Endgame Solving in Large Imperfect-Information Games*

Sam Ganzfried and Tuomas Sandholm
Computer Science Department
Carnegie Mellon University
{sganzfri, sandholm}@cs.cmu.edu

ABSTRACT

The leading approach for computing strong game-theoretic strategies in large imperfect-information games is to first solve an abstracted version of the game offline, then perform a table lookup during game play. We consider a modification to this approach where we solve the portion of the game that we have actually reached in real time to a greater degree of accuracy than in the initial computation. We call this approach endgame solving. Theoretically, we show that endgame solving can produce highly exploitable strategies in some games; however, we show that it can guarantee a low exploitability in certain games where the opponent is given sufficient exploitative power within the endgame. Furthermore, despite the lack of a general worst-case guarantee, we describe many benefits of endgame solving. We present an efficient algorithm for performing endgame solving in large imperfect-information games, and present a new variance-reduction technique for evaluating the performance of an agent that uses endgame solving. Experiments on no-limit Texas Hold'em show that our algorithm leads to significantly stronger performance against the strongest agents from the 2013 AAAI Annual Computer Poker Competition.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent Systems; J.4 [**Social and Behavioral Sciences**]: Economics

General Terms

Algorithms, Economics, Theory

Keywords

Game Theory; Game Solving; Imperfect Information

1. INTRODUCTION

Sequential games of perfect information can be solved in linear time by a straightforward backward induction procedure in which solutions to endgames are propagated up the game tree.[1] However, this procedure does not work in general in imperfect-information games because different endgames can contain nodes that belong to the same information set and cannot be treated independently. More sophisticated algorithms are needed for this class of game. One algorithm for solving two-player zero-sum imperfect-information games is based on a linear program (LP) formulation [14], which scales to games with around 10^8 nodes in their game tree [7]. Many interesting games are significantly larger; for example, two-player limit Texas Hold'em has about 10^{17} nodes, and a popular variant of two-player no-limit Texas Hold'em has about 10^{165} nodes [12]. To address such large games, newer approximate equilibrium-finding algorithms have been developed that scale to games with around 10^{14} nodes, such as counterfactual regret minimization (CFR) [21] and an algorithm based on the excessive gap technique (EGT) [10]. These algorithms are iterative and guarantee convergence to equilibrium in the limit.

The leading approach for solving extremely large games such as Texas Hold'em (TH)[2] is to abstract the game down to a game with only around 10^{12} nodes, then to compute an approximate equilibrium in the abstract game using one of the algorithms described above [2, 7]. In order to perform such a dramatic reduction in size, significant abstraction is often needed. *Information* (aka *card*) *abstraction* involves reducing the number of nodes by bundling signals (e.g., forcing a player to play the same way with two different hands), and *action* (aka *betting*) *abstraction* involves reducing the number of actions by discretizing large action spaces into a small number of actions. All of the computation (both for constructing the abstraction and computing an approximate equilibrium in the abstraction) is done offline, and a table lookup is performed in real time to implement the strategy.

We consider a modification to this approach where we retain the abstract equilibrium strategies for the initial portion of the game tree (called the *trunk*), and discard the strategies for the final portion (called the *endgames*). Then, in real time, we solve the relevant endgame that we have reached to a greater degree of accuracy than the initial abstract strategy, where we use Bayes' rule to compute the distribution of players' private information leading into the endgames from the precomputed trunk strategies. This approach, which we call *endgame solving*, is depicted in Figure 1.

We present the first theoretical analysis of endgame solv-

*This material is based on work supported by the National Science Foundation under grants IIS-1320620, IIS-0964579, and CCF-1101668, as well as XSEDE computing resources provided by the Pittsburgh Supercomputing Center.

Appears in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Bordini, Elkind, Weiss, Yolum (eds.), May 4–8, 2015, Istanbul, Turkey.*

[1]Prior work has shown that precomputing solutions to endgames offline can be effective in large perfect-information games [1, 18]. In contrast, we solve endgames online.
[2]See Appendix A for background on Texas Hold'em poker.

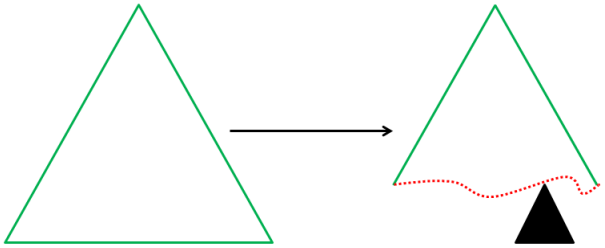

Figure 1: Endgame solving (re-)solves the relevant endgame that we have actually reached in real time to a greater degree of accuracy than in the offline computation.

ing in imperfect-information games, and show that it can actually produce highly exploitable strategies in some games. In fact, we show that it can fail even in a simple game with a unique equilibrium and a single endgame, even if our base strategy were an exact equilibrium (of the full game) and we were able to compute an exact equilibrium in the endgame. However, we show that endgame solving can guarantee a low *exploitability* (difference between game value and payoff against a nemesis) in some games when the opponent is given sufficient exploitative power within the endgame.

Endgame solving has been used by several prior agents for the limit variation of TH (where bets must be of a single fixed size). The agent GS1 precomputed strategies only for the first two rounds, using rough approximations for the payoffs at the leaves of that trunk based on the (unrealistic) assumption that there was no betting in future rounds [7]. Then in real time, the relevant endgame consisting of the final two rounds was solved using the LP algorithm. GS2 precomputed strategies for the first three rounds, using simulations to estimate the payoffs at the leaves; it then solved the endgames for the final two rounds in real time [8].

However endgame solving has not been implemented by any competitive agents for the significantly larger and more challenging domain of no-limit Texas Hold'em (NLTH) prior to our work. We present a new algorithm that is capable of scaling to extremely large games such as no-limit Texas Hold'em, and incorporates several algorithmic improvements over the prior approaches (the benefits described in this paragraph would be improvements over the prior approaches even for the limit variant). First, the prior approaches assume that the private hand distributions leading into the endgame are independent, while they are actually dependent and the full joint distribution should be computed. The naïve way of accomplishing this would require $O(n^2)$ strategy table lookups, where n is the number of private hands (1081 for the final round of poker), and computing these distributions would become the bottleneck of the algorithm and make the real-time computation intractable; however, we developed a technique for computing the joint distributions that requires just $O(n)$ strategy table lookups. Second, the prior approaches use a single perfect-recall card abstraction that has been precomputed offline (which assumes a uniform random distribution for the opponent's hand distributions). In contrast, we use an imperfect-recall card abstraction[3] that is computed in real time in a finer granularity than the initial offline abstraction and that is tailored specifically to the relevant distribution of the opponent's hands at the given hand history. Furthermore, the prior approaches did not compare performance between endgame solving and not using it (since the base strategies were not computed for the endgames), while we provide such a comparison.

Very recent work, which appeared subsequently to the first version of this work, has presented approaches for decomposing imperfect-information games into smaller games that can be solved independently offline, and provides some theoretical guarantees on full-game exploitability. One of these approaches has only been applied to the small domain of limit Leduc Hold'em, which has 936 information sets in its game tree, and is not practical for larger games such as NLTH due to its running time [3]. A second related (offline) approach includes counterfactual values for game states that could have been reached off the path to the endgames [11]. This approach has been demonstrated to be effective in limit Leduc Hold'em, and has also been implemented in NLTH, though no experimental results are given for that domain. For NLTH, it is implemented by first solving the game in a coarse abstraction, then fixing the strategies for the preflop (first) round, and re-solving for certain endgames starting at the flop (second round) after common preflop betting sequences have been played. All of this computation is done offline. In contrast, our approach enables us to solve endgames at the river (final round) in real time. It is infeasible to solve the river endgames using the prior approach for several reasons. First, there are far too many of them to be solved individually in advance (there is a different one for each sequence of public cards and betting actions). Second, by the time play gets down to the river, there are many possible alternative actions that a player could have taken to avoid reaching the given endgame, and counterfactual values for each of these would need to be computed and then included in the solution to the endgame solver; this would likely be infeasible to do in real time. Solving the river endgames, as opposed to the flop endgames which the prior approach does, is very important because CFR only occasionally samples from a specific river endgame during the course of the initial equilibrium computation, while it very frequently samples from the flop endgames that follow common preflop betting sequences. So, our approach is addressing a more pressing limitation.

Our approach has significant benefits over the standard approach for solving large imperfect-information games, including computation of exact (rather than approximate) equilibrium strategies (within a given abstraction), the ability to compute certain equilibrium refinements that cannot be computed in the full offline computation, finer-grained abstraction in the endgames, abstraction that takes into account realistic distributions of players' private information entering the endgame (as opposed to the typical assumption of uniform random distributions), and a solution to the "off-tree" problem that arises when the opponent has taken actions that are not allowed in the abstraction. We present an efficient algorithm for performing endgame solving in large imperfect-information games, and present a novel variance-reduction technique for evaluating the performance of an agent that uses endgame solving. Experiments on no-limit Texas Hold'em show that using our algorithm leads to a significantly stronger performance against the strongest 2013 poker competition agents.

[3]Imperfect-recall abstractions allow for greater flexibility in which hands can be grouped together, and significantly improve performance over perfect-recall abstractions [20, 13].

2. ENDGAME SOLVING

DEFINITION 1. *E is an* endgame *of game G if the following properties hold:*

1. *The set of E's nodes is a subset of the set of G's nodes.*
2. *If s' is a child of s in G and s is a node in E, then s' is also a node in E.*
3. *If s is in the same information set as s' in G and s is a node in E, then s' is also a node in E.*

For example, we can consider endgames in poker where several rounds of betting have taken place and several public cards have already been dealt. In these endgames, we can assume players have a joint distribution of private information from nodes prior to the endgame that are induced from the precomputed base approximate-equilibrium strategy using Bayes' rule. Given this distribution as input, we can then solve individual endgames in real time using more accurate abstractions.

Unfortunately, this approach has some fundamental theoretical shortcomings. It turns out that even if we computed an exact equilibrium in the trunk (which is an unrealistically optimistic assumption in large games) and in the endgame, the combined strategies for the trunk and endgame may fail to be an equilibrium in the full game. One obvious reason for this is that the game may contain many equilibria, and we might choose one for the trunk that does not match up correctly with the one for the endgame; or we may compute different equilibria in different endgames that do not balance appropriately. However, Proposition 1 shows that it is possible for this procedure to output a non-equilibrium strategy profile in the full game even if the full game has a unique equilibrium and a single endgame.

PROPOSITION 1. *There exist games—even with a unique equilibrium and a single endgame—for which endgame solving can produce a non-equilibrium strategy profile.*

PROOF. Consider a sequential version of Rock-Paper-Scissors where player 1 acts, then player 2 acts without observing player 1's action. This game has a single endgame—when it is player 2's turn to act—and a unique equilibrium—where each player plays each action with probability $\frac{1}{3}$. Now suppose we restrict player 1 to follow the equilibrium in the initial portion of the game. Any strategy for player 2 is an equilibrium in the endgame, because each one yields her expected payoff 0. In particular, suppose our equilibrium solver outputs the pure strategy Rock for her. This is clearly not an equilibrium of the full game. □

Rock-Paper-Scissors (RPS) is somewhat of an extreme example though, because player 1 does not actually make any moves in the endgame. At the other extreme, if the endgame were the entire game, then endgame solving would produce an exact equilibrium. As a slightly less extreme example, consider the game in Figure 2, where P1 selects an action a_i, and then a sequential imperfect-information game G_i is played. Suppose we are solving endgames after P1's initial action. Then we will solve the endgame G_i and produce strategies with zero exploitability in the full game. Endgame solving could be very useful in this game for several reasons. First, if the number of initial actions n for P1 were extremely large, it may be infeasible to solve and/or store solutions to all of the endgames in advance of game play. Endgame solving would only require solving the endgames that are actually reached during game play, and would be feasible even if n is extremely large as long as the number of game repetitions were relatively small. And second, the typical approach would actually not even involve solving each of the G_i separately in advance; it would be to solve the full game, which includes each of the G_i as well as P1's initial actions. It is very possible that equilibrium-finding algorithms would not scale to the full game and/or it would not fit in memory, while equilibria could be computed quickly and fit into memory for the individual endgames G_i.

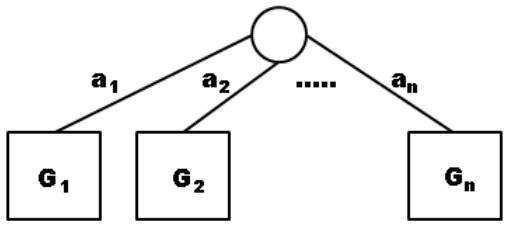

Figure 2: Player 1 selects his action a_i, then the players play imperfect-information game G_i.

One could imagine much more complex trunk games than the above example with imperfect information and multiple actions for both players where it is difficult to know for sure how "important" the trunk strategies are for the endgames. In such games, it may be possible for endgame solving to still guarantee a reasonably low exploitability in the full game. As Proposition 2 shows, in general, the more exploitative power the opponent has within the endgame, the lower the full-game exploitability of the strategies produced by (approximate) endgame solving are.

PROPOSITION 2. *If every strategy that has exploitability strictly more than ϵ in the full game has exploitability of strictly more than δ within the endgame, then the strategy output by a solver that computes a δ-equilibrium in the endgame induced by a trunk strategy t would constitute an ϵ-equilibrium of the full game when paired with t.*

PROOF. Suppose a strategy is a δ-equilibrium in the endgame induced by t, but not an ϵ-equilibrium in the full game when paired with t. Then by assumption, it has exploitability of strictly more than δ within the endgame, which leads to a contradiction. □

Intuitively, Proposition 2 says that endgame solving produces strategies with low exploitability in games where the endgame is a significant strategic portion of the full game, that is, in games where any endgame strategy with high full-game exploitability can be exploited by the opponent by modifying his strategy just within the endgame.

One could classify different games according to how they fall regarding the premise of Proposition 2, given a subdivision of the game into a trunk and endgames, and given fixed strategies for the trunk. If the premise is satisfied, then we can say that the game/subdivision satisfies the (ϵ, δ)-*endgame property*. An interesting property would be the smallest value $\epsilon^*(\delta)$ such that the game satisfies the (ϵ, δ)-endgame property for a given δ. For instance, the game in Figure 2 would have $\epsilon^*(\delta) = \delta$ for all $\delta \geq 0$, while RPS would only have $\epsilon^*(\delta) = 1$ for each $\delta \geq 0$. While Proposition 2 is admittedly somewhat trivial, such a classification could be useful in developing a better understanding of when endgame solving would be helpful in general.

3. BENEFITS OF ENDGAME SOLVING

Even though we showed in the previous section that endgame solving may lead to highly exploitable strategies in some games, it has many clear benefits in large imperfect-information games, which we now describe. These benefits and techniques are *enabled by* using endgame solving (rather than being techniques that help alongside endgame solving).

3.1 Exact Computation of Nash Equilibrium in Abstracted Endgames

The best algorithms for computing approximate equilibria in large games of imperfect information scale to about 10^{14} nodes. However, they are iterative and guarantee convergence only in the limit; in practice they only produce approximations of equilibrium strategies (within a given abstraction). Sometimes the approximation error is quite large. For example, a recent NLTH agent reported having an exploitability of 800 milli big blinds per hand (mbb/h) even within the abstract game [4] (an agent that folds every hand would only have an exploitability of 750 mbb/h). The best general-purpose LP algorithms find an exact equilibrium, though they only scale to games with around 10^8 nodes [7]. While the LP algorithms do not scale to reasonable abstractions of full TH, we can (and do) use them to exactly solve abstracted endgames that have up to around 10^8 nodes.

3.2 Ability to Compute Certain Equilibrium Refinements

The Nash equilibrium (NE) solution concept has some theoretical limitations, and several equilibrium refinements have been proposed which rule out NEs that are not rational in various senses. In general, these solution concepts guarantee that we behave sensibly against an opponent who does not follow his prescribed equilibrium strategy (i.e., he takes actions that should be taken with probability zero in equilibrium). Specialized algorithms have been developed for computing many of these concepts [15, 16, 17]. However, those algorithms do not scale to large games. In TH, computing a reasonable approximation of a single Nash equilibrium already takes months (using the leading algorithms, CFR or EGT), and there are no known algorithms for computing any of the common refinements that scale to games of that size. However, when solving endgames that are significantly smaller than the full game, it can be possible to compute certain refinements. An undominated Nash equilibrium (UNE) can be computed by solving two LPs instead of one and an ϵ-quasi-perfect-equilibrium by solving a single LP (though the second one is not technically a refinement and has documented numerical stability issues). We have implemented algorithms for computing both of these on large NLTH endgames, which demonstrates for the first time that they are feasible to compute in imperfect-information games of this magnitude. Preliminary experiments indicate that in NLTH endgames UNE is useful, though those results were not statistically significant, so we do not report on those experiments here.

3.3 Finer-Grained, History-Aware, and Strategy-Biased Abstraction

Another important benefit of endgame solving in large games is that we can compute better abstractions in the endgame that is actually played than if we are forced to abstract the entire game at once in advance. In addition to allowing us to compute finer-grained abstractions, endgame solving enables us to compute an abstraction specifically for the situation at hand. In other words, we can condition the abstraction on the path of play so far (both the players' actions and nature's actions). For example, in poker, we can condition the abstraction on the betting history (which offline game-solving approaches do not do) and on the board cards (which offline game-solving approaches cannot afford to do at an equally fine granularity).

The standard approach for performing information abstraction is to bucket information sets together for hands that perform similarly against a uniform distribution of the opponent's private information [7, 13].[4] However, the assumption that the opponent has a hand uniformly at random is extremely unrealistic in many situations; for example, if the opponent has called large bets throughout the hand, he is unlikely to hold a very weak hand. Ideally, a successful information abstraction algorithm would group hands together that perform similarly against the relevant distribution of hands the opponent actually has—not a naïve uniform random distribution. Fortunately, we can accomplish such *strategy-biased information abstraction* in endgames. Our algorithm is detailed in Section 4.

3.4 A Solution to the Off-Tree Problem

When we perform action abstraction, the opponent may take an action that falls outside of our action model for him. When this happens, an *action translation mapping* (aka *reverse mapping*) is necessary to interpret his action by mapping it to an action in our model [5, 19]. However, this mapping may ignore relevant game state information. In poker, action translation works by mapping a bet of the opponent to a 'nearby' bet size in our abstraction; however, it does not account for the size of the pot or remaining stacks. For example, suppose remaining stacks are 17,500, the pot is 5,000, and our abstraction allows for bets of size 5,000 and 17,500. Suppose the opponent bets 10,000, which we map to 5,000 (if we use a randomized mapping, we will do this with some probability). So we map his action to 5,000, and simply play as if he had bet 5,000. If we call his bet, we will think the pot has 15,000 and stacks are 12,500. However, in reality the pot has 25,000 and stacks are 7,500. These two situations are completely different and should be played very differently (for example, we should be more reluctant to bluff in the latter case because the opponent will be getting much better odds to call). This is known as the *off-tree problem*. Even if one is using a very sophisticated translation algorithm, one will run into the off-tree problem.

When performing endgame solving in real time, we can solve the off-tree problem completely. Regardless of the action translation used to interpret the opponent's actions prior to the endgame, we can take the stack and pot sizes (or any other relevant game state information) as inputs to the endgame solver. Our endgame solver in poker takes the current pot size, stack sizes, and prior distributions of the cards of both players as inputs. Therefore, even if we mapped the opponent's action to 5,000 in the above example, we correct the pot size to 25,000 (and the stack sizes accordingly) before solving the endgame.

[4]Recent work has also considered an approach where the opponent's preflop hands are first grouped into several buckets, then hands for the later rounds are grouped together if they perform similarly against each of the preflop buckets [13].

4. ENDGAME SOLVING ALGORITHM

In this section we present our algorithm for endgame solving in imperfect-information games with very large state and action spaces. Pseudocode is given in Algorithm 1. The core algorithm is domain independent, although we present the signals as card-playing hands for concreteness. An example poker hand illustrating each step of the algorithm is given in Appendix B.

Algorithm 1 Algorithm for endgame solving

Inputs: number of information buckets per agent k_i; abstraction parameter T; action abstractions B_i with b_i action sequences; clustering algorithms C_i; equilibrium-finding algorithm Q; number of private hands H; hand rankings $R[]$

Compute joint hand-strength distribution $D[i][j]$
$E_1, E_2 \leftarrow$ array of dimension H of zeroes
for $h_1 = 1$ to H **do**
  $r_1 \leftarrow R[h_1]$, $s_1, s_2 \leftarrow 0$
  for $h_2 = 1$ to H **do**
    $r_2 \leftarrow R[h_2]$, s_1 += $D[h_1][h_2]$, s_2 += $D[h_2][h_1]$
    if $r_2 < r_1$ **then**
      $E_1[h_1]$ += $D[h_1][h_2]$, $E_2[h_1]$ += $D[h_2][h_1]$
    else if $r_1 == r_2$ **then**
      $E_1[h_1]$ += $\frac{D[h_1][h_2]}{2}$, $E_2[h_1]$ += $\frac{D[h_2][h_1]}{2}$
  $E_1[h_1] = \frac{E_1[h_1]}{s_1}$, $E_2[h_1] = \frac{E_2[h_1]}{s_2}$
$k_i \leftarrow \lfloor \frac{T}{b_i} \rfloor$ for $i = 1, 2$
$A_i \leftarrow$ information abstraction created by clustering elements of E_i into k_i buckets using C_i for $i = 1, 2$
Solve game with information abstractions A_i and action abstractions B_i using Q

The first step is to compute the joint input distribution of private information using Bayes' rule. The naïve approach for doing this would require iterating over all possible private hand combinations h_1, h_2 for the players, and for each pair looking up the probability that the base agent would have taken the given action sequence. This requires $O(n^2)$ lookups to the strategy table, where n is the number of possible hands ($n = 1081$ for the final round in poker). It turns out that this computation would become the bottleneck of the entire endgame-solving algorithm and would make real-time endgame solving computationally infeasible. For this reason, prior approaches for endgame solving have made the (significantly) simplifying assumption that the distributions are independent [7, 8]. However, we developed an algorithm that does this with just $O(n)$ table lookups. Pseudocode for our algorithm is given in Algorithm 2.

In short, the algorithm first computes the distributions separately for each player (as done by the independent approach), then multiplies the probabilities together for hands that do not share a common card (and setting the joint probability to zero otherwise). In order to make sure hands are indexed properly in the array, we must make use of two helper indexing functions, Algorithms 3 and 4. The former gives an algorithm for indexing the two-card private hands, and the latter gives an algorithm for indexing the 7-card river hand consisting of the two private cards and five public cards. Then, in Algorithm 2, we iterate over all sets of private hands (p_1, p_2), and create an array called IndexMap that maps the 7-card hand index to the corresponding 2-card hand index. In the course of this loop, we also look up the probability that each player would play according to the observed betting history in the precomputed trunk strategies, which we then normalize in accordance with Bayes' rule.

Algorithm 2 Algorithm for computing hand distributions

Inputs: Public board B; number of possible private hands H; betting history of current hand h; array of index conflicts IC[][]; base strategy s^*

$D_1, D_2 \leftarrow$ array of dimension H of zeroes
for $p_1 = 0$ to 50, p_1 not already on B **do**
  for $p_2 = p_1 + 1$ to 51, p_2 not already on B **do**
    $I \leftarrow$ IndexFull(B, p_1, p_2)
    IndexMap$[I] \leftarrow$ IndexHoles(p_1, p_2)
    $P_1 \leftarrow$ probability P1 would play according to h with p_1, p_2 in s^*
    $P_2 \leftarrow$ probability P2 would play according to h with p_1, p_2 in s^*
    $D_1[I]$ += P_1, $D_2[I]$ += P_2
Normalize D_1 and D_2 so all entries sum to 1
for $i = 0$ to H **do**
  for $j = 0$ to H **do**
    if !IC[IndexMap[i]][IndexMap[j]] **then**
      $D[i][j] \leftarrow D_1[i] \cdot D_2[j]$
    else
      $D[i][j] \leftarrow 0$
Normalize D so all entries sum to 1 **return** D

Algorithm 3 Algorithm for computing private hand index

Inputs: Private hole cards h_1, h_2

if $h_2 < h_1$ **then** $t \leftarrow h_1$, $h_1 \leftarrow h_2$, $h_2 \leftarrow t$
return $\binom{h_2}{2} + \binom{h_1}{1}$

Algorithm 4 Algorithm for computing index of 7-card hands on a given board

Inputs: Private hole cards h_1, h_2, board B consisting of five public cards

if $h_2 < h_1$ **then**, $t \leftarrow h_1$, $h_1 \leftarrow h_2$, $h_2 \leftarrow t$
$n_1 \leftarrow 0, n_2 \leftarrow 0$
for $i = 1$ to 5 **do**
  for $j = 1$ to 2 **do**
    if $B[i] < h_j$ **then** $++n_j$
return $\binom{h_2 - n_2}{2} + \binom{h_1 - n_1}{1}$

In advance of applying Algorithm 2, we compute a table of the conflicts between each pair of private-hand indices, where we set IC$[i][j]$ to 1 if hand with indices i and j share a card in common, and 0 otherwise. Then, we set the joint probability $D[i][j]$ to equal the product of the two independent probabilities $D_1[i], D_2[j]$ if there is no constraint between the indices, and we set it to zero otherwise. Note that this algorithm actually runs in $O(n^2)$, where n is the number of private hands. However, the n^2 loop only involves the simple step of looking up an element in the IC array, which is performed extremely quickly. The time-consuming part of the computation is looking up the strategy probabilities P_1, P_2, which involves accessing several elements in the massive binary strategy file. Our algorithm performs this task only $O(n)$ times, while the naïve approach would do this $O(n^2)$ time, and make real-time endgame solving intractable. (Note that each private hand consists of the two

cards p_1, p_2, so while the main loop in Algorithm 2 iterates over both p_1 and p_2, it is only iterating once over the H private hands and is $O(n)$).

Next we compute arrays E_1, E_2 that contain the *equities* for each state of private information against the opponent's distribution. For player 1, we do this by adding $D[h_1][h_2]$ to $E_1[h_1]$ for each hand h_2 such that the rank of it on the given board is lower than that of h_1, and adding $\frac{D[h_1][h_2]}{2}$ for each hand with equal rank.[5] We then normalize the entries of $E_1[h_1]$, and compute E_2 analogously. $E_1[h_1]$ is now the probability that player 2 has a hand worse than h_1, given the prior distribution D and the current history of betting and public cards.

In advance of gameplay, we have computed separate action abstractions for the endgame solver to use for each pot/stack size that could be encountered. This allows us to solve the "off-tree problem," since we are taking into account the actual pot size even the opponent took an action outside the action abstraction earlier in the hand. We have constructed these abstractions so that the larger pot sizes (which have shallower stacks) have more bet sizes available for each history, for several reasons; the first is that the tree is smaller in these situations due to the shallower stack sizes (once players are "all-in," no additional bets are allowed), and the second is that hands with larger pot sizes are more important, since more money is won and lost on them, and we would like to ensure that more bet sizes are accounted for on these hands. B_i denotes the action abstraction to use for the given pot size at hand, and b_i denotes the number of betting sequences of B_i, for $i = 1, 2$.

Next, we compute a card abstraction A_i by grouping E_i into k_i buckets, using some clustering algorithm C_i, for $i = 1, 2$. Here $k_i = \frac{T}{b_i}$, where T is a parameter of the algorithm (for our agent we used $T = 7500$). While much prior work on poker has used k-means as the standard clustering algorithm, the following example demonstrates why this would be problematic. Suppose there are many hands with an equity of 0.775, and also many hands with an equity of 0.772. Then k-means would likely create separate clusters for these two equity values, and possibly group hands with very different equities (e.g., 0.2 and 0.3) together if few hands have those equities. To address this concern we used percentile hand strength, which also happens to be easier to compute. To do this, we break up the interval [0,1] into k_i regions of equal length (each of size $\frac{1}{k_i}$). We then group hand h_i into bucket $\lfloor \frac{E_i[h_i]}{k_i} \rfloor$. (For our poker agent we actually use a slight modification of this approach where we create a special bucket just for the hands with $E_i[h_i] \geq \alpha$, to ensure that the strongest hands are grouped separately (we used $\alpha = 0.99$ for our agent). Then the remaining α mass is divided according to the previously described procedure.) Sometimes this algorithm results in significantly fewer than k_i buckets, since there may be zero hands with E_i within certain intervals. We take this into account, and reduce the number of buckets in the card abstraction accordingly before solving the endgame. Note that the card abstractions A_i may be very different for the two players (and have different numbers of buckets); this differs from all the standard approaches, which use the same abstraction for all players.

[5]The rank of a hand given a set of public cards is an integral-valued mapping such that stronger hands have a higher value; for example, a royal flush has the highest rank.

Algorithm 5 Algorithm for computing endgame information abstractions

Inputs: Equity arrays E_i; desired number of buckets per agent k_i; parameter for top bucket α; total number of possible private hands H

$J \leftarrow \frac{\alpha}{k_1 - 1}$
$A_1 \leftarrow$ array of zeroes of size H
$U_1 \leftarrow$ array of booleans initialized to false of size H
for $h = 1$ to H **do**
  if $E_1[h] \geq \alpha$ **then** $b \leftarrow k_1 - 1$
  else
    $b \leftarrow \lfloor \frac{E_1[h]}{J} \rfloor$
  if $U_1[b] ==$ FALSE **then** $U_1[h] \leftarrow$ TRUE
$M_1 \leftarrow$ array of zeroes of size k_1, $g \leftarrow 0$
for $i = 0$ to k_i **do**
  $M_1[i] \leftarrow g$
  if $U_1[i] ==$ TRUE **then** $g = g + 1$
for $h = 1$ to H **do**
  if $E_1[h] \geq \alpha$ **then** $A_1[h] \leftarrow M_1[k_1 - 1]$
  else
    $A_1[h] \leftarrow M_1 \left[\lfloor \frac{E_1[h]}{J} \rfloor \right]$
Compute A_2 analogously

Finally, we compute an (exact) equilibrium in the abstracted endgame by applying an equilibrium-finding algorithm Q to the game with card abstractions A_i and betting abstractions B_i. While the card abstractions were computed independently (based on equities derived from the joint distribution), we use the joint distribution for determining the probabilities that players are dealt hands from their respective buckets when constructing the endgame. For our agent, we used Gurobi's parallel LP solver [9] as Q.

5. EXPERIMENTS ON NO-LIMIT TEXAS HOLD'EM

We tested our algorithm against the two strongest agents from the 2013 poker competition. The base agent was a version of the agent we submitted to the 2014 AAAI computer poker competition (that came in first place) from shortly before the competition. Ordinarily it would be very time consuming to differentiate the performance of the base strategies from the endgame solver with statistical significance, since the endgame solver plays relatively slowly (it averaged around 8 seconds per hand, which still kept us well within the competition time limit of 7 seconds per hand on average, since only around 25% of hands make it to the final betting round). A useful variance-reduction technique is to only consider hands where both agents make it to an endgame. In Appendix C we prove that this technique is unbiased. The results using this evaluation metric are given in Table 1, where the $\pm$ indicates 95% confidence intervals.

Hyperborean.iro	Slumbot
+87 $\pm$ 50	+29 $\pm$ 25

Table 1: Improvement by using endgame solving against the strongest agents from the 2013 poker competition over all hands where both agents made it to some endgame (i.e., to the river betting round). Units are milli big blinds per hand.

The base agent used a procedure called purification on all rounds (except for the first preflop action); this procedure selects the maximal probability action at each information set with probability 1 instead of randomizing according to the abstract equilibrium strategy (ties are broken uniformly at random) [6]. This parameter setting was shown to be the best in our thorough experiments in prior years, and we had used this as the standard setting when evaluating our base agent. The main motivation for purification is that it compensates for the failure of iterative equilibrium-finding algorithms to fully converge to equilibrium in the abstract game (a phenomenon that has been documented by prior agents, e.g., [4]). The endgame solving agent did not use any rounding for the river, as the endgame equilibria are exact (within the chosen abstraction), and the problem of the equilibrium-finding algorithm failing to converge is not present. Both agents used the pseudoHarmonic action translation mapping [5] for all rounds to interpret actions taken by the opponent that fall outside of the action abstraction.

The results are from 100 duplicate matches against Hyperborean and 155 duplicate matches against Slumbot. Since each match is 3,000 hands, this means we played 600,000 and 930,000 hands; out of these hands, both versions of our agent made it to the river round on 173,568 and 318,700 hands against the respective opponents. If we had used the standard duplicate approach for evaluating performance, we would not have been able to statistically differentiate the base agent from the endgame solver over this sample. However, we were able to obtain statistically significant results using our new evaluation approach.

6. CONCLUSION

We demonstrated that endgame solving can be successful in practice in large imperfect-information games despite the fact that the strategies it computes is not guaranteed to constitute an equilibrium in the full game (which we showed). We also showed that endgame solving guarantees a low exploitability in certain games, and presented a framework that can be used to evaluate its applicability more broadly. We described several benefits of endgame solving in large imperfect-information games, including exact computation of Nash equilibria in abstracted endgames, the ability to compute certain equilibrium refinements, the ability to compute finer-grained, history-aware, and strategy-biased abstractions in endgames, and a solution to the off-tree problem. We presented an efficient algorithm for performing endgame solving in very large imperfect-information games and showed that it led to a significantly stronger performance against the strongest no-limit Texas Hold'em agents from the 2013 computer poker competition, utilizing a new variance-reduction technique that we described.

This work opens many interesting avenues for future research. We showed that endgame solving can produce strategies with high exploitability in certain games, while it guarantees low exploitability in others. It would be interesting to study where different game classes fall on this spectrum. It is possible that for interesting classes of games—perhaps even classes that include variants of poker—endgame solving is guaranteed to produce strategies with low exploitability. We would also like to study various subdivisions of a game into a trunk and endgames, to experiment on additional game classes, to experiment with the refinements we described, and to develop improved variance-reduction techniques.

APPENDIX

A. NO-LIMIT TEXAS HOLD'EM POKER

No-limit Texas Hold'em is the most popular variant of poker among humans, and the two-player version is the game of most active research in the computer poker community currently. This game works as follows. Initially two players each have a *stack* of chips (worth $20,000 in the computer poker competition). One player, called the *small blind*, initially puts $50 worth of chips in the middle, while the other player, called the *big blind*, puts $100 worth of chips in the middle. The chips in the middle are known as the *pot*, and will go to the winner of the hand.

Next, there is an initial round of betting. The player whose turn it is can choose from three available options:

- *Fold:* Give up on the hand, surrendering the pot to the opponent.
- *Call:* Put in the minimum number of chips needed to match the number of chips put into the pot by the opponent. For example, if the opponent has put in $1000 and we have put in $400, a call would require putting in $600 more. A call of zero chips is also known as a *check*.
- *Bet:* Put in additional chips beyond what is needed to call. A bet can be of any size up to the number of chips a player has left in his stack. If the opponent has just bet, then our additional bet is also called a *raise*.

The initial round of betting ends if a player has folded, if there has been a bet and a call, or if both players have checked. If the round ends without a player folding, then three public cards are revealed face-up on the table (called the *flop*) and a second round of betting occurs. Then one more public card is dealt (called the *turn*) and a third round of betting, followed by a fifth public card (called the *river*) and a final round of betting. If a player ever folds, the other player wins all the chips in the pot. If the final betting round is completed without a player folding, then both players reveal their private cards, and the player with the best hand wins the pot (it is divided equally if there is a tie).

In the experiments, we will be solving endgames after the final public card is dealt but before the final round of betting. (Thus, the endgame contains no more chance events, and only publicly observable actions of both players remain.)

B. EXAMPLE

In this section we demonstrate the operation of our algorithm on an example hand of no-limit Texas Hold'em. Recall that blinds are $50 and $100 and that both players start with $20,000. In the example hand, we are in the small blind with 8dTh. We raise to $250, the opponent re-raises to $750, and we call (there is now $1500 in the pot). The flop is Jc6s2c. The opponent checks and we check. The turn is Kd. The opponent checks, we bet $375, and he calls (there is now $2250 in the pot). The river is Qc. Up until this point we have just played according to the precomputed base strategy; the endgame-solving algorithm begins now.

According to the pseudocode for Algorithm 1, the first step is to compute the joint prior hand distribution D from the base strategies, using Algorithm 2. This took 0.433 seconds. We then compute the equities E_i for each player, using Algorithm 1. This took 0.015 seconds.

The next step is to look at the betting abstraction that has been precomputed for this specific pot/stack size (pot size of $2250 and stack sizes of $18875). Note that for this particular hand all of the opponent's actions before the river fell inside of our betting abstraction; however, if they had not, and we were forced to use an action translation mapping to map his action to an action in our betting abstraction, we would be able to correct our misperception of the pot size at this point, by selecting the precomputed betting abstraction for the actual pot/stack size (as opposed to the size that assumed he played an action in our betting abstraction). This solves the "off-tree" problem, discussed in the paper.

The betting abstraction for a pot size of $2250 has 196 betting sequences for each player. For this hand we used a betting abstraction parameter of $T = 10000$ (while for the experiments described in the paper, we used $T = 7500$). Therefore, we will use $k_i = \lfloor \frac{10000}{196} \rfloor = 51$ card buckets for each player for this hand.

Next, we compute card abstractions for both players We used used a top bucket parameter of $\alpha = 0.995$ (while for the experiments described in the paper, we used $\alpha = 0.99$). After applying our card abstraction algorithm for both players, the resulting abstractions had 38 and 35 buckets respectively for the two players (since not all of the 51 hand equity intervals contained hands). Computing these took 0.008 seconds.

Our actual hand (8dTh) had rank 296 (out of 1081) and actually had an equity of 0 vs. the opponent's hand distribution (we thought the opponent would never play the hand the way he did so far with a worse hand than 8dTh). This places us in bucket 0 (the worst bucket, out of 35). By contrast, if the opponent had our hand, he would have an equity of 0.336 against our hand distribution, and would be in bucket 8 (where his buckets range from 0–37).

We then construct the LP matrices for the resulting abstracted endgame, which took 0.15 seconds, and then compute an exact equilibrium by solving the LP using Gurobi's parallel LP solver (it took 1.051 seconds to construct the LP instance and 5.328 seconds to solve it). Overall, the endgame-solving algorithm took 6.985 seconds for this hand.

The opponent checked for his initial action on the river. The betting abstraction for this hand had nine available options for the first action for each player: check, 0.1 pot, $\frac{1}{3}$ pot, $\frac{2}{3}$ pot, pot, 1.5 pot, 2 pot, 3 pot, all-in. The strategy from our endgame solver said for us to check with probability 0.742, bet $\frac{2}{3}$ pot with probability 0.140, bet pot with probability 0.103, and bet 2 pot with probability 0.014.

C. VARIANCE-REDUCTION TECHNIQUE

When comparing the performance of one version of an agent A_1 to another version that is identical except that it plays differently on endgames A_2, one would like to take advantage of the fact that the agents play identically up until the endgames in order to evaluate the performance difference more efficiently. Ideally, we could play A_1 against a given opponent, and when the endgame is reached, evaluate how both A_1 and A_2 would do on that same endgame given the trunk history. However, such a technique is not possible on the poker competition test server. All that is allowed is to play A_1 and A_2 against an opponent for a full set of matches. The agents may reach endgames on different hands, or may reach different endgames even on the same hands (since both our agent and the opponent may be playing randomized strategies before the endgames).

One possible approach for reducing variance would be to only consider hands where both A_1 and A_2 arrive at the same endgame (the same betting history was played). It turns out that this approach is actually biased, so it cannot be applied to accurately measure performance. A second approach, that it turns out is unbiased, would be to only consider the hands where both agents arrive at *some* endgame (though not necessarily the same one). If we only consider these hands, then the difference in performance between the two agents is an unbiased estimator of their true performance difference. This would allow us to achieve statistical significance using a smaller sample of hands.

PROPOSITION 3. *Let A_1 and A_2 be two algorithms that differ in play only for endgames. Then the difference in performance looking at only the hands where both make it to the same endgame is not an unbiased estimator of the overall performance difference.*

PROOF. Suppose there were only two betting sequences and both make it to the river, where the first one (A) happens 99% of the time and the second one (B) happens 1% of the time. Then the probability that both hands hit the river with B on any particular hand is 0.01%, and the probability that both hands hit the river with A with any particular hand is 98.01%. So if you look at all hands where both hit the river with the same sequence, there would be only 1 (B) for every 9802 (A) sequences. □

PROPOSITION 4. *Let A_1 and A_2 be two algorithms that differ in play only for endgames. Then the difference in performance looking at only the hands where both make it to some (but not necessarily the same) endgame is an unbiased estimator of the overall performance difference.*

PROOF. For each history that leads into an endgame h_i, let p_i be the probability that h_i is played when we use the base strategy against the opponent O. Then the expectation of the difference in payoff between playing A_1 (base strategy) and A_2 (endgame solver) against O is

$$\begin{aligned} & \sum_i \left[p_i \left(U(A_1, O, h_i) - U(A_2, O, h_i)\right)\right] \\ = & \sum_i \left[p_i U(A_1, O, h_i)\right] - \sum_i \left[p_i U(A_2, O, h_i)\right] \end{aligned}$$

Suppose that we look at performance over all hands where both algorithms make it to some endgame. The probability that A_1 makes it to the endgame with history h_i and A_2 makes it to the endgame with history h_j is $p_i p_j$. Thus, the expectation of the payoff difference is

$$\begin{aligned} & \sum_i \sum_j \left[p_i p_j \left(U(A_1, O, h_i) - U(A_2, O, h_j)\right)\right] \\ = & \sum_i \sum_j \left[p_i p_j U(A_1, O, h_i)\right] - \sum_i \sum_j \left[p_i p_j U(A_2, O, h_j)\right] \\ = & \sum_i \left[p_i U(A_1, O, h_i) \sum_j p_j\right] - \sum_j \left[p_j U(A_2, O, h_j) \sum_i p_i\right] \\ = & \sum_i \left[p_i U(A_1, O, h_i)\right] - \sum_j \left[p_j U(A_2, O, h_j)\right] \\ = & \sum_i \left[p_i U(A_1, O, h_i)\right] - \sum_i \left[p_i U(A_2, O, h_i)\right] \end{aligned}$$

□

REFERENCES

[1] R. E. Bellman. On the application of dynamic programming to the determination of optimal play in chess and checkers. *National Academy of Sciences of the United States of America*, 53:244–247, 1965.

[2] D. Billings, N. Burch, A. Davidson, R. Holte, J. Schaeffer, T. Schauenberg, and D. Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[3] N. Burch, M. Johanson, and M. Bowling. Solving imperfect information games using decomposition. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2014.

[4] S. Ganzfried and T. Sandholm. Tartanian5: A heads-up no-limit Texas Hold'em poker-playing program. In *Computer Poker Symposium at the National Conference on Artificial Intelligence (AAAI)*, 2012.

[5] S. Ganzfried and T. Sandholm. Action translation in extensive-form games with large action spaces: Axioms, paradoxes, and the pseudo-harmonic mapping. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

[6] S. Ganzfried, T. Sandholm, and K. Waugh. Strategy purification and thresholding: Effective non-equilibrium approaches for playing large games. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2012.

[7] A. Gilpin and T. 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.

[8] A. Gilpin and T. Sandholm. Better automated abstraction techniques for imperfect information games, with application to Texas Hold'em poker. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2007.

[9] I. Gurobi Optimization. Gurobi optimizer reference manual, 2014.

[10] S. Hoda, A. Gilpin, J. Peña, and T. Sandholm. Smoothing techniques for computing Nash equilibria of sequential games. *Mathematics of Operations Research*, 35(2):494–512, 2010.

[11] E. Jackson. A time and space efficient algorithm for approximately solving large imperfect information games. In *AAAI Workshop on Computer Poker and Incomplete Information*, 2014.

[12] M. Johanson. Measuring the size of large no-limit poker games. Technical report, University of Alberta, 2013.

[13] M. Johanson, N. Burch, R. Valenzano, and M. Bowling. Evaluating state-space abstractions in extensive-form games. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2013.

[14] D. Koller, N. Megiddo, and B. von Stengel. Fast algorithms for finding randomized strategies in game trees. In *Proceedings of the 26th ACM Symposium on Theory of Computing (STOC)*, pages 750–760, 1994.

[15] P. B. Miltersen and T. B. Sørensen. Computing proper equilibria of zero-sum games. In *Computers and Games*, pages 200–211, 2006.

[16] P. B. Miltersen and T. B. Sørensen. Fast algorithms for finding proper strategies in game ttrees. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 874–883, 2008.

[17] P. B. Miltersen and T. B. Sørensen. Computing a quasi-perfect equilibrium of a two-player game. *Economic Theory*, 42(1):175–192, 2010.

[18] J. Schaeffer, Y. Björnsson, N. Burch, R. Lake, P. Lu, and S. Sutphen. Building the checkers 10-piece endgame databases. In *Advances in Computer Games 10*, 2003.

[19] D. Schnizlein, M. Bowling, and D. Szafron. Probabilistic state translation in extensive games with large action sets. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, 2009.

[20] K. Waugh, M. Zinkevich, M. Johanson, M. Kan, D. Schnizlein, and M. Bowling. A practical use of imperfect recall. In *Proceedings of the Symposium on Abstraction, Reformulation and Approximation (SARA)*, 2009.

[21] M. Zinkevich, M. Bowling, M. Johanson, and C. Piccione. Regret minimization in games with incomplete information. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, 2007.