# Algorithms for abstracting and solving imperfect information games

Thesis Proposal

Andrew Gilpin
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

**Committee:**
Tuomas Sandholm (CMU, CSD), Chair
Avrim Blum (CMU, CSD)
Geoff Gordon (CMU, MLD)
Javier Peña (CMU, Tepper School of Business)
Bernhard von Stengel (LSE, Department of Mathematics)

April 13, 2007

**Abstract**

Game theory is the mathematical study of rational behavior in strategic environments. In many settings, most notably two-person zero-sum games, game theory provides particularly strong and appealing solution concepts. Furthermore, these solutions are efficiently computable in the complexity-theory sense. However, in most interesting potential applications in artificial intelligence, the solutions are difficult to compute using current techniques due primarily to the extremely large state-spaces of the environments.

In this thesis, we propose new algorithms for tackling these computational difficulties. In one stream of research, we introduce *automated abstraction algorithms for sequential games of imperfect information.* These algorithms take as input a description of a game and produce a description of a strategically similar, but smaller, game as output. We present algorithms that are lossless (*i.e.,* equilibrium-preserving), as well as algorithms that are lossy, but which can yield much smaller games while still retaining the most important features of the original game.

In a second stream of research, we develop *specialized optimization algorithms for finding $\epsilon$-equilibria in sequential games of imperfect information.* The algorithms are based on recent advances in non-smooth convex optimization (namely the excessive gap technique) and provide significant improvements over previous algorithms for finding $\epsilon$-equilibria.

Combining these two streams, we enable the application of game theory to games extremely larger than was previously possible. As in illustrative example, we find near-optimal solutions for a four-round model of Texas Hold'em poker, and demonstrate that the resulting player is significantly better than previous computer poker players.

In addition to the above (already completed) work, we discuss how the same techniques can be used to construct an agent for no-limit Texas Hold'em poker (a game with an infinite number of pure strategies). We propose coming up with worst-case guarantees (both *ex ante* and *ex post*) for automated abstraction algorithms. We also propose a regret-minimizing pure strategy solution concept appropriate for sequential games with many players, and propose an algorithm for computing this concept. Finally, we propose specialized interior-point algorithms for equilibrium computation in extensive form games (possibly for computing equilibrium refinements such as sequential equilibrium) as well as a prioritized updating scheme for speeding up the excessive gap technique family of algorithms.

# 1 Introduction

In settings with multiple, self-interested agents, the outcome for each individual agent depends on the actions of the other agents in the system. Consequently, *rational* and *optimal* strategies for each agent also depend on the other agents' actions. In order for an agent to achieve their best possible outcome it is necessary to take the other agents' preferences and strategies into account during the deliberation process.

Game theory is the mathematical framework that enables the study of rational behavior in competitive multiagent environments. *Inter alia*, game theory defines *solution concepts* that provide prescriptive behavior (*i.e.,* strategies) for each agent. The *Nash equilibrium* [87] is the most prevalent such solution concept. In a Nash equilibrium, no agent has any incentive to deviate to any other strategy.

In some settings, the algorithms for finding (or approximating) Nash equilibria are straightforward. In fact, there has been an enormous amount of research on algorithms for two-person perfect information games (sometimes called *combinatorial games* [7]). In these games, applying minimax search (possibly with $\alpha$-$\beta$-pruning) actually yields a Nash equilibrium (assuming that the game tree is completely searched and no internal nodes are replaced by leaves according to some evaluation function) [104]. Perhaps without realizing it, the researchers that developed these algorithms were motivated by a line of reasoning that is analogous to the reasoning behind the Nash equilibrium: the best action for one agent largely depends on the best actions for the other agents. However, as we will discuss below, these algorithms are not applicable to many interesting, real-world games.

Developing expert-level game-playing computer agents has long been a major focus of the artificial intelligence (AI) community. The most notable successes include *Chinook*, which defeated the checkers world champion Dr. Marion Tinsley in 1992 [111]; *Deep Blue*, which defeated Garry Kasparov, the chess world champion in 1997; and *TD-Gammon*, the best backgammon-playing program in the world [125]. (See [112] for a survey of other AI success stories in game-playing.) These are all very impressive applications of AI techniques and have done much to advance the standing of AI in the wider science community. However, each of these three games possess the property known as *perfect information*, *i.e.*, at any point in time, both players are fully informed about the state of the world. In contrast, most interesting potential application areas of game theory have the property of *imperfect information*: at most stages of the game, the players are only partially informed about the state of the world. Examples include poker and most other card games (in which each player does not know the cards held by the other players), economic environments (in which each player does not know the other players' preferences), and adversarial robot environments (in which each robot does not know the locations and goals of the other robots). Due to this informational difference, algorithms for perfect information games are unhelpful when designing agents for games with imperfect information.

In the last 15 years, there has been a surge of research with the goal of developing the theory and algorithms for finding equilibria in sequential games with imperfect information [61, 129, 63, 62, 65, 64]. Among other breakthroughs, it is now well-known that one can compute a Nash equilibrium in a two-person zero-sum sequential game with imperfect information in time polynomial in the size of the game tree. The prescribed method for solving this problem is to model the game as a linear program and solve for the equilibrium using general-purpose linear programming tools. However, for most interesting applications in AI, these tools do not scale. In this thesis we propose two complementary streams of research to tackle this problem.

1. We introduce *automated abstraction algorithms for sequential games of imperfect information* as a method for finding (nearly) equivalent, smaller representations of games on which the equilibrium analysis may be carried out.

2. We improve the equilibrium finding algorithms themselves via the development of *specialized optimization algorithms for finding approximate equilibria in sequential games of imperfect information.*

Combining these approaches enables the application of game theory to games many orders of magnitude larger than previously possible. In the remainder of this section, we motivate our main application area (poker), give the thesis statement, and summarize the research proposed in the remainder of this document.

## 1.1 Poker and Artificial Intelligence

Poker is an enormously popular card game that has stood the test of time. The strategies employed by expert players can be extremely sophisticated [119]. A poker player cannot succeed with just the ability to compute odds and probabilities; they also need to utilize randomized (information-hiding) strategies that attempt to deceive the opponent. When performing actions, successful poker players need to consider not only what possible private information their opponent knows, but also what their own action reveals about their own private information. Thus, players must speculate, counter-speculate, counter-counter-speculate, *etc.*, about what their actions are achieving and what they are revealing. Game theory, via its various equilibrium concepts, is particularly well-suited to providing definitive answers in these types of situations.

In addition to the challenging research issues presented by poker, it is a particularly attractive testbed for computational game theory research. Unlike many other important games with imperfect information (*e.g.* financial markets, business-to-business interactions, political negotiations, legal disputes), there are no issues with the game model. Game theory is capable of modeling the game *precisely* as it is played. Another motivation for studying poker is that it represents a frontier of machine intelligence: while artificially intelligent agents have surpassed the skill level of humans in games such as chess, checkers, and backgammon, poker remains a game where humans are superior.

For the above reasons, as well as many others, poker has been identified as an important area of research for AI [12], and it is with these challenges in mind that we present this thesis proposal.

## 1.2 Thesis statement

> *Automated state-space abstraction in conjunction with specialized equilibrium-finding algorithms enables the construction of agents for challenging competitive environments, with robust theoretical guarantees on their performance.*

## 1.3 Summary of work completed

The following work has already been completed:

- We developed a provably lossless automated abstraction algorithm, *GameShrink*, for sequential games of imperfect information. It enabled the computation of optimal strategies for Rhode Island Hold'em poker, which at that time was the largest game solved by over four orders of magnitude [44, 46]. (See Section 3.)
- We developed approximation versions of *GameShrink* for handling even larger games. These new algorithms have been used to develop a series of players for heads-up limit Texas Hold'em poker, the latest of which, *GS3*, beats all known poker-playing programs [45, 47, 48, 49]. (See Section 4.)
- We developed specialized equilibrium-finding algorithms based on recent techniques developed for non-smooth convex optimization. These algorithms have enabled the solution of games four orders of magnitude larger than was previously possible using state-of-the-art linear programming solvers [54, 43]. (See Section 5.)

## 1.4 Summary of proposed work

The following work is already in progress or is proposed for future research:

- We propose to apply the above automated abstraction algorithms and specialized equilibrium-approximating algorithms to construct an agent for *no-limit* Texas Hold'em poker, a game with an infinite strategy space. We are in the process of developing techniques for discretizing the space of infinite strategies to enable the above techniques to apply. (See Section 4.5.)

- We propose to develop new theoretical frameworks and algorithms for providing worst-case guarantees for automated abstraction algorithms. We propose to investigate both *ex ante* and *ex post* guarantees. (See Section 6.)
- We propose to improve the existing excessive gap technique algorithms by incorporating a selective updating process to speed up the search. (See Section 7.)
- We propose to investigate specialized interior-point methods for computing equilibria in two-person zero-sum sequential games, and investigate the feasibility of using such techniques for computing equilibrium refinements such as sequential equilibria. (See Section 8.)
- We propose to investigate new solution concepts and algorithms for games with more than two-players. Instead of aiming for the Nash equilibrium concept (which appears to be computationally intractable for games with more than two players), we instead propose a regret-minimization approach which can be effectively applied to games with many players. In particular, we propose its application to a multi-player Texas Hold'em poker tournament. (See Section 9.)

### 1.5 Organization

Section 2 presents the necessary game theory background used in this proposal. Sections 3–9 discuss the thesis contribution. Figure 1 illustrates how the different pieces of the thesis interact. Solid boxes indicate work that has already been completed. This work is discussed in Section 3–5. Sections 4.5 and Sections 6–9 discuss future work, denoted by dotted rectangles in Figure 1. Section 10 discusses related work. Section 11 proposes a timeline for thesis completion.

## 2 Game theory

In this section we review some definitions and algorithms from game theory. Game theory is the mathematical study of decision-making in interactive, competitive environments. The most basic assumption underlying the (classical) theory involves the embodiment of rationality in individual agents via utility-function-inducing preferences. Further, the players are assumed to act in such as a way as to maximize their utility based on their knowledge about the game. In this proposal, we do not further discuss these basic assumptions, nor detail the many objections raised against them. Instead, we simply present the basic game theory necessary for understanding and evaluating this proposal. The models and solution concepts discussed in this section mirrors the development of any standard game theory text (*e.g.*, [94, 86]).

### 2.1 Extensive form games and perfect recall

Normal form games (often called matrix (resp. bimatrix) games in two-person zero-sum (resp. non-zero-sum) games) are games in which each player simultaneously chooses an action in their action set, and these choices deterministically determine the outcome of the game. Although there is a deep theory for these games, and many interesting games that naturally fit this model, they are not our main interest.

In this proposal, we are primarily interested in *sequential games*, in which players may take moves after observing moves of chance (*e.g.*, a roll of a die) and moves of the other players. This model is much more powerful in terms of modeling capability as many real-world games can be concisely represented in this model.[1] This class of games is referred to as *extensive form games* and our definition of this class is standard:

**Definition 1** *An* $n$-person game in extensive form *is a tuple* $\Gamma = \langle I, V, E, P, H, A, u, p \rangle$ *satisfying the following conditions:*

[1] In principle, any finite sequential game can be represented in the normal form by considering cross products of all possible contingency plans [69]. However, such representations lead to exponential increases in the size of the game and are not at all suitable for computational purposes [129].

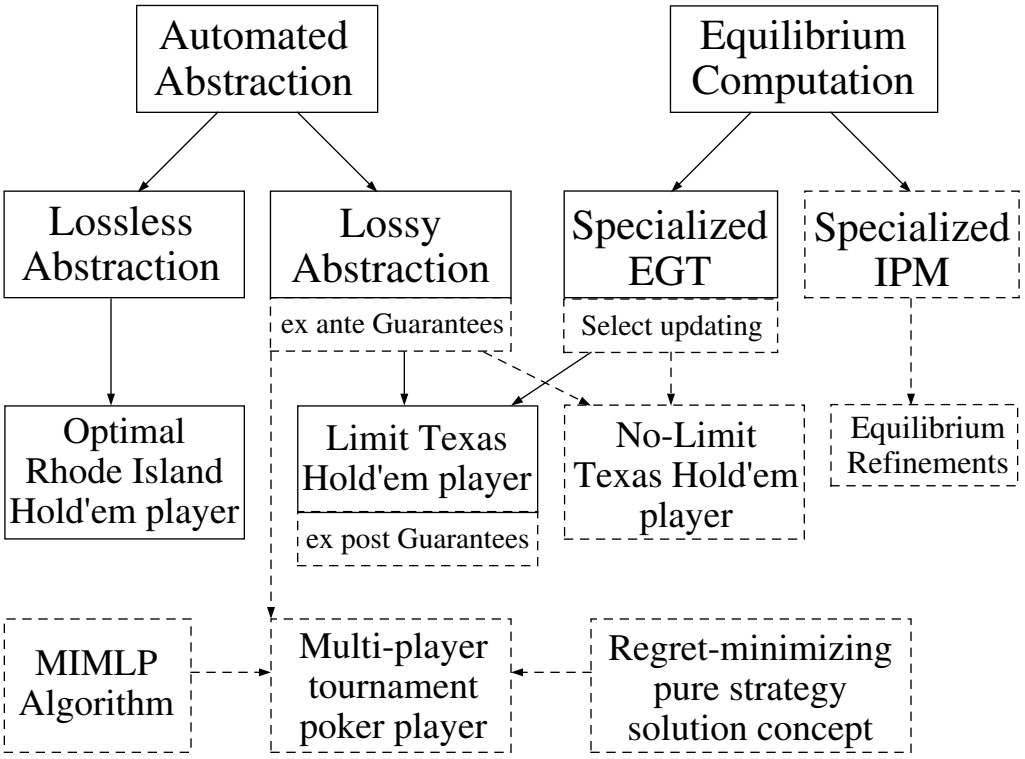


Figure 1: Graphical depiction of thesis outline.

1. $I = \{0, 1, \ldots, n\}$ *is a finite set of players. By convention, player 0 is the* chance *player.*

2. *The pair* $(V, E)$ *is a finite directed tree with nodes* $V$ *and edges* $E$*.* $Z$ *denotes the leaves of the tree, called* terminal nodes*.* $V \setminus Z$ *are* decision nodes*.* $N(x)$ *denotes* $x$*'s children and* $N^*(x)$ *denotes* $x$*'s descendants.*

3. $P : V \setminus Z \to I$ *determines which player moves at each decision node.* $P$ *induces a partition of* $V \setminus Z$ *and we define* $P_i = \{x \in V \setminus Z \,|\, P(x) = i\}$*.*

4. $H = \{H_0, \ldots, H_n\}$ *where each* $H_i$ *is a partition of* $P_i$*. For each of player* $i$*'s* information sets $h \in H_i$ *and for* $x, y \in h$*, we have* $|N(x)| = |N(y)|$*. We denote the information set of a node* $x$ *as* $h(x)$ *and the player who controls* $h$ *is* $i(h)$*.*

5. $A = \{A_0, \ldots, A_n\}$*,* $A_i : H_i \to 2^E$ *where for each* $h \in H_i$*,* $A_i(h)$ *is a partition of the set of edges* $\{(x, y) \in E \,|\, x \in h\}$ *leaving the information set* $h$ *such that the cardinalities of the sets in* $A_i(h)$ *are the same and the edges are disjoint. Each* $a \in A_i(h)$ *is called an* action *at* $h$*.*

6. $u : Z \to \mathbb{R}^N$ *is the* payoff *function. For* $x \in Z$*,* $u_i(x)$ *is the payoff to player* $i$ *in the event that the game ends at node* $x$*.*

7. $p : H_0 \times \{a \in A_0(h) \,|\, h \in H_0\} \to [0, 1]$ *where*

$$\sum_{a \in A_0(h)} p(h, a) = 1$$

*for all* $h \in H_0$ *is the transition probability for chance nodes.*

In this paper we restrict our attention to games with *perfect recall* [71], which means that players never forget information:

**Definition 2** *An n-person game in extensive form satisfies* perfect recall *if the following two constraints hold:*

1. *Every path in $(V, E)$ intersects $h$ at most once.*
2. *If $v$ and $w$ are nodes in the same information set and there is a node $u$ that precedes $v$ and $P(u) = P(v)$, then there must be some node $x$ that is in the same information set as $u$ and precedes $v$ and the paths taken from $u$ to $v$ is the same as from $x$ to $w$.*

A straightforward representation for strategies in extensive form games is the *behavior strategy* representation. This is without loss of generality since Kuhn's theorem [71] states that for any mixed strategy there is a payoff-equivalent behavioral strategy in games with perfect recall. For each information set $h \in H_i$, a behavior strategy is $\sigma_i(h) \in \Delta(A_i(h))$ where $\Delta(A_i(h))$ is the set of all probability distributions over actions available at information set $h$. A group of strategies $\sigma = (\sigma_1, \ldots, \sigma_n)$ consisting of strategies for each player is a *strategy profile*. We sometimes write $\sigma_{-i} = (\sigma_1, \ldots, \sigma_{i-1}, \sigma_{i+1}, \ldots, \sigma_n)$ and $(\sigma'_i, \sigma_{-i}) = (\sigma_1, \ldots, \sigma_{i-1}, \sigma'_i, \sigma_{i+1}, \ldots, \sigma_n)$. By an abuse of notation, we will say player $i$ receives an expected payoff of $u_i(\sigma)$ when all players are playing the strategy profile $\sigma$.

## 2.2 Solution concepts

Having defined the model of games we wish to consider, we now define the various solutions of interest.

**Definition 3** *A strategy profile $\sigma = (\sigma_1, \ldots, \sigma_n)$ for a game $\Gamma = \langle I, V, E, P, H, A, u, p \rangle$ is a* Nash equilibrium *if $u_i(\sigma_i, \sigma_{-i}) \geq u_i(\sigma'_i, \sigma_{-i})$ for all $i \in I$ and all $\sigma'_i$.*

If the game happens to be two-person zero-sum, then a Nash equilibrium may be called a *minimax solution* and satisfies the following additional properties.

1. If $(\sigma_1, \sigma_2)$ is a minimax solution to a two-person zero-sum game $\Gamma$, and $(\sigma'_1, \sigma'_2)$ is also a minimax solution to $\Gamma$, then $(\sigma_1, \sigma'_2)$ and $(\sigma'_1, \sigma_2)$ are also minimax solutions to $\Gamma$.
2. If $(\sigma_1, \sigma_2)$ is a minimax solution to a two-person zero-sum game $\Gamma$, then $u_1(\sigma_1, \sigma_2) \geq u_1(\sigma_1, \sigma'_2)$ for all $\sigma'_2$.
3. All convex combinations of minimax solutions for two-person zero-sum games are also minimax solutions. (The set of minimax solutions form a convex set.)

The first property means that there is no equilibrium selection problem, which can inhibit the application of game theory in some games. The second property means that equilibria solutions in two-person zero-sum games are robust in that they don't depend on what strategy the opponent employs. The third property will be of importance when designing some equilibrium-finding algorithms.

Due to computational limitations (and in particular the inherent finiteness of floating-point arithmetic), we are often interested in the following slightly relaxed version of Nash equilibrium:

**Definition 4** *A strategy profile $\sigma = (\sigma_1, \ldots, \sigma_n)$ for a game $\Gamma = \langle I, V, E, P, H, A, u, p \rangle$ is an* $\epsilon$-equilibrium *if $u_i(\sigma'_i, \sigma_{-i}) - u_i(\sigma_i, \sigma_{-i}) \leq \epsilon$ for all $i \in I$ and all $\sigma'_i$.*

In many algorithms, the parameter $\epsilon$ is specified as an input parameter and the algorithm guarantees finding such an $\epsilon$-equilibrium. For a small enough $\epsilon$, these solutions are acceptable in many domains.

In addition to the basic equilibrium concepts defined above, there are also many known refinements to the Nash equilibrium concept. We will discuss some of these in Section 8.

## 2.3 Algorithms for finding equilibria

In this subsection we describe existing algorithms for finding Nash equilibria and $\epsilon$-equilibria in both normal form and extensive form games.

### 2.3.1 Algorithms for finding equilibria in normal form games

The Nash equilibrium problem for two-person zero-sum (matrix) games can be modeled and solved as a linear program [28, 78, 24]. Linear programs are typically solved via the simplex algorithm or interior-point methods. The simplex algorithm has exponential worst-case complexity, but runs efficiently in practice. Interior-point methods run in polynomial time, and, increasingly, are also fast in practice. Other solution techniques include learning-based approaches, such as fictitious play [18, 102] and experts-based approaches [39]. These approaches are more interested in the learning process itself rather than in arriving at an equilibrium, and generally do not provide very good convergence bounds. Most recently, the *excessive gap technique* was proposed as a method for solving certain non-smooth convex optimization problems, and it has been applied to the problem of finding $\epsilon$-equilibria in matrix games [92, 91]. Finally, bundle-based methods have recently been shown to be effective on some large poker games, including Rhode Island Hold'em [82]. One drawback to those algorithms is that the memory usage increases every iteration, and the time to solve each iteration increases with every iteration (although there are heuristics that mitigate this).

There has been significant recent work on Nash equilibrium finding for two-person non-zero-sum normal form games. The question of how complex it is to construct an equilibrium in a 2-player game has been dubbed "a most fundamental computational problem whose complexity is wide open" and "together with factoring, [...] the most important concrete open question on the boundary of P today" [95]. Most interesting questions about optimal (for many definitions of "optimal") equilibria are NP-complete [42, 25]. An $\epsilon$-equilibrium in a normal form game with any constant number of agents can be constructed in quasi-polynomial time [75, 74], but finding an exact equilibrium is PPAD-complete even in a 2-player game [22]. The most prevalent algorithm for finding an equilibrium in a two-player bimatrix game is *Lemke-Howson* [72], but it takes exponentially many steps in the worst case [109]. For a survey of equilibrium computation in 2-player games, see [130]. Recently, equilibrium-finding algorithms that enumerate supports (*i.e.*, sets of pure strategies that are played with positive probability) have been shown efficient on many games [100], and efficient mixed integer programming algorithms that search in the space of supports have been developed [108]. For more than two players, many algorithms have been proposed, but they currently only scale to very small games [53, 81, 100].[2]

### 2.3.2 Algorithms for finding equilibria in extensive form games

As discussed in the introduction, Nash equilibria of two-person sequential games with perfect information can be found by simply searching over the tree.[3] In computer science terms, this is done using *minimax search* (often in conjunction with *$\alpha$-$\beta$-pruning* to reduce the search tree size and thus enhance speed). Minimax search runs in linear time in the size of the game tree.[4]

The differentiating feature of games of imperfect information, such as poker, is that they are not fully observable: when it is an agent's turn to move, she does not have access to all of the information about the world. In such games, the decision of what to do at a point in time cannot generally be optimally made without considering decisions at all other points in time (including ones on other paths of play) because those other decisions affect the probabilities of being at different states at the current point in time. Thus the algorithms for perfect information games do not solve games of imperfect information.

[2] Progress has also been made on algorithms for finding equilibria in restricted and/or structured games (*e.g.*, [96, 8, 73, 14, 118]), as well as for finding market equilibria (*e.g.*, [32, 33, 57, 110]).

[3] This actually yields a solution that satisfies not only the Nash equilibrium solution concept, but a stronger solution concept called *subgame perfect Nash equilibrium* [115].

[4] This type of algorithm has its limits, of course, particularly when the game tree is huge, but extremely effective game-playing agents can be developed, even when the size of the game tree prohibits complete search, by evaluating intermediate nodes using a heuristic evaluation and then treating those nodes as leaves of the tree.

As discussed previously, one could try to find an equilibrium of a sequential game by converting the normal form, but this is computationally intractable. However, by observing that one needs to consider only sequences of moves rather than pure strategies, one arrives at a more compact representation, the *sequence form*, which is linear in the size of the game tree [103, 116, 61, 129]. For two-person zero-sum games, there is a polynomial-sized (in the size of the game tree) linear programming formulation (linear complementarity in the non-zero-sum case) based on the sequence form such that strategies for players 1 and 2 correspond to primal and dual variables. Thus, the equilibria of reasonable-sized 2-player games can be computed using this method [129, 63, 65].[5] However, this approach still yields enormous (unsolvable) optimization problems for many real-world games, such as poker.

The Nash equilibrium problem for two-player zero-sum sequential games of imperfect information can be formulated using the sequence form representation [103, 61, 129] as the following saddle-point problem:

$$\max_{\mathbf{x}\in Q_1} \min_{\mathbf{y}\in Q_2} \langle A\mathbf{y}, \mathbf{x}\rangle = \min_{\mathbf{y}\in Q_2} \max_{\mathbf{x}\in Q_1} \langle A\mathbf{y}, \mathbf{x}\rangle. \tag{1}$$

In this formulation, $\mathbf{x}$ is player 1's strategy and $\mathbf{y}$ is player 2's strategy. The bilinear term $\langle A\mathbf{y}, \mathbf{x}\rangle$ is the payoff that player 1 receives from player 2 when the players play the strategies $\mathbf{x}$ and $\mathbf{y}$. The strategy spaces are represented by $Q_i \subseteq \mathbb{R}^{|S_i|}$, where $S_i$ is the set of sequences of moves of player $i$, and $Q_i$ is the *set of realization plans* of player $i$. Thus $\mathbf{x}$ ($\mathbf{y}$) encodes probability distributions over actions at each point in the game where player 1 (2) acts. The set $Q_i$ has an explicit linear description of the form $\{z \geq 0 : Ez = \mathbf{e}\}$. Consequently, as mentioned above, problem (1) can be modeled as a linear program (see [129] for details).

Recently, we have investigated the application of Nesterov's excessive gap technique [91] to the problem of finding $\epsilon$-equilibria in extensive form games by directly tackling the equations in problem (1) [54, 43]. We will describe this approach in detail in Section 5.

### 2.3.3 Algorithmic approximations

As discussed above, the equilibrium problem for two-player zero-sum games can be modeled as a linear program (LP), which can in turn be solved using the simplex method. This approach has inherent features which we can leverage into desirable properties in the context of solving games.

In the LP, primal solutions correspond to strategies of player 2, and dual solutions correspond to strategies of player 1. There are two versions of the simplex method: the primal simplex and the dual simplex. The primal simplex maintains primal feasibility and proceeds by finding better and better primal solutions until the dual solution vector is feasible, at which point optimality has been reached. Analogously, the dual simplex maintains dual feasibility and proceeds by finding increasingly better dual solutions until the primal solution vector is feasible. (The dual simplex method can be thought of as running the primal simplex method on the dual problem.) Thus, the primal and dual simplex methods serve as *anytime* algorithms (for a given abstraction) for players 2 and 1, respectively. At any point in time, they can output the best strategies found so far.

Also, for any feasible solution to the LP, we can get bounds on the quality of the strategies by examining the primal and dual solutions. (When using the primal simplex method, dual solutions may be read off of the LP tableau.) Every feasible solution of the dual yields an upper bound on the optimal value of the primal, and vice versa [24, p. 57]. Thus, without requiring further computation, we get lower bounds on the expected utility of each agent's strategy against that agent's worst-case opponent.

One problem with the simplex method is that it is not a primal-dual algorithm, that is, it does not maintain both primal and dual feasibility throughout its execution. (In fact, it only obtains primal and dual feasibility at the very end of execution.) In contrast, there are interior-point methods for linear programming that maintain primal and dual feasibility throughout the execution. For example, many interior-point path-following algorithms have this property [131, Ch. 5]. We observe that running such a linear programming method yields a method for finding $\epsilon$-equilibria (*i.e.*, strategy profiles in which no agent can increase her expected utility by more than $\epsilon$ by deviating). A threshold on $\epsilon$ can also be used as a termination criterion

[5] Recently this approach was extended to handle computing *sequential equilibria* [68] as well [84].

for using the method as an anytime algorithm. Furthermore, interior-point methods in this class have polynomial-time worst-case run time, as opposed to the simplex algorithm, which takes exponentially many steps in the worst case. In Section 8, we will review the necessary interior-point method theory, and propose how an interior-point method could be specialized to the equilibrium-finding problem to improve even further their performance in solving games.

# 3 Lossless automated abstraction [44, 46]

In this research stream, we take a different approach to tackling the difficult problem of equilibrium computation. Instead of developing an equilibrium-finding method *per se*, we instead develop a methodology for automatically abstracting games in such a way that any equilibrium in the smaller (abstracted) game corresponds directly to an equilibrium in the original game. Thus, by computing an equilibrium in the smaller game (using any available equilibrium-finding algorithm), we are able to construct an equilibrium in the original game. The motivation is that an equilibrium for the smaller game can be computed drastically faster than for the original game.

To this end, we introduce *games with ordered signals* (Section 3.2), a broad class of games that has enough structure which we can exploit for abstraction purposes. Instead of operating directly on the game tree (something we found to be technically challenging), we instead introduce the use of *information filters* (Section 3.3), which coarsen the information each player receives. They are used in our analysis and abstraction algorithm. By operating only in the space of filters, we are able to keep the strategic structure of the game intact, while abstracting out details of the game in a way that is lossless from the perspective of equilibrium finding. We introduce the *ordered game isomorphism* to describe strategically symmetric situations and the *ordered game isomorphic abstraction transformation* to take advantage of such symmetries (Section 3.5). As our main equilibrium result we have the following:

> **Theorem 2** *Let $\Gamma$ be a game with ordered signals, and let $F$ be an information filter for $\Gamma$. Let $F'$ be an information filter constructed from $F$ by one application of the ordered game isomorphic abstraction transformation, and let $\sigma'$ be a Nash equilibrium strategy profile of the induced game $\Gamma_{F'}$ (*i.e.*, the game $\Gamma$ using the filter $F'$). If $\sigma$ is constructed by using the corresponding strategies of $\sigma'$, then $\sigma$ is a Nash equilibrium of $\Gamma_F$.*

The proof of the theorem uses an equivalent characterization of Nash equilibria: $\sigma$ is a Nash equilibrium if and only if there exist beliefs $\mu$ (players' beliefs about unknown information) at all points of the game reachable by $\sigma$ such that $\sigma$ is sequentially rational (*i.e.*, a best response) given $\mu$, where $\mu$ is updated using Bayes' rule. We can then use the fact that $\sigma'$ is a Nash equilibrium to show that $\sigma$ is a Nash equilibrium considering only local properties of the game.

We also give an algorithm, *GameShrink*, for abstracting the game using our isomorphism exhaustively (Section 3.6). Its complexity is $\tilde{O}(n^2)$, where $n$ is the number of nodes in a structure we call the signal tree. It is no larger than the game tree, and on nontrivial games it is drastically smaller, so *GameShrink* has time and space complexity *sublinear* in the size of the game tree. We also present several algorithmic and data structure related speed improvements (Section 3.7).

In the following subsection, we describe some application areas that fit within our model, including one where we have already applied our technique.

## 3.1 Applications

Sequential games of imperfect information are ubiquitous, for example in negotiation and in auctions. Often aspects of a player's knowledge are not pertinent for deciding what action the player should take at a given point in the game. On the trivial end, some aspects of a player's knowledge are never pertinent (*e.g.*, whether it is raining or not has no bearing on the bidding strategy in an art auction), and such aspects can be completely left out of the model specification. However, more generally, some aspects can be pertinent in certain states of the game while they are not pertinent in other states, and thus cannot be left out of the

model completely. Furthermore, it may be highly non-obvious which aspects are pertinent in which states of the game. Our algorithm automatically discovers which aspects are irrelevant in different states, and eliminates those aspects of the game, resulting in a more compact, equivalent game representation.

One broad application area that has this property is sequential negotiation (potentially over multiple issues). Another broad application area is sequential auctions (potentially over multiple goods). For example, in those states of a 1-object auction where bidder A can infer that his valuation is greater than that of bidder B, bidder A can ignore all his other information about B's signals, although that information would be relevant for inferring B's exact valuation. Furthermore, in some states of the auction, a bidder might not care which exact other bidders have which valuations, but cares about which valuations are held by the other bidders in aggregate (ignoring their identities). Many open-cry sequential auction and negotiation mechanisms fall within the game model studied in this paper (specified in detail later), as do certain other electronic commerce settings, such as sequences of take-it-or-leave-it offers [107]. In fact, our game model captures the entire class of games for which Fudenberg and Tirole applied the perfect Bayesian equilibrium solution concept [41] (*i.e.*, Bayesian games with observable actions).

### 3.1.1 Solving Rhode Island Hold'em poker

Our techniques are in no way specific to an application. The main experiment that we present here is on a recreational game. We chose a particular poker game as the benchmark problem because it yields an extremely complicated and enormous game tree, it is a game of imperfect information, it is fully specified as a game (and the data is available), and it has been posted as a challenge problem by others [117] (to our knowledge no such challenge problem instances have been proposed for electronic commerce applications that require solving sequential games).

Rhode Island Hold'em was invented as a testbed for computational game playing [117]. It was designed so that it was similar in style to Texas Hold'em, yet not so large that devising reasonably intelligent strategies would be impossible. We applied the techniques developed in this paper to find an exact (minimax) solution to Rhode Island Hold'em, which has a game tree exceeding 3.1 billion nodes.

Applying the sequence form to Rhode Island Hold'em directly without abstraction yields a linear program 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 our *GameShrink* algorithm to reduce this through lossless abstraction, and it yielded a linear program with 1,237,238 rows and columns—with 50,428,638 non-zero coefficients. We then applied iterated elimination of dominated strategies, which further reduced this to 1,190,443 rows and 1,181,084 columns. (Applying iterated elimination of dominated strategies without *GameShrink* yielded 89,471,986 rows and 89,121,538 columns, which still would have been prohibitively large to solve.) *GameShrink* required less than one second to perform the shrinking (*i.e.*, to compute all of the ordered game isomorphic abstraction transformations). Using a 1.65GHz IBM eServer p5 570 with 64 gigabytes of RAM (the linear program solver actually needed 25 gigabytes), we solved it in 7 days and 17 hours using the interior-point barrier method of CPLEX version 9.1.2. We recently demonstrated our optimal Rhode Island Hold'em poker player at the AAAI-05 conference [44], and it is available for play on-line at `http://www.cs.cmu.edu/~gilpin/gsi.html`.

While others have worked on computer programs for playing Rhode Island Hold'em [117], no optimal strategy has been found before. This is the largest poker game solved to date by over four orders of magnitude.

## 3.2 Games with ordered signals

We work with a slightly restricted class of games, as compared to the full generality of the extensive form. This class, which we call *games with ordered signals*, is highly 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 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). For simplicity, we assume—as is the case in most recreational games—that within each round, the number of private signals received is the same across players (this could quite likely be relaxed). We also 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 partial ordering over sets of signals, and the payoffs are increasing (not necessarily strictly) in these signals. For example, in poker, this partial ordering corresponds exactly to the ranking of card hands.

**Definition 5** *A* game with ordered signals *is a tuple* $\Gamma = \langle I, G, L, \Theta, \kappa, \gamma, p, \succeq, \omega, u \rangle$ *where:*

1. $I = \{1, \ldots, n\}$ *is a finite set of players.*
2. $G = \langle G^1, \ldots, G^r \rangle$, $G^j = \left(V^j, E^j\right)$, *is a finite collection of finite directed trees with nodes* $V^j$ *and edges* $E^j$. *Let* $Z^j$ *denote the leaf nodes of* $G^j$ *and let* $N^j(v)$ *denote the outgoing neighbors of* $v \in V^j$. $G^j$ *is the* stage game *for round* $j$.
3. $L = \langle L^1, \ldots, L^r \rangle$, $L^j : V^j \setminus Z^j \to I$ *indicates which player acts (chooses an outgoing edge) at each internal node in round* $j$.
4. $\Theta$ *is a finite set of* signals.
5. $\kappa = \langle \kappa^1, \ldots, \kappa^r \rangle$ *and* $\gamma = \langle \gamma^1, \ldots, \gamma^r \rangle$ *are vectors of nonnegative integers, where* $\kappa^j$ *and* $\gamma^j$ *denote the number of public and private signals (per player), respectively, revealed in round* $j$. *Each signal* $\theta \in \Theta$ *may only be revealed once, and in each round every player receives the same number of private signals, so we require* $\sum_{j=1}^{r} \kappa^j + n\gamma^j \leq |\Theta|$. *The public information revealed in round* $j$ *is* $\alpha^j \in \Theta^{\kappa^j}$ *and the public information revealed in all rounds up through round* $j$ *is* $\tilde{\alpha}^j = \left(\alpha^1, \ldots, \alpha^j\right)$. *The private information revealed to player* $i \in I$ *in round* $j$ *is* $\beta_i^j \in \Theta^{\gamma^j}$ *and the private information revaled to player* $i \in I$ *in all rounds up through round* $j$ *is* $\tilde{\beta}_i^j = \left(\beta_i^1, \ldots, \beta_i^j\right)$. *We also write* $\tilde{\beta}^j = \left(\tilde{\beta}_1^j, \ldots, \tilde{\beta}_n^j\right)$ *to represent all private information up through round* $j$, *and* $\left(\tilde{\beta'}_i^j, \tilde{\beta}_{-i}^j\right) = \left(\tilde{\beta}_1^j, \ldots, \tilde{\beta}_{i-1}^j, \tilde{\beta'}_i^j, \tilde{\beta}_{i+1}^j, \ldots, \tilde{\beta}_n^j\right)$ *is* $\tilde{\beta}^j$ *with* $\tilde{\beta}_i^j$ *replaced with* $\tilde{\beta'}_i^j$. *The total information revealed up through round* $j$, $\left(\tilde{\alpha}^j, \tilde{\beta}^j\right)$, *is said to be* legal *if no signals are repeated.*
6. $p$ *is a probability distribution over* $\Theta$, *with* $p(\theta) > 0$ *for all* $\theta \in \Theta$. *Signals are drawn from* $\Theta$ *according to* $p$ *without replacement, so if* $X$ *is the set of signals already revealed, then*

$$p(x \mid X) = \begin{cases} \frac{p(x)}{\sum_{y \notin X} p(y)} & \text{if} \quad x \notin X \\ 0 & \text{if} \quad x \in X. \end{cases}$$

7. $\succeq$ *is a partial ordering of subsets of* $\Theta$ *and is defined for at least those pairs required by* $u$.
8. $\omega : \bigcup_{j=1}^{r} Z^j \to \{over, continue\}$ *is a mapping of terminal nodes within a stage game to one of two values:* over, *in which case the game ends, or* continue, *in which case the game continues to the next round. Clearly, we require* $\omega(z) = over$ *for all* $z \in Z^r$. *Note that* $\omega$ *is independent of the signals. Let* $\omega_{over}^j = \left\{z \in Z^j \mid \omega(z) = over\right\}$ *and* $\omega_{cont}^j = \left\{z \in Z^j \mid \omega(z) = continue\right\}$.
9. $u = (u^1, \ldots, u^r)$, $u^j : \bigtimes_{k=1}^{j-1} \omega_{cont}^k \times \omega_{over}^j \times \bigtimes_{k=1}^{j} \Theta^{\kappa^k} \times \bigtimes_{i=1}^{n} \bigtimes_{k=1}^{j} \Theta^{\gamma^k} \to \mathbb{R}^n$ *is a utility function such that for every* $j$, $1 \leq j \leq r$, *for every* $i \in I$, *and for every* $\tilde{z} \in \bigtimes_{k=1}^{j-1} \omega_{cont}^k \times \omega_{over}^j$, *at least one of the following two conditions holds:*

*(a) Utility is signal independent:* $u_i^j(\tilde{z}, \vartheta) = u_i^j(\tilde{z}, \vartheta')$ *for all legal* $\vartheta, \vartheta' \in \bigtimes_{k=1}^{j} \Theta^{\kappa^k} \times \bigtimes_{i=1}^{n} \bigtimes_{k=1}^{j} \Theta^{\gamma^k}$.

*(b)* $\succeq$ *is defined for all legal signals* $(\tilde{\alpha}^j, \tilde{\beta}_i^j)$ *and* $(\tilde{\alpha}^j, \tilde{\beta}_i'^j)$ *through round* $j$ *and a player's utility is increasing in her private signals, everything else equal:*

$$\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right) \succeq \left(\tilde{\alpha}^j, \tilde{\beta}_i'^j\right) \Longrightarrow u_i\left(\tilde{z}, \tilde{\alpha}^j, \left(\tilde{\beta}_i^j, \tilde{\beta}_{-i}^j\right)\right) \geq u_i\left(\tilde{z}, \tilde{\alpha}^j, \left(\tilde{\beta}'^j_i, \tilde{\beta}_{-i}^j\right)\right).$$

We will use the term *game with ordered signals* and the term *ordered game* interchangeably.

### 3.2.1 Rhode Island Hold'em modeled as an ordered game

As an illustration, we describe how Rhode Island Hold'em can be defined as an ordered game in accordance with Definition 5. First, we describe the rules of Rhode Island Hold'em.

1. Each player pays an *ante* of 5 chips which is added to the *pot*. Both players initially receive a single card, face down; these are known as the *hole cards*.
2. After receiving the hole cards, the players participate in one betting round. Each player may *check* (not placing any money in the pot and passing) or *bet* (placing 10 chips into the pot) if no bets have been placed. If a bet has been placed, then the player may *fold* (thus forfeiting the game along with any money they have put into the pot), *call* (adding chips to the pot equal to the last player's bet), or *raise* (calling the current bet and making an additional bet). In Rhode Island Hold'em, the players are limited to three bets each per betting round. (A raise equals two bets.) In the first betting round, the bets are equal to 10 chips.
3. After the first betting round, a community card is dealt face up. This is called the *flop* card. Another betting round take places at this point, with bets equal to 20 chips.
4. Following the second betting round, another community card is dealt face up. This is called the *turn* card. A final betting round takes place at this point, with bets again equal to 20 chips.
5. If neither player folds, then the *showdown* takes place. Both players turn over their cards. The player who has the best 3-card poker hand takes the pot. In the event of a draw, the pot is split evenly.

| **Rank** | **Hand** | **Prob.** | **Description** | **Example** |
|---|---|---|---|---|
| 1 | Straight flush | 0.00217 | 3 cards w/ consecutive rank & same suit | K♠, Q♠, J♠ |
| 2 | Three of a kind | 0.00235 | 3 cards of the same rank | Q♠, Q♡, Q♣ |
| 3 | Straight | 0.03258 | 3 cards w/ consecutive rank | 3♣, 4♠, 5♡ |
| 4 | Flush | 0.04959 | 3 cards of the same suit | 2♢, 5♢, 8♢ |
| 5 | Pair | 0.16941 | 2 cards of the same rank | 2♢, 2♠, 3♡ |
| 6 | High card | 0.74389 | None of the above | J♣, 9♡, 2♠ |

Table 1: Rankings of three-card poker hands.

Hands in 3-card poker games are ranked slightly differently than 5-card poker hands. The main differences are that the order of flushes and straights are reversed, and a three of a kind is better than straights or flushes. Table 1 describes the rankings. Within ranks, ties are broken by by ordering hands according to the rank of cards that make up the hand. If players are still tied after applying this criterion, *kickers* are used to determine the winner. A kicker is a card that is not used to make up the hand. For example, if player 1 has a pair of eights and a five, and player 2 has a pair of eights and a six, player 2 wins.

To make the definition of ordered games concrete, here we define each of the components of the tuple $\Gamma = \langle I, G, L, \Theta, \kappa, \gamma, p, \succeq, \omega, u \rangle$ for Rhode Island Hold'em. There are two players so $I = \{1, 2\}$. There are

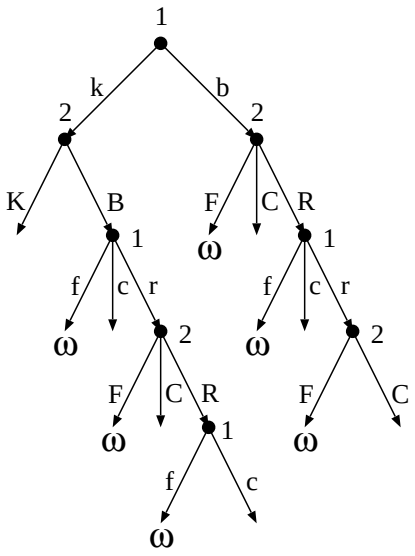


Figure 2: Stage game $G_{RI}$, player label $L$, and game-ending nodes $\omega$ for Rhode Island Hold'em. The action labels denote which action the player is taking: k (check), b (bet), f (fold), c (call), and r (raise). Lower case letters indicate player 1 actions and upper case letters indicate player 2 actions.

three rounds, and the stage game is the same in each round so we have $G = \langle G_{RI}, G_{RI}, G_{RI} \rangle$ where $G_{RI}$ is given in Figure 2, which also specifies the player label $L$. $\Theta$ is the standard deck of 52 cards. The community cards are dealt in the second and third rounds, so $\kappa = \langle 0, 1, 1 \rangle$. Each player receives a since face down card in the first round only, so $\gamma = \langle 1, 0, 0 \rangle$. $p$ is the uniform distribution over $\Theta$. $\succeq$ is defined for three card hands and is defined using the ranking given in Table 1. The game-ending nodes $\omega$ are denoted in Figure 2 by $\omega$. $u$ is defined as in the above description; it is easy to verify that it satisfies the necessary conditions.

### 3.3 Information filters

In this subsection, we define an *information filter* for ordered games. Instead of completely revealing a signal (either public or private) to a player, the signal first passes through this filter, which outputs a *coarsened* signal to the player. By varying the filter applied to a game, we are able to obtain a wide variety of games while keeping the underlying action space of the game intact. We will use this when designing our abstraction techniques. Formally, an information filter is as follows.

**Definition 6** *Let $\Gamma = \langle I, G, L, \Theta, \kappa, \gamma, p, \succeq, \omega, u \rangle$ be an ordered game. Let $S^j \subseteq \bigtimes_{k=1}^{j} \Theta^{\kappa^k} \times \bigtimes_{k=1}^{j} \Theta^{\gamma^k}$ be the set of legal signals (*i.e.*, no repeated signals) for one player through round $j$. An* information filter *for $\Gamma$ is a collection $F = \langle F^1, \ldots, F^r \rangle$ where each $F^j$ is a function $F^j : S^j \to 2^{S^j}$ such that each of the following conditions hold:*

1. *(Truthfulness) $(\tilde{\alpha}^j, \tilde{\beta}_i^j) \in F^j(\tilde{\alpha}^j, \tilde{\beta}_i^j)$ for all legal $(\tilde{\alpha}^j, \tilde{\beta}_i^j)$.*
2. *(Independence) The range of $F^j$ is a partition of $S^j$.*
3. *(Information preservation) If two values of a signal are distinguishable in round $k$, then they are distinguishable fpr each round $j > k$. Let $m^j = \sum_{l=1}^{j} \kappa^l + \gamma^l$. We require that for all legal $(\theta_1, \ldots, \theta_{m^k}, \ldots, \theta_{m^j}) \subseteq \Theta$ and $(\theta'_1, \ldots, \theta'_{m^k}, \ldots, \theta'_{m^j}) \subseteq \Theta$:*

$$(\theta'_1, \ldots, \theta'_{m^k}) \notin F^k(\theta_1, \ldots, \theta_{m^k}) \implies (\theta'_1, \ldots, \theta'_{m^k}, \ldots, \theta'_{m^j}) \notin F^j(\theta_1, \ldots, \theta_{m^k}, \ldots, \theta_{m^j}).$$

A game with ordered signals $\Gamma$ and an information filter $F$ for $\Gamma$ defines a new game $\Gamma_F$. We refer to such games as *filtered ordered games*. We are left with the original game if we use the identity filter $F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right) = \left\{\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right)\right\}$. We have the following simple (but important) result:

**Proposition 1** *A filtered ordered game is an extensive form game satisfying perfect recall.*

A simple proof proceeds by constructing an extensive form game directly from the ordered game, and showing that it satisfies perfect recall. In determining the payoffs in a game with filtered signals, we take the average over all real signals in the filtered class, weighted by the probability of each real signal occurring.

## 3.4 Strategies and Nash equilibrium in games with ordered signals

We are now ready to define behavior strategies in the context of filtered ordered games.

**Definition 7** *A* behavior strategy *for player $i$ in round $j$ of $\Gamma = \langle I, G, L, \Theta, \kappa, \gamma, p, \succeq, \omega, u\rangle$ with information filter $F$ is a probability distribution over possible actions, and is defined for each player $i$, each round $j$, and each $v \in V^j \setminus Z^j$ for $L^j(v) = i$:*

$$\sigma_{i,v}^j : \bigtimes_{k=1}^{j-1} \omega_{cont}^k \times Range\left(F^j\right) \to \Delta\left\{w \in V^j \mid (v,w) \in E^j\right\}.$$

*($\Delta(X)$ is the set of probability distributions over a finite set $X$.) A behavior strategy for player $i$ in round $j$ is $\sigma_i^j = (\sigma_{i,v_1}^j, \ldots, \sigma_{i,v_m}^j)$ for each $v_k \in V^j \setminus Z^j$ where $L^j(v_k) = i$. A behavior strategy for player $i$ in $\Gamma$ is $\sigma_i = \left(\sigma_i^1, \ldots, \sigma_i^r\right)$. A* strategy profile *is $\sigma = (\sigma_1, \ldots, \sigma_n)$. A strategy profile with $\sigma_i$ replaced by $\sigma_i'$ is $(\sigma_i', \sigma_{-i}) = (\sigma_1, \ldots, \sigma_{i-1}, \sigma_i', \sigma_{i+1}, \ldots, \sigma_n)$.*

As in extensive form games, we abuse notation to say player $i$ receives an expected payoff of $u_i(\sigma)$ when all players are playing the strategy profile $\sigma$. Strategy $\sigma_i$ is said to be player $i$'s *best response* to $\sigma_{-i}$ if for all other strategies $\sigma_i'$ for player $i$ we have $u_i(\sigma_i, \sigma_{-i}) \geq u_i(\sigma_i', \sigma_{-i})$. $\sigma$ is a *Nash equilibrium* if, for every player $i$, $\sigma_i$ is a best response for $\sigma_{-i}$. A Nash equilibrium always exists in finite extensive form games [87], and one exists in behavior strategies for games with perfect recall [71]. Using these observations, we have the following corollary to Proposition 1:

**Corollary 1** *For any filtered ordered game, a Nash equilibrium exists in behavior strategies.*

## 3.5 Equilibrium-preserving abstractions

In this section, we present our main technique for reducing the size of games. We begin by defining a *filtered signal tree* which represents all of the chance moves in the game. The bold edges (*i.e.* the first two levels of the tree) in the game trees in Figure 3 correspond to the filtered signal trees in each game.

**Definition 8** *Associated with every ordered game $\Gamma = \langle I, G, L, \Theta, \kappa, \gamma, p, \succeq, \omega, u\rangle$ and information filter $F$ is a* filtered signal tree*, a directed tree in which each node corresponds to some revealed (filtered) signals and edges correspond to revealing specific (filtered) signals. The nodes in the filtered signal tree represent the set of all possible revealed filtered signals (public and private) at some point in time. The filtered public signals revealed in round $j$ correspond to the nodes in the $\kappa^j$ levels beginning at level $\sum_{k=1}^{j-1}\left(\kappa^k + n\gamma^k\right)$ and the private signals revealed in round $j$ correspond to the nodes in the $n\gamma^j$ levels beginning at level $\sum_{k=1}^{j}\kappa^k + \sum_{k=1}^{j-1} n\gamma^k$. We denote children of a node $x$ as $N(x)$. In addition, we associate weights with the edges corresponding to the probability of the particular edge being chosen given that its parent was reached.*

In many games, there are certain situations in the game that can be thought of as being strategically equivalent to other situations in the game. By melding these situations together, it is possible to arrive at a strategically equivalent smaller game. The next two definitions formalize this notion via the introduction of the *ordered game isomorphic* relation and the *ordered game isomorphic abstraction transformation.*

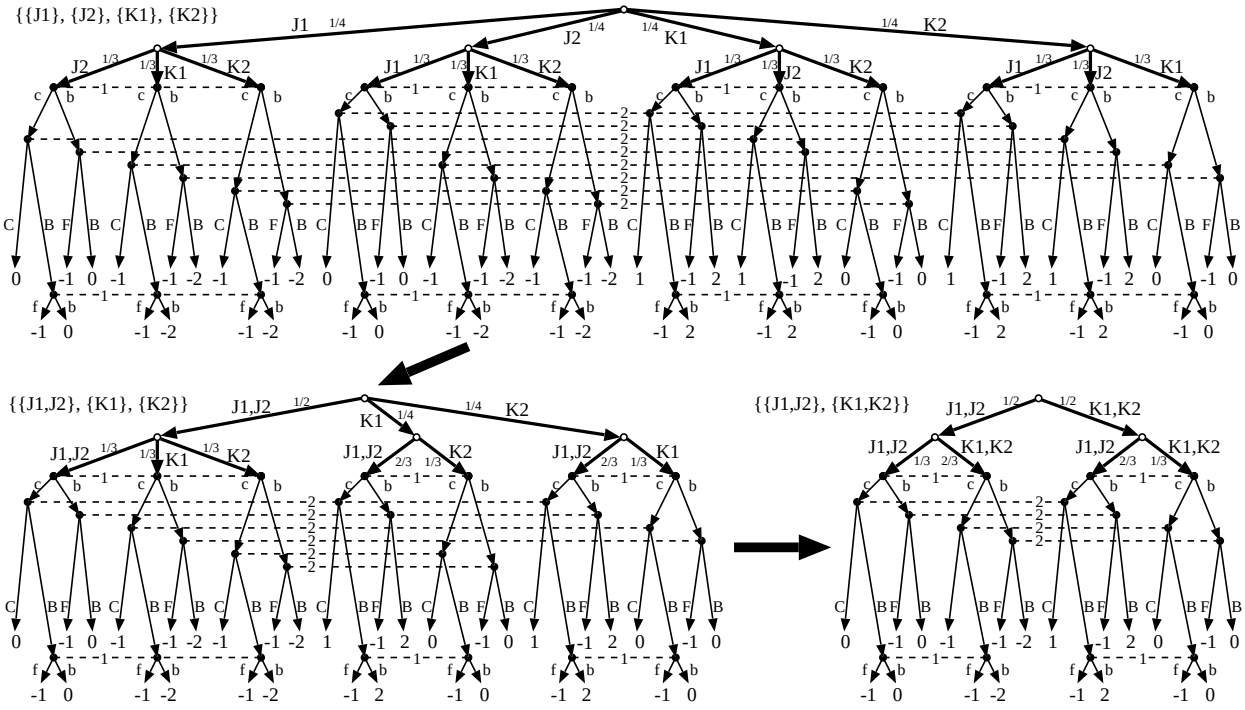


Figure 3: *GameShrink* applied to a tiny two-person four-card (two Jacks and two Kings) poker game. Next to each game tree is the range of the information filter $F$. Dotted lines denote information sets, which are labeled by the controlling player. Open circles are chance nodes with the indicated transition 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 53 nodes to 19 nodes.

**Definition 9** *Two subtrees beginning at internal nodes $x$ and $y$ of a filtered signal tree are* ordered game isomorphic *if $x$ and $y$ have the same parent and there is a bijection $f : N(x) \to N(y)$, such that for $w \in N(x)$ and $v \in N(y)$, $v = f(w)$ implies the weights on the edges $(x, w)$ and $(y, v)$ are the same and the subtrees beginning at $w$ and $v$ are ordered game isomorphic. Two leaves (corresponding to filtered signals $\vartheta$ and $\vartheta'$ up through round $r$) are ordered game isomorphic if for all $\tilde{z} \in \bigtimes_{j=1}^{r-1} \omega_{cont}^{j} \times \omega_{over}^{r}$, $u^r(\tilde{z}, \vartheta) = u^r(\tilde{z}, \vartheta')$.*

**Definition 10** *Let $\Gamma = \langle I, G, L, \Theta, \kappa, \gamma, p, \succeq, \omega, u \rangle$ be an ordered game and let $F$ be an information filter for $\Gamma$. Let $\vartheta$ and $\vartheta'$ be two information structures where the subtrees in the induced filtered signal tree corresponding to the nodes $\vartheta$ and $\vartheta'$ are ordered game isomorphic, and $\vartheta$ and $\vartheta'$ are at either level $\sum_{k=1}^{j-1}(\kappa^k + n\gamma^k)$ or $\sum_{k=1}^{j}\kappa^k + \sum_{k=1}^{j-1} n\gamma^k$ for some round $j$. The* ordered game isomorphic abstraction transformation *is given by creating a new information filter $F'$:*

$$F'^{j}\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right) = \begin{cases} F^j\left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right) & \text{if} \quad \left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right) \notin \vartheta \cup \vartheta' \\ \vartheta \cup \vartheta' & \text{if} \quad \left(\tilde{\alpha}^j, \tilde{\beta}_i^j\right) \in \vartheta \cup \vartheta'. \end{cases}$$

Figure 3 shows the ordered game isomorphic abstraction transformation applied twice to a tiny poker game. Theorem 2, our main equilibrium result, shows how the ordered game isomorphic abstraction transformation can be used to compute equilibria faster.

**Theorem 2** *Let $\Gamma = \langle I, G, L, \Theta, \kappa, \gamma, p, \succeq, \omega, u \rangle$ be an ordered game and $F$ be an information filter for $\Gamma$. Let $F'$ be an information filter constructed from $F$ by one application of the ordered game isomorphic abstraction transformation. Let $\sigma'$ be a Nash equilibrium of the induced game $\Gamma_{F'}$. If we take $\sigma^j_{i,v}\left(\tilde{z}, F^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right)\right) = \sigma'^j_{i,v}\left(\tilde{z}, F'^j\left(\tilde{\alpha}^j, \tilde{\beta}^j_i\right)\right)$, $\sigma$ is a Nash equilibrium of $\Gamma_F$.*

## 3.6 GameShrink: An efficient algorithm for computing ordered game isomorphic abstraction transformations

In this section we present an algorithm, *GameShrink*, for conducting the abstractions. The algorithm only needs to analyze the signal tree discussed above, rather than the entire game tree.

We first present a subroutine that *GameShrink* uses. It is a dynamic program for computing the ordered game isomorphic relation. Again, it operates on the signal tree.

**Algorithm 1** *OrderedGameIsomorphic?* $(\Gamma, \vartheta, \vartheta')$

1. *If $\vartheta$ and $\vartheta'$ have different parents, then return false.*
2. *If $\vartheta$ and $\vartheta'$ are both leaves of the signal tree:*
   (a) *If $u^r(\vartheta \mid \tilde{z}) = u^r(\vartheta' \mid \tilde{z})$ for all $\tilde{z} \in \bigtimes_{j=1}^{r-1} \omega^j_{cont} \times \omega^r_{over}$, then return true.*
   (b) *Otherwise, return false.*
3. *Create a bipartite graph $G_{\vartheta,\vartheta'} = (V_1, V_2, E)$ with $V_1 = N(\vartheta)$ and $V_2 = N(\vartheta')$.*
4. *For each $v_1 \in V_1$ and $v_2 \in V_2$:*

   *If OrderedGameIsomorphic?* $(\Gamma, v_1, v_2)$

   *Create edge* $(v_1, v_2)$
5. *Return true if $G_{\vartheta,\vartheta'}$ has a perfect matching; otherwise, return false.*

By evaluating this dynamic program from bottom to top, Algorithm 1 determines, in time polynomial in the size of the signal tree, whether or not any pair of equal depth nodes $x$ and $y$ are ordered game isomorphic. We can further speed up this computation by only examining nodes with the same parent, since we know (from step 1) that no nodes with different parents are ordered game isomorphic. The test in step 2(a) can be computed in $O(1)$ time by consulting the $\succeq$ relation from the specification of the game. Each call to *OrderedGameIsomorphic?* performs at most one perfect matching computation on a bipartite graph with $O(|\Theta|)$ nodes and $O(|\Theta|^2)$ edges (recall that $\Theta$ is the set of signals). Using the Ford-Fulkerson algorithm [38] for finding a maximal matching, this takes $O(|\Theta|^3)$ time. Let $S$ be the maximum number of signals possibly revealed in the game (*e.g.*, in Rhode Island Hold'em, $S = 4$ because each of the two players has one card in the hand plus there are two cards on the table). The number of nodes, $n$, in the signal tree is $O(|\Theta|^S)$. The dynamic program visits each node in the signal tree, with each visit requiring $O(|\Theta|^2)$ calls to the *OrderedGameIsomorphic?* routine. So, it takes $O(|\Theta|^S|\Theta|^3|\Theta|^2) = O(|\Theta|^{S+5})$ time to compute the entire ordered game isomorphic relation.

While this is exponential in the number of revealed signals, we now show that it is polynomial in the size of the signal tree—and thus polynomial in the size of the game tree because the signal tree is smaller than the game tree. The number of nodes in the signal tree is

$$n = 1 + \sum_{i=1}^{S} \prod_{j=1}^{i} (|\Theta| - j + 1)$$

(Each term in the summation corresponds to the number of nodes at a specific depth of the tree.) The number of leaves is

$$\prod_{j=1}^{S}(|\Theta|-j+1)=\binom{|\Theta|}{S}S!$$

which is a lower bound on the number of nodes.[6] For large $|\Theta|$ we can use the relation $\binom{n}{k}\sim\frac{n^k}{k!}$ to get

$$\binom{|\Theta|}{S}S!\sim\left(\frac{|\Theta|^S}{S!}\right)S!=|\Theta|^S$$

and thus the number of leaves in the signal tree is $\Omega(|\Theta|^S)$. Therefore, $O(|\Theta|^{S+5})=O(n|\Theta|^5)$, which proves that we can indeed compute the ordered game isomorphic relation in time polynomial in the number of nodes, $n$, of the signal tree.

The algorithm often runs 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. (Note that the input to the algorithm is not an explicit game tree, but a specification of the rules, so the algorithm does not need to read in the game tree.) See Figure 3. In general, if an ordered game has $r$ rounds, and each round's stage game has at least $b$ nonterminal leaves, then the size of the signal tree is at most $\frac{1}{b^r}$ of the size of the game tree. For example, in Rhode Island Hold'em, the game tree has 3.1 billion nodes while the signal tree only has 6,632,705.

Given the *OrderedGameIsomorphic?* routine for determining ordered game isomorphisms in an ordered game, we are ready to present the main algorithm, *GameShrink*.

**Algorithm 2** *GameShrink* ($\Gamma$)

1. *Initialize $F$ to be the identity filter for $\Gamma$.*
2. *For $j$ from 1 to $r$:*

   *For each pair of sibling nodes $\vartheta,\vartheta'$ at either level $\sum_{k=1}^{j-1}\left(\kappa^k+n\gamma^k\right)$ or $\sum_{k=1}^{j}\kappa^k+\sum_{k=1}^{j-1}n\gamma^k$ in the filtered (according to $F$) signal tree:*

   *If $OrderedGameIsomorphic?(\Gamma,\vartheta,\vartheta')$, then $F^j(\vartheta)\leftarrow F^j(\vartheta')\leftarrow F^j(\vartheta)\cup F^j(\vartheta')$.*
3. *Output $F$.*

Given as input an ordered game $\Gamma=\langle I,G,L,\Theta,\kappa,\gamma,p,\succeq,\omega,u\rangle$, *GameShrink* applies the shrinking ideas presented above as aggressively as possible. Once it finishes, there are no contractible nodes (since it compares every pair of nodes at each level of the signal tree), and it outputs the corresponding information filter $F$. The correctness of *GameShrink* follows by a repeated application of Theorem 2. Thus, we have the following result:

**Theorem 3** GameShrink *finds all ordered game isomorphisms and applies the associated ordered game isomorphic abstraction transformations. Furthermore, for any Nash equilibrium, $\sigma'$, of the abstracted game, the strategy profile constructed for the original game from $\sigma'$ is a Nash equilibrium.*

The dominating factor in the run time of *GameShrink* is in the $r^{th}$ iteration of the main for-loop. There are at most $\binom{|\Theta|}{S}S!$ nodes at this level, where we again take $S$ to be the maximum number of signals possibly revealed in the game. Thus, the inner for-loop executes $O\left(\left(\binom{|\Theta|}{S}S!\right)^2\right)$ times. As discussed in the next subsection, we use a union-find data structure to represent the information filter $F$. Each iteration of the inner for-loop possibly performs a union operation on the data structure; performing $M$ operations on a union-find

[6] Using the inequality $\binom{n}{k}\geq\left(\frac{n}{k}\right)^k$, we get the lower bound $\binom{|\Theta|}{S}S!\geq\left(\frac{|\Theta|}{S}\right)^S S!=|\Theta|^S\frac{S!}{S^S}$.

data structure containing $N$ elements takes $O(\alpha(M,N))$ amortized time per operation, where $\alpha(M,N)$ is the inverse Ackermann's function [1, 124] (which grows extremely slowly). Thus, the total time for *GameShrink* is $O\left(\left(\binom{|\Theta|}{S}S!\right)^2 \alpha\left(\left(\binom{|\Theta|}{S}S!\right)^2, |\Theta|^S\right)\right)$. By the inequality $\binom{n}{k} \leq \frac{n^k}{k!}$, this is $O\left((|\Theta|^S)^2 \alpha\left((|\Theta|^S)^2, |\Theta|^S\right)\right)$. Again, although this is exponential in $S$, it is $\tilde{O}(n^2)$, where $n$ is the number of nodes in the signal tree. Furthermore, *GameShrink* tends to actually 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, as discussed above.

### 3.7 Efficiency enhancements

We designed several speed enhancement techniques for *GameShrink*, and all of them are incorporated into our implementation. One technique is the use of the union-find data structure [26, Chapter 21] for storing the information filter $F$. This data structure uses time almost linear in the number of operations [124]. Initially each node in the signalling tree is its own set (this corresponds to the identity information filter); when two nodes are contracted they are joined into a new set. Upon termination, the filtered signals for the abstracted game correspond exactly to the disjoint sets in the data structure. This is an efficient method of recording contractions within the game tree, and the memory requirements are only linear in the size of the signal tree.

Determining whether two nodes are ordered game isomorphic requires us to determine if a bipartite graph has a perfect matching. We can eliminate some of these computations by using easy-to-check necessary conditions for the ordered game isomorphic relation to hold. One such condition is to check that the nodes have the same number of chances as being ranked (according to $\succeq$) higher than, lower than, and the same as the opponents. We can precompute these frequencies for every game tree node. This substantially speeds up *GameShrink*, and we can leverage this database across multiple runs of the algorithm (for example, when trying different abstraction levels; see next section). The indices for this database depend on the private and public signals, but not the *order* in which they were revealed, and thus two nodes may have the same corresponding database entry. This makes the database significantly more compact. (For example in Texas Hold'em, the database is reduced by a factor $\binom{50}{3}\binom{47}{1}\binom{46}{1}/\binom{50}{5} = 20$.) We store the histograms in a 2-dimensional database. The first dimension is indexed by the private signals, the second by the public signals. The problem of computing the index in (either) one of the dimensions is exactly the problem of computing a bijection between all subsets of size $r$ from a set of size $n$ and integers in $\left[0, \ldots, \binom{n}{r} - 1\right]$. We efficiently compute this using the subsets' *colexicographical ordering* [15]. Let $\{c_1, \ldots, c_r\}$, $c_i \in \{0, \ldots, n-1\}$, denote the $r$ signals and assume that $c_i < c_{i+1}$. We compute a unique index for this set of signals as follows:

$$index(c_1, \ldots, c_r) = \sum_{i=1}^{r} \binom{c_i}{i}.$$

## 4 Lossy automated abstraction [45, 47, 48, 49]

Some games are too large to compute an exact equilibrium, even after applying *GameShrink* as described in the previous section. By slightly modifying the *GameShrink* algorithm we can obtain an algorithm that yields even smaller game trees, at the expense of losing the equilibrium guarantees of Theorem 2. Instead of requiring the payoffs at terminal nodes to match exactly, we can instead compute a penalty that increases as the difference in utility between two nodes increases.

There are many ways in which the penalty function could be defined and implemented. One possibility is to create edge weights in the bipartite graphs used in Algorithm 1, and then instead of requiring perfect matchings in the unweighted graph we would instead require perfect matchings with low cost (*i.e.*, only consider two nodes to be ordered game isomorphic if the corresponding bipartite graph has a perfect matching with cost below some threshold). Thus, with this threshold as a parameter, we have a knob to turn that in one extreme (threshold = 0) yields an optimal abstraction and in the other extreme (threshold = $\infty$)

yields a highly abstracted game (this would in effect restrict players to ignoring all signals, but still observing actions).

Following this approach we developed the first version of our Texas Hold'em player, *GS1*, and we showed it to be competitive with *Sparbot* and *Vexbot* [45]. However, we observe that *GameShrink* when applied in this lossy mode suffers from three major drawbacks.

- The first, and most serious, is that the abstraction that *GameShrink* computes can be highly inaccurate because the grouping of states is in a sense greedy. For example, if *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 classes where large classes are likely to attract even more states into the class.

- The second drawback to *GameShrink* is that there is no way to directly specify how many classes the abstraction algorithm should yield (overall or at any specific betting round). Rather, there is a parameter (for each round) that specifies a threshold 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 classes in the abstraction 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 betting round) in a way that yields an LP of the desired size.

- The third drawback to *GameShrink* is its scalability. In particular, the time needed to compute an abstraction for a three-round truncated version of Texas Hold'em was over a month. Furthermore, it would have to be executed in the inner loop of the parameter guessing algorithm of the previous paragraph (*i.e.*, once for each setting of the parameters).

In this section we describe new abstraction algorithms that eliminates these problems.

## 4.1 Automated abstraction using clustering and integer programming

*GameShrink* operates on a data structure called the *filtered signal tree* (Section 3.5). This structure captures all of the information that the players receive from moves of nature, and is also used to represent the actual abstraction. We introduce a similar structure for our algorithm, which we will call the *abstraction tree*. For Texas Hold'em, the basic abstraction tree is initialized as follows. The root node contains $\binom{52}{2} = 1326$ children, one for each possible pair of hole cards that a player may be dealt. Each of these children has $\binom{50}{3}$ children, each corresponding to the possible flops that can appear after the two hole cards in the parent node have already been dealt. Similarly, the nodes at the next two levels have 47 and 46 children corresponding to the possible turn and river cards, respectively. This structure is by no means limited to poker, but here for simplicity we only describe it in terms of Texas Hold'em poker since that is the primary application of this paper.

As described in the previous section, we limit the number of strategically different hands we can consider in the first round to 15. Thus, we need to group each of the $\binom{52}{2} = 1326$ different hands into 15 classes. 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 $(w, l, d)$ be the number of possible wins, losses, and draws (based on the roll-out of the remaining cards), we compute the hand's value as $w - l + d/2$, and we take the distance between two hands to be the absolute difference between their values. This gives us the necessary ingredients to apply the $k$-means clustering algorithm [79], which we specialize here to our problem:

**Algorithm 3** *$k$-means clustering for poker hands*

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.*

This algorithm 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 stages of the game, we again want to determine what the best abstraction classes are. Here we face the additional problem of determining how many children each parent in the abstraction tree can have. We use a limit of 225 total child edges that we can use at this level.[7] How should the right to have 225 children (abstraction classes that have not yet been generated at this stage) be divided among the 15 parents? We model and solve this problem as a 0-1 integer program [89] as follows. Our objective is to minimize the expected error in the abstraction. Thus, for each of the 15 parent nodes, we run the $k$-means algorithm presented above for values of $k$ between 1 and 30.[8] 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 $i$ (*i.e.*, in parent $i$); this is simply the number of hands in $i$ divided by $\binom{52}{2}$. Based on these computations, the following 0-1 integer program finds the abstraction that minimizes the overall expected error for the second level:

$$\begin{aligned}
\min \quad & \sum_{i=1}^{15} p_i \sum_{k=1}^{30} c_{i,k} x_{i,k} & & \\
\text{s.t.} \quad & \sum_{i=1}^{15} \sum_{k=1}^{30} k x_{i,k} & \leq & \ 225 \\
& \sum_{k=1}^{30} x_{i,k} & = & \ 1 \quad \forall i \\
& x_{i,k} & \in & \ \{0,1\}
\end{aligned}$$

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 (*e.g.,* 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 the third betting round (with the second-round abstraction classes as the parents, and a limit of 900 on the maximum number of children). This completes the abstraction that is used in Phase 1.[9]

For Phase 2, we compute a third and fourth-round abstraction using the same approach. We do this separately for each of the $\binom{52}{4}$ possible flop and turn combinations.[10]

## 4.2 Estimating payoffs of a truncated game

The second main new idea of this paper is estimating the leaf payoffs for a truncated version of a game by simulating the actions in the remaining portion of the game. This allows the equilibrium-finding algorithm

[7] This limit was again determined based on the size of the LP that was solvable.

[8] The maximum number of children of a particular node in an optimal abstraction will depend on several factors. For this problem, we observed that 30 was an upper bound on this number.

[9] As discussed, our technique optimizes the abstraction round by round, *i.e.*, level by level 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.

[10] Most of the computation time of the abstraction algorithm is spent running the $k$-means clustering algorithm. Our straightforward implementation of the latter could be improved by using sophisticated data structures such as a kd-tree [97] or performing bootstrap averaging to find the initial clusters [30]. This would also allow one to run the $k$-means clustering more times and thus have an even better chance of finding the global optimum of any individual $k$-means clustering problem.

to take into account the entire game tree while having to explicitly solve only a truncated version. This section covers the idea in the context of Texas Hold'em.

In both *Sparbot* and *GS1*, the payoffs that are computed for the leaf nodes at the end of the truncated game (Phase 1) are based on the betting history leading to that node and the expected value of winning that hand assuming that no more betting takes place in later rounds (*i.e.* the payoffs are averaged over the possible fourth-round cards drawn uniformly at random, and assuming that neither player bets in the final round). This completely ignores the fact that later betting actions affect the expected payoff of a node in the game tree.

Instead of ignoring the fourth-round betting, we in effect incorporate it into the truncated game tree by simulating the betting actions for the fourth round (using reasonable fixed randomized strategies for the fourth round), and then using these payoffs as the payoffs in the truncated game. This is intended to mitigate the negative effects of performing an equilibrium analysis on a truncated game.

At the beginning of the fourth round, each player has two hole cards and there are five community cards on the table. Letting $(w, l, d)$ be the number of possible wins, losses, and draws for a player in that situation, we compute the hand's value using the formula $w - l + d/2$ (this is the same formula used by our clustering algorithm). For hands in the fourth round, this gives a value in the interval $[-990, 990]$. Using the history from 343,513 games of *Sparbot* in self-play (of which 187,850 went to the fourth round), we determined the probability of performing each possible action at each player's turn to act as a function of the hand value. To illustrate this, Figures 4–6 show these smoothed probabilities for three particular points to act.

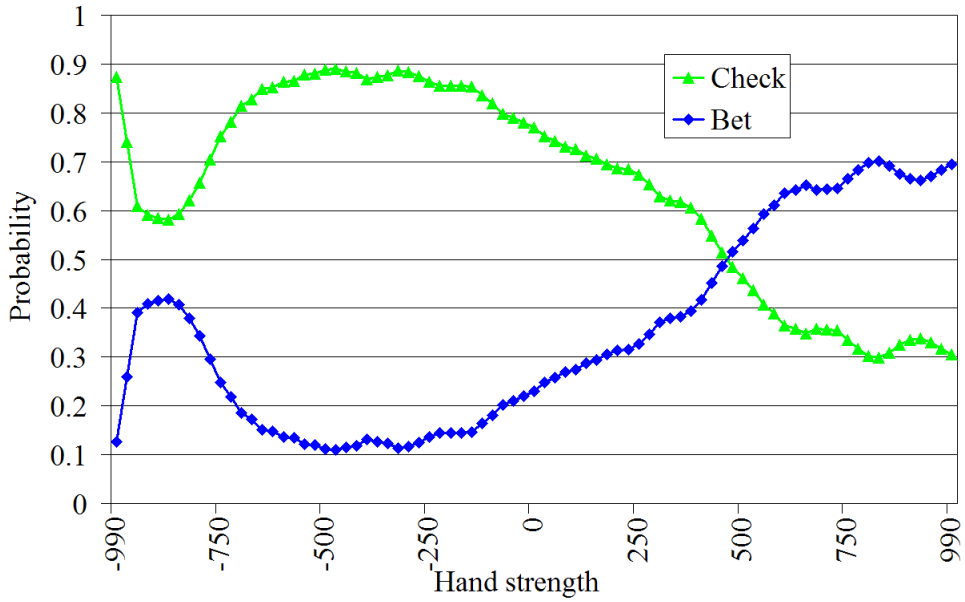


Figure 4: First player's empirical action probabilities (for folding, calling, and raising) as a function of hand strength at the beginning of the fourth betting round.

Of course, since these probabilities are only conditioned on the hand value (and ignore the betting history in the first three rounds), they do not exactly capture the strategy used by *Sparbot* in the fourth round. However, conditioning the probabilities on the betting history as well would have required a vast number of additional trials, as well as much more space to store the result. Conditioning the actions on hand value alone is a practical way of striking that trade-off.

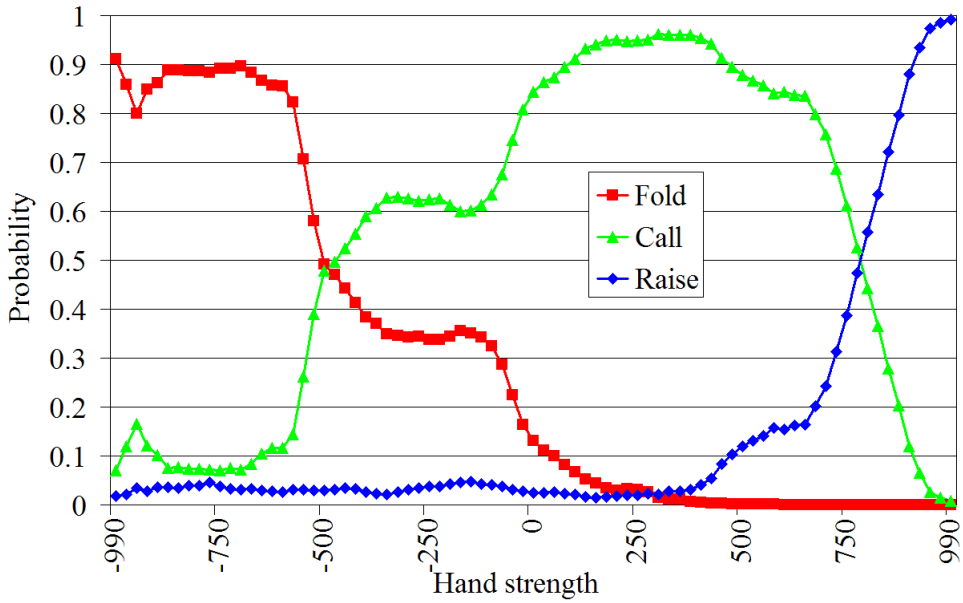


Figure 5: Second player's empirical action probabilities (for folding, calling, and raising) as a function of hand strength for the fourth betting round after the first player has bet.

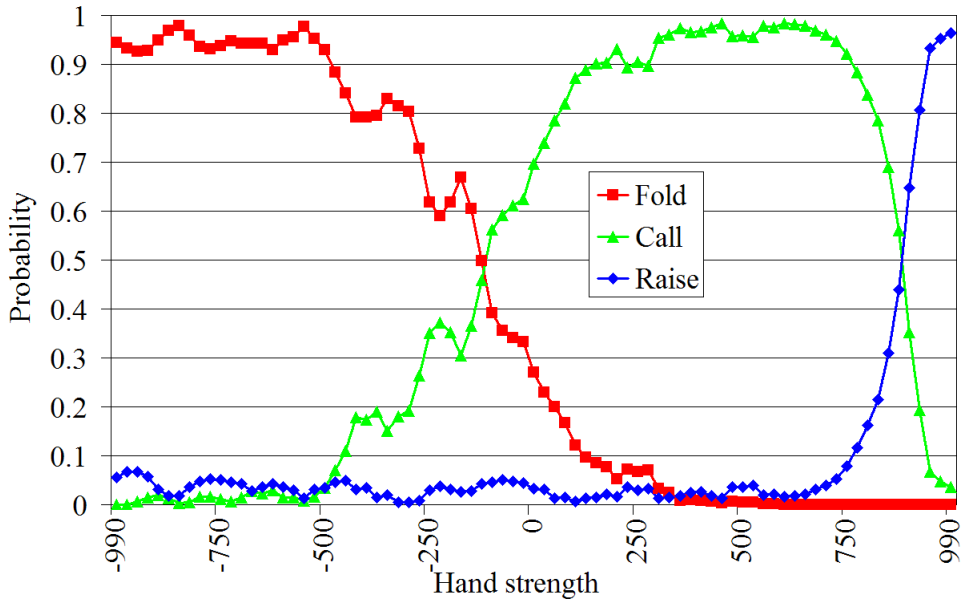


Figure 6: First player's empirical action probabilities (for folding, calling, and raising) as a function of hand strength for the fourth betting round after the first player has bet and the second player has raised.

### 4.3 Potential-aware automated abstraction

The automated abstraction algorithm presented in Section 4.1 was based on a myopic expected-value computation. A state of the game was evaluated according to the probability of winning the hand. The algorithm clustered together states with similar probabilities of winning, and it started computing the abstraction from the first round and then down through the card tree.

That approach does not take into account the *potential* of hands. For example, certain poker hands are considered *drawing hands* in which the hand is currently weak, but has a chance of becoming very strong. An important type of drawing hand is one in which the player has four cards of a certain suit (five are required to make a *flush*); at the present stage the hand is not very strong, but could become so if the required card showed up later in the game. 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 the same chance of winning, but without as much potential to improve.[11] However, if using the difference between probabilities of winning as the metric for performing the clustering, the automated abstraction algorithm would consider these two very different situations to be quite 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. In other words, the algorithm would be able to differentiate between two hands that have the same probability of winning, but where one hand faces more uncertainty about what its final strength will be, while the other hand strength is unlikely to change much. Although this would likely improve the basic abstraction algorithm, it 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 be determined after one more step, while the other hand needs two more steps before its final strength is determined.

To address this, we introduce an approach where we associate with each state of the game a *histogram* over future possible states. This representation can encode all the pertinent information from the rest of the game, such as the probability of winning, the variance of winning, and the paths of information revelation. In such a scheme, the $k$-means clustering step requires a distance function to measure the dissimilarity between different states. The metric we use in this paper is the usual $L_2$-distance metric. Given a finite set $\mathcal{S}$ of states where each hand $i$ is associated with histogram $h_i$ over the future possible states $\mathcal{S}$, the $L_2$-distance between hands $i$ and $j$ is

$$dist(i,j) = \left[\sum_{s \in \mathcal{S}} \left(h_i(s) - h_j(s)\right)^2\right]^{\frac{1}{2}}.$$

There are at least two prohibitive problems with the 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 (*i.e.,* 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 non-overlapping 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 (clusters), which contain a number of states each. We will have, for each cluster, a histogram over clusters later in the game. Second, we restrict the histogram of each cluster to be over clusters at the next level of the game tree only (rather than over clusters at all future levels). However, we introduce a technique (a bottom-up pass of constructing abstractions up the tree) that allows the clusters at the next level to capture information from all later levels.

One way of constructing the histograms would be to perform a bottom-up pass of the card deal tree: abstracting level four (*i.e.,* betting round 4) first, creating histograms for level 3 nodes based on the level 4 clusters, then abstracting level 3, creating histograms for level 2 nodes based on the level 3 clusters, and so on. This is indeed what we do to find the abstraction for level 1.

However, for later betting rounds, we improve on this algorithm further by leveraging our knowledge of the fact that abstracted children of any cluster at the level above should only include states that can actually

[11] In the manual abstraction used in *Sparbot*, there are six buckets of hands where the hands are selected based on likelihood of winning and one extra bucket for hands that an expert considered to have high potential [11]. In contrast, our approach is automated, and does its bucketing holistically based on a multi-dimensional notion of potential (so it does not separate buckets into ones based on winning probability and ones based on potential). Furthermore, its abstraction is drastically finer grained.

be children of the states in that cluster. We do this by *multiple* bottom-up passes, one for each cluster at the level above. For example, if a cluster at level 1 contains only those states where the hand consists of two Aces, then when we are doing abstraction for level 2, the bottom-up pass for that level-1 cluster 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 levels. The following subsections describe our abstraction algorithm in detail.

#### 4.3.1 Computing the abstraction for round 1

The first piece of the abstraction we computed was for the first round, *i.e.,* the pre-flop. In this round we have a target of 20 buckets, out of the $\binom{52}{2} = 1{,}326$ possible combinations of cards. As discussed above, we will have, for each pair of hole cards, a histogram over clusters of cards at level 2. (These clusters are not necessarily the same that we will eventually use in the abstraction for level 2, discussed later.)

To obtain the level-2 clusters, we perform a bottom-up pass of the card tree as follows. Starting with the fourth round, we cluster the $\binom{52}{2}\binom{50}{5} = 2{,}809{,}475{,}760$ hands into 5 clusters[12] based on the probability of winning. Next, we consider the $\binom{52}{2}\binom{50}{4} = 305{,}377{,}800$ third-round hands. For each hand we compute its histogram over the 5 level-4 clusters we computed. Then, we perform $k$-means clustering on these histograms to identify 10 level-3 clusters. We repeat a similar procedure for the $\binom{52}{2}\binom{50}{3} = 25{,}989{,}600$ hands in the second round to identify 20 level-2 clusters.

Using those level-2 clusters, we compute the 20-dimensional histograms for each of the $\binom{52}{2} = 1{,}326$ possible hands at level 1 (*i.e.,* in the first betting round). Then we perform $k$-means clustering on these histograms to obtain the 20 buckets that constitute our abstraction for the first betting round.

#### 4.3.2 Computing the abstraction for rounds 2 and 3

Just as we did in computing the abstraction for the first round, we start by performing a bottom-up clustering, beginning in the fourth round. However, instead of doing this bottom-up pass once, we do it once for each bucket in the first round. Thus, instead of considering all $\binom{52}{2}\binom{50}{5} = 2{,}809{,}475{,}760$ hands in each pass, we only consider those hands which contain as the hole cards those pairs that exist in the particular first-round bucket we are looking at.

At this point we have, for each first-round bucket, a set of second-round clusters. For each first-round bucket, we have to determine how many child buckets it should actually have. For each first-round bucket, we run $k$-means clustering on its second-round clusters for $k \in \{1..80\}$. (In other words, we are clustering those second-round clusters (*i.e.,* data points) into $k$ clusters.) This yields, for each first-round bucket and each value of $k$, an error measure for that bucket assuming it will have $k$ children. (The error is the sum of each data point's $L_2$ distance from the centroid of its assigned cluster.)

Based on our design of the coarseness of the abstraction, we know that we have a total limit of 800 children (*i.e.,* buckets at level 2) to be spread across the 20 first-round buckets. As in the abstraction algorithm used by *GS2* [48], we formulate and solve an integer program (IP) to determine how many children each first-round bucket should have (*i.e.,* what $k$ should be for that bucket). The IP simply minimizes the sum of the errors of the level-1 buckets under the constraint that their $k$-values do not sum to more than 800. (The optimal $k$-value for different level-1 buckets varied between 18 and 51.) This determines the final bucketing for the second betting round.

The bucketing for the third betting round is computed analogously. We use level-2 buckets as the starting point (instead of level-1 buckets), and in the integer program we allow a total of 4,800 buckets for the third betting round. (The optimal $k$-value for different level-2 buckets varied between 1 and 10.)

[12] For this algorithm, the number of clusters at each level (5, 10, 20) was chosen to reflect the fact that when clustering data, the number of clusters needed to represent meaningful information should be at least the level of dimensionality of the data. So, the number of clusters on level $k$ should be at least as great as on level $k+1$.

#### 4.3.3 Computing the abstraction for round 4

In round 4 there is no need to use the sophisticated clustering techniques discussed above since the players will not receive any more information. Instead, we simply compute the fourth-round abstraction based on each hand's probability of winning, exactly the way as was done for computing the abstraction for *GS2* [48]. Specifically, for each third-round bucket, we consider all possible rollouts of the fourth round. Each of them constitutes a data point (whose value is computed as the probability of winning plus half the probability of tying), and we run $k$-means clustering on them for $k \in \{1..18\}$. (The optimal $k$-value for different level-3 buckets varied between 1 and 14.) The error, for each third-round bucket and each $k$, is the sum over the bucket's data points, of the data point's $L_2$ distance from the centroid of its cluster.

Finally, we run an IP to decide the $k$ for each third-round bucket, with the objective of minimizing the sum of the third-round buckets' errors under the constraint that the sum of those buckets' $k$-values does not exceed 28,800 (which is the number of buckets allowed for the fourth betting round, as discussed earlier). This determines the final bucketing for the fourth betting round.[13]

### 4.4 Experiments on *GS3*

We tested *GS3* against the leading prior poker programs: *GS2*, *Sparbot*, and *Vexbot*. For the learning opponent, *Vexbot*, we allowed it to learn throughout the 100,000 hands (rather than flushing its memory every so often as is customary in computer poker competitions).

*GS3* outperformed all three players, by statistically significant margins. Table 2 summarizes our results. The variance of heads-up Texas Hold'em has been empirically observed to be $\pm 6/\sqrt{N}$ small bets per hand when $N$ hands are played [9]. This value (for the actual number of hands played) is displayed in the last column. A win rate greater than this value indicates statistical significance. Figure 7 plots *GS3*'s bankroll as a function of the number of hands played.

| **Opponent** | **# of hands** ($N$) | **Small bets we won** (per hand) | $6/\sqrt{N}$ |
|---|---|---|---|
| *Bluffbot* | 20,000 | 0.291 | 0.042 |
| *GS2* | 16,000 | 0.279 | 0.047 |
| *Sparbot* | 200,000 | 0.033 | 0.013 |
| *Vexbot* | 100,000 | 0.130 | 0.019 |

Table 2: Experiments against the leading prior programs.

### 4.5 Applying the techniques to no-limit Texas Hold'em poker

*GS3* was designed to play *limit* Texas Hold'em poker. In that game, the players are limited to folding, calling, or betting a fixed amount. Thus, each information set has at most three actions. In *no-limit* Texas Hold'em poker, on the other hand, the players are allowed to bet any amount (above a certain threshold). Thus, there are actually infinitely many actions at each information set.

Currently we are working on developing techniques for constucting a discretized model of the strategy space based on important features of the game. There are many different ways in which this discretization can take place, and we intend to develop and evaluate several alternatives.

[13] As discussed, our overall technique optimizes the abstraction one betting round at a time. A better abstraction could conceivably be obtained by optimizing all rounds together. However, that seems infeasible. First, the optimization problem would be nonlinear because the probabilities at a given level depend on the abstraction at all previous levels of the tree. Second, the number of decision variables in the problem would be exponential in the size of the card tree (even if the number of abstraction classes for each level is fixed). Third, one would have to solve a $k$-means clustering problem for each of those variables to determine its coefficient in the optimization problem.

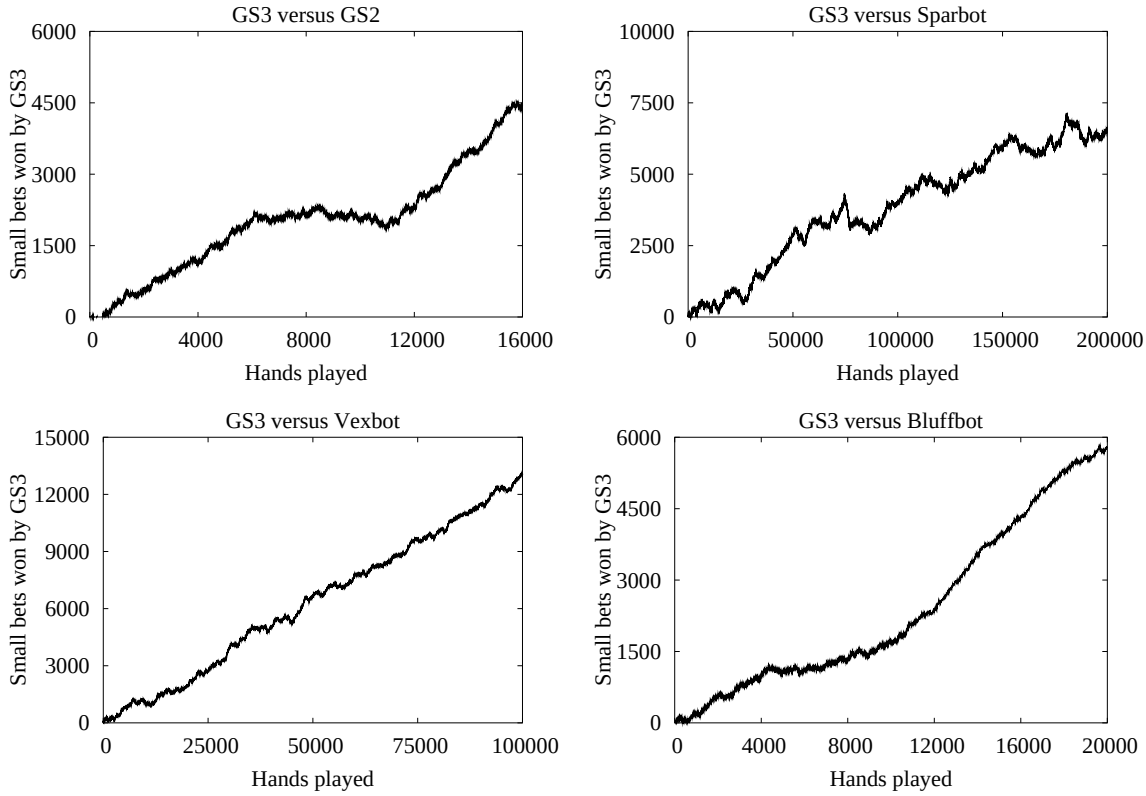


Figure 7: Performance against *GS2*, *Sparbot*, *Bluffbot*, and *Vexbot*.

# 5 Specialized algorithms for finding approximate equilibria [54, 43]

As described in Section 2.3.2, the Nash equilibrium problem for two-player zero-sum sequential games of imperfect information with perfect recall can be formulated using the sequence form representation [103, 61, 129] as the following saddle-point problem:

$$\max_{\mathbf{x}\in Q_1} \min_{\mathbf{y}\in Q_2} \langle A\mathbf{y}, \mathbf{x}\rangle = \min_{\mathbf{y}\in Q_2} \max_{\mathbf{x}\in Q_1} \langle A\mathbf{y}, \mathbf{x}\rangle. \tag{2}$$

In this formulation, $\mathbf{x}$ is player 1's strategy and $\mathbf{y}$ is player 2's strategy. The bilinear term $\langle A\mathbf{y}, \mathbf{x}\rangle$ is the payoff that player 1 receives from player 2 when the players play the strategies $\mathbf{x}$ and $\mathbf{y}$. The strategy spaces are represented by $Q_i \subseteq \mathbb{R}^{|S_i|}$, where $S_i$ is the set of sequences of moves of player $i$, and $Q_i$ is the *set of realization plans* of player $i$. Thus $\mathbf{x}$ ($\mathbf{y}$) encodes probability distributions over actions at each point in the game where player 1 (2) acts. The set $Q_i$ has an explicit linear description of the form $\{z \geq 0 : Ez = \mathbf{e}\}$. Consequently, problem (2) can be modeled as a linear program (see [129] for details).

The linear programs that result from this formulation have size linear in the size of the game tree. Thus, in principle, these linear programs can be solved using any algorithm for linear programming such as the simplex or interior-point methods. For some smaller games, this approach is successful [65]. However, for many games the size of the game tree and the corresponding linear program is enormous.

We propose a new approach to the approximation of Nash equilibria that directly tackles the saddle-point formulation of Equation 2. In particular, we are able to compute, in $O(1/\epsilon)$ iterations, strategies $\mathbf{x}^*$ and $\mathbf{y}^*$

such that

$$\max_{\mathbf{x}\in Q_1}\langle A\mathbf{y}^*,\mathbf{x}\rangle - \min_{\mathbf{y}\in Q_2}\langle A\mathbf{y},\mathbf{x}^*\rangle \le \epsilon. \tag{3}$$

Strategies that satisfy this inequality are called $\epsilon$-*equilibria* (Definition 4). This class of game-theoretic solution concepts encapsulates strategies in which either player can gain at most $\epsilon$ by deviating to another strategy. For most applications this type of approximation is acceptable if $\epsilon$ is small. The algorithms of this section are anytime algorithms and guarantee that $\epsilon$ approaches zero, and quickly find solutions that have a very small $\epsilon$.

Our approach is based on modern smoothing techniques for saddle-point problems. A particularly attractive feature of our approach is its simple work per iteration as well as the low cost per iteration: the most complicated operation is a matrix-vector multiplication involving the payoff matrix $A$. In addition, we can take advantage of the structure of the problem to improve the performance of this operation both in terms of time and memory requirements. As a result, we are able to handle games that are several orders of magnitude larger than games that can be solved using conventional linear programming solvers. For example, we compute approximate solutions to an abstracted version of Texas Hold'em poker whose LP formulation has 18,536,842 rows and 18,536,852 columns, and has 61,450,990,224 non-zeros in the payoff matrix. This is more than 1,200 times the number of non-zeros in the Rhode Island Hold'em problem mentioned above. Since conventional LP solvers require an explicit representation of the problem (in addition to their internal data structures), this would require such a solver to use more than 458 GB of memory *simply to represent the problem*. On the other hand, our algorithm only requires 2.49 GB of memory.

The algorithm we present herein can be seen as a primal-dual first-order algorithm applied to the pair of optimization problems

$$\max_{\mathbf{x}\in Q_1} f(\mathbf{x}) = \min_{\mathbf{y}\in Q_2} \phi(\mathbf{y})$$

where

$$f(\mathbf{x}) = \min_{\mathbf{y}\in Q_2} \langle A\mathbf{y},\mathbf{x}\rangle \;\text{ and }\; \phi(\mathbf{y}) = \max_{\mathbf{x}\in Q_1} \langle A\mathbf{y},\mathbf{x}\rangle.$$

It is easy to see that $f$ and $\phi$ are respectively concave and convex non-smooth (*i.e.* not differentiable) functions. Our algorithm is based on a modern smoothing technique for non-smooth convex minimization [91]. This smoothing technique provides first-order algorithms whose theoretical complexity to find a feasible primal-dual solution with gap $\epsilon > 0$ is $O(1/\epsilon)$ iterations. We note that this is a substantial improvement to the black-box generic complexity bound $O(1/\epsilon^2)$ of general first-order methods for non-smooth convex minimization (concave maximization) [90].

Some recent work has applied smoothing techniques to the solution of large-scale semidefinite programming problems [77] and to large-scale linear programming problems [23]. However, our work appears to be the first application of smoothing techniques to Nash equilibrium computation in sequential games.

## 5.1 Nesterov's excessive gap technique (EGT)

We next describe Nesterov's excessive gap smoothing technique [91], specialized to extensive form games. For $i = 1, 2$, assume that $S_i$ is the set of sequences of moves of player $i$ and $Q_i \subseteq \mathbb{R}^{|S_i|}$ is the *set of realization plans* of player $i$. For $i = 1, 2$, assume that $d_i$ is a strongly convex function on $Q_i$, *i.e.* there exists $\rho_i > 0$ such that

$$d_i(\alpha\mathbf{z} + (1-\alpha)\mathbf{w}) \le \alpha d_i(\mathbf{z}) + (1-\alpha)d_i(\mathbf{w}) - \frac{1}{2}\rho\alpha\|\mathbf{z}-\mathbf{w}\|^2 \tag{4}$$

for all $\alpha \in [0,1]$ and $\mathbf{z}, \mathbf{w} \in Q_i$. The largest $\rho_i$ satisfying (4) is the *strong convexity parameter* of $d_i$. For convenience, we assume that $\min_{\mathbf{z}\in Q_i} d_i(\mathbf{z}) = 0$.

The *prox functions* $d_1$ and $d_2$ can be used to *smooth* the non-smooth functions $f$ and $\phi$ as follows. For $\mu_1, \mu_2 > 0$ consider

$$f_{\mu_2}(\mathbf{x}) = \min_{\mathbf{y}\in Q_2} \{\langle A\mathbf{y},\mathbf{x}\rangle + \mu_2 d_2(\mathbf{y})\}$$

and

$$\phi_{\mu_1}(\mathbf{y}) = \max_{\mathbf{x}\in Q_1} \{\langle A\mathbf{y}, \mathbf{x}\rangle - \mu_1 d_1(\mathbf{x})\}.$$

Because $d_1$ and $d_2$ are strongly convex, it follows [91] that $f_{\mu_2}$ and $\phi_{\mu_1}$ are smooth (*i.e.* differentiable). Notice that $f(\mathbf{x}) \leq \phi(\mathbf{y})$ for all $\mathbf{x} \in Q_1, \mathbf{y} \in Q_2$. Consider the following related *excessive gap condition:*

$$f_{\mu_2}(\mathbf{x}) \geq \phi_{\mu_1}(\mathbf{y}). \tag{5}$$

Let $D_i := \max_{\mathbf{z}\in Q_i} d_i(\mathbf{z})$. If $\mu_1, \mu_2 > 0$, $\mathbf{x} \in Q_1, \mathbf{y} \in Q_2$ and $(\mu_1, \mu_2, \mathbf{x}, \mathbf{y})$ satisfies (5), then [91, Lemma 3.1] yields

$$0 \leq \phi(\mathbf{y}) - f(\mathbf{x}) \leq \mu_1 D_1 + \mu_2 D_2. \tag{6}$$

This suggests the following strategy to find an approximate solution to (2): generate a sequence $(\mu_1^k, \mu_2^k, \mathbf{x}^k, \mathbf{y}^k)$, $k = 0, 1, \ldots$, with $\mu_1^k$ and $\mu_2^k$ decreasing to zero as $k$ increases, while $\mathbf{x}^k \in Q_1$, $\mathbf{y}^k \in Q_2$ and while maintaining the loop invariant that $(\mu_1^k, \mu_2^k, \mathbf{x}^k, \mathbf{y}^k)$ satisfies (5). This is the strategy underlying the EGT algorithms we present in this paper.

The building blocks of our algorithms are the mapping sargmax and the procedures `initial` and `shrink`. Let $d$ be a strongly convex function with a convex, closed, and bounded domain $Q \subseteq \mathbb{R}^n$. Let $\text{sargmax}(d, \cdot) : \mathbb{R}^n \to Q$ be defined as

$$\text{sargmax}(d, \mathbf{g}) := \operatorname*{argmax}_{\mathbf{x}\in Q} \{\langle \mathbf{g}, \mathbf{x}\rangle - d(\mathbf{x})\}. \tag{7}$$

By [91, Lemma 5.1], the following procedure `initial` yields an initial point that satisfies the excessive gap condition (5). The notation $\|A\|$ indicates an appropriate operator norm (see [91] and Examples 1 and 2 for details), and $\nabla d_2(\hat{\mathbf{x}})$ is the gradient of $d_2$ at $\hat{\mathbf{x}}$.

`initial`$(A, d_1, d_2)$

1. $\mu_1^0 := \mu_2^0 := \frac{\|A\|}{\sqrt{\rho_1\rho_2}}$
2. $\hat{\mathbf{y}} := \text{sargmax}(d_2, \mathbf{0})$
3. $\mathbf{x}^0 := \text{sargmax}\left(d_1, \frac{1}{\mu_1^0} A\hat{\mathbf{y}}\right)$
4. $\mathbf{y}^0 := \text{sargmax}\left(d_2, \nabla d_2(\hat{\mathbf{x}}) + \frac{1}{\mu_2^0} A^{\mathrm{T}}\mathbf{x}^0\right)$
5. `return` $(\mu_1^0, \mu_2^0, \mathbf{x}^0, \mathbf{y}^0)$

The following procedure `shrink` enables us to reduce $\mu_1$ and $\mu_2$ while maintaining (5).

`shrink`$(A, \mu_1, \mu_2, \tau, \mathbf{x}, \mathbf{y}, d_1, d_2)$

1. $\breve{\mathbf{y}} := \operatorname{sargmax}\left(d_2, -\frac{1}{\mu_2} A^{\mathrm{T}}\mathbf{x}\right)$
2. $\hat{\mathbf{y}} := (1-\tau)\mathbf{y} + \tau\breve{\mathbf{y}}$
3. $\hat{\mathbf{x}} := \operatorname{sargmax}\left(d_1, \frac{1}{\mu_1} A\hat{\mathbf{y}}\right)$
4. $\tilde{\mathbf{y}} := \operatorname{sargmax}\left(d_2, \nabla d_2\left(\breve{\mathbf{y}}\right) + \frac{\tau}{(1-\tau)\mu_2} A^{\mathrm{T}}\hat{\mathbf{x}}\right)$
5. $\mathbf{x}^+ := (1-\tau)\mathbf{x} + \tau\hat{\mathbf{x}}$
6. $\mathbf{y}^+ := (1-\tau)\mathbf{y} + \tau\tilde{\mathbf{y}}$
7. $\mu_2^+ := (1-\tau)\mu_2$
8. `return` $(\mu_2^+, \mathbf{x}^+, \mathbf{y}^+)$

By [91, Theorem 4.1], if the input $(\mu_1, \mu_2, \mathbf{x}, \mathbf{y})$ to `shrink` satisfies (5) then so does $(\mu_1, \mu_2^+, \mathbf{x}^+, \mathbf{y}^+)$ as long as $\tau$ satisfies $\tau^2/(1-\tau) \leq \mu_1\mu_2\rho_1\rho_2\|A\|^2$. Consequently, the iterates generated by procedure `EGT` below satisfy (5). In particular, after $N$ iterations, Algorithm `EGT` yields points $\mathbf{x}^N \in Q_1$ and $\mathbf{y}^N \in Q_2$ with

$$0 \leq \max_{\mathbf{x}\in Q_1} \langle A\mathbf{y}^N, \mathbf{x}\rangle - \min_{\mathbf{y}\in Q_2} \langle A\mathbf{y}, \mathbf{x}^N\rangle \leq \frac{4\,\|A\|}{N}\sqrt{\frac{D_1 D_2}{\rho_1\rho_2}}.$$

`EGT`

1. $(\mu_1^0, \mu_2^0, \mathbf{x}^0, \mathbf{y}^0) =$ `initial`$(A, d_1, d_2)$
2. `For` $k = 0, 1, \ldots$:
   (a) $\tau := \frac{2}{k+3}$
   (b) `If` $k$ `is even:  // shrink` $\mu_2$
      i. $(\mu_2^{k+1}, \mathbf{x}^{k+1}, \mathbf{y}^{k+1}) :=$ `shrink`$(A, \mu_1^k, \mu_2^k, \tau, \mathbf{x}^k, \mathbf{y}^k, d_1, d_2)$
      ii. $\mu_1^{k+1} := \mu_1^k$
   (c) `If` $k$ `is odd:  // shrink` $\mu_1$
      i. $(\mu_1^{k+1}, \mathbf{y}^{k+1}, \mathbf{x}^{k+1}) :=$ `shrink`$(A^{\mathrm{T}}, -\mu_1^k, -\mu_2^k, \tau, \mathbf{y}^k, \mathbf{x}^k, d_2, d_1)$
      ii. $\mu_2^{k+1} := \mu_2^k$

Notice that Algorithm `EGT` is a *conceptual* algorithm that finds an $\epsilon$-solution to (2). It is conceptual only because the algorithm requires that the mappings $\operatorname{sargmax}(d_i, \cdot)$ be computed several times at each iteration. Consequently, a specific choice of the functions $d_1$ and $d_2$ is a critical step to convert Algorithm `EGT` into an actual algorithm.

## 5.2 Nice prox functions

Assume $Q$ is a convex, closed, and bounded set. We say that a function $d : Q \to \mathbb{R}$ is a *nice prox function* for $Q$ if it satisfies the following three conditions:

1. $d$ is strongly convex and continuous everywhere in $Q$ and is differentiable in the relative interior of $Q$;

2. $\min\{d(\mathbf{z}) : \mathbf{z} \in Q\} = 0$;

3. The mapping $\text{sargmax}(d, \cdot)\colon \mathbb{R}^n \to Q$ is easily computable, *e.g.*, it has a closed-form expression.

We next provide two specific examples of nice prox functions for the simplex

$$\Delta_n = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{x} \geq 0, \sum_{i=1}^{n} x_i = 1\}.$$

**Example 5.1** *Consider the* entropy *function*

$$d(x_1, \ldots, x_n) = \ln n + \sum_{i=1}^{n} x_i \ln x_i.$$

*The function $d$ is strongly convex and continuous in $\Delta_n$ and $\min_{\mathbf{x}\in\Delta_n} d(\mathbf{x}) = 0$. It is also differentiable in the relative interior of $\Delta_n$. It has strong convexity parameter $\rho = 1$ for the 1-norm in $\mathbb{R}^n$, namely, $\|\mathbf{x}\| = \sum_{i=1}^{n} |x_i|$. The corresponding operator norm, $\|A\|$, for this setting is simply the value of the largest entry in $A$ in absolute value. Finally, the mapping* $\text{sargmax}(d, \mathbf{g})$ *has the closed-form expression*

$$\text{sargmax}(d, \mathbf{g})_j = \frac{e^{g_j}}{\sum_{i=1}^{n} e^{g_i}}.$$

**Example 5.2** *Consider the (squared)* Euclidean distance *to the center of $\Delta_n$, that is,*

$$d(x_1, \ldots, x_n) = \sum_{i=1}^{n} \left(x_i - \frac{1}{n}\right)^2.$$

*This function is strongly convex, continuous and differentiable in $\Delta_n$, and $\min_{\mathbf{x}\in\Delta_n} d(\mathbf{x}) = 0$. It has strong convexity parameter $\rho = 1$ for the Euclidean norm, namely, $\|\mathbf{x}\| = \left(\sum_{i=1}^{n} |x_i|^2\right)^{1/2}$. The corresponding operator norm, $\|A\|$, for this setting is the spectral norm of $A$,* i.e. *the square root of the largest eigenvalue of $A^T A$. Although the mapping* $\text{sargmax}(d, \mathbf{g})$ *does not have a closed-form expression, it can easily be computed in $O(n \log n)$ steps [23].*

In order to apply Algorithm `EGT` to problem (2) for sequential games we need nice prox-functions for the realization sets $Q_1$ and $Q_2$ (which are more complex than the simplex discussed above in Examples 1 and 2). This problem was recently solved [54]:

**Theorem 4** *Any nice prox-function $\psi$ for the simplex induces a nice prox-function for a set of realization plans $Q$. The mapping* $\text{sargmax}(d, \cdot)$ *can be computed by repeatedly applying* $\text{sargmax}(\psi, \cdot)$.

## 5.3 Customizing the algorithm for poker games

The bulk of the computational work at each iteration of the EGT algorithm consists of some matrix-vector multiplications $\mathbf{x} \mapsto A^{\mathrm{T}}\mathbf{x}$ and $\mathbf{y} \mapsto A\mathbf{y}$ in addition to some calls to the mappings $\text{smax}(d_i, \cdot)$ and $\text{sargmax}(d_i, \cdot)$. Of these operations, the matrix-vector multiplications are by far the most expensive, both in terms of memory (for storing $A$) and time (for computing the product).

### 5.3.1 Addressing the space requirements

To address the memory requirements, we exploit the problem structure to obtain a concise representation for the payoff matrix $A$. This representation relies on a uniform structure that is present in poker games and many other games. For example, the betting sequences that can occur in most poker games are independent

of the cards that are dealt. This conceptual separation of betting sequences and card deals is used by automated abstraction algorithms [46]. Analogously, we can decompose the payoff matrix based on these two aspects.

The basic operation we use in this decomposition is the *Kronecker product*, denoted by $\otimes$. Given two matrices $B \in \mathbb{R}^{m \times n}$ and $C \in \mathbb{R}^{p \times q}$, the Kronecker product is

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

For ease of exposition, we explain the concise representation in the context of Rhode Island Hold'em poker [117], although the general technique applies much more broadly. The payoff matrix $A$ can be written as

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

where $A_1 = F_1 \otimes B_1$, $A_2 = F_2 \otimes B_2$, and $A_3 = F_3 \otimes B_3 + S \otimes W$ 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.

Given this concise representation of $A$, computing $\mathbf{x} \mapsto A^{\mathrm{T}}\mathbf{x}$ and $\mathbf{y} \mapsto A\mathbf{y}$ is straightforward, and the space required is sublinear in the size of the game tree. For example, in Rhode Island Hold'em, the dimensions of the $F_i$ and $S$ matrices are $10 \times 10$, and the dimensions of $B_1$, $B_2$, and $B_3$ are $13 \times 13$, $205 \times 205$, and 1,774 $\times$ 1,774, respectively—in contrast to the $A$-matrix, which is 883,741 $\times$ 883,741. Furthermore, the matrices $F_i$, $B_i$, $S$, and $W$ are themselves sparse which allows us to use the Compressed Row Storage (CRS) data structure (which stores only non-zero entries).

Table 3 provides the sizes of the four test instances; each models some variant of poker, an important challenge problem in AI [12]. The first three instances, `10k`, `160k`, and `RI`, are abstractions of Rhode Island Hold'em [117] computed using the *GameShrink* automated abstraction algorithm [46]. The first two instances are lossy (non-equilibrium preserving) abstractions, while the `RI` instance is a lossless abstraction. The last instance, `Texas`, is a lossy abstraction of Texas Hold'em. We wanted to test the algorithms on problems of widely varying sizes, which is reflected by the data in Table 3. We also chose these four problems because we wanted to evaluate the algorithms on real-world instances, rather than on randomly generated games (which may not reflect any realistic setting).

| **Name** | **Rows** | **Columns** | **Non-zeros** |
|---|---:|---:|---:|
| `10k` | 14,590 | 14,590 | 536,502 |
| `160k` | 226,074 | 226,074 | 9,238,993 |
| `RI` | 1,237,238 | 1,237,238 | 50,428,638 |
| `Texas` | 18,536,842 | 18,536,852 | 61,498,656,400 |

Table 3: Problem sizes (when formulated as an LP) for the instances used in our experiments.

Table 4 clearly demonstrates the extremely low memory requirements of the EGT algorithms. Most notably, on the `Texas` instance, both of the CPLEX algorithms require more than 458 GB simply to *represent* the problem. In contrast, using the decomposed payoff matrix representation, the EGT algorithms require only 2.49 GB. Furthermore, in order to solve the problem, both the simplex and interior-point algorithms would require additional memory for their internal data structures.[14] Therefore, the EGT family of

[14]The memory usage for the CPLEX simplex algorithm reported in Table 4 is the memory used after 10 minutes of execution (except for the `Texas` instance which did not run at all as described above). This algorithm's memory requirements grow and shrink during the execution depending on its internal data structures. Therefore, the number reported is a lower bound on the maximum memory usage during execution.

algorithms is already an improvement over the state-of-the-art (even without the heuristics).

| 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 |

Table 4: Memory footprint in gigabytes of CPLEX interior-point method (IPM), CPLEX Simplex, and EGT algorithms. CPLEX requires more than 458 GB for the `Texas` instance.

#### 5.3.2 Speedup from parallelizing the matrix-vector product

To address the time requirements of the matrix-vector product, we can effectively parallelize the operation by simply partitioning the work into $n$ pieces when $n$ CPUs are available. The speedup we can achieve on parallel CPUs is demonstrated in Table 5. The instance used for this test is the `Texas` instance described above. The matrix-vector product operation scales linearly in the number of CPUs, and the time to perform one iteration of the algorithm (using the entropy prox function and including the time for applying Heuristic 1) scales nearly linearly, decreasing by a factor of 3.72 when using 4 CPUs.

| CPUs | matrix-vector product | | EGT iteration | |
|---|---|---|---|---|
| | time (s) | speedup | time (s) | speedup |
| 1 | 278.958 | 1.00x | 1425.786 | 1.00x |
| 2 | 140.579 | 1.98x | 734.366 | 1.94x |
| 3 | 92.851 | 3.00x | 489.947 | 2.91x |
| 4 | 68.831 | 4.05x | 383.793 | 3.72x |

Table 5: Effect of parallelization for the `Texas` instance.

### 5.4 Heuristics for speeding up the excessive gap technique

We mention that we have developed heuristics for further speeding up the excessive gap technique family of algorithms. These have led to speed improvements of more than an order of magnitude. However, these heuristics are not a component of this thesis proposal and will not be described here.

## 6 Worst-case guarantees

In this section we propose a line of research that aims to develop strong guarantees for the strategies computed. These guarantees would be bounds on the performance of a computed strategy (measured by expected utility). We are primarily interested in the two-person zero-sum setting, although we do not wish to artificially limit ourselves to this setting.

### 6.1 *Ex ante* guarantees

The plan of attack for developing guarantees is twofold. First, we are interested in *ex ante* guarantees, in which the guarantee is a function of an input parameter to a lossy automated abstraction algorithm. For example, it would be helpful to have a characterization of how far from optimal in the original game our computed strategies are as a function of the number of buckets allowed in the abstraction.

### 6.2 *Ex post* guarantees

The second line of attack is the development of techniques for computing *ex post* guarantees in the original game. In this approach, we would like to design algorithms for computing bounds in the original game, given an approximately optimal strategy for that game. In other words, we would be determining a precise lower bound on the performance of the strategy in the real game.

## 7 Selective updating with the excessive gap technique

As discussed in Section 5, the biggest computational bottleneck in running the EGT algorithm is in the computation of the three matrix-vector products in each iteration. In this section we describe a possible modification to the algorithm that can eliminate parts of this computation by saving solutions from previous iterations. The motivating idea is that certain parts of the game are easy to play, and the algorithm quickly finds assignments for the variables corresponding to these parts of the game. Once these assignments are found, the values of these variables do not change very much and we can afford to just consider them as fixed. On the other hand, some parts of the game are difficult to play, and the later stages of the convergence process are primarily geared towards finding appropriate assignments to the variables corresponding to these parts of the game. By saving the portion of the matrix-vector product corresponding to the easy parts of the game (whose values are fixed), we can eliminate a portion of the matrix-vector computation by reusing the answer from the previous iteration.

More formally, let $A$ be the payoff matrix and let $I$ be the index set for the part of the game we are considering fixed (the easy part of the game). Let $J = I^c$ be the index set for the part of the game where we are still focusing our computational efforts. In each iteration, we want to compute the product $Ax$ for a given vector $x$. Let $A_I$ and $A_J$ be the columns of $A$ indexed by $I$ and $J$, and similarly let $x_I$ and $x_J$ be the entries of $x$ indexed by $I$ and $J$. Thus we can write

$$Ax = A_I x_I + A_J x_J.$$

By saving the portion $A_I x_I$ from the previous iteration, we can eliminate some of the computations required to compute $Ax$.

### 7.1 Research questions

There are many design decisions that will need to be determined in the course of this development. The first is how to determine if a part of the game has already converged or not. Along these lines, we would like to determine what additional information from the game we can incorporate into making this decision. For example, we may want to use the concept of the betting round in poker to guide these decisions.

A second research question is how to design the internal data structures so that they can efficiently handle the matrix-vector operations discussed above. Actually splitting the matrix $A$ into the two pieces $A_I$ and $A_J$ will be computationally burdensome. This is particularly relevant since we will likely want to change the index sets $I$ and $J$ often to take advantage of stationary portions of the game. Hence, we will need to figure out a representation for the payoff matrix (while still using the matrix representation based on composition of Kronecker products discussed in Section 5.3.1) that can efficiently support the needed operations. Furthermore, we will want this data structure to enable the high-level of parallelization achievable currently in the matrix-vector product computation.

A third research question is how to maintain the convergence properties of the algorithm. Performing the selective updates means that certain parts of the game remain fixed. However, once $\mu$ gets small enough, it may be required to adjust these fixed variables (probably by small amounts).

As the research in this area is still quite preliminary, likely even more design decisions will reveal themselves during the course of the research.

# 8 Specialized interior-point methods for computing equilibria and equilibrium refinements

Although we have found first-order (gradient-based) methods highly effective at quickly finding $\epsilon$-equilibria in large sequential games, the convergence that these algorithms guarantee is $O(1/\epsilon)$. On the other hand, interior-point methods have $O(\log 1/\epsilon)$ convergence. Unfortunately, current interior-point methods do not scale to the same problem sizes as do first-order methods. An interesting line of research concerns improving the scalability of interior-point methods by taking advantage of certain problem structure. We discuss interior-point methods (Section 8.1) and some research possibilities in Section 8.1.3.

Another intriguing possibility is the use of interior-point methods for computing equilibrium refinements. We discuss this in Section 8.2.

## 8.1 Specialized interior-point methods

In this subsection, we review some basic facts about the sequence form representation and interior-point methods, and discuss how such a method could possibly be specialized to the equilibrium problem in order to boost performance and scalability.

### 8.1.1 Sequence form representation and LP formulation

Instead of representing strategies as probability distributions over all possible contingency plans (which would lead to an exponential strategy representation in the worst-case), it is more efficient to represent strategies in terms of their *sequences*. This observation has led to the development of the sequence form representation [103, 61, 129] of extensive form games. In this representation, strategies are represented as *realization weights* over sequences. Specifically, player 1's space of possible realization weights can be encoded as the solutions to a simple system of linear inequalities: $S_1 = \{x \geq 0 : Ex = e\}$. Here, the number of rows in $E$ is equal to the number of information sets of player 1 (plus one), and the number of columns of $E$ is equal to the number of actions of player 1 (plus one). The entries of $E$ are from the set $\{-1, 0, 1\}$, and encode the hierarchical relationship of player 1's actions. (Hence the representation of the set of strategies is linear in the size of the game tree.) The vector $e$ contains a one as the first entry, and zeros as the remaining entries. Similarly, player 2's set of realization weights can be represented as $S_2 = \{y \geq 0 : Fy = f\}$ for a similarly defined matrix $F$ and vector $f$. (See [129] for details.)

In addition to the compact representation of strategies via linear inequalities, we can represent the payoff function as a bilinear product. Specifically, if player 1 uses (randomized) strategy $x \in S_1$ and player 2 uses (randomized) strategy $y \in S_2$, there is a payoff matrix $A$ such that $x^T Ay$ is player 1's payoff and $x^T(-A)y$ is player 2's payoff. Each entry of $A$ corresponds to one or more leaves in the original game (hence the encoding size of $A$ is linear in the size of the game tree). Thus, we can state the Nash equilibrium problem for two-person zero-sum extensive form games in terms of $A$, $E$, $e$, $F$, and $f$. In particular, we are interested in solving the following saddle-point problem:

$$\max_{x \in S_1} \min_{y \in S_2} x^T Ay = \min_{y \in S_2} \max_{x \in S_1} x^T(-A)y$$

This problem can be solved via the following (primal) LP [129].

$$(P) \quad \begin{array}{lrcrcl} \min\limits_{y,p} & e^T p & & & & \\ \text{s.t.} & E^T p & - & Ay & \geq & 0 \\ & & & -Fy & = & -f \\ & & & y & \geq & 0 \end{array}$$

The dual of this linear program is as follows.

$$
(D) \quad \begin{array}{lrcrcl} \max\limits_{x,q} & -f^T q & & & & \\ \text{s.t.} & -F^T q & - & A^T x & \leq & 0 \\ & & & Ex & = & e \\ & & & x & \geq & 0 \end{array}
$$

Solving (either of) the above problems yields Nash equilibrium strategies $x$ and $y$ for the game.

It will be useful to write the primal and dual problems in which they consist only of equality constraints (except for the non-negativity constraints on some of the variables). This is done by adding *slack variables* $s$ and $t$. The primal is:

$$
(P') \quad \begin{array}{lrcrcrcl} \min\limits_{y,p} & e^T p & & & & & & \\ \text{s.t.} & E^T p & - & Ay & - & s & = & 0 \\ & & & -Fy & & & = & -f \\ & & & y & & & \geq & 0 \\ & & & & & s & \geq & 0 \end{array}
$$

And the dual is:

$$
(D') \quad \begin{array}{lrcrcrcl} \max\limits_{x,q} & -f^T q & & & & & & \\ \text{s.t.} & -F^T q & - & A^T x & + & t & = & 0 \\ & & & Ex & & & = & e \\ & & & x & & & \geq & 0 \\ & & & & & t & \geq & 0 \end{array}
$$

### 8.1.2 Linear programming central path

Using the complementary slackness optimality condition (see, *e.g.,* [24]), we can write the optimality conditions for the primal and dual problems as the following nonlinear feasibility problem (see, *e.g.,* [131]):

$$
(PD) \quad \begin{array}{rcrcrcl} E^T p & - & Ay & - & s & = & 0 \\ -F^T q & - & A^T x & + & t & = & 0 \\ & & Ex & & & = & e \\ & & Fy & & & = & f \\ & & & & XS\mathbf{1} & = & 0 \\ & & & & YT\mathbf{1} & = & 0 \\ & & & & x, s & \geq & 0 \\ & & & & y, t & \geq & 0 \end{array}
$$

In the above equations, we are denoting by $X$ the diagonal matrix whose diagonal entry $X_{ii}$ is $x_i$ and other entries are zero. We are using a similar notion for $S$, $Y$, and $T$. The symbol $\mathbf{1}$ represents the vector of all ones.

Note that this problem is only "mildly" nonlinear: the constraints $XS\mathbf{1} = 0$ and $YT\mathbf{1} = 0$ are mildly nonlinear in the sense that each of them includes just a multiplication of two variables, and the rest of the constraints are linear. The strategy employed by interior-point methods is to relax these two sets of nonlinear constraints, and solve a sequence of relaxed problems that leads to an optimal solution to the original primal and dual problems. To this end, consider the following relaxed version of the optimality conditions:

$$
(PD(\nu)) \qquad
\begin{array}{rcrcrcl}
E^T p & - & Ay & - & s & = & 0 \\
-F^T q & - & A^T x & + & t & = & 0 \\
 & & Ex & & & = & e \\
 & & Fy & & & = & f \\
 & & & & XS\mathbf{1} & = & \nu\mathbf{1} \\
 & & & & YT\mathbf{1} & = & \nu\mathbf{1} \\
 & & & & x, s & > & 0 \\
 & & & & y, t & > & 0
\end{array}
$$

The first four equations are the same. The nonlinear constraints have a different right-hand side, and the non-negativity constraints are replaced by positivity constraints. Here, $\nu > 0$ is the *smoothing parameter*. For a given $\nu > 0$, let $\{x(\nu), y(\nu), p(\nu), q(\nu), s(\nu), t(\nu)\}$ denote a solution to $(PD(\nu))$. The curve $\{x(\nu), y(\nu), p(\nu), q(\nu), s(\nu), t(\nu) : \nu > 0\}$ is called the *central path* of problem $(PD)$.

We now review three basic results from the theory of interior-point methods (see, *e.g.,* [131]), which will turn out to be useful for our purposes.

**Fact 1** *For every $\nu > 0$, there is a unique solution $\{x(\nu), y(\nu), p(\nu), q(\nu), s(\nu), t(\nu)\}$ to $(PD(\nu))$.*

(The above result can be proven by showing that solutions to $PD(\nu)$ are the minimizers of a strictly convex function (namely the duality gap plus the logarithmic barrier function for the nonnegative orthant), and hence unique.)

**Fact 2** *The central path $\{x(\nu), y(\nu), p(\nu), q(\nu), s(\nu), t(\nu) : \nu > 0\}$ is continuous in $\nu$.*

(This result can be proven via an application of the implicit function theorem (see, *e.g.,* [67]).)

Finally, the central path converges to an optimal solution of the primal-dual problem $(PD)$:

**Fact 3** *If*

$$\{x^*, y^*, p^*, q^*, s^*, t^*\} = \lim_{\nu \to 0} \{x(\nu), y(\nu), p(\nu), q(\nu), s(\nu), t(\nu)\}$$

*then $\{x^*, y^*, p^*, q^*, s^*, t^*\}$ is an optimal solution to $(PD)$. (In fact, $\{x^*, y^*, p^*, q^*, s^*, t^*\}$ is the analytic center of the optimal face of the LP polytope.) In other words, $(x^*, y^*)$ specifies a Nash equilibrium for the game.*

### 8.1.3 Specializing interior-point algorithms for equilibrium problems

The main computational work in each iteration is in solving a linear system whose matrix is the Jacobian of problem (PD). Developing a new solution technique for this linear system is the key to improving the scalability of interior-point methods as applied to sequential games.

Typically, this linear system is solved by computing a symbolic Cholesky factorization[15] at the beginning of the problem solve. Then, a numerical factorization is performed at each iteration (based on the symbolic factorization and the actual values of $x$, $s$, $y$, and $t$ at that iteration). The numerical factorization runs in time linear in the size of the Cholesky factorization. However, the Cholesky factorization can be quite dense. In fact, even a single dense column in $A$ can lead to an extremely dense Cholesky factorization. (See [131] for details.)

Avoiding the Cholesky factorization (and other such *direct methods*) thus seems desirable. One possibility is to consider iterative methods for solving linear systems. These algorithms do not require factorizations, but instead require simply computing matrix-vector products—something that we can do very well thanks to this operation's importance in the EGT algorithm. However, applying an iterative method is not a straightforward process. In order for an iterative method to be successful, it will most likely require finding appropriate preconditioners for the problem. It is not clear at all if this is even possible, but if it can be

[15] If $A$ is positive semidefinite, then an upper-triangular matrix $U$ is the Cholesky factorization of $A$ if $A = U^T U$.

made to work it could potentially have a major impact on finding equilibria in games even larger than those solvable by the EGT algorithms, due to the faster convergence of interior-point methods.

Along these lines, another possible improvement that could be made for interior-point methods as applied to solving games involves *warm starting* the algorithm. This is the process by which an algorithm is fed a reasonable (non-optimal) solution as a starting point. Until now, warm starting methods have been unsuccessful when applied to the general linear programming problem. The difficulty lies in finding a solution that also satisfies certain technical properties, namely lying in a certain neighborhood of the central path [131]. However, it may be the case that for special cases of LP (such as finding equilibria in games) it will be possible to devise such a warm start procedure.

As discussed previously, the excessive gap technique family of algorithms achieves an $\epsilon$-solution in $O(1/\epsilon)$ iterations, while interior-point methods typically run in $O(\log 1/\epsilon)$ iterations. Thus, one particularly intriguing approach to solving games would be to run the excessive gap technique to obtain a reasonably good solution, and then warm start a specialized interior-point algorithm to take this good solution to an optimal solution, thus taking advantage of the fast local convergence properties of interior-point methods. This process could potentially leverage information about the solution that can be inferred from properties of the iterates generated by the excessive gap technique.

## 8.2 Equilibrium refinements

The concept of Nash equilibrium in extensive form games suffers from the fact that it does not specify rational behavior at points off of the equilibrium path. In particular, if a player expects a particular point of the game to be reached with probability zero, then it is acceptable under the concept of Nash equilibrium for the player's strategy to be *sequentially irrational*, *i.e.,* to not be the best strategy choice, given the fact that the point of the game was reached. In other words, if the opponent plays in a way that leads to a state of the game that should have been reached with zero probability in equilibrium, an agent with a Nash equilibrium strategy might play nonsensically from then on.

The *sequential equilibrium* [68] is a *refinement* of Nash equilibrium for extensive form games. It strengthens the notion of equilibrium by requiring the specification of rational behavior at states of the game that are "off the equilibrium path", *i.e.,* are reached with probability zero according to the equilibrium strategies. This feature of the sequential equilibrium concept is especially attractive for applications where the computed strategy is going to play against a human opponent who is likely to make mistakes, or against a computerized opponent that does not play Nash equilibrium (*e.g.,* opponents that try to learn during play).

Recently, it was shown that a sequential equilibrium of a two-player zero-sum extensive form game can, in theory, be found in polynomial time [84].[16] That work introduces two algorithms: one that is polynomial-time in the worst case but unusable (due primarily to numerical instability) and one that is usable but exponential in the worst case. That paper acknowledged the dilemma between practicality and polynomial-time performance.

We would like to show that that dilemma can be overcome. We propose a research plan to investigate efficient, practical algorithms for finding sequential equilibria in sequential games.

### 8.2.1 Sequential equilibrium

As discussed above, for many applications, Nash equilibrium is not a sufficient solution concept. It allows for the specification of irrational behavior at information sets that are reached with probability zero. For example, if the opponent plays in a way that leads to a state of the game that should have been reached with zero probability in equilibrium, an agent's prescribed Nash equilibrium strategy might be nonsensical from then on. The *sequential equilibrium* [68] solution concept was developed as a way to eliminate such equilibria. Before giving the definition of sequential equilibrium, we need to introduce the concepts of *beliefs* in extensive form games.

[16] There are also fast algorithms for computing other equilibrium refinements, including proper equilibria [83] and normal form perfect equilibria [128].

**Definition 11** *Let* $\Gamma = \langle V, E, P, H, A, u, p \rangle$ *be a two-player extensive form game. A* belief system $\mu : V \setminus Z \to [0,1]$ *is a mapping from the decision nodes of the game to* $[0,1]$*, where* $\sum_{x \in h} \mu(x) = 1$ *for all information sets* $h$ *in the game.*

Beliefs are typically discussed with respect to an associated strategy profile:

**Definition 12** *An* assessment *of a two-player extensive form game* $\Gamma$ *is a pair* $((\beta_1, \beta_2), \mu)$ *where* $(\beta_1, \beta_2)$ *is a behavioral strategy profile for* $\Gamma$ *and* $\mu$ *is a belief system for* $\Gamma$*.*

The following definition captures the intuitive notion that strategies should be optimal with respect to the beliefs.

**Definition 13** *An assessment* $((\beta_1, \beta_2), \mu)$ *is* sequentially rational *if*
$\mathrm{E}[u_1 | (\beta_1, \beta_2), \mu, h] \geq \mathrm{E}[u_1 | (\beta_1', \beta_2), \mu, h]$ *for all* $h \in H_1$*;*
$\mathrm{E}[u_2 | (\beta_1, \beta_2), \mu, h] \geq \mathrm{E}[u_2 | (\beta_1, \beta_2'), \mu, h]$ *for all* $h \in H_2$*.*

The following standard definition is a technical tool that enables us to talk about sequentially rational behavior at information sets that are reached with probability zero.

**Definition 14** *An assessment* $((\beta_1, \beta_2), \mu)$ *of a two-player extensive form game* $\Gamma$ *is* consistent *if there exists a sequence* $\{\beta_1^k, \beta_2^k, \mu^k\}_{k=0}^{\infty}$ *such that*

- $\beta_1^k$ *and* $\beta_2^k$ *are completely mixed (*i.e.*, every action in the game has positive probability);*
- $\mu^k$ *is derived from* $(\beta_1^k, \beta_2^k)$ *using Bayes' rule; and*
- $\lim_{k \to \infty} ((\beta_1^k, \beta_2^k), \mu^k) = ((\beta_1, \beta_2), \mu)$*.*

Combining the above definitions we finally arrive at the definition of sequential equilibrium:

**Definition 15 (Kreps & Wilson 1982)** *A* sequential equilibrium *is an assessment* $((\beta_1, \beta_2), \mu)$ *that is both sequentially rational and consistent.*

An intuitive description of sequential equilibrium is that the beliefs should be sensible given the strategies, and the strategies should be sensible given the beliefs. It is immediate from the definitions above that every sequential equilibrium is a Nash equilibrium. However, there are numerous examples that show that the converse is not true (see, *e.g.,* [68, 94, 84]).

As a way of comparing the Nash and sequential equilibrium concepts, the following fact about Nash equilibria in extensive form games is illustrative. This result can be found in many game theory textbooks (*e.g.* [80, Proposition 9.C.1]).

**Proposition 5** $(\beta_1, \beta_2)$ *is a Nash equilibrium of a two-person zero-sum extensive form game* $\Gamma = \langle V, E, P, H, A, u, p \rangle$ *if and only if there exists a belief system* $\mu$ *such that*

- $\mathrm{E}[u_1 \mid (\beta_1, \beta_2), \mu, h] \geq \mathrm{E}[u_1 \mid (\beta_1', \beta_2), \mu, h]$ *for all* $h \in H_1$ *with* $\Pr[h \mid (\beta_1, \beta_2)] > 0$*; and*
- $\mathrm{E}[u_2 \mid (\beta_1, \beta_2), \mu, h] \geq \mathrm{E}[u_2 \mid (\beta_1, \beta_2'), \mu, h]$ *for all* $h \in H_2$ *with* $\Pr[h \mid (\beta_1, \beta_2)] > 0$*.*

The main difference between this characterization of Nash equilibrium and the definition of sequential equilibrium lies in the fact that Nash equilibria are only required to be sequentially rational at information sets that are reached with positive probability. Therefore, Nash equilibrium is silent about what happens if the game reaches one of the zero-probability information sets (for example, because the opponent did not play according to equilibrium).

### 8.2.2 Fixing interior-point methods to compute sequential equilibria

Ideally, we could state that the central path defined above converges to a sequential equilibrium.[17] However, it is not clear that the limit point of the central path is sequentially rational. In particular, it may be possible for the probability of reaching a particular information set to go to zero faster than the deviation incentive at that information set.

Rather than considering the (problem-independent) central path commonly studied in linear programming, we propose considering alternate central path definitions that are designed to prevent the probability of reaching an information set going to zero faster than the duality gap. One promising possibility is to modify the logarithmic barrier function to normalize the probabilities at lower levels in the tree, and study the path of solutions induced by that barrier function.

We propose to investigate such barrier functions and evaluate their usefulness in computing sequential equilibria in sequential games.

# 9 Multi-player approaches

In this section we propose a novel solution concept applicable in games with many players. We suggest an algorithm for solving such games (with respect to our solution concept), and discuss its applicability to multi-player poker tournaments.

## 9.1 Regret-minimizing pure strategy solution concept

The problem of finding Nash equilibria is PPAD-complete [22], even in two-player non-zero-sum games. Hence, finding Nash equilibria is PPAD-complete in zero-sum games with three or more players. Given this, it is unlikely that there are efficient algorithms for finding (or even approximating) Nash equilibria in most games with three or more players. Indeed, the present general-purpose algorithms for games with many players do not scale very well at all [53, 81, 100]. Hence, we propose an alternative solution concept which may permit practical algorithms for finding solutions, at the expense of the stronger guarantees of an equilibrium.

The *regret-minimizing pure strategy solution concept* can be described as follows. Let $S_i$ be the set of pure strategies for player $i \in \{1, \ldots, n\} = N$ and let $S = S_1 \times \cdots \times S_n$. Let $s = (s_1, \ldots, s_n) \in S$ be a profile of pure strategy selections $s_i \in S_i$ for each player. Let $u_i(s)$ be the utility player $i$ receives when the profile $s$ is played, and let $regret_i(s)$ be the regret player $i$ experiences when playing $s_i$:

$$regret_i(s) = \left[\max_{s'_i \in S_i} u_i(s'_i, s_{-i})\right] - u_i(s_i, s_{-i}).$$

We observe that if $regret_i(s) = 0$ for each player $i$, then $s$ is a Nash equilibrium. However, pure strategy equilibria do not exist in general. One way to weaken the Nash equilibrium concept is to relax the requirement that every player have zero regret for their strategy. The *regret-minimizing pure strategy solution concept* is the solution to the following problem:

$$\operatorname*{argmin}_{s \in S} \max_{i \in N} regret_i(s).$$

This problem can be formulated and solved as a mixed-integer multi-linear program as described in the following section.[18]

One reason to expect that a solution concept confined to pure strategies only would work well in at least some applications is the empirical observation that in the late stages of a no-limit Texas Hold'em tournament (when considering jam-fold strategies), very little randomization is required [85].

[17] We do know that the central path converges to a well-defined behavior strategy. Furthermore, the resulting assessment is consistent.

[18] Instead of requiring the strategy profile which minimizes the maximum regret experienced by any player, we could instead require the profile minimizing *average* regret. This type of concept is also solvable by the algorithm we propose below.

### 9.2 Mixed-integer multi-linear programming (MIMLP)

Mixed-integer linear programming has been developed as an approach for finding Nash equilibria in two-person non-zero-sum games [108]. In that paper, it was pointed out that the technique does not work for games with three or more players because of the non-linearity that is required in the constraints. In particular, it required constraints containing terms with products of probabilities, which are continuous variables. On the other hand, when considering pure strategies only, this yields constraints containing products of binary variables. Although these constraints are multi-linear, there are standard techniques for linearizing these constraints [89]. Furthermore, with this approach, it is just as easy to consider sequential games, rather than the simpler normal form games previously considered [108]. In fact, the linearization will only introduce a number of binary variables equal to the number of leaves in the game tree.

A straightforward algorithm for finding regret-minimizing pure strategy solutions would be to linearize the abovementioned multilinear constraints, and directly solve the resulting mixed-integer linear program using an integer programming solver. However, one can potentially do even better by incorporating a branch-and-price approach [4], in which variables are brought into the problem on an as-needed basis. For many problems, this greatly improves the scalability of the basic branch-and-bound framework.

It is quite reasonable to expect that this technique will be applicable for our problem. At a given point in the search process, if an action that takes place at the beginning of the game is currently considered to be taken with probability zero, then it is not necessary to consider any of the actions in the game coming after that. Due to the inherent exponential increase in tree sizes as the depth increases, this could save a tremendous number of variables from having to be introduced into the problem.

Another way of finding regret-minimizing pure strategy equilibria would be to directly tackle the multi-linear formulation using a general-purpose mixed-integer non-linear programming (MINLP) solver, such as *Bonmin* [16]. However, that algorithm does not guarantee finding optimal solutions when the problem (like ours) is non-convex, and hence would not be the most appropriate for our purposes.

## 10 Related work

Almost since the field's founding, game theory has been used to analyze different aspects of poker [17, 127, 6, 70, 88, 5, 56, 58, 93, 52, 40, 27, 105, 106]. That early work was limited to tiny games that could be solved by hand. In fact, most of that work was focused on computing *analytical solutions* for various stylized versions of poker. For example, one typical assumption in that line of work is for the cards to be drawn uniformly at random from the unit interval, followed by a simple betting protocol. Although this approach seems likely to be of use only for extremely small games, recently there has been renewed interest in extending some of these models and determining analytical solutions from them with the goal of applying them to certain situations in real poker games [35, 36]. Simplified versions of poker have also been developed for illustrative examples in education [101]. From a cognitive modeling and analysis perspective, poker has proved to be a fertile environment for research [37, 19, 20].

Using the sequence form representation in conjunction with linear programming, Koller and Pfeffer (1997) determined solutions to poker games with up to 140,000 nodes. That approach scales to games with about a million nodes [46]. For a medium-sized (3.1 billion nodes) variant of poker called Rhode Island Hold'em, game theory-based solutions have been developed using a lossy abstraction followed by linear programming [117] (of course we have since solved this same problem as discussed previously [46]).

Recently there has been a surge of research into new techniques for developing players for Texas Hold'em poker [66, 114, 123, 12, 11, 10, 46, 45, 85, 2]. One approach has been *opponent modeling*, in which a poker-playing program attempts to identify and exploit weaknesses in the opponents [13, 29, 10, 55, 121, 122, 113]. The most successful Texas Hold'em program from that line of research is *Vexbot* [10], which combines opponent modeling with miximax search (a variant of minimax search which allows the players to move probabilistically according to some model to account for the presence of imperfect information), and is available in the commercial product *Poker Academy Pro*. The first notable game theory-based player for Texas Hold'em used expert-designed manual abstractions and is competitive with advanced human players [11].

A player based on the techniques developed in that paper is available in *Poker Academy Pro* as *Sparbot*. Differences between that player and *GS3* are discussed in Section 4.

## 10.1 Related research on abstraction

The main technique applied in this paper is that of transforming large extensive form games into smaller extensive form games for which an equilibrium can be computed. Then, the equilibrium strategies of the smaller game are mapped back into the original larger game. One of the first pieces of research addressing functions which transform extensive form games into other extensive form games, although not for the purpose of making the game smaller, was in an early paper [126], which was later extended [34]. In these papers, several distinct *transformations*, now known as Thompson-Elmes-Reny transformations, are defined. The main result is that one game can be derived from another game by a sequence of those transformations if and only if the games have the same *pure reduced normal form*. The pure reduced normal form is the extensive form game represented as a game in normal form where duplicates of pure strategies (*i.e.*, ones with identical payoffs) are removed and players essentially select equivalence classes of strategies [69]. An extension to this work shows a similar result, but for slightly different transformations and *mixed reduced normal form* games [60]. Modern treatments of this previous work on game transformations have also been written [98, Ch. 6],[31].

The recently introduced notion of *weak isomorphism* in extensive form games [21] is related to our notion of restricted game isomorphism. The motivation of that work was to justify solution concepts by arguing that they are invariant with respect to isomorphic transformations. Indeed, the author shows, among other things, that many solution concepts, including Nash, perfect, subgame perfect, and sequential equilibrium, are invariant with respect to weak isomorphisms. However, that definition requires that the games to be tested for weak isomorphism are of the same size. Our focus is totally different: we find strategically equivalent *smaller* games. Another difference is that their paper does not provide any algorithms.

Abstraction techniques have been used in AI research before. In contrast to our work, most (but not all) research involving abstraction has been for single-agent problems (*e.g.* [59, 76]). Furthermore, the use of abstraction typically leads to sub-optimal solutions, unlike the techniques presented in this paper, which yield optimal solutions. A notable exception is the use of abstraction to compute optimal strategies for the game of Sprouts [3]. However, a significant difference to our work is that Sprouts is a game of perfect information.

One of the first pieces of research to use abstraction in multi-agent settings was the development of *partition search*, which is the algorithm behind *GIB*, the world's first expert-level computer bridge player [50, 51]. In contrast to other game tree search algorithms which store a particular game position at each node of the search tree, partition search stores *groups* of positions that are similar. (Typically, the similarity of two game positions is computed by ignoring the less important components of each game position and then checking whether the abstracted positions are similar—in some domain-specific expert-defined sense—to each other.) Partition search can lead to substantial speed improvements over $\alpha$-$\beta$-search. However, it is not game theory-based (it does not consider information sets in the game tree), and thus does not solve for the equilibrium of a game of imperfect information, such as poker.[19] Another difference is that the abstraction is defined by an expert human while our abstractions are determined automatically.

There has been some research on the use of abstraction for imperfect information games. Most notably, Billings *et al* [11] describe a manually constructed abstraction for the game of Texas Hold'em poker, and include promising results against expert players. However, this approach has significant drawbacks. First, it is highly specialized for Texas Hold'em. Second, a large amount of expert knowledge and effort was used in constructing the abstraction. Third, the abstraction does not preserve equilibrium: even if applied to a smaller game, it might not yield a game-theoretic equilibrium. Promising ideas for abstraction in the context

[19] Bridge is also a game of imperfect information, and partition search does not find the equilibrium for that game either. Instead, partition search is used in conjunction with statistical sampling to simulate the uncertainty in bridge. There are also other bridge programs that use search techniques for perfect information games in conjunction with statistical sampling and expert-defined abstraction [120]. Such (non-game-theoretic) techniques are unlikely to be competitive in poker because of the greater importance of information hiding and bluffing.

of general extensive form games have been described in an extended abstract [99], but to our knowledge, have not been fully developed.

# 11 Thesis timeline

An estimated schedule of the proposed research is as follows:

| **Task** | **Schedule estimate** |
|---|---|
| Heads-up **no-limit** Texas Hold'em player | Now – July 2007 |
| EGT selective updating | July 2007 – October 2007 |
| Specialized IPM for sequential games | July 2007 – October 2007 |
| IPM for equilibrium refinements | July 2007 – October 2007 |
| MIMLP and multi-player tournament poker player | November 2007 – February 2008 |
| *ex ante* and *ex post* guarantees | November 2007 – February 2008 |
| Thesis writing, journal submissions | March 2008 – August 2008 |

The topics "specialized IPM for sequential games", "IPM for equilibrium refinements" and "*ex ante* and *ex post* guarantees" are risky research avenues. The research in these areas is very preliminary at this point, and there is a chance that this research could be unsuccessful. Therefore, these topics are considered optional pieces of the thesis.

# References


[1] Wilhelm Ackermann. Zum Hilbertschen Aufbau der reellen Zahlen. *Math. Annalen*, 99:118–133, 1928.

[2] Rickard Andersson. Pseudo-optimal strategies in no-limit poker. Master's thesis, Umeå University, May 2006.

[3] David Applegate, Guy Jacobson, and Daniel Sleator. Computer analysis of sprouts. Technical Report CMU-CS-91-144, Carnegie Mellon University, 1991.

[4] Cynthia Barnhart, Ellis Johnson, George Nemhauser, Martin Savelsbergh, and Pamela Vance. Branch-and-price: column generation for solving huge integer programs. *Operations Research*, 46:316–329, 1998.

[5] Richard Bellman. On games involving bluffing. *Rendiconti del Circolo Matematico di Palermo*, 1(2):139–156, 1952.

[6] Richard Bellman and David Blackwell. Some two-person games involving bluffing. *Proceedings of the National Academy of Sciences*, 35:600–605, 1949.

[7] Elwyn R. Berlekamp, John H. Conway, and Richard K. Guy. *Winning Ways for Your Mathematical Plays*. Academic Press, New York, 1983.

[8] Nivan A. R. Bhat and Kevin Leyton-Brown. Computing Nash equilibria of action-graph games. In *Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, Banff, Canada, 2004.

[9] Darse Billings. Web posting at Poker Academy Forums, Meerkat API and AI Discussion, December 2005. `http://www.poker-academy.com/forums/viewtopic.php?t=1872`.

[10] Darse Billings, Michael Bowling, Neil Burch, Aaron Davidson, Rob Holte, Jonathan Schaeffer, Terrance Schauenberg, and Duane Szafron. Game tree search with adaptation in stochastic imperfect information games. In *Proceedings of the 4th International Conference on Computers and Games (CG)*, pages 21–34, Ramat-Gan, Israel, July 2004. Springer-Verlag.

[11] Darse Billings, Neil Burch, Aaron Davidson, Robert Holte, Jonathan Schaeffer, Terence Schauenberg, and Duane Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 661–668, Acapulco, Mexico, 2003.

[12] Darse Billings, Aaron Davidson, Jonathan Schaeffer, and Duane Szafron. The challenge of poker. *Artificial Intelligence*, 134(1-2):201–240, 2002.

[13] Darse Billings, Denis Papp, Jonathan Schaeffer, and Duane Szafron. Opponent modeling in poker. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 493–499, Madison, WI, 1998.

[14] Ben Blum, Christian R. Shelton, and Daphne Koller. A continuation method for Nash equilibria in structured games. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)*, Acapulco, Mexico, 2003.

[15] Béla Bollobás. *Combinatorics*. Cambridge University Press, 1986.

[16] Pierre Bonami, Lorenz T. Biegler, Andrew R. Conn, Gérard Cornuéjols, Ignacio E. Grossmann, Carl D. Laird, Jon Lee, Andrea Lodi, François Margot, Nicolas Sawaya, and Andreas Wachter. An algorithmic framework for convex mixed integer nonlinear programs. Research Report RC23771, IBM, October 2005.

[17] Émile Borel. *Traité du calcul des probabilités et ses applications*, volume IV of *Applications aux jeux des hazard*. Gauthier-Villars, Paris, 1938.

[18] George W. Brown. Iterative solutions of games by fictitious play. In Tjalling C. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 374–376. John Wiley & Sons, 1951.

[19] Kevin Burns. Heads-up face-off: On style and skill in the game of poker. In *Style and Meaning in Language, Art, Music, and Design: Papers from the 2004 Fall Symposium*, pages 15–22, Menlo Park, California, 2004. AAAI Press.

[20] Kevin Burns. Pared-down poker: Cutting to the core of command and control. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 234–241, Colchester, UK, 2005.

[21] André Casajus. Weak isomorphism of extensive games. *Mathematical Social Sciences*, 46:267–290, 2003.

[22] Xi Chen and Xiaotie Deng. Settling the complexity of 2-player Nash equilibrium. *Electronic Colloquium on Computational Complexity*, Report No. 150, 2005.

[23] Fabian A. Chudak and Vania Eleutério. Improved approximation schemes for linear programming relaxations of combinatorial optimization problems. In *IPCO*, pages 81–96, Berlin, Germany, 2005.

[24] Vasek Chvátal. *Linear Programming*. W. H. Freeman and Company, 1983.

[25] Vincent Conitzer and Tuomas Sandholm. Complexity results about Nash equilibria. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 765–771, Acapulco, Mexico, 2003.

[26] Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.

[27] William H. Cutler. An optimal strategy for pot-limit poker. *American Mathematical Monthly*, 82:368–376, 1975.

[28] George Dantzig. A proof of the equivalence of the programming problem and the game problem. In Tjalling Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 330–335. John Wiley & Sons, 1951.

[29] Aaron Davidson. Opponent modeling in poker: Learning and acting in a hostile environment. Master's thesis, University of Alberta, 2002.

[30] Ian Davidson and Ashwin Satyanarayana. Speeding up k-means clustering by bootstrap averaging. In *IEEE Data Mining Workshop on Clustering Large Data Sets*, 2003.

[31] Boudewijn P. de Bruin. Game transformations and game equivalence. Technical note x-1999-01, University of Amsterdam, Institute for Logic, Language, and Computation, 1999.

[32] Xiaotie Deng, Christos Papadimitriou, and Shmuel Safra. On the complexity of equilibria. In *Proceedings of the 34th Annual ACM Symposium on the Theory of Computing*, pages 67–71, 2002.

[33] Nikhil R. Devanar, Christos H. Papadimitriou, Amin Saberi, and Vijay V. Vazirani. Market equilibrium via a primal-dual-type algorithm. In *Proceedings of the 43rd Annual Symposium on Foundations of Computer Science*, pages 389–395, 2002.

[34] Susan Elmes and Philip J. Reny. On the strategic equivalence of extensive form games. *Journal of Economic Theory*, 62:1–23, 1994.

[35] Chris Ferguson and Thomas S. Ferguson. On the Borel and von Neumann poker models. *Game Theory and Applications*, 9:17–32, 2003.

[36] Chris Ferguson, Tom Ferguson, and Céphas Gawargy. Uniform(0,1) two-person poker models, 2004. Available at `http://www.math.ucla.edu/~tom/papers/poker2.pdf`.

[37] Nicholas V. Findler. Studies in machine cognition using the game of poker. *Communications of the ACM*, 20(4):230–245, 1977.

[38] Lester R. Ford, Jr. and Delbert R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[39] Yoav Freund and Robert Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29:79–103, 1999.

[40] Lawrence Friedman. Optimal bluffing strategies in poker. *Management Science*, 17(12):764–771, 1971.

[41] Drew Fudenberg and Jean Tirole. Perfect Bayesian equilibrium and sequential equilibrium. *Journal of Economic Theory*, 53(2):236–260, 1991.

[42] Itzhak Gilboa and Eitan Zemel. Nash and correlated equilibria: Some complexity considerations. *Games and Economic Behavior*, 1:80–93, 1989.

[43] Andrew Gilpin, Samid Hoda, Javier Peña, and Tuomas Sandholm. Gradient-based algorithms for finding Nash equilibria in extensive form games. Mimeo, 2007.

[44] Andrew Gilpin and Tuomas Sandholm. Optimal Rhode Island Hold'em poker. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 1684–1685, Pittsburgh, PA, 2005. Intelligent Systems Demonstration.

[45] Andrew Gilpin and Tuomas 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)*, Boston, MA, 2006.

[46] Andrew Gilpin and Tuomas Sandholm. Finding equilibria in large sequential games of imperfect information. In *Proceedings of the ACM Conference on Electronic Commerce (ACM-EC)*, pages 160–169, Ann Arbor, MI, 2006.

[47] Andrew Gilpin and Tuomas Sandholm. A Texas Hold'em poker player based on automated abstraction and real-time equilibrium computation. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1453–1454, Hakodate, Japan, 2006. Demonstration Track.

[48] Andrew Gilpin and Tuomas Sandholm. Better automated abstraction techniques for imperfect information games, with application to Texas Hold'em poker. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Honolulu, HI, USA, 2007. To appear.

[49] Andrew Gilpin, Tuomas Sandholm, and Troels Sørensen. Potential-aware automated abstraction of sequential games, and holistic equilibrium analysis of Texas Hold'em poker. Mimeo, 2007.

[50] Matthew L. Ginsberg. Partition search. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 228–233, Portland, OR, 1996.

[51] Matthew L. Ginsberg. GIB: Steps toward an expert-level bridge-playing program. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, Stockholm, Sweden, 1999.

[52] A. J. Goldman and J. J. Stone. A symmetric continuous poker model. *Journal of Research of the National Institute of Standards and Technology*, 64(B):35–40, 1960.

[53] Srihari Govindan and Robert Wilson. A global Newton method to compute Nash equilibria. *Journal of Economic Theory*, 110:65–86, 2003.

[54] Samid Hoda, Andrew Gilpin, and Javier Peña. A gradient-based approach for computing Nash equilibria of large sequential games. Manuscript. Presented at INFORMS-06., 2006.

[55] Bret Hoehn, Finnegan Southey, Robert C. Holte, and Valeriy Bulitko. Effective short-term opponent exploitation in simplified poker. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 783–788, July 2005.

[56] R. Isaacs. A card game with bluffing. *American Mathematical Monthly*, 62:99–108, 1955.

[57] Kamal Jain, Mohammad Mahdian, and Amin Saberi. Approximating market equilibria. In *Proceedings of the 6th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, 2003.

[58] S. Karlin and R. Restrepo. Multi-stage poker models. In *Contributions to the Theory of Games*, volume 3 of *Annals of Mathematics Studies, Number 39*, pages 337–363. Princeton University Press, Princeton, New Jersey, 1957.

[59] Craig A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68(2):243–302, 1994.

[60] Elon Kohlberg and Jean-Francois Mertens. On the strategic stability of equilibria. *Econometrica*, 54:1003–1037, 1986.

[61] Daphne Koller and Nimrod Megiddo. The complexity of two-person zero-sum games in extensive form. *Games and Economic Behavior*, 4(4):528–552, October 1992.

[62] Daphne Koller and Nimrod Megiddo. Finding mixed strategies with small supports in extensive form games. *International Journal of Game Theory*, 25:73–92, 1996.

[63] Daphne Koller, Nimrod Megiddo, and Bernhard von Stengel. Efficient computation of equilibria for extensive two-person games. *Games and Economic Behavior*, 14(2):247–259, 1996.

[64] Daphne Koller and Brian Milch. Multi-agent influence diagrams for representing and solving games. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1027–1034, Seattle, WA, 2001.

[65] Daphne Koller and Avi Pfeffer. Representations and solutions for game-theoretic problems. *Artificial Intelligence*, 94(1):167–215, July 1997.

[66] K. Korb, A. Nicholson, and N. Jitnah. Bayesian poker. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 343–350, Stockholm, Sweden, 1999.

[67] Steven G. Krantz and Harold R. Parks. *The Implicit Function Theorem*. Birkhäuser, 2002.

[68] David M. Kreps and Robert Wilson. Sequential equilibria. *Econometrica*, 50(4):863–894, 1982.

[69] H. W. Kuhn. Extensive games. *Proc. of the National Academy of Sciences*, 36:570–576, 1950.

[70] H. W. Kuhn. A simplified two-person poker. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games*, volume 1 of *Annals of Mathematics Studies, 24*, pages 97–103. Princeton University Press, Princeton, New Jersey, 1950.

[71] H. W. Kuhn. Extensive games and the problem of information. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games*, volume 2 of *Annals of Mathematics Studies, 28*, pages 193–216. Princeton University Press, 1953.

[72] Carlton Lemke and J. Howson. Equilibrium points of bimatrix games. *Journal of the Society of Industrial and Applied Mathematics*, 12:413–423, 1964.

[73] Kevin Leyton-Brown and Moshe Tennenholtz. Local-effect games. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)*, Acapulco, Mexico, 2003.

[74] Richard Lipton, Evangelos Markakis, and Aranyak Mehta. Playing large games using simple strategies. In *Proceedings of the ACM Conference on Electronic Commerce (ACM-EC)*, pages 36–41, San Diego, CA, 2003.

[75] Richard J. Lipton and Neal E. Young. Simple strategies for large zero-sum games with applications to complexity theory. In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*, pages 734–740, Montreal, Quebec, Canada, 1994.

[76] Chao-Lin Liu and Michael Wellman. On state-space abstraction for anytime evaluation of Bayesian networks. *SIGART Bulletin*, 7(2):50–57, 1996. Special issue on Anytime Algorithms and Deliberation Scheduling.

[77] Zhaosong Lu, Arkadi Nemirovski, and Renato D. C. Monteiro. Large-scale semidefinite programming via a saddle point mirror-prox algorithm. *Mathematical Programming, Series B*, 2007. Forthcoming.

[78] R. Duncan Luce and Howard Raiffa. *Games and Decisions*. John Wiley and Sons, New York, 1957. Dover republication 1989.

[79] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, Berkeley, California, 1967. University of California Press.

[80] Andreu Mas-Colell, Michael Whinston, and Jerry R. Green. *Microeconomic Theory*. Oxford University Press, 1995.

[81] Richard D. McKelvey and Andrew McLennan. Computation of equilibria in finite games. In H. Amann, D. Kendrick, and J. Rust, editors, *Handbook of Computational Economics*, volume 1, pages 87–142. Elsevier, 1996.

[82] H. Brendan McMahan and Geoffrey J. Gordon. A fast bundle-based anytime algorithm for poker and other convex games. In *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2007.

[83] Peter Bro Miltersen and Troels Bjerre Sørensen. Computing proper equilibria of zero-sum games. In *Computer and Games*, 2006.

[84] Peter Bro Miltersen and Troels Bjerre Sørensen. Computing sequential equilibria for two-player games. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 107–116, 2006.

[85] Peter Bro Miltersen and Troels Bjerre Sørensen. A near-optimal strategy for a heads-up no-limit Texas Hold'em poker tournament. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Honolulu, HI, USA, 2007. To appear.

[86] Roger Myerson. *Game Theory: Analysis of Conflict*. Harvard University Press, Cambridge, 1991.

[87] John Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36:48–49, 1950.

[88] John F. Nash and Lloyd S. Shapley. A simple three-person poker game. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games*, volume 1, pages 105–116. Princeton University Press, 1950.

[89] George Nemhauser and Laurence Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1999.

[90] Yurii Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Kluwer Academic Publishers, 2004.

[91] Yurii Nesterov. Excessive gap technique in nonsmooth convex minimization. *SIAM Journal of Optimization*, 16(1):235–249, 2005.

[92] Yurii Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103:127–152, 2005.

[93] Donald J. Newman. A model for "real" poker. *Operations Research*, 7:557–560, 1959.

[94] Martin J Osborne and Ariel Rubinstein. *A Course in Game Theory*. MIT Press, 1994.

[95] Christos Papadimitriou. Algorithms, games and the Internet. In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*, pages 749–753, 2001.

[96] Christos Papadimitriou and Tim Roughgarden. Computing equilibria in multi-player games. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 82–91, 2005.

[97] Dan Pelleg and Andrew Moore. Accelerating exact k-means algorithms with geometric reasoning. In *Knowledge Discovery and Data Mining*, pages 277–281, 1999.

[98] Andrés Perea. *Rationality in extensive form games*. Kluwer Academic Publishers, 2001.

[99] Avi Pfeffer, Daphne Koller, and Ken Takusagawa. State-space approximations for extensive form games, July 2000. Talk given at the First International Congress of the Game Theory Society, Bilbao, Spain.

[100] Ryan Porter, Eugene Nudelman, and Yoav Shoham. Simple search methods for finding a Nash equilibrium. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 664–669, San Jose, CA, 2004.

[101] David H. Reiley, Michael B. Urbancic, and Mark Walker. Stripped-down poker: A classroom game with signaling and bluffing, February 2005. Working paper. Available at `http://economics.eller.arizona.edu/downloads/working_papers/Econ-WP-05-11.pdf`.

[102] Julia Robinson. An iterative method of solving a game. *Annals of Mathematics*, 54:296–301, 1951.

[103] I. Romanovskii. Reduction of a game with complete memory to a matrix game. *Soviet Mathematics*, 3:678–681, 1962.

[104] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2003.

[105] M. Sakaguchi. A note on the disadvantage for the sente in poker. *Mathematica Japonica*, 29:483–489, 1984.

[106] M. Sakaguchi and S. Sakai. Partial information in a simplified two person poker. *Mathematica Japonica*, 26:695–705, 1981.

[107] Tuomas Sandholm and Andrew Gilpin. Sequences of take-it-or-leave-it offers: Near-optimal auctions without full valuation revelation. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1127–1134, Hakodate, Japan, 2006.

[108] Tuomas Sandholm, Andrew Gilpin, and Vincent Conitzer. Mixed-integer programming methods for finding Nash equilibria. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 495–501, Pittsburgh, PA, 2005.

[109] Rahul Savani and Bernhard von Stengel. Exponentially many steps for finding a Nash equilibrium in a bimatrix game. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, pages 258–267, 2004.

[110] Herbert E. Scarf. The approximation of fixed points of a continuous mapping. *SIAM Journal of Applied Mathematics*, 15:1328–1343, 1967.

[111] Jonathan Schaeffer. *One Jump Ahead: Challenging Human Supremacy in Checkers*. Springer-Verlag, New York, 1997.

[112] Jonathan Schaeffer. The games computers (and people) play. In Marvin V. Zelkowitz, editor, *Advances in Computers*, volume 50, pages 189–266. Academic Press, 2000.

[113] Terence Conrad Schauenberg. Opponent modelling and search in poker. Master's thesis, University of Alberta, 2006.

[114] Alex Selby. Optimal heads-up preflop poker, 1999. `http://www.archduke.demon.co.uk/simplex/`.

[115] Reinhard Selten. Spieltheoretische behandlung eines oligopolmodells mit nachfrageträgheit. *Zeitschrift für die gesamte Staatswissenschaft*, 12:301–324, 1965.

[116] Reinhard Selten. Evolutionary stability in extensive two-person games – correction and further development. *Mathematical Social Sciences*, 16:223–266, 1988.

[117] Jiefu Shi and Michael Littman. Abstraction methods for game theoretic poker. In *Computers and Games*, pages 333–345. Springer-Verlag, 2001.

[118] Satinder P Singh, Vishal Soni, and Michael P Wellman. Computing approximate Bayes-Nash equilibria in tree-games of incomplete information. In *Proceedings of the ACM Conference on Electronic Commerce (ACM-EC)*, pages 81–90, New York, NY, 2004.

[119] David Sklansky. *The Theory of Poker*. Two Plus Two Publishing, fourth edition, 1999.

[120] Stephen J. J. Smith, Dana S. Nau, and Thomas Throop. Computer bridge: A big win for AI planning. *AI Magazine*, 19(2):93–105, 1998.

[121] Finnegan Southey, Michael Bowling, Bryce Larson, Carmelo Piccione, Neil Burch, Darse Billings, and Chris Rayner. Bayes' bluff: Opponent modelling in poker. In *Proceedings of the 21st Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 550–558, July 2005.

[122] Nathan Sturtevant, Martin Zinkevich, and Michael Bowling. Prob-max$^n$: Opponent modeling in n-player games. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 1057–1063, Boston, MA, 2006.

[123] Ken Takusagawa. Nash equilibrium of Texas Hold'em poker, 2000. Undergraduate thesis, Stanford University.

[124] Robert E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, 1975.

[125] Gerald Tesauro. Temporal difference learning and TD-gammon. *Communications of the ACM*, 38(3), 1995.

[126] F. Thompson. Equivalence of games in extensive form. RAND Memo RM-759, The RAND Corporation, January 1952.

[127] John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1947.

[128] B. von Stengel, A. H. van den Elzen, and A. J. J. Talman. Computing normal form perfect equilibria for extensive two-person games. *Econometrica*, 70:693–715, 2002.

[129] Bernhard von Stengel. Efficient computation of behavior strategies. *Games and Economic Behavior*, 14(2):220–246, 1996.

[130] Bernhard von Stengel. Computing equilibria for two-person games. In Robert Aumann and Sergiu Hart, editors, *Handbook of game theory*, volume 3. North Holland, Amsterdam, 2002.

[131] Stephen J. Wright. *Primal-Dual Interior-Point Methods*. SIAM, 1997.