Gradient-based algorithms for finding Nash equilibria in extensive form games

Andrew Gilpin
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA, USA
gilpin@cs.cmu.edu

Samid Hoda
Tepper School of Business
Carnegie Mellon University
Pittsburgh, PA, USA
shoda@andrew.cmu.edu

Javier Peña
Tepper School of Business
Carnegie Mellon University
Pittsburgh, PA, USA
jfp@andrew.cmu.edu

Tuomas Sandholm
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA, USA
sandholm@cs.cmu.edu

March 31, 2007

Abstract

We present a computational approach to the saddle-point formulation for the Nash equilibria of two-person, zero-sum sequential games of imperfect information. The algorithm is a first-order gradient method based on modern smoothing techniques for non-smooth convex optimization. The algorithm requires $O(1/\epsilon)$ iterations to compute an ϵ-equilibrium, and the work per iteration is extremely low. These features enable us to find approximate Nash equilibria for sequential games with a tree representation of about 10^{10} nodes. This is three orders of magnitude larger than what previous algorithms can handle. We present two heuristic improvements to the basic algorithm and demonstrate their efficacy on a range of real-world games. Furthermore, we demonstrate how the algorithm can be customized to a specific class of problems with enormous memory savings.

1 Introduction

Extensive form games model the interaction of multiple, self-interested agents in stochastic environments with hidden information. The goal of each agent is to maximize its own utility. Since the outcome for a particular agent depends on the actions of the other agents, each agent must reason about the other agents' behavior before acting. A fundamental solution concept for these games is the *Nash equilibrium*, *i.e.* a specification of strategies for each agent such that no agent is better off by deviating from their prescribed equilibrium strategy. Generally, Nash equilibrium strategies involve randomized actions (called *mixed strategies*). For two-player zero-sum sequential games of imperfect information, the Nash equilibrium problem can be formulated using the sequence form representation [13, 7, 15] 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, problem (1) can be modeled as a linear program (see [15] 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 [8]. However, for many games the size of the game tree and the corresponding linear program is enormous. For example, the Nash equilibrium problem for Rhode Island Hold'em [14], after a substantial reduction in size via the *GameShrink* lossless abstraction algorithm [5], leads to a linear program with about 10^6 variables and constraints, whose solution using the state-of-the-art CPLEX interior-point linear programming solver takes over one week on a 1.65GHz IBM eServer p5 570, and consumes 25 GB of memory [5]. Prior to the work presented in this paper, this was the largest poker game instance solved to date. Recently there has been substantial interest in two-player limit Texas Hold'em poker, whose game tree has about 10^{18} variables and constraints. The latter problem is well beyond current computational technology.

A recent and fruitful approach to this problem is to solve *lossy abstractions* [1, 4, 5] to approximate the Nash equilibrium. These abstractions yield smaller games that capture some of the main features of the full game. The quality of the approximate Nash equilibrium solution depends on the coarseness of the abstraction. The main current limitation on the degree of coarseness is the magnitude of the abstracted game that standard linear programming solvers can handle. With the current state-of-the art CPLEX solver the dimension is limited to games whose tree representation has about 10^7 nodes (due primarily to memory limitations of the interior-point method; the simplex method is unusable due to slowness [5]).

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

$$\max_{\mathbf{x}\in Q_1} \langle A\mathbf{y}^*, \mathbf{x}\rangle - \min_{\mathbf{y}\in Q_2} \langle A\mathbf{y}, \mathbf{x}^*\rangle \leq \epsilon. \tag{2}$$

Strategies that satisfy this inequality are called *ϵ-equilibria*. This class of game-theoretic solution concepts encapsulates strategies in which either player can gain at most ϵ by deviating to another strategy. For most applications this type of approximation is acceptable if ϵ is small.[1] The algorithms of this paper are anytime algorithms and guarantee that ϵ approaches zero, and quickly find solutions that have a very small ϵ.

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 ϕ 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 [12]. 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) [11].

[1] There has been work on finding ϵ-equilibria *in matrix games* [9], but those algorithms are based on sampling, and thus quite different than the ones discussed here.

Some recent work has applied smoothing techniques to the solution of large-scale semidefinite programming problems [10] and to large-scale linear programming problems [3]. However, our work appears to be the first application of smoothing techniques to Nash equilibrium computation in sequential games.

2 Nesterov's excessive gap technique (EGT)

We next describe Nesterov's excessive gap smoothing technique [12], 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

$$d_i(\alpha \mathbf{z} + (1-\alpha)\mathbf{w}) \leq \alpha d_i(\mathbf{z}) + (1-\alpha) d_i(\mathbf{w}) - \frac{1}{2}\rho\alpha\|\mathbf{z} - \mathbf{w}\|^2 \tag{3}$$

for all $\alpha \in [0, 1]$ and $\mathbf{z}, \mathbf{w} \in Q_i$. The largest ρ_i satisfying (3) 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 ϕ 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 [12] that f_{μ_2} and ϕ_{μ_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:*

$$f_{\mu_2}(\mathbf{x}) \geq \phi_{\mu_1}(\mathbf{y}). \tag{4}$$

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 (4), then [12, Lemma 3.1] yields

$$0 \leq \phi(\mathbf{y}) - f(\mathbf{x}) \leq \mu_1 D_1 + \mu_2 D_2. \tag{5}$$

This suggests the following strategy to find an approximate solution to (1): generate a sequence $(\mu_1^k, \mu_2^k, \mathbf{x}^k, \mathbf{y}^k)$, $k = 0, 1, \ldots$, with μ_1^k and μ_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 (4). This is the strategy underlying the EGT algorithms we present in this paper.

The building blocks of our algorithms are the mapping $\operatorname{sargmax}$ 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 $\operatorname{sargmax}(d, \cdot) : \mathbb{R}^n \to Q$ be defined as

$$\operatorname{sargmax}(d, \mathbf{g}) := \operatorname*{argmax}_{\mathbf{x} \in Q} \{\langle \mathbf{g}, \mathbf{x}\rangle - d(\mathbf{x})\}. \tag{6}$$

By [12, Lemma 5.1], the following procedure `initial` yields an initial point that satisfies the excessive gap condition (4). The notation $\|A\|$ indicates an appropriate operator norm (see [12] 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. $\hat{\mathbf{y}} := \operatorname{sargmax}(d_2, \mathbf{0})$
3. $\mathbf{x}^0 := \operatorname{sargmax}\left(d_1, \frac{1}{\mu_1^0} A\hat{\mathbf{y}}\right)$
4. $\mathbf{y}^0 := \operatorname{sargmax}\left(d_2, \nabla d_2(\hat{\mathbf{x}}) + \frac{1}{\mu_2^0} A^{\mathrm{T}}\mathbf{x}^0\right)$
5. `return` $(\mu_1^0, \mu_2^0, \mathbf{x}^0, \mathbf{y}^0)$

The following procedure `shrink` enables us to reduce μ_1 and μ_2 while maintaining (4).

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

1. $\check{\mathbf{y}} := \operatorname{sargmax}\left(d_2, -\frac{1}{\mu_2}A^{\mathrm{T}}\mathbf{x}\right)$
2. $\hat{\mathbf{y}} := (1-\tau)\mathbf{y} + \tau\check{\mathbf{y}}$
3. $\hat{\mathbf{x}} := \operatorname{sargmax}\left(d_1, \frac{1}{\mu_1}A\hat{\mathbf{y}}\right)$
4. $\tilde{\mathbf{y}} := \operatorname{sargmax}\left(d_2, \nabla d_2\left(\check{\mathbf{y}}\right) + \frac{\tau}{(1-\tau)\mu_2}A^{\mathrm{T}}\hat{\mathbf{x}}\right)$
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 [12, Theorem 4.1], if the input $(\mu_1, \mu_2, \mathbf{x}, \mathbf{y})$ to `shrink` satisfies (4) then so does $(\mu_1, \mu_2^+, \mathbf{x}^+, \mathbf{y}^+)$ as long as τ 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 (4). 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` μ_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` μ_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 ϵ-solution to (1). 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.

2.1 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 $\mathrm{sargmax}(d, \cdot) \colon \mathbb{R}^n \to Q$ 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 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 Δ_n and $\min_{\mathbf{x} \in \Delta_n} d(\mathbf{x}) = 0$. It is also differentiable in the relative interior of Δ_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 $\mathrm{sargmax}(d, \mathbf{g})$ has the closed-form expression

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

Example 2 *Consider the (squared)* Euclidean distance *to the center of Δ_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 Δ_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 $\mathrm{sargmax}(d, \mathbf{g})$ does not have a closed-form expression, it can easily be computed in $O(n \log n)$ steps [3].*

In order to apply Algorithm EGT to problem (1) 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 [6]:

Theorem 1 *Any nice prox-function ψ for the simplex induces a nice prox-function for a set of realization plans Q. The mapping $\mathrm{sargmax}(d, \cdot)$ can be computed by repeatedly applying $\mathrm{sargmax}(\psi, \cdot)$.*

Later in this paper we will present experiments regarding the two nice prox functions (entropy and Euclidean).

3 Heuristics for improving speed of convergence

While Algorithm EGT has theoretical complexity $O(1/\epsilon)$, and (as our experiments on EGT show later in this paper) EGT is already an improvement over the state of the art (in particular, the simplex method and standard interior point methods for solving the game modeled as a linear program), we introduce two heuristics for making EGT drastically faster. The heuristics attempt to speed up the decrease in μ_1 and μ_2, and thus the overall convergence time of the algorithm, while maintaining the excessive gap condition (4).

3.1 Heuristic 1: Aggressive μ reduction

The first heuristic is based on the following observation: although the value of $\tau = 2/(k+3)$ computed in step 2(a) of EGT guarantees the excessive gap condition (4), computational experiments indicate that this is an overly conservative value, particularly during the first few iterations. Instead we can use an adaptive procedure to choose a larger value of τ. Since we now can no longer guarantee the excessive gap condition (4) *a priori*, we are required to do a *posterior* verification which occasionally leads to adjustments in the parameter τ. In order to check (4), we need to compute the values of f_{μ_2} and ϕ_{μ_1}. To that end, consider the following mapping smax, which is simply a variation of $\mathrm{sargmax}$. Assume d is a prox-function with domain $Q \subseteq \mathbb{R}^n$. Let $\mathrm{smax}(d, \cdot) : \mathbb{R}^n \to \mathbb{R}$ be defined as

$$\mathrm{smax}(d, \mathbf{g}) := \max_{\mathbf{x} \in Q} \{\langle \mathbf{g}, \mathbf{x} \rangle - d(\mathbf{x})\}. \tag{7}$$

It is immediate that $\mathrm{smax}(d, \cdot)$ is easily computable provided $\mathrm{sargmax}(d, \cdot)$ is. Notice that $\phi_{\mu_1}(\mathbf{y}) = \mathrm{smax}(d_1, \frac{1}{\mu_1} A\mathbf{y})$ and $f_{\mu_2}(\mathbf{x}) = -\mathrm{smax}(d_2, -\frac{1}{\mu_2} A^{\mathrm{T}}\mathbf{x})$. To incorporate Heuristic 1 in Algorithm EGT we modify the procedure shrink as follows.

```
decrease(A, μ1, μ2, τ, x, y, d1, d2)
  1. (μ2+, x+, y+) := shrink(A, μ1, μ2, τ, x, y, d1, d2)
  2. while smax(d1, (1/μ1) A y+) > −smax(d2, (−1/μ2+) Aᵀ x+)
       // reduced too much, τ is too big
     (a) τ := τ/2
     (b) (μ2+, x+, y+) := shrink(A, μ1, μ2, τ, x, y, d1, d2)
  3. return (μ2+, x+, y+)
```

By [12, Theorem 4.1], when the input $(\mu_1, \mu_2, \mathbf{x}, \mathbf{y})$ to decrease satisfies (4), the procedure decrease will halt.

3.2 Heuristic 2: balancing and reduction of μ_1 and μ_2

Our second heuristic is motivated by the observation that after several calls of the decrease procedure, one of μ_1 and μ_2 may be much smaller than the other. This imbalance is undesirable because the larger one dominates in the bound given by (5). Hence after a certain number of iterations we perform a *balancing* step to bring these values closer together. The balancing consists of repeatedly shrinking the larger one of μ_1 and μ_2.

We also observed that after such balancing, the values of μ_1 and μ_2 can sometimes be further reduced without violating the excessive gap condition (4). We thus include a final reduction step in the balancing heuristic.

This balancing and reduction heuristic is incorporated via the following procedure.[2]

[2] We set the parameters (0.9 and 1.5) based on some initial experimentation (details omitted in this short paper).

`balance`$(\mu_1, \mu_2, \mathbf{x}, \mathbf{y}, A)$

1. `while` $\mu_2 > 1.5\mu_1$ `// shrink` μ_2
   $(\mu_2, \mathbf{x}, \mathbf{y}) :=$ `decrease`$(A, \mu_1, \mu_2, \tau, \mathbf{x}, \mathbf{y}, d_1, d_2)$
2. `while` $\mu_1 > 1.5\mu_2$ `// shrink` μ_1
   $(\mu_1, \mathbf{y}, \mathbf{x}) :=$ `decrease`$(A^{\mathrm{T}}, -\mu_2, -\mu_1, \tau, \mathbf{y}, \mathbf{x}, d_2, d_1)$
3. `while` $\operatorname{smax}(d_1, \frac{1}{0.9\mu_1} A\mathbf{y}) > -\operatorname{smax}(d_2, \frac{-1}{0.9\mu_2} A^{\mathrm{T}}\mathbf{x})$
   `// decrease` μ_1 `and` μ_2 `if possible`
   $\mu_1 := 0.9\mu_1$
   $\mu_2 := 0.9\mu_2$

We are now ready to describe the variant of `EGT` with Heuristics 1 and 2.

`EGT-2`

1. $(\mu_1^0, \mu_2^0, \mathbf{x}^0, \mathbf{y}^0) =$ `initial`(A, Q_1, Q_2)
2. $\tau := 0.5$
3. `For` $k = 0, 1, \ldots$:
   (a) `If` k `is even:` `// Shrink` μ_2
      i. $(\mu_1^{k+1}, \mathbf{x}^{k+1}, \mathbf{y}^{k+1}) :=$ `decrease`$(A, \mu_1^k, \mu_2^k, \tau, \mathbf{x}^k, \mathbf{y}^k, d_1, d_2)$
      ii. $\mu_1^{k+1} := \mu_1^k$
   (b) `If` k `is odd:` `// Shrink` μ_1
      i. $(\mu_1^{k+1}, \mathbf{y}^{k+1}, \mathbf{x}^{k+1}) :=$ `decrease`$(-A^{\mathrm{T}}, \mu_2^k, \mu_1^k, \tau, \mathbf{y}^k, \mathbf{x}^k, d_2, d_1)$
      ii. $\mu_2^{k+1} := \mu_2^k$
   (c) `If` $k \mod 10 = 0$ `// balance & reduce`
      `balance`$(\mu_1^k, \mu_2^k, \mathbf{x}^k, \mathbf{y}^k, A)$

4 Customizing the algorithm for poker games

The bulk of the computational work at each iteration of Algorithms `EGT` and `EGT-2` 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 $\operatorname{smax}(d_i, \cdot)$ and $\operatorname{sargmax}(d_i, \cdot)$. 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).

4.1 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 [5]. 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 [14], 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. (We omit the exact details of these matrices due to limited space.)

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 10×10, and the dimensions of B_1, B_2, and B_3 are 13×13, 205×205, 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 1 provides the sizes of the four test instances; each models some variant of poker, an important challenge problem in AI [2]. The first three instances, `10k`, `160k`, and `RI`, are abstractions of Rhode Island Hold'em [14] computed using the *GameShrink* automated abstraction algorithm [5]. 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 1. 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 1: Problem sizes (when formulated as an LP) for the instances used in our experiments.

Table 2 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.[3] Therefore, the EGT family of algorithms is already an improvement over the state-of-the-art (even without the heuristics).

4.2 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

[3]The memory usage for the CPLEX simplex algorithm reported in Table 2 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.

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 2: 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.

demonstrated in Table 3. 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 3: Effect of parallelization for the `Texas` instance.

The rest of the experiments in this paper do not reflect any parallelization.

5 Experiments regarding prox functions

Figure 1 displays the relative performance of the entropy and Euclidean prox functions, described in Examples 1 and 2, respectively. (Heuristics 1 and 2 were enabled in this experiment.)

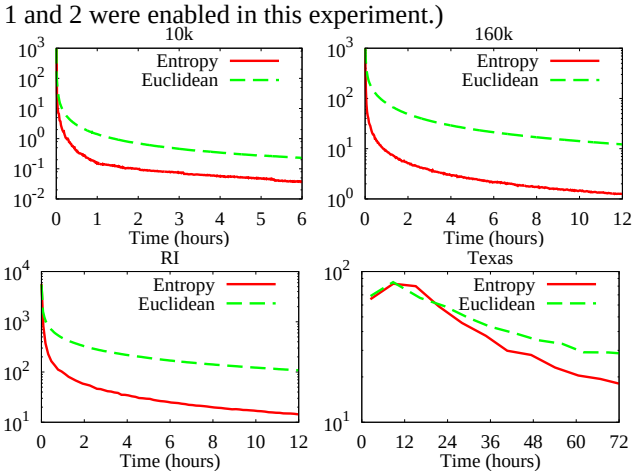

Figure 1: Comparison of the entropy and Euclidean prox functions. The value axis is the gap ϵ (Equation 2).

The entropy prox function outperformed the Euclidean prox function on all four instances. Therefore, in the remaining experiments we use the entropy prox function.

6 Experiments regarding Heuristics 1 and 2

Figure 2 demonstrates the impact of applying Heuristic 1 only. On all four instances, Heuristic 1 reduced the gap significantly; on the larger instances, this reduction was an order of magnitude.

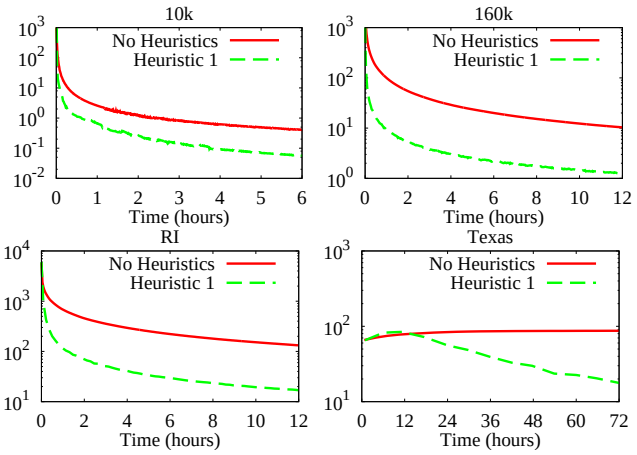

Figure 2: Experimental evaluation of Heuristic 1.

Because Heuristic 2 takes more time to compute, we experimented with how often the algorithm should run it. (We did this by varying the constant in line 3(c) of Algorithm `EGT-2`. In this experiment, Heuristic 1 was turned off.) Figure 3 shows that it is better to run it than to not run it, and on most instances, it is better to run it every 100 iterations than every 10 iterations.

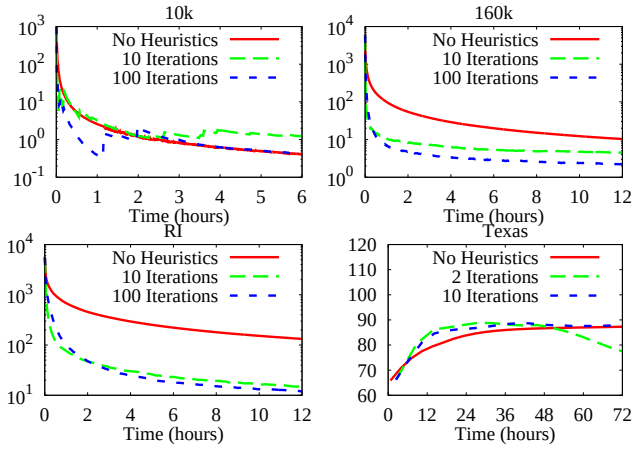

Figure 3: Heuristic 2 applied at different intervals.

7 Conclusions and future research

We applied Nesterov's excessive gap technique to extensive form games. We introduced two heuristics for improving convergence speed, and showed that each of them reduces the gap by an order of magnitude. Best results were achieved by using Heuristic 2 only every so often. It was best to use both heuristics together (Figure 1 vs. Figures 2 and 3). We also observed that the entropy prox function yieled faster convergence than the Euclidean prox function. For poker games and similar games, we introduced a decomposed matrix representation that reduces storage requirements drastically. We also showed near-perfect efficacy of parallelization. Overall, our techniques enable one to solve orders of magnitude larger games than the prior state of the art.

Although current general-purpose simplex and interior-point solvers cannot handle problems of more than around 10^6 nodes [5], it is conceivable that specialized versions of these algorithms could be effective. However, taking advantage of the problem structure in these linear programming methods appears to be quite challenging. For example,

a single interior-point iteration requires the solution of a symmetric non-definite system of equations whose matrix has the payoff matrix A and its transpose A^{T} in some blocks. Such a step is inherently far more complex than the simple matrix-vector multiplications required in `EGT-2`. On the upside, overcoming this obstacle would enable us to capitalize on the superb speed of convergence of interior-point methods. While first-order methods require $O(1/\epsilon)$ iterations to find an ϵ-solution, interior-point methods require only $O(\log(1/\epsilon))$ iterations. We leave the study of these alternative algorithms for Nash equilibrium finding as future work.

References

[1] D. Billings, N. Burch, A. Davidson, R. Holte, J. Schaeffer, T. Schauenberg, and D. Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 661–668, Acapulco, Mexico, 2003.

[2] D. Billings, A. Davidson, J. Schaeffer, and D. Szafron. The challenge of poker. *Artificial Intelligence*, 134(1-2):201–240, 2002.

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

[4] A. Gilpin and T. Sandholm. A competitive Texas Hold'em poker player via automated abstraction and real-time equilibrium computation. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, Boston, MA, 2006.

[5] A. Gilpin and T. 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.

[6] S. Hoda, A. Gilpin, and J. Peña. A gradient-based approach for computing Nash equilibria of large sequential games. Manuscript. Presented at INFORMS-06., 2006.

[7] D. Koller and N. Megiddo. The complexity of two-person zero-sum games in extensive form. *Games and Economic Behavior*, 4(4):528–552, Oct. 1992.

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

[9] R. J. Lipton and N. 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.

[10] Z. Lu, A. Nemirovski, and R. D. C. Monteiro. Large-scale semidefinite programming via a saddle point mirror-prox algorithm. *Mathematical Programming, Series B*, 2007. Forthcoming.

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

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

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

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

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