First-Order Algorithm with $\mathcal{O}(\ln(1/\epsilon))$ Convergence for ϵ-Equilibrium in Two-Person Zero-Sum Games

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

Javier Peña
Tepper School of Business
Carnegie Mellon University
Pittsburgh, PA, USA

Tuomas Sandholm
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA, USA

Abstract

We propose an iterated version of Nesterov's first-order smoothing method for the two-person zero-sum game equilibrium problem

$$\min_{x\in Q_1}\max_{y\in Q_2} x^{\mathrm{T}}Ay = \max_{y\in Q_2}\min_{x\in Q_1} x^{\mathrm{T}}Ay.$$

This formulation applies to matrix games as well as sequential games. Our new algorithmic scheme computes an ϵ-equilibrium to this min-max problem in $\mathcal{O}(\kappa(A)\ln(1/\epsilon))$ first-order iterations, where $\kappa(A)$ is a certain condition measure of the matrix A. This improves upon the previous first-order methods which required $\mathcal{O}(1/\epsilon)$ iterations, and it matches the iteration complexity bound of interior-point methods in terms of the algorithm's dependence on ϵ. Unlike the interior-point methods that are inapplicable to large games due to their memory requirements, our algorithm retains the small memory requirements of prior first-order methods. Our scheme supplements Nesterov's algorithm with an outer loop that lowers the target ϵ between iterations (this target affects the amount of smoothing in the inner loop). We find it surprising that such a simple modification yields an exponential speed improvement. Finally, computational experiments both in matrix games and sequential games show that a significant speed improvement is obtained in practice as well, and the relative speed improvement increases with the desired accuracy (as suggested by the complexity bounds).

Introduction

Game-theoretic solution concepts provide an appealing normative basis for designing agents for multi-agent settings. The concepts are particularly robust in two-person zero-sum games. Equilibrium-finding algorithms for computing approximately optimal strategies have been recently successfully applied to games as large as two-person Texas Hold'em poker (Gilpin, Sandholm, and Sørensen 2007; Zinkevich, Bowling, and Burch 2007; Zinkevich et al. 2007).

The Nash equilibrium problem for a two-person zero-sum game can be formulated as a saddle-point problem (we will describe this in detail later). The latter can in turn be cast as a linear program (LP). However, for many interesting instances of games, such as those that arise in real poker, these LPs are enormous and unsolvable via standard algorithms such as the simplex or interior-point methods.

To address this, some alternative algorithms have been developed and have been shown to be effective in finding ϵ-equilibria, where neither player can benefit more than ϵ by deviating. These include an algorithm based on regret minimization (Zinkevich et al. 2007) and iterative bundle-based algorithms (Zinkevich, Bowling, and Burch 2007; McMahan and Gordon 2007). There are no known convergence rates for those algorithms in terms of ϵ.

Another recent approach (Gilpin et al. 2007) is based on Nesterov's (2005a; 2005b) first-order smoothing techniques. The main strength is simplicity and low computational cost of each iteration. The algorithm finds an ϵ-equilibrium within $\mathcal{O}(1/\epsilon)$ iterations. In contrast, interior-point methods find an ϵ-equilibrium within $\mathcal{O}(\ln(1/\epsilon))$ iterations (Wright 1997), but do not scale to large games due to memory requirements.

In this paper we propose an iterated version of Nesterov's smoothing algorithm for nonsmooth convex optimization (Nesterov 2005b) that runs in $\mathcal{O}(\kappa(A)\ln(1/\epsilon))$ iterations. In terms of ϵ, the iteration complexity is thus the same as that of interior-point methods and exponentially better than that of prior first-order methods. The complexity also depends on a certain condition measure, $\kappa(A)$, of the payoff matrix A. Unlike interior-point methods, we inherit the manageable memory usage of prior first-order methods. So, our algorithm scales to large games and small ϵ.

First-Order Methods

Assume $Q \subseteq \mathbf{R}^n$ is a compact convex set and $f : Q \to \mathbf{R}$ is convex. Consider the convex optimization problem

$$\min\{f(x) : x \in Q\} \tag{1}$$

This paper is concerned with *first-order methods* for solving a particular form of problem (1). The defining feature of these methods is that the search direction at each main iteration is obtained using only first-order information, such as the gradient or subgradient of the function $f(x)$. This feature makes their computational overhead per iteration extremely low, and hence makes them attractive for large-scale problems.

The complexity of first-order methods for finding an approximate solution to (1) depends on the properties of f and

Q. For the setting where f is differentiable and ∇f, the gradient of f, is Lipschitz[1] and continuous, Nesterov (1983) proposed a gradient-based algorithm with convergence rate $\mathcal{O}(1/\sqrt{\epsilon})$. In other words, within $\mathcal{O}(1/\sqrt{\epsilon})$ iterations, the algorithm outputs a value $x \in Q$ such that $f(x) \leq f(x') + \epsilon$ for all $x' \in Q$, including the optimal one. We refer to this algorithm as *Nesterov's optimal method* since it can be shown that for that smooth class of problems, no gradient-based algorithm has faster convergence. A variant by Lan, Lu, and Monteiro (2006) also features $\mathcal{O}(1/\sqrt{\epsilon})$ convergence and outperformed the original in experiments.

For the setting where f is non-differentiable, *subgradient* algorithms are often used. They have complexity $\Theta(1/\epsilon^2)$ (Goffin 1977). However, this pessimistic result is based on treating f as a black box, whose value and subgradient are available through an oracle. For a function f with a suitable structure, Nesterov (2005a; 2005b) devised a first-order algorithm with convergence rate $\mathcal{O}(1/\epsilon)$.[2] The algorithm is based on a *smoothing* technique. The idea is that the structure of f can be used to construct a smooth function with Lipschitz gradient that resembles f. Then, the optimal gradient algorithm applied to the smooth function yields an approximate minimizer for f. This latter technique in particular applies to equilibrium problems arising in two-person zero-sum games, as explained next.

Smoothing Scheme for Matrix Games

In this subsection we describe a smoothing method for the min-max matrix game problem

$$\min_{x\in\Delta_m}\max_{y\in\Delta_n} x^{\mathrm{T}}Ay = \max_{y\in\Delta_n}\min_{x\in\Delta_m} x^{\mathrm{T}}Ay \tag{2}$$

where $\Delta_m := \{x \in \mathbf{R}^m : \sum_{i=1}^m x_i = 1, x \geq 0\}$ is the set of mixed strategies for a player with m pure strategies. The game interpretation is that if player 1 plays $x \in \Delta_m$ and player 2 plays $y \in \Delta_n$, then 1 gets payoff $-x^{\mathrm{T}}Ay$ and 2 gets payoff $x^{\mathrm{T}}Ay$.

Nesterov (2005b) formulated a first-order smoothing technique for solving for each agent's strategy in a matrix game separately. We present that idea here, but applied to a formulation where we solve for both players' strategies at once.

Problem (2) can be rewritten as the primal-dual pair of nonsmooth optimization problems

$$\min\{f(x) : x \in \Delta_m\} = \max\{\phi(y) : y \in \Delta_n\}$$

where

$$\begin{aligned} f(x) &:= \max\left\{x^{\mathrm{T}}Av : v \in \Delta_n\right\}, \\ \phi(y) &:= \min\left\{u^{\mathrm{T}}Ay : u \in \Delta_m\right\}. \end{aligned}$$

[1]Recall that a function f is *Lipschitz with constant* L if $|f(x) - f(y)| \leq L|x - y|$ for all x and y in the domain of f.

[2]First-order methods have also proven to be effective for finding approximate solutions to large-scale LPs (Bienstock 2002) and to large-scale nonlinear convex programs (Smola, Vishwanathan, and Le 2007). These approaches use $\mathcal{O}(1/\epsilon^2)$ iterations on non-smooth problems. (For a special class of continuously differentiable minimization problems (which is very different from our non-differentiable setting) the first-order method presented by Smola *et al.* (2007) runs in $\mathcal{O}(\ln(1/\epsilon))$ iterations.)

For our purposes it will be convenient to cast this as the primal-dual nonsmooth convex minimization problem

$$\min\{F(x, y) : (x, y) \in \Delta_m \times \Delta_n\}, \tag{3}$$

where

$$F(x, y) = \max\left\{x^{\mathrm{T}}Av - u^{\mathrm{T}}Ay : (u, v) \in \Delta_m \times \Delta_n\right\}. \tag{4}$$

Observe that $F(x, y) = f(x) - \phi(y)$ is convex and $\min\{F(x, y) : (x, y) \in \Delta_m \times \Delta_n\} = 0$. Also, a point $(x, y) \in \Delta_m \times \Delta_n$ is an ϵ-solution to (2) if and only if $F(x, y) \leq \epsilon$.

Since the objective function $F(x, y)$ in (3) is nonsmooth, a subgradient method would be appropriate. Thus, without making any attempt to exploit the structure of our problem, we would be faced with a worst-case bound on a subgradient-based algorithm of $\mathcal{O}(1/\epsilon^2)$. However, we can get a much better bound by exploiting the structure of our problem as we now show.

The following objects associated to Equation (3) play a central role in the sequel. Let

$$\mathsf{Opt} := \mathrm{Argmin}\{F(x, y) : (x, y) \in \Delta_m \times \Delta_n\}$$

be the set of all optimal solutions and let $\mathsf{dist} : \Delta_m \times \Delta_n \to \mathbf{R}$ be the distance function to the set Opt, *i.e.*,

$$\mathsf{dist}(x, y) := \min\{\|(x, y) - (u, v)\| : (u, v) \in \mathsf{Opt}\}.$$

Let $(\bar{u}, \bar{v}) \in \Delta_m \times \Delta_n$ and $\mu > 0$. Consider the following smoothed version of F:

$$\begin{aligned} F_\mu(x, y) = \max\{&x^{\mathrm{T}}Av - u^{\mathrm{T}}Ay - \frac{\mu}{2}\|(u, v) - (\bar{u}, \bar{v})\|^2 \\ &: (u, v) \in \Delta_m \times \Delta_n\}. \end{aligned} \tag{5}$$

Let $(u(x, y), v(x, y)) \in \Delta_m \times \Delta_n$ denote the maximizer in (5). This maximizer is unique since the function

$$x^{\mathrm{T}}Av - u^{\mathrm{T}}Ay - \mu\,\|(u, v) - (\bar{u}, \bar{v})\|^2$$

is strictly concave in u and v (Nesterov 2005b). It follows from (Nesterov 2005b, Theorem 1) that F_μ is smooth with gradient

$$\nabla F_\mu(x, y) = \begin{bmatrix} 0 & A \\ -A^{\mathrm{T}} & 0 \end{bmatrix} \begin{bmatrix} u(x, y) \\ v(x, y) \end{bmatrix},$$

and ∇F_μ is Lipschitz with constant $\frac{\|A\|^2}{\mu}$.[3] Let

$$D := \max\left\{\frac{\|(u, v) - (\bar{u}, \bar{v})\|^2}{2} : (u, v) \in \Delta_m \times \Delta_n\right\}.$$

Nesterov's optimal gradient algorithm applied to the problem

$$\min\{F_\mu(x, y) : (x, y) \in \Delta_m \times \Delta_n\} \tag{6}$$

yields the following algorithm. Assume $(x_0, y_0) \in \Delta_m \times \Delta_n$ and $\epsilon > 0$ are given.

[3]$\|A\|$ denotes the *matrix norm* of matrix A which is associated with some vector norm. In this paper, we will use the Euclidean norm (L_2-norm) for which it can be shown that $\|A\| = \sqrt{\lambda(A^{\mathrm{T}}A)}$ where $\lambda(M)$ denotes the largest eigenvalue of matrix M.

$\mathbf{smoothing}(A, x_0, y_0, \epsilon)$

1. Let $\mu = \frac{\epsilon}{2D}$ and $(w_0, z_0) := (x_0, y_0)$
2. For $k = 0, 1, \ldots$
   - $(u_k, v_k) = \frac{2}{k+2}(w_k, z_k) + \frac{k}{k+2}(x_k, y_k)$
   - $(x_{k+1}, y_{k+1}) =$ $\operatorname{argmin}\{\nabla F_\mu(u_k, v_k)^{\mathrm{T}}((x, y) - (u_k, v_k)) +$ $\frac{\|A\|^2}{2\mu}\|(x, y) - (u_k, v_k)\|^2 : (x, y) \in \Delta_m \times \Delta_n\}$
   - if $F(x_{k+1}, y_{k+1}) < \epsilon$ return
   - $(w_{k+1}, z_{k+1}) =$ $\operatorname{argmin}\{\sum_{i=0}^{k} \frac{i+1}{2}\nabla F_\mu(u_i, v_i)^{\mathrm{T}}((w, z) - (u_i, v_i)) +$ $\frac{\|A\|^2}{2\mu}\|(w, z) - (x_0, y_0)\|^2 : (w, z) \in \Delta_m \times \Delta_n\}$

Proposition 1 *Algorithm* **smoothing** *finishes in at most*

$$k = \frac{2\sqrt{2} \cdot \|A\| \cdot \sqrt{D} \cdot \mathsf{dist}(x_0, y_0)}{\epsilon}$$

first-order iterations.

Proof. This readily follows from (Lan, Lu, and Monteiro 2006, Theorem 9) applied to the prox-function

$$d(u, v) = \frac{1}{2}\|(u, v) - (\bar{u}, \bar{v})\|^2,$$

which we used for smoothing in Equation (5). □

Note that the vectors $\bar{u}$, $\bar{v}$ can be *any* vectors in Δ_m and Δ_n. In our implementation, we take these vectors to be those corresponding to a uniformly random strategy.

Iterated Smoothing Scheme for Matrix Games

We are now ready to present our main contribution. The new algorithm is a simple modification of the smoothing algorithm. At each iteration we call the basic smoothing algorithm with a target accuracy. Between the iterations, we reduce the target accuracy by $\gamma > 1$. Consider the following iterated first-order algorithm for minimizing $F(x, y)$.

$\mathbf{iterated}(A, x_0, y_0, \gamma, \epsilon)$

1. Let $\epsilon_0 = F(x_0, y_0)$
2. For $i = 0, 1, \ldots$
   - $\epsilon_{i+1} = \frac{\epsilon_i}{\gamma}$
   - $(x_{i+1}, y_{i+1}) = \mathbf{smoothing}(A, x_i, y_i, \epsilon_{i+1})$
   - If $F(x_{i+1}, y_{i+1}) < \epsilon$ halt

While the modification to the algorithm is simple, it yields an exponential speedup with respect to reaching the target accuracy ϵ:

Theorem 2 *Each call to* **smoothing** *in Algorithm* **iterated** *halts in at most*

$$\frac{2\sqrt{2} \cdot \gamma \cdot \|A\| \cdot \sqrt{D}}{\delta(A)} \tag{7}$$

first-order iterations, where $\delta(A)$ is a finite condition measure of the matrix A.

Algorithm **iterated** *halts in at most*

$$\frac{\ln(2\|A\|/\epsilon)}{\ln(\gamma)}$$

outer iterations, that is, in at most

$$\frac{2\sqrt{2} \cdot \gamma \cdot \|A\| \cdot \ln(2\|A\|/\epsilon) \cdot \sqrt{D}}{\ln(\gamma) \cdot \delta(A)} \tag{8}$$

first-order iterations.

By setting $\gamma = e \approx 2.718...$ above gives the bound

$$\frac{2\sqrt{2} \cdot e \cdot \|A\| \cdot \ln(2\|A\|/\epsilon)}{\delta(A)}.$$

It can be shown that this is the optimal setting of γ for the overall complexity bound in Theorem 2.

For the proof of Theorem 2, we need to introduce the condition measure $\delta(A)$.

The Condition Measure $\delta(A)$

We define the condition measure of a matrix A as

$$\delta(A) = \sup_{\delta}\left\{\delta : \mathsf{dist}(x, y) \le \frac{F(x, y)}{\delta} \forall (x, y) \in \Delta_m \times \Delta_n\right\}.$$

Notice that $\delta(A)$ can be geometrically visualized as a measure of "steepness" of the function $F(x, y)$. We can relate this to $\kappa(A)$ by defining $\kappa(A) := \|A\|/\delta(A)$. The following technical lemma shows that $\delta(A) > 0$ for all A.

Lemma 3 *Assume $A \in \mathbf{R}^{m \times n}$ and F is as in* (4)*. There exists $\delta > 0$ such that*

$$\mathsf{dist}(x, y) \le \frac{F(x, y)}{\delta} \text{ for all } (x, y) \in \Delta_m \times \Delta_n. \tag{9}$$

Proof. Since the function $F : \Delta_m \times \Delta_n \to \mathbf{R}$ is polyhedral, its epigraph $\mathsf{epi}(F) = \{(x, y, t) : t \ge F(x, y),\ (x, y) \in \Delta_m \times \Delta_n\}$ is polyhedral. It thus follows that

$$\mathsf{epi}(F) = \mathsf{conv}\{(x_i, y_i, t_i) : i = 1...M\} + \{0\} \times \{0\} \times [0, \infty)$$

for a finite set of points $(x_i, y_i, t_i) \in \Delta_m \times \Delta_n \times \mathbf{R}_+$, $i = 1, \ldots, M$. Therefore F can be expressed as

$$F(x, y) = \min\left\{\sum_{i=1}^{M} t_i\lambda_i : \sum_{i=1}^{M}(x_i, y_i)\lambda_i = (x, y),\ \lambda \in \Delta_M\right\}. \tag{10}$$

Since $\min\{F(x, y) : (x, y) \in \Delta_m \times \Delta_n\} = 0$, we have $\min\{t_i, i = 1, \ldots, M\} = 0$. Without loss of generality assume $t_1 \ge t_2 \ge \cdots \ge t_N > 0 = t_{N+1} = \cdots = t_M$. We assume $N \ge 1$ as otherwise $\mathsf{Opt} = \Delta_m \times \Delta_n$ and (9) readily holds for any $\delta > 0$. Thus $\mathsf{Opt} = \mathsf{conv}\{(x_i, y_i) : i = N + 1, \ldots, M\}$. Let

$$\delta := \frac{t_N}{\max\{\|(x_i, y_i) - (x, y)\| : i = 1...N, (x, y) \in \mathsf{Opt}\}}$$
$$= \frac{t_N}{\max\{\|(x_i, y_i) - (x_j, y_j)\| : i = 1...N, j = N + 1...M\}}$$

We claim that δ satisfies (9). To prove this claim, let $(x, y) \in \Delta_m \times \Delta_n$ be any arbitrary point. We need to show that $\mathsf{dist}(x, y) \leq F(x, y)/\delta$. Assume $F(x, y) > 0$ as otherwise there is nothing to show. From (10) it follows that

$$(x, y) = \sum_{i=1}^{M} (x_i, y_i)\lambda_i,\ F(x, y) = \sum_{i=1}^{M} t_i\lambda_i = \sum_{i=1}^{N} t_i\lambda_i$$

for some $\lambda \in \Delta_M$. Let $\mu := \sum_{i=1}^{N} \lambda_i > 0$, and let $\tilde{\lambda} \in \Delta_N$ be the vector defined by putting $\tilde{\lambda}_i := \lambda_i/\mu,\ i = 1, \ldots, N$. In addition, let $(\hat{x}, \hat{y}) = \sum_{i=1}^{N} (x_i, y_i)\tilde{\lambda}_i = \sum_{i=1}^{N} (x_i, y_i)\lambda_i/\mu \in \Delta_m \times \Delta_n$, and $(\tilde{x}, \tilde{y}) \in \mathsf{Opt}$ be as follows

$$(\tilde{x}, \tilde{y}) := \begin{cases} \sum_{i=N+1}^{M} (x_i, y_i)\lambda_i/(1-\mu) & \text{if} \quad \mu < 1 \\ (x_M, y_M) & \text{if} \quad \mu = 1 \end{cases}$$

Then $(x, y) = \mu(\hat{x}, \hat{y}) + (1-\mu)(\tilde{x}, \tilde{y})$ and consequently

$$\begin{aligned} \|(x, y) - (\tilde{x}, \tilde{y})\| &= \mu\|(\hat{x}, \hat{y}) - (\tilde{x}, \tilde{y})\| \\ &= \mu \left\| \sum_{i=1}^{N} \tilde{\lambda}_i((x_i, y_i) - (\tilde{x}, \tilde{y})) \right\| \\ &\leq \mu \sum_{i=1}^{N} \tilde{\lambda}_i \|(x_i, y_i) - (\tilde{x}, \tilde{y})\| \\ &\leq \mu \max\{\|(x_i, y_i) - (x, y)\| : \\ &\quad i = 1, \ldots, N,\ (x, y) \in \mathsf{Opt}\} \\ &= \tfrac{\mu t_N}{\delta}. \end{aligned}$$

To finish, observe that

$$F(x, y) = \sum_{i=1}^{N} t_i\lambda_i = \mu \sum_{i=1}^{N} t_i\tilde{\lambda}_i \geq \mu t_N.$$

Therefore,

$$\mathsf{dist}(x, y) \leq \|(x, y) - (\tilde{x}, \tilde{y})\| \leq \mu t_N/\delta \leq F(x, y)/\delta.$$

□

Proof of Theorem 2

By construction, for each $i = 0, 1, \ldots$ we have

$$\mathsf{dist}(x_i, y_i) \leq \frac{\epsilon_i}{\delta(A)} = \frac{\gamma \cdot \epsilon_{i+1}}{\delta(A)}.$$

The iteration bound (7) then follows from Proposition 1.

After N outer iterations Algorithm **iterated** yields $(x_N, y_N) \in \Delta_m \times \Delta_n$ with

$$F(x_N, y_N) < \epsilon_N = \frac{F(x_0, y_0)}{\gamma^N} \leq \frac{2\|A\|}{\gamma^N}.$$

Thus, $F(x_N, y_N) < \epsilon$ for $N = \frac{\ln(2\|A\|/\epsilon)}{\ln(\gamma)}$ and (8) follows from (7).

The Subroutine *smoothing* for Matrix Games

Algorithm **smoothing** involves fairly straightforward operations except for the solution of a subproblem of the form

$$\operatorname{argmin}\left\{\frac{1}{2}\|(u, v)\|^2 - (g, h)^{\mathrm{T}}(u, v) : (u, v) \in \Delta_m \times \Delta_n\right\}.$$

This problem in turn separates into two subproblems of the form

$$\operatorname{argmin}\left\{\frac{1}{2}\|u\|^2 - g^{\mathrm{T}}u : u \in \Delta_m\right\}. \tag{11}$$

Problem (11) can easily be solved via its Karush-Kuhn-Tucker optimality conditions:

$$u - g = \lambda\mathbf{1} + \mu,\ \lambda \in \mathbf{R},\ \mu \in \mathbf{R}^m_+,\ u \in \Delta_m,\ u^{\mathrm{T}}\mu = 0.$$

From these conditions it follows that the solution to (11) is given by

$$u_i = \max\{0, g_i - \lambda\},\ i = 1, \ldots, m,$$

where $\lambda \in \mathbf{R}$ is such that $\sum_{i=1}^{m} \max\{0, (g_i - \lambda)\} = 1$. This value of λ can be computed in $\mathcal{O}(m \ln(m))$ steps via a binary search in the sorted components of the vector g.

Smoothing Scheme for Sequential Games

Algorithm **iterated** and its complexity bound can be extended to sequential games. The Nash equilibrium problem of a two-player zero-sum sequential game with imperfect information can be formulated using the sequence form representation as the following saddle-point problem (Koller and Megiddo 1992; Romanovskii 1962; von Stengel 1996):

$$\min_{x \in Q_1} \max_{y \in Q_2} x^{\mathrm{T}}Ay = \max_{y \in Q_2} \min_{x \in Q_1} x^{\mathrm{T}}Ay. \tag{12}$$

In this formulation, the vectors x and y represent the strategies of players 1 and 2 respectively. The strategy spaces $Q_i \subseteq \mathbf{R}^{|S_i|},\ i = 1, 2$ are the sets of realization plans of players 1 and 2 respectively, where S_i is the set of sequences of moves of player i.

The approach we presented for equilibrium finding in matrix games extends to sequential games in the natural way: recast (12) as a nonsmooth convex minimization problem

$$\min\{F(x, y) : (x, y) \in Q_1 \times Q_2\},\ \text{for} \tag{13}$$

$$F(x, y) = \max\{x^{\mathrm{T}}Av - u^{\mathrm{T}}Ay : (u, v) \in Q_1 \times Q_2\}. \tag{14}$$

Algorithms **smoothing** and **iterated** extend to this context by replacing Δ_m and Δ_n with Q_1 and Q_2, respectively. Proposition 1 and Theorem 2 also extend in the same fashion. However, the critical subproblem in the subroutine **smoothing** becomes more challenging, as described next.

The Subroutine *smoothing* for Sequential Games

Here we describe how to solve each of the two argmin subproblems of **smoothing** in the sequential game case. Each of those two subproblems decomposes into two subproblems of the form

$$\operatorname{argmin}\left\{\frac{1}{2}\|u\|^2 - g^{\mathrm{T}}u : u \in Q\right\}, \tag{15}$$

where Q is a set of realization plans.

Our algorithm for this is a generalization of the solution approach described above for the case $Q = \Delta_k$. In order to describe it, we use some features of the sets of realization plans in the sequence form representation of sequential games. A detailed discussion of the sequence form can be found in (von Stengel 1996). Recall that an extensive form sequential game is given by a tree, payoffs at the leaves, chance moves, and information sets (Osborne and Rubinstein 1994). Each node in the tree determines a unique *sequence* of choices from the root to that node for each one of the players. Under the assumption of perfect recall, all nodes in an information set u of a player define the same sequence σ_u of choices.

Assume U is the set of information sets of a particular player. For each $u \in U$ let C_u denote the set of choices for that player. Then the set of sequences S of the player can be written as

$$S = \{\emptyset\} \cup \{\sigma_u c : u \in U,\, c \in C_u\}$$

where the notation $\sigma_u c$ denotes the sequence of moves σ_u followed by the move c. A *realization plan* for this player is a non-negative vector $x : S \to \mathbf{R}$ that satisfies $x(\emptyset) = 1$, and

$$-x(\sigma_u) + \sum_{c \in C_u} x(\sigma_u c) = 0$$

for all $u \in U$.

It is immediate that the set of realization plans of the player as above can be written in the form

$$\{x \geq 0 : Ex = e\}$$

for some $(1+|U|)\times|S|$ matrix E with entries $\{0, 1, -1\}$ and the $(1 + |U|)$-dimensional vector $e = (1, 0, \ldots, 0)^{\mathrm{T}}$. It also follows that sets of realization plans are *complexes.* A complex is a generalization of a simplex, and can be recursively defined as follows:

(C1) The empty set $\emptyset$ is a complex.

(C2) Assume $Q_j \subseteq \mathbf{R}^{d_j}$ for $j = 1, \ldots, k$ are complexes. Then the following set is a complex

$$\{(u^0, u^1, \ldots, u^k) \in \mathbf{R}^{k+d_1+\cdots+d_k} : u^0 \in \Delta_k, \\ u^j \in u^0_j \cdot Q_j,\ j = 1, \ldots, k\}.$$

(The operation $u^0_j \cdot Q_j$ multiplies all elements of Q_j by u^0_j.) Note that any simplex is a complex: Δ_k is obtained by applying **(C2)** with $Q_j = \emptyset,\ j = 1, \ldots, k$.

Given a complex $Q \subseteq \mathbf{R}^d$ and a vector $g \in \mathbf{R}^d$, define the *value function* $v_{Q,g} : \mathbf{R}_+ \to \mathbf{R}$ as

$$v_{Q,g}(t) := \min\left\{\frac{1}{2}\|u\|^2 - g^{\mathrm{T}}u : u \in t \cdot Q\right\}.$$

It is easy to see that $v_{Q,g}$ is differentiable in $\mathbf{R}_+$. Let $\lambda_{Q,g} = v'_{Q,g}$ and let $\theta_{Q,g}$ be the inverse function of $\lambda_{Q,g}$. It is easy to see that $\lambda_{Q,g}$ is strictly increasing in $\mathbf{R}_+$. In particular, its minimum value is $\lambda_{Q,g}(0)$. The function $\theta_{Q,g}$ can be defined in all of $\mathbf{R}$ by putting $\theta_{Q,g}(\lambda) := 0$ for all $\lambda \leq \lambda_{Q,g}(0)$. Finally, define the minimizer function $u_{Q,g} : \mathbf{R}_+ \to Q$ as

$$u_{Q,g}(t) := \operatorname{argmin}\left\{\frac{1}{2}\|u\|^2 - g^{\mathrm{T}}u : u \in t \cdot Q\right\}.$$

The recursive algorithm **ComplexSubproblem** below computes the functions $v_{Q,g}, \lambda_{Q,g}, \theta_{Q,g}$, and $u_{Q,g}$ for any given complex Q. In particular, it computes the solution $u_{Q,g}(1)$ to the subproblem (15). The algorithm assumes that Q is as in **(C2)** and $g = \left(g^0, g^1, \ldots, g^k\right) \in \mathbf{R}^{k+d_1+\cdots+d_k}$.

ComplexSubproblem(Q, g)

1. For $i = 1, \ldots, k$ let $\tilde{\lambda}_i : \mathbf{R}_+ \to \mathbf{R}$ and $\tilde{\theta}_i : \mathbf{R} \to \mathbf{R}_+$ be
$$\tilde{\lambda}_i(t) := \lambda_{Q_i,g^i}(t) + t - g^0_i, \quad \tilde{\theta}_i := \tilde{\lambda}_i^{-1}.$$
2. Let $\theta_{Q,g} := \sum_{i=1}^k \tilde{\theta}_i$ and $\lambda_{Q,g} := \theta^{-1}_{Q,g}$
3. Let $u_{Q,g} : \mathbf{R}_+ \to Q$ be
$$u_{Q,g}(t)^0_i := \max\left\{0, \left(g^0_i - \lambda_{Q,g}(t)\right)\right\}$$
and
$$u_{Q,g}(t)^i := u_{Q^i,g^i}\left(u_{Q,g}(t)^0_i\right)$$
for $i = 1, \ldots, k$.

While we presented Algorithm **ComplexSubproblem** in recursive form for pedagogical reasons, for efficiency purposes we implemented it as a dynamic program. The implementation first performs a bottom-up pass that computes and stores the functions $\lambda_{Q,g}$. Subsequently a top-down pass computes the components of the minimizer $u_{Q,g}(t)$.

Theorem 4 *Algorithm* **ComplexSubproblem** *is correct. In addition, the function $\lambda_{Q,g}$ is piecewise linear. Furthermore, if Q is as in* **(C2)***, then the total number of breakpoints $B(Q, g)$ of $\lambda_{Q,g}$ is at most*

$$\sum_{i=1}^k \max\{B(Q_i, g_i), 1\}.$$

If the breakpoints of λ_{Q^i,g^i} are available, then the breakpoints of $\lambda_{Q,g}$ can be constructed in $\mathcal{O}(B(Q, g)\ln(B(Q, g)))$ steps, i.e.*, this is the run time of Algorithm* **ComplexSubproblem**.

Proof. The value function $v_{Q,g}(t)$ can be written as

$$v_{Q,g}(t) = \min\left\{\frac{1}{2}\left\|u^0\right\|^2 - \left(g^0\right)^{\mathrm{T}}u^0 + \sum_{j=1}^k v_{Q_j,g^j}\left(u^0_j\right) : \\ u^0 \in t \cdot \Delta_k\right\}. \quad (16)$$

This is a constrained optimization problem in the variables u^0. Its Karush-Kuhn-Tucker optimality conditions are

$$u^0_j - g^0_j + \lambda_{Q_j,g^j}\left(u^0_j\right) = \lambda + \mu_j, \\ \lambda \in \mathbf{R},\ \mu \in \mathbf{R}^k_+, \quad (17) \\ u^0 \in t \cdot \Delta_k,\ \ \mu^{\mathrm{T}}u^0 = 0.$$

By basic differentiability properties from convex analysis (see, *e.g.* (Hirriart-Urruty and Lemaréchal 2001, Chapter D)), it follows that $\lambda_{Q,g}(t) = v'_{Q,g}(t)$ is precisely the value of λ that solves the optimality conditions (17). From these optimality conditions, we get $u_j^0 = \tilde{\theta}_j(\lambda),\ j = 1, \ldots, k$ for the functions $\tilde{\theta}_j$ constructed in step 1 of Algorithm **ComplexSubproblem**. Hence

$$t = \sum_{j=1}^{k} u_j^0 = \sum_{j=1}^{k} \tilde{\theta}_j(\lambda).$$

Therefore, $\theta_{Q,g} = \sum_{j=1}^{k} \tilde{\theta}_j$. This shows the correctness of Steps 1 and 2 of Algorithm **ComplexSubproblem**. Finally, the correctness of Step 3 follows from (16) and (17).

The piecewise linearity of $\lambda_{Q,g}$ readily follows from the correctness of Algorithm **ComplexSubproblem**. As for the number of breakpoints, observe that the number of breakpoints of $\tilde{\theta}_i$ is the same as that of $\tilde{\lambda}_i$, which is either the same as that of λ_{Q^i,g^i} (if $Q_i \neq \emptyset$) or 1 (if $Q_i = \emptyset$). To get the bound on $B(Q, g)$, note that the total number of breakpoints of $\lambda_{Q,g}$ is the same as that of $\theta_{Q,g}$, which is at most the sum of the number of breakpoints of all $\tilde{\theta}_i,\ i = 1, \ldots, k$. Finally, the breakpoints of $\theta_{Q,g}$ can be obtained by sorting the breakpoints of all of the θ_i together. This can be done in $\mathcal{O}(B(Q, g) \ln(B(Q, g)))$ steps. □

ComplexSubproblem Example

We include a simple example to illustrate Algorithm **ComplexSubproblem**, as well as the use of our recursive definition of complexes. For simplicity of the example, let $Q_1 = \Delta_2$ and $Q_2 = \emptyset$. Then applying the recursive definition of complexes, **(C2)**, we get that Q is the set

$$\left\{ \left(u^0, u^1\right) : u^0 \in \Delta_2, u^1 \in u_1^0 \cdot Q_1 \right\}.$$

In a sequential game corresponding to this set of realization plans, the player first chooses among actions a_1^0 and a_2^0, with probabilities u_1^0 and u_2^0, respectively, and conditioned on choosing action a_1^0, the player may be asked to choose among actions a_1^1 and a_2^1, which are played with probabilities u_1^1/u_1^0 and u_2^1/u_1^0, respectively. (Note that there is no u^2 in the above equation since $Q_2 = \emptyset$, *i.e.*, if the agent plays a_2^0, he will have no further actions.)

Now, given input vector $g^1 \in \mathbf{R}^2$, we define the value function for Q_1 as

$$v_{Q_1,g^1}\left(u_1^0\right) := \min \left\{ \frac{1}{2} \left\|u^1\right\|^2 - \left(g^1\right)^{\mathrm{T}} u^1 : u^1 \in u_1^0 \cdot Q_1 \right\}.$$

Then, as was done in the proof of Theorem 4, we can write the value function for Q as

$$v_{Q,g}(t) := \min \left\{ \frac{1}{2} \left\|u^0\right\|^2 - \left(g^0\right)^{\mathrm{T}} u^0 + v_{Q_1,g^1}\left(u_1^0\right) : u^0 \in t \cdot \Delta_k \right\}$$

for $g = \left(g^0, g^1\right) \in \mathbf{R}^4$. This is the problem that **ComplexSubproblem** is trying to solve in our example.

We first demonstrate the algorithm as it executes **ComplexSubproblem**(Q_1, g^1), *i.e.*, the bottom of the recursion. Since Q_1 has no "sub-complexes", we have

$$\begin{aligned} \tilde{\lambda}_1(t) &:= t - g_1^1, \\ \tilde{\lambda}_2(t) &:= t - g_2^1. \end{aligned}$$

The equations are graphed on the left in Figure 1. Step 1 of the algorithm constructs the $\tilde{\theta}_i$ functions to be the inverse of the $\tilde{\lambda}_i(t)$ functions. Once these inverses are computed, Step 2 of the algorithm adds the $\tilde{\theta}_i$ functions to obtain the θ_{Q_1,g^1} function, which is in turn inverted to construct the λ_{Q_1,g^1} function. This process of inverting, adding, and inverting again has a more intuitive description in the form of a "horizontal addition" operation on the $\tilde{\lambda}_i$ functions. In such an operation, two functions are added as normal, except we flip the axis of the graph so that the x-axis and y-axis are switched. This operation is illustrated in Figure 1. The graph on the left in Figure 1 contains the $\tilde{\lambda}_i(t)$ functions. These functions are "horizontally added" to obtain λ_{Q_1,g^1} on the right in Figure 1.

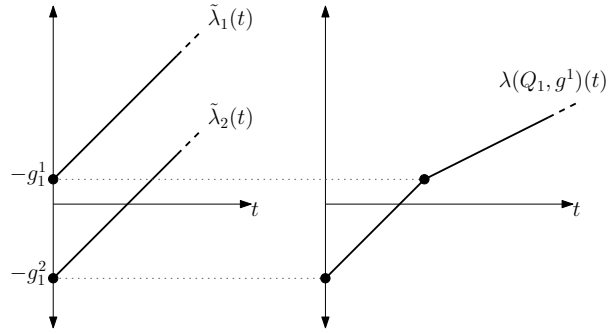

Figure 1: An illustration of Steps 1 and 2 of Algorithm **ComplexSubproblem** applied to Q_1 and g^1.

At non-bottom parts of the recursion ($\lambda_{Q,g}$ in our example) we construct the piecewise linear functions similarly, except that we have to take into account subsequent actions using the piecewise linear functions (function $\lambda_{Q_1,g^1}(t)$ in our example) already computed for the nodes below the current node in the recursion tree:

$$\begin{aligned} \tilde{\lambda}_1(t) &:= t - g_1^0 + \lambda_{Q_1,g^1}(t), \\ \tilde{\lambda}_2(t) &:= t - g_2^0 \end{aligned}$$

The "horizontal addition" operation for this case is depicted in Figure 2.

Since $\lambda_{Q_1,g^1}(t)$ and $\lambda_{Q,g}$ are piecewise linear, our implementation simply represents them as a set of breakpoints, which are represented by solid circles in Figures 1 and 2. Given that we have finally constructed the piecewise linear function at the root, we can determine the values of u^0 and u^1 that solve the optimization problem in (15) as described in Step 3 of Algorithm **ComplexSubproblem**. Specifically, we first take $t = 1$ and solve for u^0. To do this, we evaluate

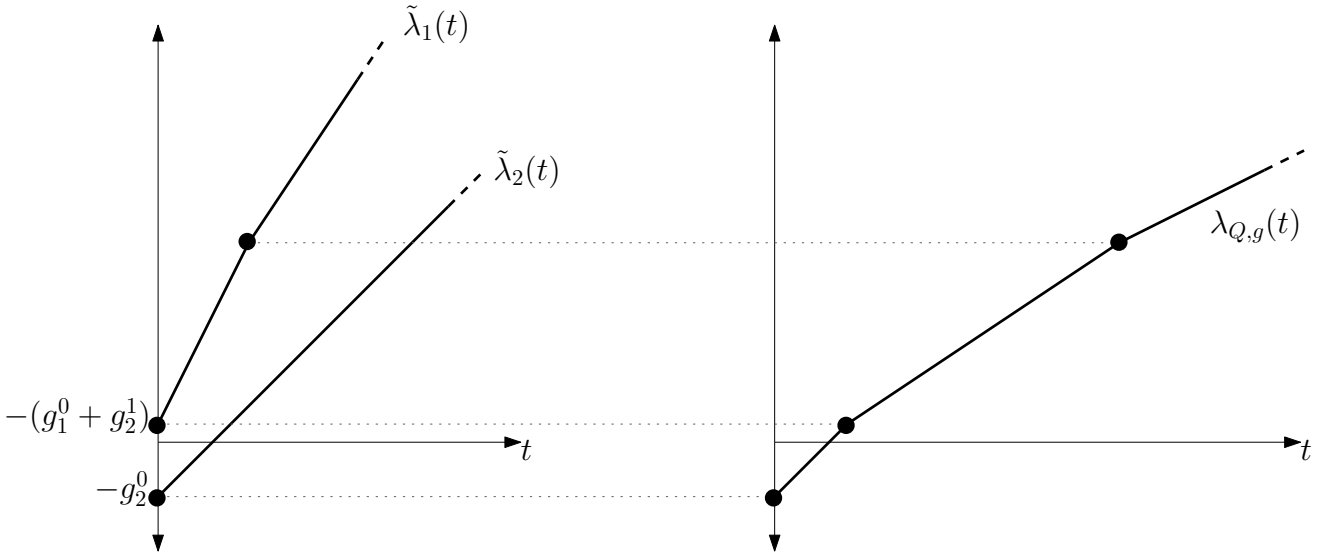

Figure 2: An illustration of Steps 1 and 2 of Algorithm **ComplexSubproblem** applied to Q and g.

$\lambda_{Q,g}(1)$. Then we find the values of u_1^0 and u_2^0 such that

$$\begin{aligned} \tilde{\lambda}_1(u_1^0) &= \lambda_{Q,g}(1), \\ \tilde{\lambda}_2(u_2^0) &= \lambda_{Q,g}(1). \end{aligned}$$

This last operation is straightforward since the functions in question are monotonically increasing and piecewise linear.

Once we have computed u_1^0, we can evaluate $\lambda(Q_1, g^1)(u_1^0)$ and find u_1^1 and u_2^1 that satisfy

$$\begin{aligned} \tilde{\lambda}_1(u_1^1) &= \lambda_{Q_1,g_1}(u_1^0), \\ \tilde{\lambda}_2(u_2^1) &= \lambda_{Q_1,g_1}(u_1^0). \end{aligned}$$

Again, this operation is easy due to the functions being monotonically increasing and piecewise linear. This completes the execution of Algorithm **ComplexSubproblem** on our example.

Computational Experiments

In this section we report on our computational experience with our new method. We compared our **iterated** algorithm against the basic **smoothing** algorithm. We tested the algorithms on matrix games as well as sequential games.

For matrix games, we generated 100 games of three different sizes where the payoffs are drawn uniformly at random from the interval $[-1, 1]$. This is the same instance generator as in Nesterov's (2005b) experiments.

For sequential games, we used the benchmark instances `81`, `10k`, and `160k` which have been used in the past for benchmarking equilibrium-finding algorithms for sequential imperfect-information games (Gilpin et al. 2007). These instances are all abstracted versions of Rhode Island Hold'em poker (Shi and Littman 2002), and they are named to indicate the number of variables in each player's strategy vector.

Figure 3 displays the results. Each graph is plotted with ϵ on the x-axis (using an inverse logarithmic scale). The y-axis is the number of seconds (using a logarithmic scale) needed to find ϵ-equilibrium for the given ϵ. The matrix game graphs also display the standard deviation.

In all settings we see that our **iterated** algorithm indeed outperforms the **smoothing** algorithm (as the worst-case complexity results would suggest). In fact, as the desired accuracy increases, the relative speed difference also increases.

We also tested a version of our algorithm using the Lan *et al.* (2006) variant of Nesterov's optimal method (details omitted). Although the guarantee of Theorem 2 does not hold, that version performed nearly the same.

Conclusions

We presented a new algorithm for finding ϵ-equilibria in two-person zero-sum games. It applies to both matrix and sequential games. The algorithm has convergence rate $\mathcal{O}(\kappa(A)\ln(1/\epsilon))$, where $\kappa(A)$ is a condition measure of the matrix A. In terms of the dependence on ϵ, this matches the complexity of interior-point methods and is exponentially faster than prior first-order methods. Furthermore, our algorithm, like other first-order methods, uses dramatically less memory than interior-point methods, indicating that it can scale to games much larger than previously possible.

Our scheme supplements Nesterov's first-order smoothing algorithm with an outer loop that lowers the target ϵ between iterations (this target affects the amount of smoothing in the inner loop). We find it surprising that such a simple modification yields an exponential speed improvement, and wonder whether a similar phenomenon might occur in other optimization settings as well. Finally, computational experiments both in matrix games and sequential games show that a significant speed improvement is obtained in practice as well, and the relative speed improvement increases with the desired accuracy (as suggested by the complexity bounds).

Acknowledgments

This material is based upon work supported by the National Science Foundation under ITR grant IIS-0427858.

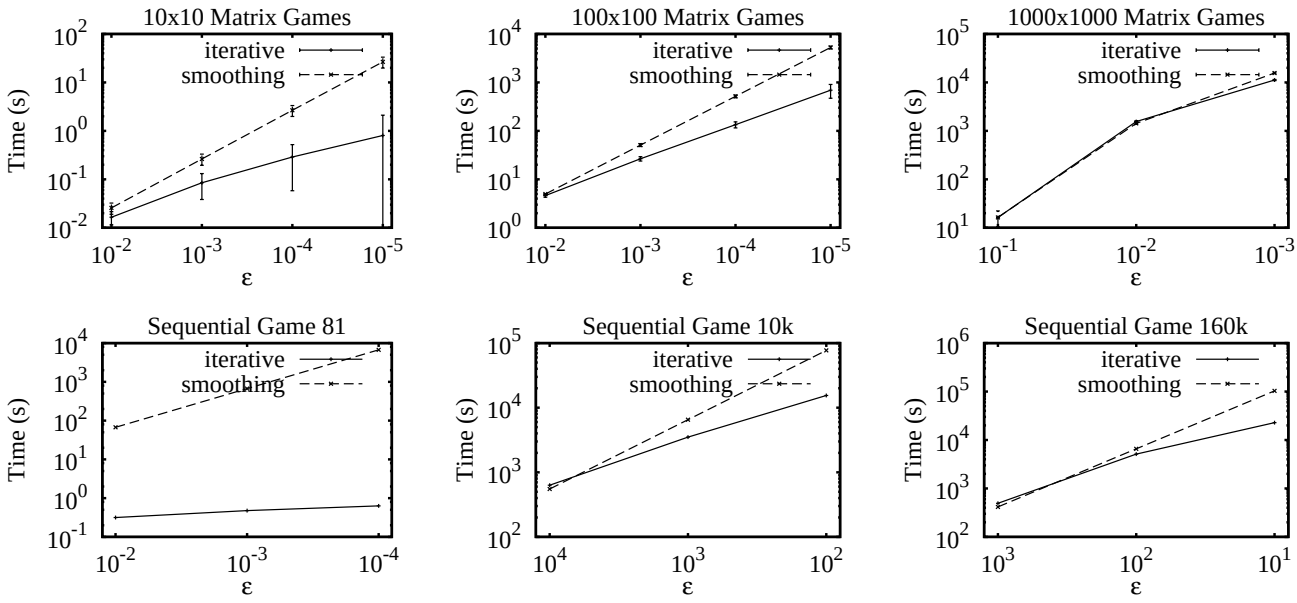

Figure 3: Time taken (in seconds) for each algorithm to find an ϵ-equilibrium for various values of ϵ.

References

Bienstock, D. 2002. *Potential Function Methods for Approximately Solving Linear Programming Problems*. Dordrecht: Kluwer International Series.

Gilpin, A.; Hoda, S.; Peña, J.; and Sandholm, T. 2007. Gradient-based algorithms for finding Nash equilibria in extensive form games. In *3rd International Workshop on Internet and Network Economics (WINE)*.

Gilpin, A.; Sandholm, T.; and Sørensen, T. B. 2007. Potential-aware automated abstraction of sequential games, and holistic equilibrium analysis of Texas Hold'em poker. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 50–57. Vancouver, Canada: AAAI Press.

Goffin, J.-L. 1977. On the convergence rate of subgradient optimization methods. *Mathematical Programming* 13:329–347.

Hirriart-Urruty, J., and Lemaréchal, C. 2001. *Fundamentals of Convex Analysis*. Berlin: Springer-Verlag.

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

Lan, G.; Lu, Z.; and Monteiro, R. D. C. 2006. Primal-dual first-order methods with $O(1/\epsilon)$ iteration-complexity for cone programming. Manuscript.

McMahan, H. B., and Gordon, G. J. 2007. 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)*.

Nesterov, Y. 1983. A method for unconstrained convex minimization problem with rate of convergence $O(1/k^2)$. *Doklady AN SSSR* 269:543–547. Translated to English as *Soviet Math. Docl.*

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

Nesterov, Y. 2005b. Smooth minimization of non-smooth functions. *Mathematical Programming* 103:127–152.

Osborne, M., and Rubinstein, A. 1994. *A Course in Game Theory*. Cambridge, MA: MIT Press.

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

Shi, J., and Littman, M. 2002. Abstraction methods for game theoretic poker. In *CG '00: Revised Papers from the Second International Conference on Computers and Games*, 333–345. London, UK: Springer-Verlag.

Smola, A. J.; Vishwanathan, S. V. N.; and Le, Q. 2007. Bundle methods for machine learning. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*.

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

Wright, S. J. 1997. *Primal-Dual Interior-Point Methods*. Philadelphia, PA: SIAM.

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

Zinkevich, M.; Bowling, M.; and Burch, N. 2007. A new algorithm for generating equilibria in massive zero-sum games. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.