The State of Solving Large Incomplete-Information Games, and Application to Poker

Tuomas Sandholm

■ *Game-theoretic solution concepts prescribe how rational parties should act, but to become operational the concepts need to be accompanied by algorithms. I will review the state of solving incomplete-information games. They encompass many practical problems such as auctions, negotiations, and security applications. I will discuss them in the context of how they have transformed computer poker. In short, game-theoretic reasoning now scales to many large problems, outperforms the alternatives on those problems, and in some games beats the best humans.*

Game-theoretic solution concepts prescribe how rational parties should act in multiagent settings. This is nontrivial because an agent's utility-maximizing strategy generally depends on the other agents' strategies. The most famous solution concept for this is a Nash equilibrium: a strategy profile (one strategy for each agent) where no agent has incentive to deviate from her strategy given that others do not deviate from theirs.

In this article I will focus on incomplete-information games, that is, games where the agents do not entirely know the state of the game at all times. The usual way to model them is a game tree where the nodes (that is, states) are further grouped into *information sets*. In an information set, the player whose turn it is to move cannot distinguish between the states in the information set, but knows that the actual state is one of them. Incomplete-information games encompass most games of practical importance, including most negotiations, auctions, and many applications in information security and physical battle.

Such games are strategically challenging. A player has to reason about what others' actions signal about their knowledge. Conversely, the player has to be careful about not signaling too much about her own knowledge to others through her actions. Such games cannot be solved using methods for complete-information games like checkers, chess, or Go. Instead, I will review new game-independent algorithms for solving them.

Poker has emerged as a standard benchmark in this space (Shi and Littman 2002; Billings et al. 2002) for a number of reasons, because (1) it exhibits the richness of reasoning about a probabilistic future, how to interpret others' actions as signals, and information hiding through careful action selection, (2) the game is unambiguously specified, (3) the game can be scaled to the desired complexity, (4) humans of a broad range of skill exist for comparison, (5) the game is fun, and (6) computers find interesting strategies automatically. For example, time-tested behaviors such as *bluffing* and *slow play* arise from the game-theoretic algorithms automatically rather than having to be explicitly programmed.

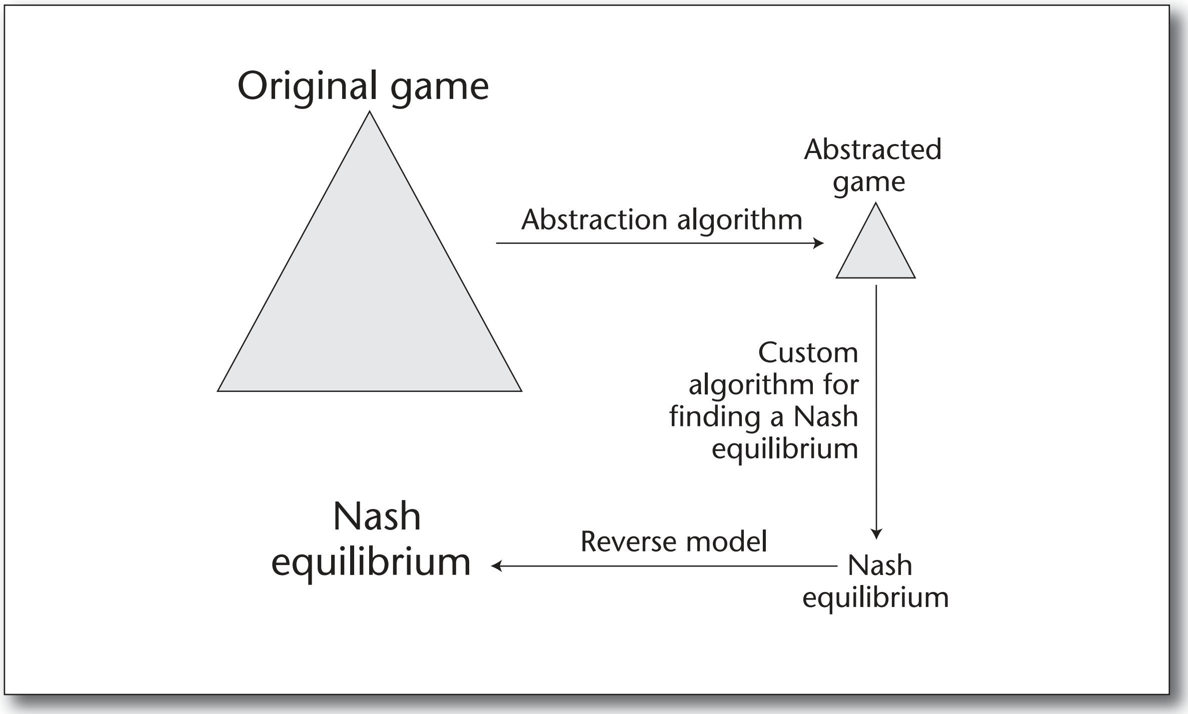

Figure 1. Current Paradigm for Solving Large Incomplete-Information Games.

Kuhn poker — a game with three cards — was among the first applications discussed in game theory, and it was solved analytically by hand (Kuhn 1950). On large-scale poker games, the best computerized strategies for a long time were rule based. Nowadays, the best poker-playing programs are generated automatically using algorithms that are based on game-theoretic principles.

There has been tremendous progress on equilibrium-finding algorithms since 2005. Two-player zero-sum game trees with 10^{12} leaves can now be solved near optimally. However, many real games are even larger. For example, two-player Limit Texas Hold'em poker has 10^{18} leaves. For such large games, abstraction algorithms have emerged as practical preprocessors.

Most competitive poker-playing programs are nowadays generated using an abstraction algorithm followed by using a custom equilibrium-finding algorithm to solve the abstracted game. See figure 1. This paradigm was first used in Gilpin, Sandholm, and Sørensen (2007). Predecessors of the paradigm included handcrafting small abstractions (Billings et al. 2003), as well as solving automatically generated abstractions with general-purpose linear programming algorithms (Gilpin and Sandholm 2006; 2007a; 2007b).

In this article I will discuss abstraction algorithms first and equilibrium-finding algorithms second. After that I will address opponent exploitation and other topics.

Abstraction Algorithms

Abstraction algorithms take as input a description of the game and output a smaller but strategically similar — or even equivalent — game. The abstraction algorithms discussed here work with any finite number of players and do not assume a zero-sum game.

Information Abstraction

The most popular kind of abstraction is information abstraction. The game is abstracted so that the agents cannot distinguish some of the states that they can distinguish in the actual game. For example in an abstracted poker hand, an agent is not able to observe all the nuances of the cards that she would normally observe.

Lossless Information Abstraction. It turns out that it is possible to do lossless information abstraction, which may seem like an oxymoron at first. The method I will describe (Gilpin and Sandholm 2007b) is for a class of games that we call *games with ordered signals*. It is structured, but still general enough to capture a wide range of strategic situations. A game with ordered signals consists of a finite number of rounds. Within a round, the players play a game on a directed tree (the tree can be different in different rounds). The only uncertainty players face stems from private signals the other players have received and from the unknown future signals. In other words, players observe each

Any Nash equilibrium of the shrunken game corresponds to a Nash equilibrium of the original game.

Theorem 1.

(Gilpin and Sandholm 2007b.)

others' actions, but potentially not nature's actions. In each round, there can be public signals (announced to all players) and private signals (confidentially communicated to individual players). We assume that the legal actions that a player has are independent of the signals received. For example, in poker, the legal betting actions are independent of the cards received. Finally, the strongest assumption is that there is a total ordering among complete sets of signals, and the payoffs are increasing (not necessarily strictly) in this ordering. In poker, this ordering corresponds to the ranking of card hands.

The abstraction algorithm operates on the *signal tree,* which is the game tree with all the agents' action edges removed. We say that two sibling nodes in the signal tree are *ordered game isomorphic* if (1) if the nodes are leaves, the payoff vectors of the players (which payoff in the vector materializes depends on how the agents play) are the same at both nodes, and (2) if the nodes are interior nodes, there is a bipartite matching of the nodes' children so that only ordered game isomorphic children get matched.

The GameShrink algorithm is a bottom-up dynamic program that merges all ordered game isomorphic nodes. It runs in $\tilde{O}(n^2)$ time, where n is the number of nodes in the signal tree. GameShrink tends to run in sublinear time and space in the size of the game tree because the signal tree is significantly smaller than the game tree in most nontrivial games. The beautiful aspect of this method is that it is lossless (theorem 1). A small example run of GameShrink is shown in figure 2.

We applied GameShrink to Rhode Island Hold'em poker (Gilpin and Sandholm 2007b). That two-player game was invented as a testbed for computational game playing (Shi and Littman 2002). Applying the sequence form to Rhode Island Hold'em directly without abstraction yields a linear program (LP) with 91,224,226 rows, and the same number of columns. This is much too large for (current) linear programming algorithms to handle. We used GameShrink to shrink this, yielding an LP with 1,237,238 rows and columns — with 50,428,638 nonzero coefficients. We then applied iterated elimination of dominated strategies, which further reduced this to 1,190,443 rows and 1,181,084 columns. GameShrink required less than one second to run. Then, using a 1.65 GHz IBM eServer p5 570 with 64 gigabytes of RAM (the LP solver actually needed 25 gigabytes), we solved the resulting LP in 8 days using the interior-point barrier method of CPLEX version 9.1.2. In summary, we found an exact solution to a game with 3.1 billion nodes in the game tree (the largest incomplete-information game that had been solved previously had 140,000 (Koller and Pfeffer 1997)). To my knowledge, this is still the largest incomplete-information game that has been solved exactly.[1]

Lossy Information Abstraction. Some games are so large that even after applying the kind of lossless abstraction described above, the resulting LP would be too large to solve. To address this problem, such games can be abstracted more aggressively, but this incurs loss in solution quality.

One approach is to use a lossy version of GameShrink where siblings are considered ordered game isomorphic if their children can be approximately matched in the bipartite matching part of the algorithm (Gilpin and Sandholm 2007b; 2006). However, lossy GameShrink suffers from three drawbacks.

First, the resulting abstraction can be highly inaccurate because the grouping of states into buckets is, in a sense, greedy. For example, if lossy GameShrink determines that hand A is similar to hand B, and then determines that hand B is similar to hand C, it will group A and C together, despite the fact that A and C may not be very similar. The quality of the abstraction can be even worse when a longer sequence of such comparisons leads to grouping together extremely different hands. Stated differently, the greedy aspect of the algorithm leads to lopsided buckets where large buckets are likely to attract even more states into the bucket.

Second, one cannot directly specify how many buckets lossy GameShrink should yield (overall or at any specific betting round). Rather, there is a parameter (for each round) that specifies a thresh-

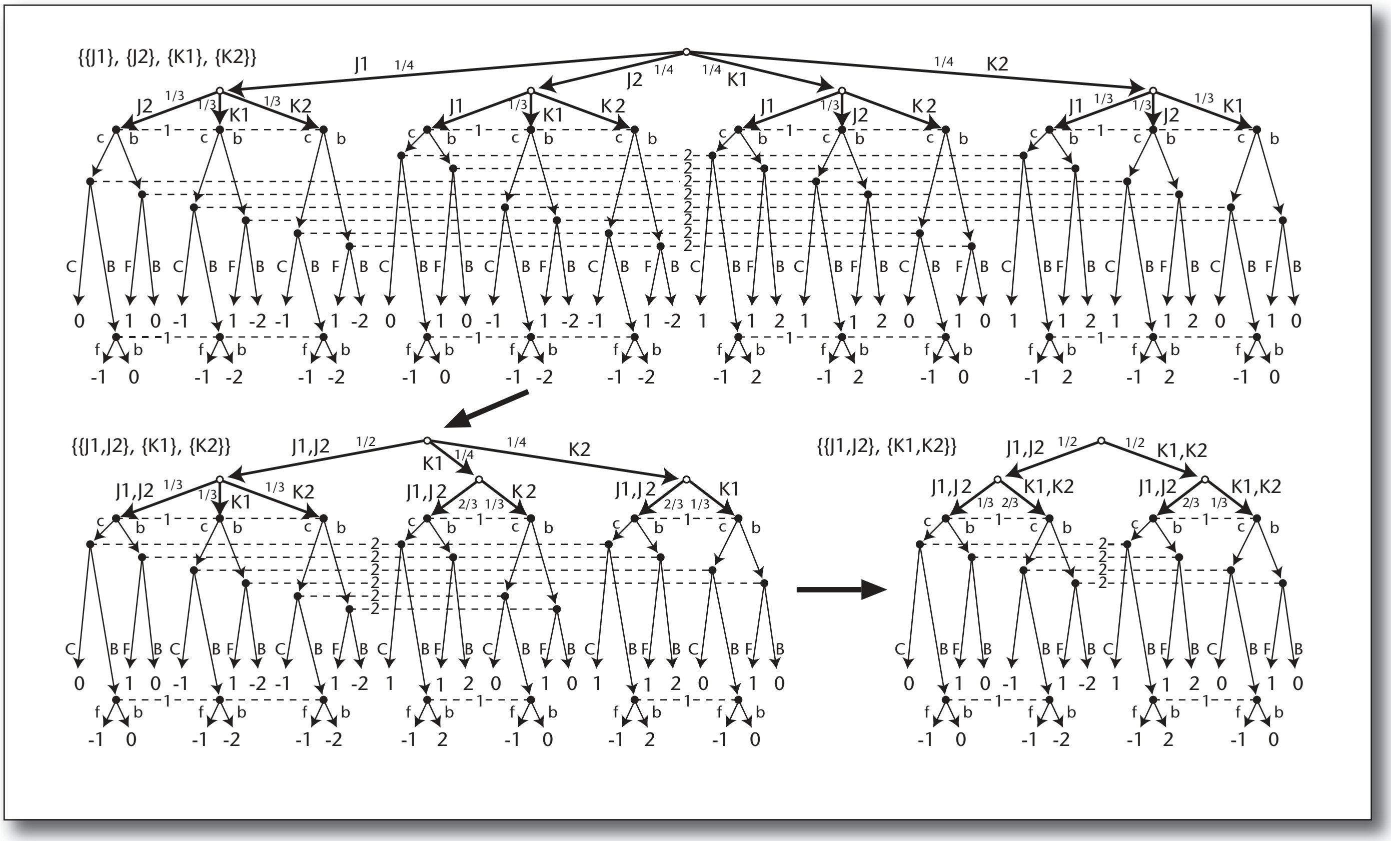

Figure 2. GameShrink Applied to a Tiny Two-Player Four-Card Poker Game.

(The game consists of two jacks and two kings) (Gilpin and Sandholm 2007b). Next to each game tree is the range of the information filter, which shows the abstraction. Dotted lines denote information sets, which are labeled by the controlling player. Open circles are chance nodes with the indicated transition probabilities. The root node is the chance node for player 1's card, and the next level is for player 2's card. The payment from player 2 to player 1 is given below each leaf. In this example, the algorithm reduces the game tree from 113 nodes to 39 nodes.

old of how different states can be and still be considered the same. If one knows how large an LP can be solved, one cannot create an LP of that size by specifying the number of buckets directly; rather one must use trial-and-error (or some variant of binary search applied to the setting of multiple parameters) to pick the similarity thresholds (one for each round) in a way that yields an LP of roughly the desired size.

The third drawback is scalability. The time needed to compute an abstraction for a three-round truncated version of two-player Limit Texas Hold'em was more than a month. Furthermore, it would have to be executed in the inner loop of the parameter guessing algorithm of the previous paragraph.

Expectation-Based Abstraction Using Clustering and Integer Programming. In this subsection I describe a new abstraction algorithm that eliminates the above problems (Gilpin and Sandholm 2007a). It is not specific to poker, but for concreteness I will describe it in the context of two-player Texas Hold'em.

The algorithm operates on an abstraction tree, which, unlike the signal tree, looks at the signals from one player's perspective only: the abstraction is done separately (and possibly with different numbers of buckets) for different players. For Texas Hold'em poker, it is initialized as follows. The root node contains (52 choose 2) = 1326 children, one for each possible pair of hole cards that a player may be dealt in round 1. Each of these children has (50 choose 3) children, each corresponding to the possible 3-card flops that can appear in round 2. Similarly, the branching factor at the next two levels is 47 and 46, corresponding to the 1-card draws in rounds 3 and 4, respectively.

The algorithm takes as input the number of buckets, K_r, allowed in the abstraction in each round, r. For example, we can limit the number of buckets of hands in round 1 to $K_1 = 20$. Thus, we would need to group (that is, abstract) each of the

(52 choose 2) = 1326 hands into 20 buckets. We treat this as a clustering problem. To perform the clustering, we must first define a metric to determine the similarity of two hands. Letting *w, l,* and *d* be the number of possible wins, losses, and draws (based on the rollout of the remaining cards), we compute the hand's value as $w + d/2$, and we take the distance between two hands to be the absolute difference between their values. This gives us the necessary ingredients to apply clustering to form *k* buckets, for example, *k*-means clustering (algorithm 1).

```
1. Create k centroid points in the interval between the
   minimum and maximum hand values.
2. Assign each hand to the nearest centroid.
3. Adjust each centroid to be the mean of their assigned
   hand values.
4. Repeat steps 2 and 3 until convergence.
```

Algorithm 1. k-means Clustering for Poker Hands.

Algorithm 1 is guaranteed to converge, but it may find a local optimum. Therefore, in our implementation we run it several times with different starting points to try to find a global optimum. For a given clustering, we can compute the error (according to the value measure) that we would expect to have when using the abstraction.

For the later rounds we again want to determine the buckets. Here we face the additional problem of determining how many children each parent in the abstraction tree can have. For example, we can put a limit of $K_2 = 800$ on the number of buckets in round 2. How should the right to have 800 children (buckets that have not yet been generated at this stage) be divided among the 20 parents? We model and solve this problem as a 0–1 integer program (Nemhauser and Wolsey 1999) as follows. Our objective is to minimize the expected error in the abstraction. Thus, for each of the 20 parent nodes, we run the *k*-means algorithm presented above for values of *k* between 1 and the largest number of children we might want any parent to have, *MAX*. We denote the expected error when node *i* has *k* children by $c_{i,k}$. We denote by p_i the probability of getting dealt a hand that is in abstraction class *i* (that is, in parent *i*); this is simply the number of hands in *i* divided by (52 choose 2). Based on these computations, the following 0–1 integer program finds the abstraction that minimizes the overall expected error for the second level:

$$\begin{array}{llcl} \min & \sum_{i=1}^{K_1} p_i \sum_{k=1}^{MAX} c_{i,k}\, x_{i,k} & & \\ \text{s.t.} & \sum_{i=1}^{K_1} \sum_{k=1}^{MAX} k x_{i,k} & \leq & K_2 \\ & \sum_{k=1}^{MAX} x_{i,k} & = & 1 \quad \forall i \\ & x_{i,k} & \in & \{0,1\} \end{array}$$

The decision variable $x_{i,k}$ is set to 1 if and only if node *i* has *k* children. The first constraint ensures that the limit on the overall number of children is not exceeded. The second constraint ensures that a decision is made for each node. This problem is a generalized knapsack problem, and although NP-complete, can be solved efficiently using off-the-shelf integer programming solvers (for example, CPLEX solves this problem in less than one second at the root node of the branch-and-bound search tree).

We repeat this procedure for round 3 (with the round-2 buckets as the parents and a different (larger) limit (say, $K_3 = 4{,}800$) on the maximum number of buckets. Then we compute the bucketing for round 4 analogously, for example with $K_4 = 28{,}800$.

Overall, our technique optimizes the abstraction round by round in the abstraction tree. A better abstraction (even for the same similarity metric) could conceivably be obtained by optimizing all rounds in one holistic optimization. However, that seems infeasible. First, the optimization problem would be nonlinear because the probabilities at a given level depend on the abstraction at previous levels of the tree. Second, the number of decision variables in the problem would be exponential in the size of the initial abstraction tree (which itself is large), even if the number of abstraction classes for each level is fixed.

Potential-Aware Abstraction. The expectation-based abstraction approach previously described does not take into account the *potential* of hands. For instance, certain poker hands are considered *drawing hands* in which the hand is currently weak, but has a chance of becoming very strong. Since the strength of such a hand could potentially turn out to be much different later in the game, it is generally accepted among poker experts that such a hand should be played differently than another hand with a similar chance of winning, but without as much potential (Sklansky 1999). However, if using the difference between probabilities of winning as the clustering metric, the abstraction algorithm would consider these two very different situations similar.

One possible approach to handling the problem that certain hands with the same probability of winning may have different potential would be to consider not only the expected strength of a hand, but also its variance. Although this would likely be an improvement over basic expectation-based abstraction, it fails to capture two important issues that prevail in many sequential imperfect information games, including poker.

First, mean and variance are a lossy representation of a probability distribution, and the lost

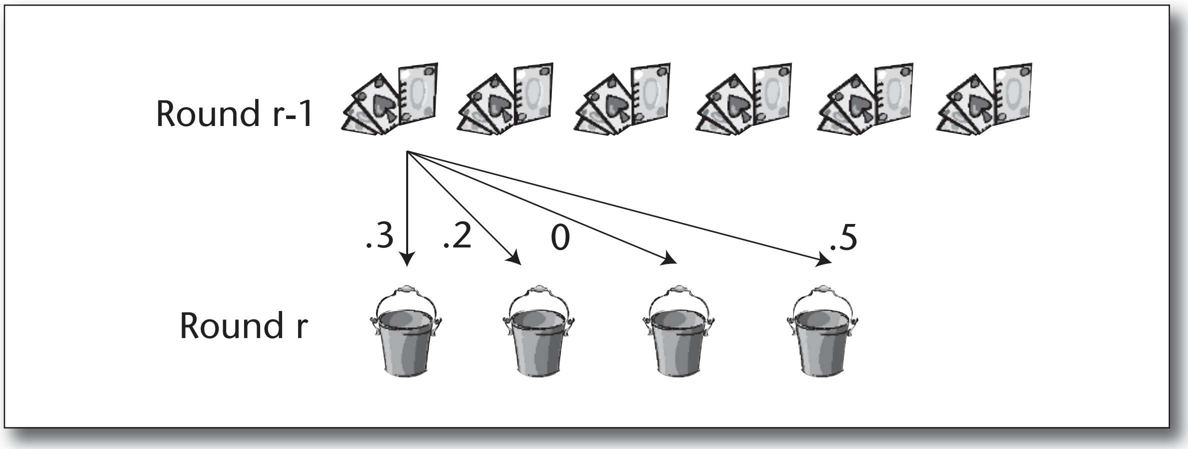

Figure 3. States and Transitions Used in Potential-Aware Abstraction.

In our potential-aware abstraction algorithm, the similarity of states is measured based on the states' transition histograms to buckets at the next round. This definition is then operationalized by conducting bottom-up passes in the abstraction tree so those later-round buckets get defined first.

aspects of the probability distribution over hand strength can be significant for deciding how one should play in any given situation.

Second, the approach based on mean and variance does not take into account the different paths of information revelation that hands take in increasing or decreasing in strength. For example, two hands could have similar means and variances, but one hand may get the bulk of its uncertainty resolved in the next round, while the other hand needs two more rounds before the bulk of its final strength is determined. The former hand is better because the player has to pay less to find out the essential strength of his hand.

To address these issues, we introduced potential-aware abstraction, where we associate with each state of the game a histogram over future possible states (Gilpin, Sandholm, and Sørensen 2007); see figure 3. This representation can encode all the pertinent information from the rest of the game (such as paths of information revelation), unlike the approach based on mean and variance.

As in expectation-based abstraction, we use a clustering algorithm and an integer program for allocating children. They again require a distance metric to measure the dissimilarity between different states. The metric we now use is the L_2-distance over the histograms of future states. Specifically, let S be a finite set of future states, and let each hand i be associated with a histogram, h_i, over the future states S. Then, the distance between hands i and j is

$$dist(i,j) = \sqrt{\sum_{s \in S} (h_i(s) - h_j(s))^2}.$$

Under this approach, another design dimension of the algorithm is in the construction of the possible future states. There are at least two prohibitive problems with this vanilla approach as stated. First, there are a huge number of possible reachable future states, so the dimensionality of the histograms is too large to do meaningful clustering with a reasonable number of clusters (that is, small enough to lead to an abstracted game that can be solved for equilibrium). Second, for any two states at the same level of the game, the descendant states are disjoint. Thus the histograms would have nonoverlapping supports, so any two states would have maximum dissimilarity and thus no basis for clustering.

For both of these reasons (and for reducing memory usage and enhancing speed), we coarsen the domains of the histograms. First, instead of having histograms over individual states, we use histograms over abstracted states (buckets, that is, clusters), which contain a number of states each (and we use those buckets' centroids to conduct the similarity calculations). We will have, for each bucket, a histogram over buckets later in the game. Second, we restrict the histogram of each bucket to be over buckets at the next round only (rather than over buckets at all future rounds). However, we introduce a technique (a bottom-up pass of constructing abstractions up the tree) that allows the buckets at the next round to capture information from all later rounds.

One way of constructing the histograms would be to perform a bottom-up pass of a tree representing the possible card deals: abstracting round 4 first, creating histograms for round 3 nodes based on the round 4 clusters, then abstracting round 3, creating histograms for round 2 nodes based on the round 3 clusters, and so on. This is indeed what we do to find the abstraction for round 1.

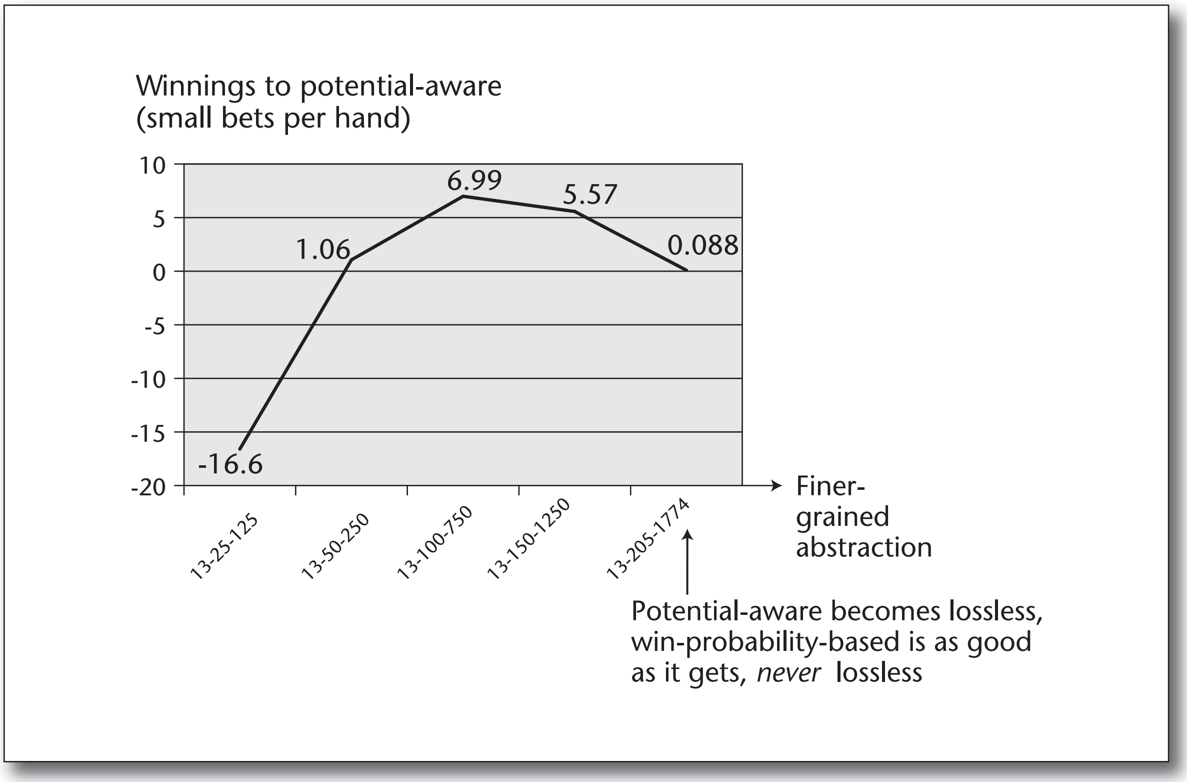

Figure 4. Potential-Aware Versus Expectation-Based Abstraction.

(Gilpin and Sandholm 2008a).

However, for later betting rounds, we improve on this by leveraging our knowledge of the fact that abstracted children of any bucket at the level above should only include states that can actually be children of the states in that bucket. We do this by multiple bottom-up passes, one for each bucket at the round above. For example, if a round-1 bucket contains only those states where the hand consists of two aces, then when we conduct abstraction for round 2, the bottom-up pass for that level-1 bucket should only consider future states where the hand contains two aces as the hole cards. This enables the abstraction algorithm to narrow the scope of analysis to information that is relevant given the abstraction that it made for earlier rounds.

In the last round there is no need to use the potential-aware techniques discussed above since the players will not receive any more information, that is, there is no potential. Instead, we simply compute the fourth-round abstraction based on each hand's probability of winning (based on different possible rollouts of the cards), using clustering and integer programming as in expectation-based abstraction.

Comparison of Expectation-Based Versus Potential-Aware Abstraction. Now, which is better, expectation-based or potential-aware abstraction? Both types of abstraction algorithms run very quickly compared to the time to even approximately solve the abstracted game, so the comparison comes down to how good the approximate equilibria are when generated from abstractions of each of the two types. It turns out that this depends on the granularity that the abstraction is allowed to have! (This in turn is dictated in practice by the speed of the equilibrium-finding algorithm that is used to approximately solve the abstracted game.) We conducted experiments on this question in the context of Rhode Island Hold'em so that we could solve the abstracted game exactly (Gilpin and Sandholm 2008a). For both types of abstraction algorithms we allowed the same number of abstract buckets at each of the three rounds of the game. We denote an abstraction granularity by the string K_1-K_2-K_3. For example, 13-25-125 means 13 first-round buckets, 25 second-round buckets, and 125 third-round buckets. The abstraction granularities we considered range from coarse (13-25-125) to fine (13-205-1774). At this latter granularity an equilibrium-preserving abstraction exists (Gilpin and Sandholm 2007b). In the experiments we fix the first-round granularity to 13 (which allows for a lossless solution in principle due to suit isomorphisms), and vary the granularity allowed in rounds two and three. Figure 4 shows the results when the programs generated with these two abstraction methods were played against each other. For very coarse abstractions, the expectation-based approach does better. For medium granularities, the potential-aware approach does better. For fine granularities, the potential-aware approach does better, but the difference is small. Interestingly, the expectation-based approach never yielded a lossless abstraction no matter how fine a granularity was allowed.

These conclusions hold also when (1) comparing the players against an optimal player (that is, an equilibrium strategy), or (2) comparing each player against its nemesis (that is, a strategy that exploits the player as much as possible in expectation), or (3) evaluating the abstraction based on its ability to estimate the true value of the game.

These conclusions also hold for a variant of the expectation-based algorithm that considers the square of the probability of winning ($v_i = (w_i + d/2)^2$), rather than simply the probability (Zinkevich et al. 2007). A motivation for this is that the hands with the higher probabilities of winning should be more finely abstracted than lower-value hands. This is because low-value hands are likely folded before the last round, and because it is important to know very finely how good a high-value hand one is holding if there is a betting escalation in the final round. Another suggested motivation is that this captures some of the variance and "higher variance is preferred as it means the player eventually will be more certain about their ultimate chances of winning prior to a showdown." The version of this we experimented with is somewhat different than the original since we are using an integer program to allocate the buckets, which enables nonuniform bucketing.

One possible explanation for the crossover between expectation-based and potential-aware abstraction is that the dimensionality of the temporary states used in the bottom-up pass in the third-round (which must be smaller than the number of available second-round buckets in order for the clustering to discover meaningful centroids) is insufficient for capturing the strategically relevant aspects of the game. Another hypothesis is that since the potential-aware approach is trying to learn a more complex model (in a sense, clusters of paths of states) and the expectation-based model is trying to learn a less complex model (clusters of states), the former requires a larger dimension to capture this richness.

The existence of a cross-over suggests that for a given game — such as Texas Hold'em — as computers become faster and equilibrium-finding algorithms more scalable so games with finer-grained abstractions become solvable, the potential-aware approach will become the better method of choice.

Problems with Lossy Information Abstraction. In single-agent settings, lossy abstraction has the desirable property of monotonicity: As one refines the abstraction, the solution improves. There is only a weak analog of this in (even two-player zero-sum) games: If the opponent's strategy space is not abstracted lossily at all, refining our player's abstraction causes our player to become less exploitable. (Exploitability is measured by how much our player loses to its nemesis in expectation. This can be computed in reasonably sizeable games by carrying out a best-response calculation to our agent's strategy.) There are no proven guarantees on the amount of exploitability as a function of the coarseness of the lossy information abstraction used. Furthermore, sometimes the exploitability of a player increases as its abstraction or its opponent's abstraction is refined (Waugh et al. 2009a). This nonmonotonicity holds for information abstraction even if one carefully selects the least exploitable equilibrium strategy for the player in the abstracted games. The nonmonotonicity has been shown in small artificial poker variants. For Texas Hold'em (even with just two players), experience from years of the AAAI Computer Poker Competition suggests that in practice finer abstractions tend to yield better strategies. However, further research on this question is warranted.

Another problem is that current lossy abstraction algorithms do not yield lossless abstractions even if enough granularity is allowed for a lossless abstraction to exist. For the expectation-based abstraction algorithm this can already be seen in figure 4, and this problem arises in some games also with the potential-aware abstraction algorithm. One could trivially try to circumvent this problem by running lossless GameShrink first, and only running a lossy abstraction algorithm if the abstraction produced by GameShrink has a larger number of buckets than desired.

Strategy-Based Abstraction. It may turn out that abstraction is as hard a problem as equilibrium finding itself. After all, two states should fall in the same abstract bucket if the optimal action probabilities in them are (almost) the same, and determining the action probabilities is done by finding an equilibrium.

This led us to develop the strategy-based abstraction approach. It iterates between abstraction and equilibrium finding. The equilibrium finding operates on the current abstraction. Once a (near) equilibrium is found for that abstraction, we redo the abstraction using the equilibrium strategies to inform the bucketing. Then we find a near equilibrium for the new abstraction, and so on.

We have applied this approach for the AAAI Computer Poker Competition. However, definitive results have not yet been obtained on the approach because for the fine-grained abstractions used in the competition (about 10^{12} leaves in the abstracted game tree), approximate equilibrium finding takes months on a shared-memory supercomputer using 96 cores. Therefore, we have so far had time to go over the abstraction/equilibrium finding cycle only twice. Future research should explore this approach more systematically both experimentally and theoretically.

Action Abstraction

So far in this article I have discussed information abstraction. Another way of making games easier to solve is *action abstraction,* where in the abstracted game the players have fewer actions available than in the original game. This is especially important in games with large or infinite action spaces, such as No-Limit poker. So far action abstraction has been done by selecting some of the actions from the original game into the abstracted game, although in principle one could generate some abstract actions that are not part of the original game. Also, so far action abstractions have been generated manually.[2] Future research should also address automated action abstraction.

Action abstraction begets a fundamental problem because real opponent(s) may select actions outside the abstract model. To address this, work has begun on studying what are good *reverse mappings* (figure 1), that is, how should opponents' actions that do not abide to the abstraction be interpreted in the abstracted game? One objective is to design a reverse mapping that tries to minimize the player's exploitability. Conversely, one would like to have actions in one's own abstraction that end up exploiting other players' action abstractions. These remain largely open research areas, but some experimental results already exist on the former. In No-Limit poker it tends to be better to use logarithmic rather than linear distance when measuring how close an opponent's real bet is to the bet sizes in the abstraction (Gilpin, Sandholm, and Sørensen 2008). Furthermore, a randomized reverse mapping that weights the abstract betting actions based on their distance to the opponent's real bet tends to help (Schnizlein, Bowling, and Szafron 2009). As with information abstraction, in some games refining the action abstraction can actually increase the player's exploitability (Waugh et al. 2009a).

Phase-Based Abstraction, Real-Time Equilibrium Finding, and Strategy Grafting

Beyond information and action abstraction, a third form of abstraction that has been used for incomplete-information games is *phase-based abstraction*. The idea is to solve earlier parts (which we call phases) of the game separately from later phases of the game. This has the advantage that each part can use a finer abstraction than would be tractable if the game were solved holistically. The downside is that gluing the phases together soundly is tricky. For one, when solving a later phase separately from an earlier phase, a strategy may disclose to the opponent information about which exact later phase version is being played (in poker, information about the private cards the player is holding). From the perspective of the later phase alone this will seem like no loss, but from the perspective of the entire game it fails to hide information as effectively as a holistically solved game.

This approach has been used to tackle two-player Texas Hold'em poker in two (or in principle more) phases. The first phase includes the early rounds. It is solved offline. To be able to solve it, one needs a model of what would happen in the later rounds, that is, what are the payoffs at the end of each path of the first phase. The first approach was to assume rollout of the cards in the later phase(s), that is, no betting and no folding (Billings et al. 2003). Better results were achieved by taking the strategies for the later phase(s) directly from strong prior poker bots or from statistical observations of such bots playing if the bots' strategies themselves are not available (Gilpin and Sandholm 2007a). I call this *bootstrapping with base strategies*.

It has also been shown experimentally that having the phases overlap yields better results than having the phases be disjoint. For example, the first phase can include rounds 1, 2, and 3, while the second phase can include rounds 3 and 4 (Gilpin and Sandholm 2007a). Or the first phase can include all rounds, and the second phase a more refined abstraction of one or more of the later rounds.

The second phase can be solved in real time during the game so that a finer abstraction can be used than if all possible second-phase games would have to be solved (that is, all possible sequences of cards and actions from the rounds before the start of the second phase) (Gilpin and Sandholm 2007a). Whether or not the second phase is solved offline or in real time, at the beginning of the second phase, before the equilibrium finding for the second phase takes place, the players' beliefs are updated using Bayes' rule based on the cards and actions each player has observed in the first phase.

The idea of bootstrapping with base strategies has been extended to base strategies that cover the entire game, not just the end (Waugh, Bard, and Bowling 2009). The base strategies can be computed using some abstraction followed by equilibrium finding. Then, one can isolate a part of the abstracted game at a time, construct a finer-grained abstraction for that part, require that Player 1 follow his base strategy in all his information sets except those in that part (no such restriction is placed on the opponent), and solve the game anew. Then one can pick a different part and do this again, using the same base strategy. Once all

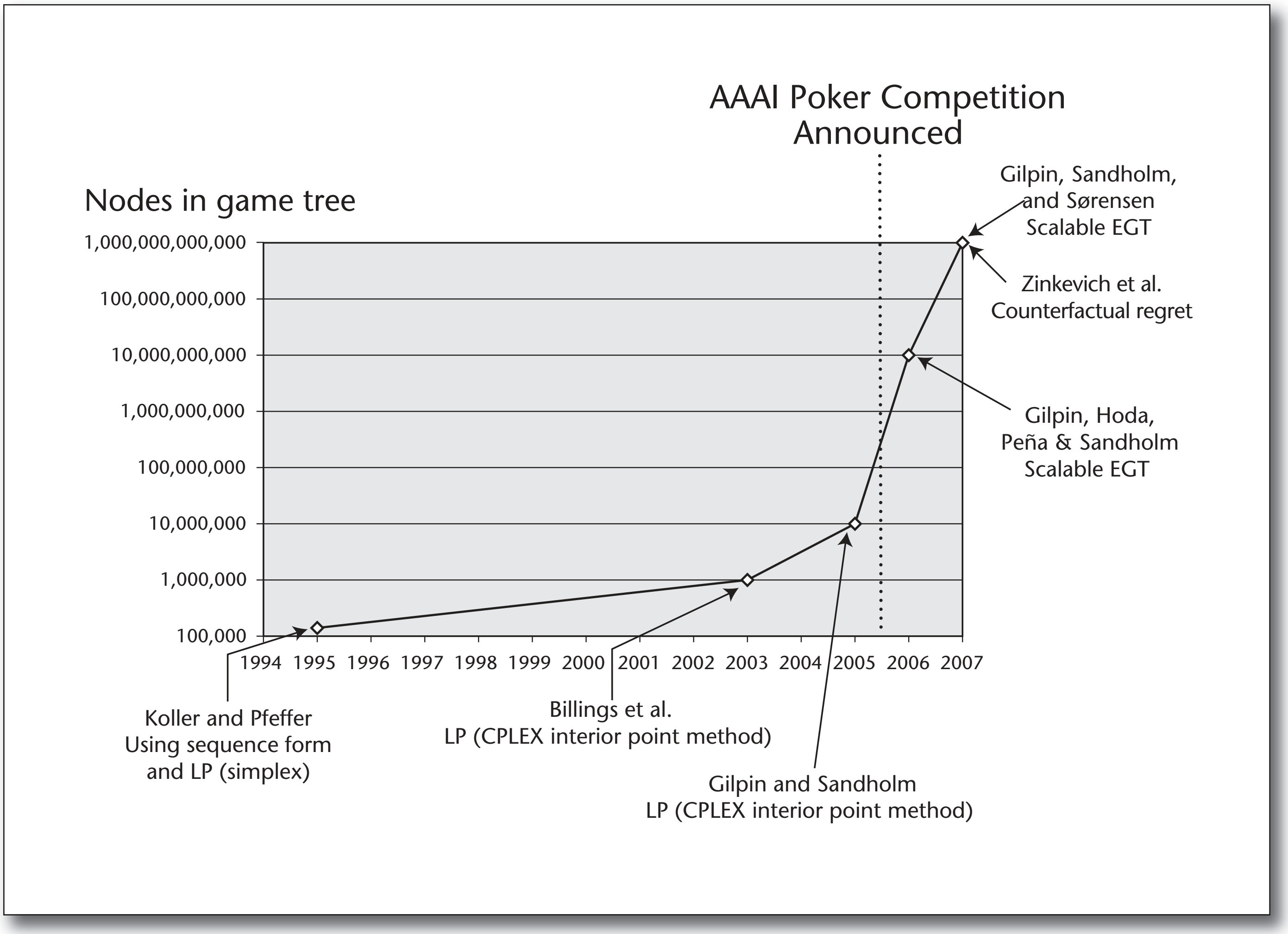

Figure 5. Progress on Algorithms for Solving Two-Player Zero-Sum Games.

the parts (which constitute a partition of the information sets where it is Player 1's turn to move) have been finished, we have a strategy for Player 1. Then, a strategy for Player 2 is computed analogously. This *grafting* approach allows one to focus on one part of the game at a time with fine abstraction while having a holistic view of the game through the base strategies. In principle this can increase the player's exploitability, but in practice it improves performance. Similar approaches can be used for more than two players, but nothing has been published on that yet.

Equilibrium-Finding Algorithms for Two-Player Zero-Sum Games.

So far I have discussed abstraction. I will now move to algorithms for solving the (abstracted) game. This section focuses on two-player zero-sum games. The next section covers more general games.

The most common solution concept (that is, definition of what it means to be a solution) is Nash equilibrium. A strategy for an agent defines for each information set where it is the agent's turn to move a probability distribution over the agent's actions. The two agents' strategies form a Nash equilibrium if neither agent can benefit in expectation by deviating from her strategy given that the other agent does not deviate from his.

Formally, the Nash equilibria of two-player zero-sum sequential games are the solutions to

$$\min_{\mathbf{x}\in\mathcal{X}} \max_{\mathbf{y}\in\mathcal{Y}} \mathbf{y}^{T}A\mathbf{x} = \max_{\mathbf{y}\in\mathcal{Y}} \min_{\mathbf{x}\in\mathcal{X}} \mathbf{y}^{T}A\mathbf{x} \tag{1}$$

where $\mathcal{X}$ and $\mathcal{Y}$ are polytopes defining the players' strategies and A is the payoff matrix (Romanovskii 1962; Koller, Megiddo, and von Stengel 1996; von Stengel 1996). When the minimizer plays a strategy $\mathbf{x} \in \mathcal{X}$ and the maximizer plays $\mathbf{y} \in \mathcal{Y}$, the expected utility to the maximizer is $\mathbf{y}^T A\mathbf{x}$ and, since the game is zero-sum, the minimizer's expected utility is $-\mathbf{y}^T A\mathbf{x}$. Problem 1 can be expressed as a linear program (LP) whose size is linear in the size of the game tree. Thus the problem is solvable in polynomial time. Today's best general-purpose LP solvers can solve games with up to 10^7 or 10^8 leaves in the game tree (corresponding to nonzero entries in A) (Gilpin and Sandholm 2006). For example, losslessly abstracted Rhode Island Hold'em poker has a $10^6 \times 10^6$ payoff matrix containing 50 million nonzeros, and solving it (to near machine precision) with CPLEX's barrier method (an interior-point LP method) took a week and used 25 gigabytes of RAM (Gilpin and Sandholm 2007b). Interestingly, on these kinds of problems the barrier method does better than the simplex method.

The LP approach does not scale to most interesting games. For instance, the payoff matrix A in (1) for two-player Limit Texas Hold'em poker has dimension $10^{14} \times 10^{14}$ and contains more than 10^{18} nonzero entries. There has been tremendous progress in developing equilibrium-finding algorithms in the last few years, spurred in part by the AAAI Computer Poker Competition, (see figure 5). These new algorithms find an ε-equilibrium, that is, strategies $\mathbf{x} \in \mathcal{X}$ and $\mathbf{y} \in \mathcal{Y}$ such that neither player can benefit more than ε in expectation by deviating from her strategy.

Algorithms Based on Smoothing and Gradient Descent

In this section I will describe recent custom equilibrium-finding algorithms based on smoothing and gradient descent.

Extending the Excessive Gap Technique to Sequential Games and Making it Scalable. In this section I will describe the first custom equilibrium-finding algorithm for sequential incomplete-information games (Gilpin et al. 2007; Hoda et al. 2010). It took equilibrium finding to a new level by solving poker games with 10^{12} leaves in the game tree (Gilpin, Sandholm, and Sørensen 2007), and it found an ε-equilibrium with a small ε (0.027 small bets per hand in abstracted Limit Texas Hold'em). It remains one of the fastest equilibrium-finding algorithms in practice.

The algorithm is an adaptation of Nesterov's excessive gap technique (Nesterov 2005a; 2005b) to sequential games, and it also includes techniques for significantly improving scalability both in terms of time and memory usage.

Problem 1 can be stated as

$$\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}) = \max_{\mathbf{y}\in\mathcal{Y}} \phi(\mathbf{y}) \tag{2}$$

where

$$f(\mathbf{x}) = \max_{\mathbf{y}\in Y} \mathbf{y}^T A\mathbf{x} \text{ and } \phi(\mathbf{y}) = \min_{\mathbf{x}\in X} \mathbf{y}^T A\mathbf{x}.$$

There is a procedure (algorithm 4 presented later) based on the above smoothing technique that after N iterations generates a pair of points $(\mathbf{x}^N, \mathbf{y}^N) \in \mathcal{X} \times \mathcal{Y}$ such that

$$0 \le f(\mathbf{x}^N) - \phi(\mathbf{y}^N) \le \frac{4\,\|A\|}{N+1}\sqrt{\frac{D_{\mathcal{X}} D_{\mathcal{Y}}}{\sigma_{\mathcal{X}}\sigma_{\mathcal{Y}}}}.$$

Furthermore, each iteration of the procedure performs some elementary operations, three matrix-vector multiplications by A, and requires the exact solution of three subproblems of the form

$$\max_{\mathbf{x}\in\mathcal{X}}\{\mathbf{g}^T\mathbf{x} - d_{\mathcal{X}}(\mathbf{x})\} \quad \text{or} \quad \max_{\mathbf{y}\in\mathcal{Y}}\{\mathbf{g}^T\mathbf{y} - d_{\mathcal{Y}}(\mathbf{y})\}. \tag{4}$$

Theorem 2.

(Nesterov 2005a, 2005b).

The functions f and ϕ are respectively convex and concave nonsmooth functions. The left-hand side of equation 2 is a standard convex minimization problem of the form

$$\min\{h(\mathbf{x}) : \mathbf{x} \in \mathcal{X}\}. \tag{3}$$

First-order methods for solving equation 3 are algorithms for which a search direction at each iteration is obtained using only the first-order information of h, such as its gradient or subgradient. When h is nonsmooth, subgradient algorithms can be applied, but they have a worst-case complexity of $\mathcal{O}(1/\varepsilon^2)$ iterations (Goffin 1977). However, that pessimistic result is based on treating h as a black box where the value and subgradient are accessed through an oracle. For nonsmooth functions with a suitable max structure, Nesterov devised first-order algorithms requiring only $\mathcal{O}(1/\varepsilon)$ iterations.

The key component of Nesterov's smoothing technique is a pair of prox-functions for the sets $\mathcal{X}$ and $\mathcal{Y}$. These prox-functions are used to construct smooth approximations $f_\mu \approx f$ and $\phi_\mu \approx \phi$. To obtain approximate solutions to equation 2, gradient-based algorithms can then be applied to f_μ and ϕ_μ.

We say that a function is a prox-function if it is strongly convex and its minimum is zero. Assume $d_{\mathcal{X}}$ and $d_{\mathcal{Y}}$ are prox-functions for the sets $\mathcal{X}$ and $\mathcal{Y}$ respectively. Then for any given $\mu > 0$, the smooth approximations $f_\mu \approx f$ and $\phi_\mu \approx \phi$ are

$$f_\mu(\mathbf{x}) := \max\{\mathbf{y}^T A\mathbf{x} - \mu d_{\mathcal{Y}}(\mathbf{y}) : \mathbf{y} \in \mathcal{Y}\},$$

$$\phi_\mu(\mathbf{y}) := \min\{\mathbf{y}^T A\mathbf{x} + \mu d_{\mathcal{X}}(\mathbf{x}) : \mathbf{x} \in \mathcal{X}\}.$$

Let $D_{\mathcal{X}} := \max\{d_{\mathcal{X}}(\mathbf{x}) : \mathbf{x} \in \mathcal{X}\}$, and let $\sigma_{\mathcal{X}}$ denote the strong convexity modulus of $d_{\mathcal{X}}$. Let $D_{\mathcal{Y}}$ and $\sigma_{\mathcal{Y}}$ be defined likewise for $\mathcal{Y}$ and $d_{\mathcal{Y}}$.[3]

For an algorithm based on theorem 2 to be practical, the subproblems (equation 4) must be solvable quickly. They can be phrased in terms of the

There is a way to construct nice prox-functions for sequence form games.

Theorem 3.
(Hoda et al. 2010).

```
initial $(A, d_{\mathcal{X}}, d_{\mathcal{Y}})$
1. $\mu^0_{\mathcal{X}} := \mu^0_{\mathcal{Y}} := \frac{\|A\|}{\sqrt{\sigma_{\mathcal{X}}\sigma_{\mathcal{Y}}}}$
2. $\hat{\mathbf{x}} := \nabla d^*_{\mathcal{X}}(\mathbf{0})$
3. $\mathbf{y}^0 := \nabla d^*_{\mathcal{Y}}\left(\frac{1}{\mu^0_{\mathcal{Y}}}A\hat{\mathbf{x}}\right)$
4. $\mathbf{x}^0 := \nabla d^*_{\mathcal{X}}\left(\nabla d_{\mathcal{X}}(\hat{\mathbf{x}}) + \frac{1}{\mu^0_{\mathcal{X}}}A^{\mathrm{T}}\mathbf{y}^0\right)$
5. Return $(\mu^0_{\mathcal{X}}, \mu^0_{\mathcal{Y}}, \mathbf{x}^0, \mathbf{y}^0)$
```

Algorithm 2.

```
shrink $(A, \mu_{\mathcal{X}}, \mu_{\mathcal{Y}}, \tau, \mathbf{x}, \mathbf{y}, d_{\mathcal{X}}, d_{\mathcal{Y}})$
1. $\breve{\mathbf{x}} := \nabla d^*_{\mathcal{X}}\left(-\frac{1}{\mu_{\mathcal{X}}}A^{\mathrm{T}}\mathbf{y}\right)$
2. $\hat{\mathbf{x}} := (1-\tau)\mathbf{x} + \tau\breve{\mathbf{x}}$
3. $\hat{\mathbf{y}} := \nabla d^*_{\mathcal{Y}}\left(\frac{1}{\mu_{\mathcal{Y}}}A\hat{\mathbf{x}}\right)$
4. $\tilde{\mathbf{x}} := \nabla d^*_{\mathcal{X}}\left(\nabla d_{\mathcal{X}}(\breve{\mathbf{x}}) - \frac{\tau}{(1-\tau)\mu_{\mathcal{X}}}A^{\mathrm{T}}\hat{\mathbf{y}}\right)$
5. $\mathbf{y}^+ := (1-\tau)\mathbf{y} + \tau\hat{\mathbf{y}}$
6. $\mathbf{x}^+ := (1-\tau)\mathbf{x} + \tau\tilde{\mathbf{x}}$
7. $\mu^+_{\mathcal{X}} := (1-\tau)\mu_{\mathcal{X}}$
8. Return $(\mu^+_{\mathcal{X}}, \mathbf{x}^+, \mathbf{y}^+)$
```

Algorithm 3.

conjugate of the functions $d_{\mathcal{X}}$ and $d_{\mathcal{Y}}$. The conjugate of $d : Q \to \mathbb{R}$ is the function $d^* : \mathbb{R}^n \to \mathbb{R}$ defined by

$$d^*(\mathbf{s}) := \max\{\mathbf{s}^{\mathrm{T}}\mathbf{x} - d(\mathbf{x}) : \mathbf{x} \in Q\}.$$

If d is strongly convex and Q is compact, then the conjugate d^* is Lipschitz continuous, differentiable everywhere, and

$$\nabla d^*(\mathbf{s}) = \operatorname{argmax}\{\mathbf{s}^{\mathrm{T}}\mathbf{x} - d(\mathbf{x}) : \mathbf{x} \in Q\}.$$

If the prox-function's conjugate and the conjugate's gradient are computable quickly (and the prox-function is continuous, strongly convex, and differentiable), we say that the prox-function is *nice* (Hoda et al. 2010). With nice prox-functions the overall algorithm is fast.

In normal form (also known as bimatrix) games, the strategy of each player lives in a simplex, that is, the probabilities on her actions sum to one. For the k-dimensional simplex Δ_k, the entropy function

$$d(\mathbf{x}) = \ln k + \sum_{i=1}^{k} x_i \ln x_i,$$

and the Euclidean distance function

$$d(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{k}(x_i - 1/k)^2$$

are nice prox-functions.

In multistep games, the strategies live in a more complex space. The common way to represent multistep games of perfect recall is the sequence form (Romanovskii 1962; Koller, Megiddo, and von Stengel 1996; von Stengel 1996). Our theorem 3 enables the use of Nesterov's excessive gap technique for these games.

Nesterov's Excessive Gap Technique (EGT). For $\mu_{\mathcal{X}}, \mu_{\mathcal{Y}} > 0$, consider the pair of problems:

$$f_{\mu_{\mathcal{Y}}}(\mathbf{x}) := \max\{\mathbf{y}^{\mathrm{T}}A\mathbf{x} - \mu_{\mathcal{Y}}d_{\mathcal{Y}}(\mathbf{y}) : \mathbf{y} \in \mathcal{Y}\},$$

$$\phi_{\mu_{\mathcal{X}}}(\mathbf{y}) := \min\{\mathbf{y}^{\mathrm{T}}A\mathbf{x} + \mu_{\mathcal{X}}d_Q(\mathbf{x}) : \mathbf{x} \in \mathcal{X}\}.$$

Algorithm 4 generates iterates $(\mathbf{x}^k, \mathbf{y}^k, \mu^k_{\mathcal{X}}, \mu^k_{\mathcal{Y}})$ with $\mu^k_{\mathcal{X}}, \mu^k_{\mathcal{Y}}$ decreasing to zero and such that the following excessive gap condition is satisfied at each iteration:

$$f_{\mu_{\mathcal{Y}}}(\mathbf{x}) \le \phi_{\mu_{\mathcal{X}}}(\mathbf{y}). \qquad (5)$$

From equation 4 and the fact $f(\mathbf{x}) \ge \phi(\mathbf{y})$, we see that

$$0 \le \phi(\mathbf{y}) - f(\mathbf{x}) \le \mu_{\mathcal{X}}D_{\mathcal{X}} + \mu_{\mathcal{Y}}D_{\mathcal{Y}}. \qquad (6)$$

Consequently, $f(\mathbf{x}^k) \approx \phi(\mathbf{y}^k)$ when $\mu^k_{\mathcal{X}}$ and $\mu^k_{\mathcal{Y}}$ are small.

Algorithm 2 finds a starting point that satisfies the excessive gap condition (equation 5).

Algorithm 3 decreases $\mu_{\mathcal{X}}$ and $\mu_{\mathcal{Y}}$ while maintaining equation 5.

If the input $(\mu_{\mathcal{X}}, \mu_{\mathcal{Y}}, x, y)$ to algorithm 3 satisfies equation 5, then so does $(\mu^+_{\mathcal{X}}, \mu_{\mathcal{Y}}, \mathbf{x}^+, \mathbf{y}^+)$ as long as τ satisfies $\tau^2/(1-\tau) \le \mu_{\mathcal{X}}\,\mu_{\mathcal{Y}}\,\sigma_{\mathcal{X}}\,\sigma_{\mathcal{Y}}/||A||^2$ (Nesterov 2005a).

We are now ready to describe Nesterov's excessive gap technique specialized to equation 1 (see algorithm 4). By theorem 2, algorithm 4 — which we will refer to as EGT — finds an ε-equilibrium in $\mathcal{O}(1/\varepsilon)$ iterations. Furthermore, for games with ordered signals, each iteration runs in linear time in the size of the game tree (Hoda et al. 2010).

Heuristics. EGT can be sped up by applying heuristics that decrease $\mu_{\mathcal{X}}$ and $\mu_{\mathcal{Y}}$ faster, while maintaining the excessive gap condition (4) (Gilpin et al. 2007). This leads to faster convergence in practice without compromising any of the theoretical guarantees.

The first heuristic is based on the following observation: although the value $\tau = 2/(k+3)$ computed in step 2(a) of EGT guarantees the excessive gap condition (equation 5), this is potentially an overly conservative value. Instead we can use an adaptive procedure to choose a larger value of τ. Since we then can no longer guarantee equation 5 a priori, we do a posterior verification, which occasionally necessitates an adjustment in the parameter τ.

The second heuristic is motivated by the observation that after several iterations, one of $\mu_{\mathcal{X}}$ and $\mu_{\mathcal{Y}}$ may be much smaller than the other. This imbalance is undesirable because the larger one contributes the most to the worst-case bound (equation 6). Hence, every so many iterations we perform a balancing to bring these values closer together. The balancing consists of repeatedly shrinking the larger one of $\mu_{\mathcal{X}}$ and $\mu_{\mathcal{Y}}$.

We also observed that after such balancing, the values of $\mu_{\mathcal{X}}$ and $\mu_{\mathcal{Y}}$ can sometimes be further reduced without violating the excessive gap condition (equation 5). We thus include a final reduction step in the balancing.

Experiments on automatically abstracted poker games show that each of the two heuristics tends to reduce ε by about an order of magnitude. Those experiments also show that using the entropy prox-function at the leaves performs better than using the Euclidian prox-function.

Decomposed Game Representation to Save Memory. One attractive feature of first-order methods like EGT is that the only operation performed on the matrix A is a matrix-vector product. We can thus exploit the problem structure to store only an implicit representation of A (Gilpin et al. 2007). This representation relies on a certain type of decomposition that is present in games with ordered signals. For example, the betting sequences that can occur in most poker games are independent of the cards that are dealt. We can decompose the payoff matrix based on these two aspects.

For ease of exposition, we explain the concise representation in the context of Rhode Island Hold'em. The payoff matrix A can be written as

$$A = \begin{bmatrix} A_1 & & \\ & A_2 & \\ & & A_3 \end{bmatrix}$$

where

$$\begin{aligned} A_1 &= F_1 \otimes B_1, \\ A_2 &= F_2 \otimes B_2, \text{ and} \\ A_3 &= F_3 \otimes B_3 + S \otimes W \end{aligned}$$

for much smaller matrices F_i, B_i, S, and W. The matrices F_i correspond to sequences of moves in round i that end with a fold, and S corresponds to the sequences in round 3 that end in a showdown. The matrices B_i encode the betting structures in round i, while W encodes the win/lose/draw information determined by poker hand ranks. The symbol $\otimes$ denotes the Kronecker product. The Kronecker product of two matrices $B \in \mathbb{R}^{m\times n}$ and $C \in \mathbb{R}^{p\times q}$ is

```
EGT $(A, d_{\mathcal{X}}, d_{\mathcal{Y}})$

1. $(\mu^0_{\mathcal{X}}, \mu^0_{\mathcal{Y}}, \mathbf{x}^0, \mathbf{y}^0)$ = initial$(A, d_{\mathcal{X}}, d_{\mathcal{Y}})$
2. For $k = 0, 1, \ldots$:
   (a) $\tau := \frac{2}{k+3}$
   (b) If $k$ is even:    // shrink $\mu_{\mathcal{X}}$
       i. $(\mu^{k+1}_{\mathcal{X}}, \mathbf{x}^{k+1}, \mathbf{y}^{k+1})$ :=
          shrink$(A, \mu^k_{\mathcal{X}}, \mu^k_{\mathcal{Y}}, \tau, \mathbf{x}^k, \mathbf{y}^k, d_{\mathcal{X}}, d_{\mathcal{Y}})$
       ii. $\mu^{k+1}_{\mathcal{Y}} := \mu^k_{\mathcal{Y}}$
   (c) If $k$ is odd:    // shrink $\mu_{\mathcal{Y}}$
       i. $(\mu^{k+1}_{\mathcal{Y}}, \mathbf{y}^{k+1}, \mathbf{x}^{k+1})$ :=
          shrink$(-A^{\mathrm{T}}, \mu^k_{\mathcal{Y}}, \mu^k_{\mathcal{X}}, \tau, \mathbf{y}^k, \mathbf{x}^k, d_{\mathcal{Y}}, d_{\mathcal{X}})$
       ii. $\mu^{k+1}_{\mathcal{X}} := \mu^k_{\mathcal{X}}$
```

Algorithm 4.

$$B \otimes C = \begin{bmatrix} b_{11}C & \cdots & b_{1n}C \\ \vdots & \ddots & \vdots \\ b_{m1}C & \cdots & b_{mn}C \end{bmatrix} \in \mathbb{R}^{mp \times nq}.$$

Given the above decomposed representation of A, the space required is sublinear in the size of the game tree. For example, in Rhode Island Hold'em, the dimensions of the F_1 and F_2 matrices are 10×10 and 70×70 respectively. The dimension of the F_3 and S matrices are 490×490. The dimensions of B_1, B_2, and B_3 are 13×13, 205×205, and $1,774 \times 1,774$, respectively. By contrast, the matrix A is $883,741 \times 883,741$. Furthermore, the matrices F_i, B_i, S, and W are themselves sparse, so we capitalize on the Compressed Row Storage (CRS) data structure that only stores nonzero entries. Table 1 demonstrates that the decomposed game representation enables equilibrium finding in the large.

Speeding Up the Matrix-Vector Products Using Parallelization and Sampling. The matrix-vector operations dominate the run time of first-order algorithms like EGT. They can be parallelized on multiple cores with near-perfect efficiency (Gilpin et al. 2007). For further speedups, one can sample the payoff matrix A to construct a sparser matrix, and then run the algorithm on the latter. One can also redo the sampling dynamically if overfitting to the sparser matrix occurs (Gilpin and Sandholm 2010).

Poker Players Created. In 2008, the Association for the Advancement of Artificial Intelligence (AAAI) held the third annual AAAI Computer Poker Competition, where computer programs submitted by teams worldwide competed against each other. We generated our players using lossy abstraction algorithms followed by equilibrium finding in the abstracted game using the EGT algorithm described above. GS4-Beta placed first (out

Name	CPLEX IPM	CPLEX Simplex	EGT
10k	0.082 GB	> 0.051 GB	0.012 GB
160k	2.25 GB	> 0.664 GB	0.035 GB
RI	25.2 GB	> 3.45 GB	0.15 GB
Texas	> 458 GB	> 458 GB	2.49 GB
GS4	> 80,000 GB	> 80,000 GB	43.96 GB

Table 1.

Memory footprint of CPLEX interior-point method (IPM), CPLEX simplex, and our memory-efficient version of EGT. 10k and 160k are lossy abstractions of Rhode Island Hold'em, and RI is lossless. Texas and GS4 are lossy abstractions of Texas Hold'em.

The algorithm finds an ϵ-equilibrium in

$$2\sqrt{2} \cdot e \cdot \kappa(A) \cdot \ln(2\|A\|/\epsilon) \cdot \sqrt{D}$$

first-order iterations, where $\|\cdot\|$ is the Euclidean matrix norm, D is the maximum Euclidean distance between strategies, and $\kappa(A)$ is a condition measure of A.

Theorem 4.

(Gilpin and Sandholm 2008a).

of nine) in the Limit Bankroll Competition and Tartanian placed third (out of four) in the No-Limit Competition. Tartanian actually had the highest winning rate in the competition, but due to the winner determination rule for the competition, it got third place.

A Gradient-Based Algorithm with $\mathcal{O}(\log 1/\varepsilon)$ Convergence. Recently we developed a gradient-based equilibrium-finding algorithm that finds an ε-equilibrium exponentially faster: in $\mathcal{O}(\log 1/\varepsilon)$ iterations (Gilpin, Peña, and Sandholm 2008). It uses as a subroutine a procedure we call *smoothing,* which is a recent smoothing technique for nonsmooth convex optimization (Nesterov 2005b). The algorithm is unlike EGT in that there is only one function that is being optimized and smoothed instead of two. Also unlike in EGT, the target ε needs to be specified in advance. This is used to set the constant μ that is used as the multiplier on the prox-function. Unlike in EGT, μ does not change inside smoothing.

We showed that smoothing can be extended to sequential games. This entailed developing a custom dynamic program.

We added an outer loop that decreases ε by a factor e in between calls to smoothing. The key was then to prove that each such call to smoothing uses only a constant number of first-order iterations (gradient steps). It follows immediately that the overall algorithm runs in $\mathcal{O}(\log 1/\varepsilon)$ first-order iterations (theorem 4).

Experiments verified that this algorithm converges faster than EGT, and the speed difference increases systematically the smaller ε is. Current work includes developing robust software implementations of this algorithm that can scale to the large.

Our $\mathcal{O}(\log 1/\varepsilon)$ convergence rate matches the best known convergence rate: that of interior-point methods. At the same time, our algorithm — being a first-order method — is scalable while current interior-point methods require a prohibitive amount of memory.

Algorithm Based on Counterfactual Regret Minimization (CFR)

Soon after we developed an EGT-based algorithm for sequential games, another algorithm, counterfactual regret (CFR) was introduced that can also find an ε-equilibrium with small ε in games with 10^{12} leaves in the game tree (Zinkevich et al. 2007). It is based on totally different principles than the gradient-based equilibrium-finding algorithms described above. Specifically, it is based on regret minimization principles, and it is crafted cleverly so they can be applied at individual information sets (usually they are employed in the space of strategies; that would not scale to large sequential games).

The *average overall regret* for agent *i*, R_i^N is how much better off *i* would have been on average in repetitions 1..*N* of the game by using his or her best fixed strategy than by having played the way he or she did. If both players have average overall regret less than ε, then Agent 1's time-averaged strategy and Agent 2's time-averaged strategy constitute a 2ε-equilibrium.

The design by Zinkevich et al. starts by studying one information set *I* and player *i*'s choices made in *I*. Define *counterfactual utility* $u_i(\sigma, I)$ to be the expected utility given that information set *I* is reached and all players play using strategy σ except that player *i* plays to reach *I*. Formally, letting $\pi^\sigma(h, h')$ be the probability of going from history *h* to *h*′, and letting *Z* be the set of terminal histories:

$$u_i(\sigma, I) = \frac{\sum_{h \in I, h' \in Z} \pi^\sigma_{-i}(h)\pi^\sigma(h, h')u_i(h')}{\pi^\sigma_{-i}(I)}$$

For all actions $a \in A(I)$, define $\sigma|_{I \to a}$ to be a strategy profile like σ except that player *i* always chooses action *a* in *I*. The *immediate counterfactual regret* is

$$R^N_{i,imm}(I) = \frac{1}{N}\max_{a\in\mathcal{A}(I)}\sum_{t=1}^{N}\pi^{\sigma^t}_{-i}(I)(u_i(\sigma^t|_{I\to a}, I) - u_i(\sigma^t, I))$$

This is the player's regret in its decisions at I in terms of counterfactual utility, with an additional weighting term for the counterfactual probability that I would be reached *if the player had tried to do so*. Denoting

$$R^{N,+}_{i,imm}(I) = \max\{R^N_{i,imm}(I), 0\}$$

it turns out that that the average overall regret can be bounded by the local ones:

$$R^N_i \le \sum_{I\in\mathcal{I}_i} R^{N,+}_{i,imm}(I).$$

The key is that immediate counterfactual regret can be minimized by controlling only $\sigma_i(I)$. The CFR algorithm maintains for all $I \in \mathcal{I}_i$, for all $a \in \mathcal{A}(I)$

$$R^N_i(I, a) = \frac{1}{N}\sum_{t=1}^{N}\pi^{\sigma^t}_{-i}(I)(u_i(\sigma^t|_{I\to a}, I) - u_i(\sigma^t, I))$$

Let the positive counterfactual regret be

$$R^{N,+}_i(I, a) = \max\{R^N_i(I, a), 0\}.$$

The CFR algorithm simulates the players playing the game repeatedly. Actions for each agent are selected in proportion to their positive counterfactual regret. (If no action has positive counterfactual regret, then the action is selected randomly.) The output is a strategy for each agent; it is the time-averaged strategy computed from the agent's strategies in repetitions 1..N.

Let Δ_i be the difference between i's highest and lowest payoff in the game. With the CFR algorithm, $R^N_{i,imm}(I) \le \Delta_i\sqrt{|\mathcal{A}_i|}/\sqrt{N}$ and thus $R^N_i \le \Delta_i|\mathcal{I}_i|\sqrt{|\mathcal{A}_i|}/\sqrt{N}$, where $|\mathcal{A}_i| = \max_{h:Player(h)=i}|\mathcal{A}(h)|$.

Theorem 5.

(Zinkevich et al. 2007.)

CFR can be sped up by sampling the chance moves, that is, sampling bucket sequences in the abstraction, rather than considering all the histories that lead to the information set that is being updated. This enables approximately 750 iterations per second on a Texas Hold'em abstraction, and yields a smaller ε faster (Zinkevich et al. 2007). Other forms of sampling can also be used (Lanctot et al. 2009). By theorem 5, CFR runs in $\mathcal{O}(1/\varepsilon^2)$ iterations (and the chance sampling incurs only a linear increase in the number of iterations) which is significantly worse than the $\mathcal{O}(1/\varepsilon)$ and $\mathcal{O}(\log(1/\varepsilon))$ guarantees of the smoothed gradient based techniques described above. In practice, on Texas Hold'em abstractions with 10^{12} leaves, CFR is run for about a billion sampled iterations while EGT needs to be run only for about a hundred iterations. On the other hand, each sampled CFR iteration runs much faster than an iteration of those other algorithms: EGT takes hours per iteration (even when the matrix-vector products are parallelized across 96 cores). Our initial direct comparisons between EGT and CFR on small and medium-sized games show that either can have a significant overall speed advantage over the other depending on the game.

A Practical Use of Imperfect Recall

All of the results above are for games with perfect recall, which includes most games played by computers. A recent idea has been to model a game of perfect recall like poker as a game of imperfect recall by allowing the player to use fine-grained abstraction at the poker round that she is in and then forgetting some of those details of that round so as to be able to have a larger branching factor in the information abstraction for the next round while keeping the overall abstracted game size manageable (Waugh et al. 2009b). Such imperfect recall abstracted games have been approached using CFR, but there are no convergence guarantees.

Equilibrium-Finding Algorithms for Multiplayer and Nonzero-Sum Games

While the abstraction methods discussed earlier are for n-player general-sum games, the equilibrium-finding algorithms above are for two-player zero-sum games. The problem of finding a Nash equilibrium in two-player general-sum games is PPAD-complete even in normal form games with complete information (Daskalakis, Goldberg, and Papadimitriou 2008; Chen and Deng 2006), suggesting that there likely is no polynomial-time algorithm for all instances. The PPAD-completeness holds even when payoffs are restricted to be binary (Abbott, Kane, and Valiant 2005). With three or more players, the problem becomes FIXP-complete even in the zero-sum case (Etessami and Yannakakis 2007).

To my knowledge the best equilibrium-finding algorithms for general multiplayer incomplete-information games are continuation methods (Govindan and Wilson 2003). They perturb the

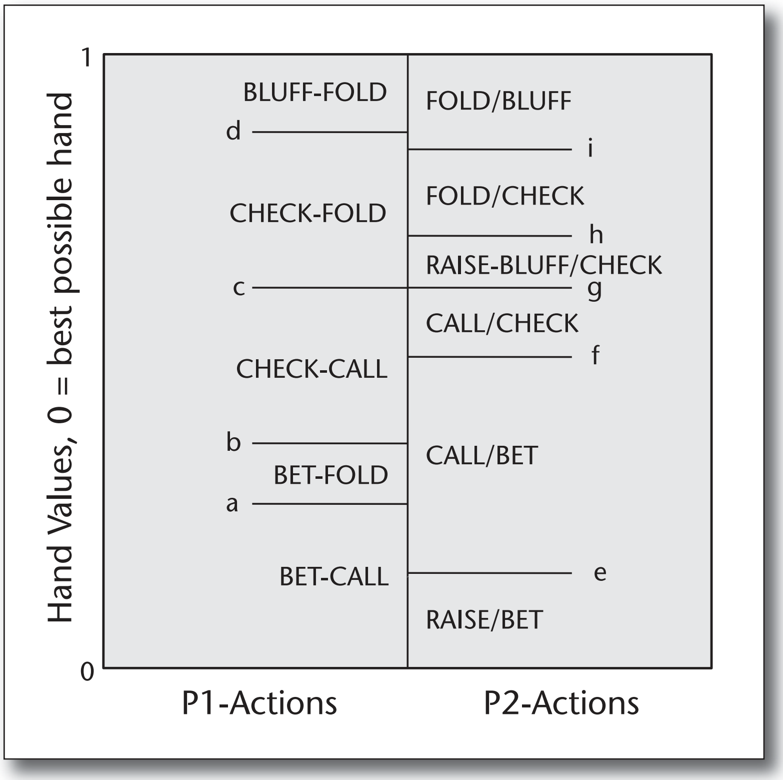

Figure 6. A Qualitative Model for a Simplified Poker Game.

(Ganzfried and Sandholm 2010). Player 1's action regions are on the left, Player 2's on the right.

game by giving agents fixed bonuses, scaled by λ, for each of their actions. If the bonuses are large enough (and unique), they dominate the original game, so the agents need not consider their opponents' actions. There is thus a unique pure-strategy equilibrium easily determined at $\lambda = 1$. The continuation method can then be used to follow a path in the space of λ and equilibrium profiles for the resulting perturbed game, decreasing λ until it is zero, at which point the original game has been solved. The algorithm scales better for games that can be represented in a structured way using multiagent influence diagrams (Blum, Shelton, and Koller 2006). However, even then it has only been applied to relatively small games.

Leveraging Qualitative Models

A recent idea that scales to significantly larger games takes advantage of the fact that in many settings it is easier to infer qualitative models about the structure of equilibrium than it is to actually compute an equilibrium. For example, in (sequences of) take-it-or-leave-it offers, equilibria involve accepting offers above a certain threshold and rejecting offers below it. Threshold strategies are also common in auctions and in deciding when to make and break partnerships and contracts. In poker, the cards in the hand are private signals, and in equilibrium, often the same action is taken in continuous regions of the signal space (for example, Ankenman and Chen [2006]). The idea of using qualitative models as an extra input for equilibrium finding has been applied to continuous (and finite) multiplayer Bayesian games, a broad class of imperfect-information games that includes many variants of poker (Ganzfried and Sandholm 2010). Figure 6 shows an example of a qualitative model for a simplified poker game.

Given a qualitative model that is correct, there is a mixed integer linear feasibility program (MILFP) that finds an equilibrium (a mixed strategy equilibrium in games with a finite number of types and a pure strategy equilibrium in games with a continuum of types) (Ganzfried and Sandholm 2010). The paper also presents extensions of the algorithm to games with dependent private signal distributions, many players, and multiple candidate qualitative models of which only some are correct. Experiments show that the algorithm can still compute an equilibrium even when it is not clear whether any of the models are correct, and an efficient procedure is given for checking the output in the event that they are not correct. The MILFP finds an exact equilibrium in two-player games, and an ε-equilibrium in multiplayer games. It also yields a MILFP for solving general multiplayer imperfect-information games given in extensive form without any qualitative models. For most of these games classes, no prior algorithm was known.

Experiments suggest that modeling a finite game with an infinite one with a continuum of types can significantly outperform abstraction-based approaches on some games. Thus, if one is able to construct a correct qualitative model, solving the MILFP formulation of the infinite approximation of a game could potentially be the most efficient approach to solving certain classes of large finite games (and the only approach to solving the infinite version). The main algorithm was used to improve play in two-player limit Texas hold'em by solving endgames. In addition, experiments demonstrated that the algorithm was able to efficiently compute equilibria in several infinite three-player games.

Solving Multiplayer Stochastic Games of Imperfect Information

Significant progress has also been made on equilibrium finding in multiplayer stochastic games of imperfect information (Ganzfried and Sandholm 2008; 2009). For example, consider a No-Limit Texas Hold'em tournament. The best way to play differs from how one should play an individual hand because there are considerations of bankroll management (one gets eliminated once one runs out of chips) and the payoffs are based on ranks in the tournament rather than chips. This becomes especially important near the end of a tournament (where the antes are large). One simple strategy restriction is to always go all-in or fold in Round 1 (that is, once the private cards have been dealt but no public cards have). In the two-player case, the best strategy in that restricted space is almost optimal against an unrestricted opponent (Miltersen and Sørensen 2007). It turns out that if all players are restricted in this way, one can find an ε-equilibrium for the multiplayer game (Ganzfried and Sandholm 2008; 2009). The algorithms have an inner loop to determine ε-equilibrium strategies for playing a hand at a given state (stack vector, one stack of chips per player) given the values of possible future states. This is done for all states. The iteration of the outer loop adjusts the values of the different states in light of the new payoffs obtained from the inner loop. Then the inner loop is executed again until convergence, then the outer loop, and so on.

For instance, fictitious play can be used for the inner loop and policy iteration for solving Markov decision processes for the outer loop. Several other variants were also studied. None of the variants are guaranteed to converge, but some of them have the property that if they converge, they converge to an equilibrium. In practice, both the inner and outer loop converge quickly in all of the tested variants. This suggests that fictitious play is another promising algorithm for multiplayer imperfect-information games.

Opponent Exploitation

So far I have discussed approaches based on game theory. A totally different approach is to try to learn to exploit opponents.

Two-player zero-sum games have the nice property that our player can only benefit if the opponent does not play an equilibrium strategy. Furthermore, it does not matter which equilibrium strategies are selected: if (x, y) and (x', y') are equilibria, then so are (x, y') and (x', y). However, even in two-player zero-sum games, an equilibrium strategy might not maximally exploit an opponent that does not play equilibrium. In multiplayer games, there is the further complications of equilibrium selection.

There is a long history of opponent-exploitation research in AI (for example, by building models of opponents). That has also been studied for poker (for example, Billings et al. [1998], Southey et al. [2005], and Bard and Bowling [2007]). In practice — at least in large games like Texas Hold'em (even in the two-player case), even with relatively large numbers of hands to learn from — those approaches are far inferior to the game-theory-based approaches. For example, our poker player that was constructed using potential-aware abstraction and EGT, and used no learning, won the Bankroll Competition in the AAAI 2008 Computer Poker Competition. This was noteworthy because the Bankroll Competition is designed to favor learning programs that can take advantage of weak opponents.

One weakness in the learning approach is the *get-taught-and-exploited problem* (Sandholm 2007): An opponent might play in a way to teach the learner a model, and then exploit the learner that attempts to use that model. Furthermore, the opponent might lose significantly less from the teaching than he gains from the exploitation.

One recent approach that has been pursued both at University of Alberta's poker research group (Johanson, Zinkevich, and Bowling 2007; Johanson and Bowling 2009) and mine is to start with a game-theory-based strategy and then adjust it in limited ways to exploit the opponent as we learn more about the opponent. This already yielded a win in the Bankroll Competition in the AAAI 2009 Computer Poker Competition, and is a promising direction for the future.

There are some fundamental limits, however. Can this be done in a safe way? That is, can one exploit to some extent beyond the game-theoretic equilibrium strategy while still maintaining at least the same expected payoff as the equilibrium strategy? Recently Sam Ganzfried and I proved that this is impossible. So, in order to increase exploitation, one needs to sacrifice some on the game-theoretic safety guarantee.

Additional Topics

Beyond what I discussed so far, there are other interesting developments in the computation of solutions to incomplete-information games. Let me briefly discuss some of them here.

One question is whether Nash equilibrium is the right solution concept. In two-player zero-sum games it provides a safety guarantee as discussed above, but in more general games it does not because equilibrium selection can be an issue. Even in two-player zero-sum games, the equilibrium strategy may not play the rest of the game optimally if the opponent makes a mistake. Various equilibrium refinements can be used to prune such equilibrium strategies from consideration, and there has been some work on computing equilibrium strategies that honor such refinements (for example, (Miltersen and Sørensen 2010; 2006; 2008)). In multiplayer games there is also the possibility of collusion (coordination of strategies and/or information), and there are coalitional equilibrium refinements (for example, Milgrom and Roberts [1996], Moreno and Wooders [1996], and Ray [1996]).

There is also work on other general classes of games. An optimal polynomial algorithm was recently developed for repeated incomplete-information games (Gilpin and Sandholm 2008b). Work has also been done on Kriegspiel (chess where the players do not observe each others' moves), but the best-performing techniques are still based on sampling of the game tree rather than game-theoretic approaches (Ciancarini and Favini 2009). There has been work on computing commitment (Stackelberg) strategies (where Player 1 has to commit to a mixed strategy first, and then Player 2 picks a strategy) in normal form games, with significant security applications (for example, Conitzer and Sandholm [2006]; Jain et al. [2010]). Recently that was studied also in sequential incomplete-information games (Letchford and Conitzer 2010; Kiekintveld, Tambe, and Marecki 2010).

Conclusions

There has been tremendous progress on solving incomplete-information games in the last five years. For some rather large games like two-player Rhode Island Hold'em, an optimal strategy has been computed. An optimal strategy is not yet known for any variant of Texas Hold'em, but in two-player Limit Texas Hold'em — a game that is frequently used in competitions among professional poker players — computers have surpassed humans. In the No-Limit and multiplayer variants humans remain ahead.

This is a very active and fertile research area. I hope this article helps newcomers enter the field, spurs interest in further pushing the boundary of what is computationally possible, and facilitates adoption of these approaches to additional games of importance, for example, negotiation, auctions, and various security applications.

Acknowledgments

I want to thank Andrew Gilpin, Javier Peña, Sam Ganzfried, Troels Bjerre Sørensen, and Sam Hoda. Their contributions to this research were essential. I also thank Kevin Waugh for conducting experiments between EGT and CFR while he was in my laboratory.

This material is based upon work supported by the National Science Foundation under grants ITR-0427858, IIS-0905390, and IIS-0964579. I also thank IBM and Intel for their machine gifts, and the Pittsburgh Supercomputing Center for compute time and help.

Notes

1. The reader is invited to play against this strategy at www.cs.cmu.edu/~gilpin/gsi.html.

2. An extreme form of action abstraction is to restrict analysis to a small number of strategies and conduct equilibrium analysis among them (Wellman 2006).

3. The operator norm of A is defined as $\|A\| := \max\{\mathbf{y}^T A\mathbf{x}: \|\mathbf{x}\|, \|\mathbf{y}\| \leq 1\}$, where the norms $\|\mathbf{x}\|$, $\|\mathbf{y}\|$ are those associated with σ_X and σ_Y.

References

Abbott, T.; Kane, D.; and Valiant, P. 2005. On the Complexity of Two-Player Win-Lose Games. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS-05).* Los Alamitos, CA: IEEE Computer Society.

Ankenman, J., and Chen, B. 2006. *The Mathematics of Poker.* Pittsburgh, PA: ConJelCo LLC.

Bard, N., and Bowling, M. 2007. Particle Filtering for Dynamic Agent Modelling in Simplified Poker. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence* (AAAI-07), 515–521. Menlo Park, CA: AAAI Press.

Billings, D.; Burch, N.; Davidson, A.; Holte, R.; Schaeffer, J.; Schauenberg, T.; and Szafron, D. 2003. Approximating Game-Theoretic Optimal Strategies for Full-scale Poker. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence* (IJCAI-03). San Francisco: Morgan Kaufmann Publishers.

Billings, D.; Davidson, A.; Schaeffer, J.; and Szafron, D. 2002. The Challenge of Poker. *Artificial Intelligence* 134(1-2): 201–240.

Billings, D.; Papp, D.; Schaeffer, J.; and Szafron, D. 1998. Opponent Modeling in Poker. In *Proceedings of the 15th National Conference on Artificial Intelligence* (AAAI-98), 493–499. Menlo Park, CA: AAAI Press.

Blum, B.; Shelton, C. R.; and Koller, D. 2006. A Continuation Method for Nash Equilibria in Structured Games. *Journal of Artificial Intelligence Research* 25: 457–502.

Chen, X., and Deng, X. 2006. Settling the Complexity of 2-Player Nash Equilibrium. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS).* Los Alamitos, CA: IEEE Computer Society.

Ciancarini, P., and Favini, G. P. 2009. Monte Carlo Tree Search Techniques in the Game of Kriegspiel. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence* (IJCAI-09). Menlo Park, CA: AAAI Press.

Conitzer, V., and Sandholm, T. 2006. Computing the Optimal Strategy to Commit to. In *Proceedings of the 11th ACM Conference on Electronic Commerce,* 83–92. New York: Association for Computing Machinery.

Daskalakis, C.; Goldberg, P.; and Papadimitriou, C. 2008. The Complexity of Computing a Nash Equilibrium. *SIAM Journal on Computing* 39(1): 195–259

Etessami, K., and Yannakakis, M. 2007. On the Complexity of Nash Equilibria and Other Fixed Points (Extended Abstract). In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS-07)*, 113–123. Los Alamitos, CA: IEEE Computer Society.

Ganzfried, S., and Sandholm, T. 2010. Computing Equilibria by Incorporating Qualitative Models. In *Proceedings of the Ninth International Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2009).* Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Ganzfried, S., and Sandholm, T. 2009. Computing Equilibria in Multiplayer Stochastic Games of Imperfect Information. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09)*. Menlo Park, CA: AAAI Press.

Ganzfried, S., and Sandholm, T. 2008. Computing an Approximate Jam/Fold Equilibrium for 3-Agent No-Limit Texas Hold'em Tournaments. In *Proceedings of the 8th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Gilpin, A., and Sandholm, T. 2010. Speeding Up Gradient-Based Algorithms for Sequential Games. In *Proceedings of the Ninth International Conference on Autonomous Agents and MultiAgent Systems* (AAMAS 2010). Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Gilpin, A., and Sandholm, T. 2008a. Expectation-Based Versus Potential-Aware Automated Abstraction in Imperfect Information Games: An Experimental Comparison Using Poker. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-08)*. Short paper. Menlo Park, CA: AAAI Press.

Gilpin, A., and Sandholm, T. 2008b. Solving Two-Person Zero-Sum Repeated Games of Incomplete Information. In *Proceedings of the Seventh International Conference on Autonomous Agents and MultiAgent Systems* (AAMAS 2008). Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Gilpin, A., and Sandholm, T. 2007a. Better Automated Abstraction Techniques for Imperfect Information Games, with Application to Texas Hold'em Poker. In *Proceedings of the 6th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2007)*, 1168–1175. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Gilpin, A., and Sandholm, T. 2007b. Lossless Abstraction of Imperfect Information Games. *Journal of the ACM* 54(5): 1–32.

Gilpin, A., and Sandholm, T. 2006. A Competitive Texas Hold'em Poker Player via Automated Abstraction and Real-Time Equilibrium Computation. In *Proceedings of the 21st National Conference on Artificial Intelligence* (AAAI-06), 1007–1013. Menlo Park, CA: AAAI Press.

Gilpin, A.; Hoda, S.; Peña, J.; and Sandholm, T. 2007. Gradient-Based Algorithms for Finding Nash Equilibria in Extensive Form Games. In *Proceedings of the 3rd International Workshop on Internet and Network Economics (WINE '07)*. Berlin: Springer-Verlag.

Gilpin, A.; Peña, J.; and Sandholm, T. 2008. First-Order Algorithm with $O(\log(1/\varepsilon))$ Convergence for ε-Equilibrium in Games. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-08)*. Menlo Park, CA: AAAI Press.

Gilpin, A.; Sandholm, T.; and Sørensen, T. B. 2008. A Heads-Up No-Limit Texas Hold'em Poker Player: Discretized Betting Models and Automatically Generated Equilibrium-Finding Programs. In *Proceedings of the Seventh International Conference on Autonomous Agents and MultiAgent Systems* (AAMAS 2008). Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Gilpin, A.; Sandholm, T.; and Sørensen, T. B. 2007. Potential-Aware Automated Abstraction of Sequential Games, and Holistic Equilibrium Analysis of Texas Hold'em Poker. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence* (AAAI-07). Menlo Park, CA: AAAI Press.

Goffin, J.-L. 1977. On the Convergence Rate of Subgradient Optimization Methods. *Mathematical Programming* 13(1): 329–347.

Govindan, S., and Wilson, R. 2003. A Global Newton Method to Compute Nash Equilibria. *Journal of Economic Theory* 110(1): 65–86.

Hoda, S.; Gilpin, A.; Peña, J.; and Sandholm, T. 2010. Smoothing Techniques for Computing Nash Equilibria of Sequential Games. *Mathematics of Operations Research* 35(2): 494–512.

Jain, M.; Tsai, J.; Pita, J.; Kiekintveld, C.; Rathi, S.; Ordóñez, F.; and Tambe, M. 2010. Software Assistants for Randomized Patrol Planning for The LAX Airport Police and The Federal Air Marshals Service. *Interfaces* 40(4): 267–290.

Johanson, M., and Bowling, M. 2009. Data Biased Robust Counter Strategies. Paper presented at the 12th International Conference on Artificial Intelligence and Statistics (AISTATS), Clearwater Beach, FL. 16–18 April.

Johanson, M.; Zinkevich, M.; and Bowling, M. 2007. Computing Robust Counter-Strategies. In *Proceedings of the 21st Annual Conference on Neural Information Processing Systems* (NIPS 2007). Cambridge, MA: The MIT Press.

Kiekintveld, C.; Tambe, M.; and Marecki, J. 2010. Robust Bayesian Methods for Stackelberg Security Games (Extended Abstract). In *Proceedings of the Ninth International Conference on Autonomous Agents and MultiAgent Systems* (AAMAS 2010). Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Koller, D., and Pfeffer, A. 1997. Representations and Solutions for Game-Theoretic Problems. *Artificial Intelligence* 94(1): 167–215.

Koller, D.; Megiddo, N.; and von Stengel, B. 1996. Efficient Computation of Equilibria for Extensive Two-Person Games. *Games and Economic Behavior* 14(2): 247–259.

Kuhn, H. W. 1950. A Simplified Two-Person Poker. In *Contributions to the Theory of Games,* Annals of Mathematics Studies 24, ed. H. W. Kuhn and A. W. Tucker, volume 1, 97–103. Princeton, NJ: Princeton University Press.

Lanctot, M.; Waugh, K.; Zinkevich, M.; and Bowling, M. 2009. Monte Carlo Sampling for Regret Minimization in Extensive Games. In *Proceedings of the 23rd Annual Conference on Neural Information Processing Systems* (NIPS 2009), 1078–1086. Cambridge, MA: The MIT Press.

Letchford, J., and Conitzer, V. 2010. Computing Optimal Strategies to Commit to in Extensive-Form Games. In *Proceedings of the 11th ACM Conference on Electronic Commerce,* 83–92. New York: Association for Computing Machinery.

Milgrom, P., and Roberts, J. 1996. Coalition-Proofness and Correlation with Arbitrary Communication Possibilities. *Games and Economic Behavior* 17(1): 113–128.

Miltersen, P. B., and Sørensen, T. B. 2010. Computing a Quasi-Perfect Equilibrium of a Two-Player Game. *Economic Theory* 42(1): 175–192.

Miltersen, P. B., and Sørensen, T. B. 2008. Fast Algorithms for Finding Proper Strategies in Game Trees. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA),* 874–883. Philadelphia, PA: Society for Industrial and Applied Mathematics.

Miltersen, P. B., and Sørensen, T. B. 2007. A Near-Optimal Strategy for a Heads-Up No-Limit Texas Hold'em Poker Tournament. In *Proceedings of the Sixth International Conference on Autonomous Agents and MultiAgent Systems* (AAMAS 2007). Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Miltersen, P. B., and Sørensen, T. B. 2006. Computing Proper Equilibria of Zero-Sum Games. In *Proceedings of the 5th International Conference on Computers and Games*. Lecture Notes in Computer Science. Berlin: Springer-Verlag.

Moreno, D., and Wooders, J. 1996. Coalition-Proof Equilibrium. *Games and Economic Behavior* 17(1): 80–112.

Nemhauser, G., and Wolsey, L. 1999. Integer and Combinatorial Optimization. John Wiley & Sons.

Nesterov, Y. 2005a. Excessive Gap Technique in Nonsmooth Convex Minimization. *SIAM Journal of Optimization* 16(1): 235–249.

Nesterov, Y. 2005b. Smooth Minimization of NonSmooth Functions. Mathematical Programming 103:127–152.

Ray, I. 1996. Coalition-Proof Correlated Equilibrium: A Definition. *Games and Economic Behavior* 17(1): 56–79.

Romanovskii, I. 1962. Reduction of a Game with Complete Memory to a Matrix Game. *Soviet Mathematics* 3: 678–681.

Sandholm, T. 2007. Perspectives on Multiagent Learning. *Artificial Intelligence* 171(7): 382–391.

Schnizlein, D.; Bowling, M.; and Szafron, D. 2009. Probabilistic State Translation in Extensive Games with Large Action Sets. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence* (IJCAI-2009). Menlo Park, CA: AAAI Press.

Shi, J., and Littman, M. 2002. Abstraction Methods for Game Theoretic Poker. In *Revised Papers from the Second International Conference on Computers and Games,* Lecture Notes in Computer Science, 333–345. Berlin: Springer-Verlag.

Sklansky, D. 1999. *The Theory of Poker,* fourth edition. Las Vegas, NV: Two Plus Two Publishing.

Southey, F.; Bowling, M.; Larson, B.; Piccione, C.; Burch, N.; Billings, D.; and Rayner, C. 2005. Bayes' Bluff: Opponent Modelling in Poker. In *Proceedings of the 21st Annual Conference on Uncertainty in Artificial Intelligence* (UAI 2005), 550–558. Redmond, WA: AUAI Press.

von Stengel, B. 1996. Efficient Computation of Behavior Strategies. *Games and Economic Behavior* 14(2): 220–246.

Waugh, K.; Bard, N.; and Bowling, M. 2009. Strategy Grafting in Extensive Games. In *Proceedings of the 23rd Annual Conference on Neural Information Processing Systems* (NIPS 2009). Cambridge, MA: The MIT Press

Waugh, K.; Schnizlein, D.; Bowling, M.; and Szafron, D. 2009a. Abstraction Pathologies in Extensive Games. In *Proceedings of the 8th International Conference on Autonomous Agents and MultiAgent Systems* (AAMAS 2009). Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Waugh, K.; Zinkevich, M.; Johanson, M.; Kan, M.; Schnizlein, D.; and Bowling, M. 2009b. A Practical Use of Imperfect Recall. In *Proceedings of the 8th Symposium on Abstraction, Reformulation and Approximation* (SARA2009). Menlo Park, CA: AAAI Press.

Wellman, M. 2006. Methods for Empirical Game-Theoretic Analysis (Extended Abstract). In *Proceedings of the 21st AAAI Conference on Artificial Intelligence* (AAAI-2006), 1552–1555. Menlo Park, CA: AAAI Press. Menlo Park, CA: AAAI Press.

Zinkevich, M.; Bowling, M.; Johanson, M.; and Piccione, C. 2007. Regret Minimization in Games with Incomplete Information. In *Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems* (NIPS 2007). Cambridge, MA: The MIT Press.

Tuomas Sandholm is a professor in the Computer Science Department at Carnegie Mellon University. He has published more than 400 papers on artificial intelligence, game theory, electronic commerce, multiagent systems, auctions and exchanges, automated negotiation and contracting, coalition formation, voting, search and integer programming, safe exchange, normative models of bounded rationality, resource-bounded reasoning, machine learning, and networks. He has 20 years of experience building optimization-based electronic marketplaces and has fielded several of his systems. He was founder, chairman, and CTO/chief scientist of CombineNet, Inc. from 1997 until its acquisition in 2010. During this period the company commercialized over 800 large-scale generalized combinatorial auctions, with over $50 billion in total spending and over $6 billion in generated savings. His technology also runs the nationwide kidney exchange. He received his Ph.D. and M.S. degrees in computer science from the University of Massachusetts at Amherst in 1996 and 1994. He earned an M.S. (B.S. included) with distinction in industrial engineering and management science from the Helsinki University of Technology, Finland, in 1991. He is recipient of the NSF Career Award, the inaugural ACM Autonomous Agents Research Award, the Alfred P. Sloan Foundation Fellowship, the Carnegie Science Center Award for Excellence, and the Computers and Thought Award. He is Fellow of the ACM and AAAI.