}_{2([\tilde{\sigma}_1, BR(\tilde{\sigma}_1)])}(I) \leq (|\mathcal{I}_2^R| - 1)\epsilon_S + \epsilon_R + \sum_{I} c_I \tilde{u}_{2([\sigma_1, BR(\sigma_1)])}(I)
\]

**Proof** $\sigma$ and $\tilde{\sigma}$ have the following properties.

\[
\tilde{u}_{2([\tilde{\sigma}_1, BR(\tilde{\sigma}_1)])} \leq \epsilon_R + \tilde{u}^{\tilde{\sigma}_1, BR(\tilde{\sigma}_1)} 
\leq \epsilon_R + \tilde{u}_{2([\sigma_1, BR(\sigma_1)])}(I)
\]

\[
\tilde{v}^2_2(I) \leq \tilde{u}^2_2([\sigma_1, BR(\sigma_1)]) \leq \epsilon_S + \tilde{u}^{\sigma_1, BR(\sigma_1)}(I) = \epsilon_S + \tilde{v}^2_2(I)
\]

Given this, the maximum difference between $c \cdot \tilde{u}_{2([\tilde{\sigma}_1, BR(\tilde{\sigma}_1)])}$ and $c \cdot \tilde{u}_{2([\sigma_1, BR(\sigma_1)])}$ occurs when the difference of these sums is concentrated at a single $I$. That is, for some $I$

\[
\tilde{u}_{2([\tilde{\sigma}_1, BR(\tilde{\sigma}_1)])}(I) = (|\mathcal{I}_2^R| - 1)\epsilon_S + \tilde{u}^2_2([\sigma_1, BR(\sigma_1)])(I)
\]

$c_I = 1$

and for all $I' \neq I$

\[
\tilde{u}_{2([\tilde{\sigma}_1, BR(\tilde{\sigma}_1)])}(I') = \tilde{u}^{\sigma_1, BR(\sigma_1)}(I')
\]

\[
\tilde{u}_{2([\sigma_1, BR(\sigma_1)])}(I') = \epsilon_S + \tilde{u}^{\sigma_1, BR(\sigma_1)}(I')
\]

$c_{I'} = 0$

In this case, the difference is $(|\mathcal{I}_2^R| - 1)\epsilon_S + \epsilon_R$.  

\[\square\]
Theorem 1 Let $\epsilon$ be a equilibrium profile approximation, where $\epsilon_S$ is an upper bound on the $p_2$ counterfactual regret so that $R_2(I) \leq \epsilon_S$ over all $I$ in $\mathcal{I}^R_2$. Let $\hat{\sigma}$ be the recovered strategy, with a bound $\epsilon_R$ on the exploitability in the recovery game. Then the exploitability of $\sigma$ is increased by no more than $(|\mathcal{I}^R_2| - 1)\epsilon_S + \epsilon_R$ if we use $\hat{\sigma}$ in the subgame:

$$u_2^{(\sigma_1[SG \leftarrow \hat{\sigma}], BR(\sigma_1[SG \leftarrow \hat{\sigma}]))} \leq (|\mathcal{I}^R_2| - 1)\epsilon_S + \epsilon_R + u_2^{(\sigma_1, BR(\sigma_1))}$$

Proof Let $\hat{\sigma} = (\sigma_1[SG \leftarrow \hat{\sigma}], BR(\sigma_1[SG \leftarrow \hat{\sigma}]))$. In this case,

$$u_2^\hat{\sigma} = \sum_{z \in SG} \pi^\hat{\sigma}(z)u_2(z) + \sum_{z \in SG} \pi^\sigma(z)u_2(z)$$

$$= \sum_{z \in SG} \pi^{(\sigma_1, BR(\sigma_1))}(z)u_2(z) + \sum_{z \in SG} \pi^\sigma(z)u_2(z)$$ (1)

Considering only the second sum, rearranging the terms and using Lemma 1

$$\sum_{z \in SG} \pi^\sigma(z)u_2(z) = \sum_{I \in \mathcal{I}^R_2} u_2^\sigma(I) = \sum_{I \in \mathcal{I}^R_2} \pi^\sigma(I)\hat{u}_2^{(\sigma_1, \sigma_2 \leftarrow F)}(I)$$

A best response must have no less utility than $\hat{\sigma}_2^F$, and we can then apply Lemma 2

$$\leq \sum_{I} \pi^\sigma_2(I)\hat{u}_2^{(\sigma_1, BR(\sigma_1))}(I) \leq (|\mathcal{I}^R_2| - 1)\epsilon_S + \epsilon_R + \sum_{I} \pi^\sigma_2(I)\hat{u}_2^{(\sigma_1, BR(\sigma_1))}(I)$$

Because $\hat{u}_2^{\sigma_2}(I) = v_2^\sigma(I)$ and $\hat{u}_2(I \cdot T) = v_2^\sigma(I)$ for all $I$, $BR(\sigma_1^F)$ can always pick action $F$, and we can directly use $BR(\sigma_1^F)$ in the real game, with the same counterfactual value.

$$= (|\mathcal{I}^R_2| - 1)\epsilon_S + \epsilon_R + \sum_{I} \pi^\sigma_2(I)u_2^{(\sigma_1, BR(\sigma_1))}(I)$$

Putting this back into line 1 and noting that a best response can only increase the utility, we get

$$u_2^\sigma = (|\mathcal{I}^R_2| - 1)\epsilon_S + \epsilon_R + u_2^{(\sigma_1, \hat{\sigma}[SG \leftarrow BR(\sigma)])} \leq (|\mathcal{I}^R_2| - 1)\epsilon_S + \epsilon_R + u_2^{(\sigma_1, BR(\sigma_1))}$$

□

References

[1] Darse Billings, Neil Burch, Aaaron Davidson, Robert Holte, Jonathan Schaeffer, Terence Schauenberg, and Duane Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI), pages 661–668, 2003.
[2] Sam Ganzfried and Tuomas Sandholm. Improving performance in imperfect-information games with large state and action spaces by solving endgames. In *Proceedings of the Computer Poker and Imperfect Information Workshop at AAAI 2013*, 2013.

[3] Sam Ganzfried, Tuomas Sandholm, and Kevin Waugh. Strategy purification. In *Applied Adversarial Reasoning and Risk Modeling*, 2011.

[4] Andrew Gilpin, Javier Peña, and Tuomas Sandholm. First-order algorithm with $O(\ln(1/\epsilon))$ convergence for $\epsilon$-equilibrium in two-person zero-sum games. *Math. Program.*, 133(1-2):279–298, 2012.

[5] Andrew Gilpin and Toumas Sandholm. A competitive texas hold’em poker player via automated abstraction and real-time equilibrium computation. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2006.

[6] Michael Johanson. Measuring the size of large no-limit poker games. Technical Report TR13-01, Department of Computing Science, University of Alberta, 2013.

[7] Michael Johanson, Nolan Bard, Neil Burch, and Michael Bowling. Finding optimal abstract strategies in extensive-form games. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, AAAI ’12, 2012.

[8] Michael Johanson, Nolan Bard, Marc Lanctot, Richard Gibson, and Michael Bowling. Efficient nash equilibrium approximation through monte carlo counterfactual regret minimization. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS ’12, pages 837–846, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems.

[9] 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 Multiagent Systems (AAMAS-13)*, 2013.

[10] Daphne Koller, Nimrod Megiddo, and Bernhard Von Stengel. Efficient computation of equilibria for extensive two-person games. *Games and Economic Behavior*, 14:247–259, 1996.

[11] Richard E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

[12] J. Schaeffer, N. Burch, Y. Björnsson, A. Kishimoto, M. Müller, R. Lake, P. Lu, and S. Sutphen. Checkers is solved. *Science*, 317(5844):1518, 2007.

[13] Kevin Waugh, Nolan Bard, and Michael Bowling. Strategy grafting in extensive games. In *Advances in Neural Information Processing Systems 22 (NIPS-09)*, 2009.
[14] Kevin Waugh, Dave Schnizlein, Michael Bowling, and Duane Szafron. Abstraction pathologies in extensive games. In Proceedings of the Eighth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), pages 781–788, 2009.

[15] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In Advances in Neural Information Processing Systems 20 (NIPS), pages 905–912, 2008.