2008

Smoothing Techniques for Computing Nash Equilibria of Sequential Games

Samid Hod
Carnegie Mellon University

Andrew Gilpin
Carnegie Mellon University, agilpin@andrew.cmu.edu

Javier F. Peña
Carnegie Mellon University, jfp@andrew.cmu.edu

Tuomas W. Sandholm
Carnegie Mellon University, sandholm@cs.cmu.edu

Smoothing Techniques for Computing Nash Equilibria of Sequential Games

Samid Hoda
Tepper School of Business, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213,
shoda@andrew.cmu.edu, http://www.andrew.cmu.edu/user/shoda/

Andrew Gilpin
Computer Science Department, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213,
gilpin@cs.cmu.edu, http://www.cs.cmu.edu/~gilpin/

Javier Peña
Tepper School of Business, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213,
jfp@andrew.cmu.edu, http://www.andrew.cmu.edu/user/jfp/

Tuomas Sandholm
Computer Science Department, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213,
sandholm@cs.cmu.edu, http://www.cs.cmu.edu/~sandholm/

We develop first-order smoothing techniques for saddle-point problems that arise in finding a Nash equilibrium of sequential games. The crux of our work is a construction of suitable prox-functions for a certain class of polytopes that encode the sequential nature of the game. We also introduce heuristics that significantly speed up the algorithm, and decomposed game representations that reduce the memory requirements, enabling the application of the techniques to drastically larger games. An implementation based on our smoothing techniques computes approximate Nash equilibria for games that are more than four orders of magnitude larger than what prior approaches can handle. Finally, we show near-linear further speedups from parallelization.

1. Introduction. The Nash equilibria of two-person, zero-sum sequential games are the solutions to

$$\min_{x \in \mathscr{X}} \max_{y \in \mathscr{Y}} \langle \mathbf{y}, A\mathbf{x} \rangle = \max_{y \in \mathscr{Y}} \min_{x \in \mathscr{X}} \langle \mathbf{y}, A\mathbf{x} \rangle, \tag{1}$$

where $\mathscr{X}$ and $\mathscr{Y}$ are polytopes defining the players' strategies and A is the payoff matrix (Koller et al. [11]; Romanovskii [19]; von Stengel [21, 22]). When the minimizer plays a strategy $\mathbf{x} \in \mathscr{X}$ and the maximizer plays $\mathbf{y} \in \mathscr{Y}$, the expected utility to the maximizer is $\langle \mathbf{y}, A\mathbf{x} \rangle$, and since the game is zero sum, the minimizer's expected utility is $\langle \mathbf{y}, -A\mathbf{x} \rangle$. Problem (1) can be expressed as a linear program, but the resulting formulations are prohibitively large for most interesting games. For instance, the payoff matrix A in (1) for limit Texas Hold'em poker has dimension $10^{14} \times 10^{14}$ and contains more than 10^{18} nonzero entries. Problems of this magnitude are far beyond the capabilities of state-of-the-art general-purpose linear programming solvers. Even solving a substantially smaller game with a $10^6 \times 10^6$ payoff matrix containing 50 million nonzeros with conventional linear programming solvers is computationally demanding both in terms of time and memory (Gilpin and Sandholm [5]).

We present a novel algorithmic approach for finding approximate solutions to (1). To this end, we define polytopes called *treeplexes* and concentrate on solving (1) when $\mathscr{X}$ and $\mathscr{Y}$ are polytopes of this type. Treeplexes generalize simplexes and include as a special case the strategy sets of sequential games. Our approach follows a current trend of applying first-order algorithms to nonsmooth optimization problems (Juditsky et al. [10]; Lan et al. [12]; Nemirovski [13]; Nesterov [16, 17]). A key feature of these algorithms is their low computational cost per iteration, which makes them particularly attractive for large problems. We adapt Nesterov's [16, 17] smoothing techniques for approximating (1). In particular, we develop first-order algorithms that take $\mathcal{O}(1/\epsilon)$ iterations to compute $\mathbf{x} \in \mathscr{X}$ and $\mathbf{y} \in \mathscr{Y}$ such that

$$0 \le \max_{v \in \mathscr{Y}} \langle \mathbf{v}, A\mathbf{x} \rangle - \min_{u \in \mathscr{X}} \langle \mathbf{y}, A\mathbf{u} \rangle \le \epsilon. \tag{2}$$

Such a pair of strategies is called an ϵ-equilibrium.

The simplicity and the low computational cost per iteration of our algorithm enables the computation of near equilibria for enormous sequential games. An implementation based on our approach has been successful in obtaining ϵ-equilibria for sequential games where the payoff matrix A is of size $10^8 \times 10^8$ and contains more than 10^{12} entries (§6). These games are abstracted poker games with 10^8 information sets and 10^{12} leaves in the game tree. This problem size (as measured by the number of leaves) is more than four orders of magnitude larger than what can be handled by solving the linear programming formulation via conventional solvers, such as interior-point methods (IPM) (Gilpin and Sandholm [4, 5]). Our implementation is a key component of several successful poker-playing computer programs for full-scale Heads-Up Texas Hold'em poker (Gilpin et al. [6, 7]).

The paper is organized as follows. Section 2 summarizes Nesterov's [16, 17] smoothing technique as it applies to problem (1). We highlight that technique's crucial ingredient, a pair of suitable *prox-functions* for the sets $\mathscr{X}$ and $\mathscr{Y}$. Section 3 presents our main idea, a template for constructing suitable prox-functions for treeplexes. Section 4 considers the special case of *uniform treeplexes*. For these treeplexes, we provide explicit bounds on the number of iterations needed for finding an ϵ-equilibrium. Sections 5 and 6 present some computational experience with an implementation based on our approach. Finally, §7 summarizes the main conclusions and discusses ideas for future work.

2. Smoothing techniques.

Problem (1) can be stated as

$$\min_{x \in \mathscr{X}} f(\mathbf{x}) = \max_{y \in \mathscr{Y}} \phi(\mathbf{y}), \tag{3}$$

where

$$f(\mathbf{x}) = \max_{y \in \mathscr{Y}} \langle \mathbf{y}, A\mathbf{x} \rangle \qquad \text{and} \qquad \phi(\mathbf{y}) = \min_{x \in \mathscr{X}} \langle \mathbf{y}, A\mathbf{x} \rangle.$$

The functions f and ϕ are respectively convex and concave nonsmooth functions. The left-hand side of (3) is a standard convex minimization problem of the form

$$\bar{h} := \min\{h(\mathbf{x}) : \mathbf{x} \in \mathscr{X}\}. \tag{4}$$

First-order methods for solving (4) are algorithms for which a search direction at each iteration is obtained using only the first-order information of h such as its gradient or subgradient. When h is smooth with Lipschitz gradient, there is a first-order algorithm for finding a point $\mathbf{x} \in \mathscr{X}$ such that $h(\mathbf{x}) \leq \bar{h} + \epsilon$ after $\mathcal{O}(1/\sqrt{\epsilon})$ iterations (Nesterov [14]). When h is nonsmooth, subgradient algorithms can be applied, but they have a worst-case complexity of $\mathcal{O}(1/\epsilon^2)$ iterations (Goffin [8]). However, that pessimistic result is based on treating h as a *black-box* where the value and subgradient are accessed via an oracle. For nonsmooth functions with a suitable max structure, Nesterov [16, 17] devised first-order algorithms requiring only $\mathcal{O}(1/\epsilon)$ iterations by applying a clever *smoothing technique*. In this paper, we adapt that smoothing technique for solving problem (1).

The key component of Nesterov's [16, 17] smoothing technique is a pair of *prox-functions* for the sets $\mathscr{X}$ and $\mathscr{Y}$. These prox-functions are used to construct smooth approximations $f_\mu \approx f$ and $\phi_\mu \approx \phi$. To obtain approximate solutions to (3), gradient-based algorithms can then be applied to f_μ and ϕ_μ.

DEFINITION 2.1. Assume $Q \subseteq \mathbb{R}^n$ is a convex compact set. A function $d : Q \to \mathbb{R}$ is a *prox-function* if it satisfies the following properties:

(i) d is strongly convex in Q, i.e., there exists $\sigma > 0$ such that for all $\mathbf{x}, \mathbf{y} \in Q$, and $\alpha \in [0, 1]$

$$d(\alpha\mathbf{x} + (1-\alpha)\mathbf{y}) \leq \alpha d(\mathbf{x}) + (1-\alpha)d(\mathbf{y}) - \tfrac{1}{2}\sigma\alpha(1-\alpha)\|\mathbf{x} - \mathbf{y}\|^2. \tag{5}$$

The largest value of the constant σ that satisfies (5) for a particular norm $\|\cdot\|$ is the *strong convexity modulus* of d with respect to $\|\cdot\|$. Note that the specific value of the strong convexity modulus σ depends on its associated norm $\|\cdot\|$.

(ii) $\min\{d(\mathbf{x}) : \mathbf{x} \in Q\} = 0$.

When $d : Q \to \mathbb{R}$ is differentiable, (5) can be equivalently stated in either of the following two forms (Nesterov [15]):

$$d(\mathbf{y}) \geq d(\mathbf{x}) + \langle \nabla d(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \tfrac{1}{2}\sigma\|\mathbf{x} - \mathbf{y}\|^2 \quad \text{for all } \mathbf{x}, \mathbf{y} \in Q, \tag{6}$$

$$\langle \nabla d(\mathbf{x}) - \nabla d(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle \geq \sigma\|\mathbf{x} - \mathbf{y}\|^2 \quad \text{for all } \mathbf{x}, \mathbf{y} \in Q. \tag{7}$$

Assume $d_{\mathcal{X}}$ and $d_{\mathcal{Y}}$ are prox-functions for the sets $\mathcal{X}$ and $\mathcal{Y}$, respectively. Then for any given $\mu > 0$, the smooth approximations $f_\mu \approx f$ and $\phi_\mu \approx \phi$ are

$$f_\mu(\mathbf{x}) := \max\{\langle \mathbf{y}, A\mathbf{x} \rangle - \mu d_{\mathcal{Y}}(\mathbf{y}) \colon \mathbf{y} \in \mathcal{Y}\}, \qquad \phi_\mu(\mathbf{y}) := \min\{\langle \mathbf{y}, A\mathbf{x} \rangle + \mu d_{\mathcal{X}}(\mathbf{x}) \colon \mathbf{x} \in \mathcal{X}\}.$$

The following result of Nesterov [16, 17] provides the theoretical foundation of our first-order algorithms for solving (1). Let $D_{\mathcal{X}} := \max\{d_{\mathcal{X}}(\mathbf{x}) \colon \mathbf{x} \in \mathcal{X}\}$, and let $\sigma_{\mathcal{X}}$ denote the strong convexity modulus of $d_{\mathcal{X}}$. Let $D_{\mathcal{Y}}$ and $\sigma_{\mathcal{Y}}$ be defined likewise for $\mathcal{Y}$ and $d_{\mathcal{Y}}$. The operator norm of A used below is defined as $\|A\| := \max\{\langle \mathbf{y}, A\mathbf{x} \rangle \colon \|\mathbf{x}\|, \|\mathbf{y}\| \le 1\}$, where the norms $\|\mathbf{x}\|$, $\|\mathbf{y}\|$ are those associated with $\sigma_{\mathcal{X}}$ and $\sigma_{\mathcal{Y}}$.

THEOREM 2.2 (NESTEROV [16, 17]). *There is a procedure based on the above smoothing technique that after N iterations generates a pair of points $(\mathbf{x}^N, \mathbf{y}^N) \in \mathcal{X} \times \mathcal{Y}$ such that*

$$0 \le f(\mathbf{x}^N) - \phi(\mathbf{y}^N) \le \frac{4\|A\|}{N+1} \sqrt{\frac{D_{\mathcal{X}} D_{\mathcal{Y}}}{\sigma_{\mathcal{X}} \sigma_{\mathcal{Y}}}}. \tag{8}$$

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

$$\max_{x \in \mathcal{X}}\{\langle \mathbf{g}, \mathbf{x} \rangle - d_{\mathcal{X}}(\mathbf{x})\} \qquad or \qquad \max_{y \in \mathcal{Y}}\{\langle \mathbf{g}, \mathbf{y} \rangle - d_{\mathcal{Y}}(\mathbf{y})\}. \tag{9}$$

In §5, we will present an explicit algorithm as stated in Theorem 2.2. Before that, we first provide a method for solving the subproblems in (9) as these are critical steps in the algorithm. These subproblems can be phrased in terms of the conjugate of the functions $d_{\mathcal{X}}$ and $d_{\mathcal{Y}}$ (Hirriart-Urruty and Lemaréchal [9]). The conjugate of $d \colon Q \to \mathbb{R}$ is the function $d^* \colon \mathbb{R}^n \to \mathbb{R}$ defined by

$$d^*(\mathbf{s}) := \max\{\langle \mathbf{s}, \mathbf{x} \rangle - d(\mathbf{x}) \colon \mathbf{x} \in Q\}.$$

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

$$\nabla d^*(\mathbf{s}) = \arg\max\{\langle \mathbf{s}, \mathbf{x} \rangle - d(\mathbf{x}) \colon \mathbf{x} \in Q\}.$$

(For a detailed discussion, see Hirriart-Urruty and Lemaréchal [9].)

For an algorithm based on Theorem 2.2 to be practical, the subproblems (9) must be solvable quickly because their solution is required three times at each iteration of the algorithm. In other words, the conjugates $d_{\mathcal{X}}^*$ and $d_{\mathcal{Y}}^*$ and their gradients $\nabla d_{\mathcal{X}}^*$ and $\nabla d_{\mathcal{Y}}^*$ should be easily computable. This motivates the following definition.

DEFINITION 2.3. Assume $Q \subseteq \mathbb{R}^n$ is a compact convex set. We say that $d \colon Q \to \mathbb{R}$ is a *nice prox-function* for Q if it satisfies the following three conditions:

(i) d is continuous and strongly convex in Q, and differentiable in the relative interior of Q.

(ii) The conjugate d^* satisfies $d^*(\mathbf{0}) = 0$.

(iii) The conjugate function d^* and its gradient ∇d^* are easily computable.

EXAMPLE 1. For the k-dimensional simplex Δ_k, the entropy function $d(\mathbf{x}) = \ln k + \sum_{i=1}^k x_i \ln x_i$, and the Euclidean distance function $d(\mathbf{x}) = \frac{1}{2} \sum_{i=1}^k (x_i - 1/k)^2$ are nice prox-functions. Indeed, for the entropy prox-function, the gradient of the conjugate $\nabla d^*(\mathbf{s})$ is given by the closed-form expression

$$\nabla_i d^*(\mathbf{s}) = \frac{e^{s_i}}{\sum_{j=1}^k e^{s_j}}, \quad i = 1, \ldots, k.$$

Furthermore, as discussed in Juditsky et al. [10] and Nesterov [16], the entropy function has strong convexity modulus equal to one for the L^1-norm $\|\mathbf{x}\| := \sum_{j=1}^k |x_j|$.

For the Euclidean prox-function, the gradient of the conjugate $\nabla d^*(\mathbf{s})$ is given by the expression

$$\nabla_i d^*(\mathbf{s}) = (s_i - \lambda)^+, \quad i = 1, \ldots, k,$$

where $\lambda \in \mathbb{R}$ is such that $\sum_{j=1}^k (s_j - \lambda)^+ = 1$. This value of λ can be found in $\mathcal{O}(k \ln k)$ steps via a binary search in the sorted components of $\mathbf{s}$. Furthermore, from (7), it follows that the Euclidean prox-function has strong convexity modulus equal to one for the Euclidean norm $\|\mathbf{x}\| := \sqrt{\sum_{j=1}^k x_j^2}$.

3. Treeplexes. This section presents the essential elements of our approach. We define the class of *treeplex* polytopes and provide a generic technique for constructing nice prox-functions for treeplexes, using as building blocks any family of nice prox-functions for simplexes. This allows us to create practical first-order algorithms based on Theorem 2.2 for solving the saddle-point problem (1) over treeplexes $\mathscr{X}$ and $\mathscr{Y}$.

A treeplex can be seen as a tree whose nodes are simplexes. The tree structure endows the treeplex with a certain kind of sequential characteristic. In particular, treeplexes include the types of polytopes that arise in the computation of Nash equilibria of sequential games. The latter is an immediate consequence of the *sequence form* formulation of Nash equilibria for sequential games, as detailed in Koller et al. [11], Romanovskii [19], and von Stengel [21, 22].

DEFINITION 3.1. The class of treeplexes is recursively defined as follows:
   (i) *Basic sets*: Every standard simplex $\Delta_m := \{\mathbf{x} \in [0, 1]^m: \sum_{j=1}^m x_j = 1\}$ is a treeplex.
   (ii) *Cartesian product*: If $Q_1, \ldots, Q_k$ are treeplexes, then $Q_1 \times \cdots \times Q_k$ is a treeplex.
   (iii) *Branching*: If $P \subseteq [0, 1]^p$ and $Q \subseteq [0, 1]^q$ are treeplexes and $i \in \{1, \ldots, p\}$, then

$$P \boxed{i} Q := \{(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^{p+q}: \mathbf{x} \in P, \mathbf{y} \in x_i \cdot Q\}$$

is a treeplex.

The Branching operation in Definition 3.1 has the following sequential interpretation: the vector $\mathbf{x}$ is the set of "current stage" decision variables, and the vector $\mathbf{y}$ is the set of "next stage" decision variables following the i-th current decision variable x_i. Notice that a treeplex can be written in the form $\{\mathbf{x} \geq 0: E\mathbf{x} = \mathbf{e}\}$ for some matrix E with entries in $\{-1, 0, 1\}$ and vector $\mathbf{e}$ with entries in $\{0, 1\}$, see von Stengel [21, 22].

In the sequel, we will often need to compare the norm of a vector $(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^{p+q}$ with those of $\mathbf{x} \in \mathbb{R}^p$, $\mathbf{y} \in \mathbb{R}^q$. This requires a certain compatibility of the norms in the spaces $\mathbb{R}^p$, $\mathbb{R}^q$, and $\mathbb{R}^{p+q}$. Henceforth we shall make the following mild *norm-embedding assumption*:

$$\|\mathbf{x}\| = \|(\mathbf{x}, \mathbf{0})\|, \qquad \|\mathbf{y}\| = \|(\mathbf{0}, \mathbf{y})\|. \tag{10}$$

We now present our general procedure for constructing nice prox-functions for treeplexes. The construction relies on the following *dilation* operation from convex analysis (Hirriart-Urruty and Lemaréchal [9]). Given a compact set $K \subseteq \mathbb{R}^d$ and a function $\Phi: K \to \mathbb{R}$, define the set $\bar{K} \subseteq \mathbb{R}^{d+1}$ as

$$\bar{K} := \{(x, \mathbf{y}) \in \mathbb{R}^{d+1}: x \in [0, 1], \mathbf{y} \in x \cdot K\},$$

and define the function $\bar{\Phi}: \bar{K} \to \mathbb{R}$ as

$$\bar{\Phi}(x, \mathbf{y}) = \begin{cases} x \cdot \Phi(\mathbf{y}/x) & \text{if } x > 0, \\ 0 & \text{if } x = 0. \end{cases}$$

PROPOSITION 3.2. *If K is compact and Φ is continuous in K, then $\bar{\Phi}$ is continuous in $\bar{K}$. Also, if $(x, \mathbf{y}) \in \bar{K}$ is such that $x > 0$ and $\nabla\Phi(\mathbf{y}/x)$ exists, then $\nabla\bar{\Phi}(x, \mathbf{y})$ exists and*

$$\begin{aligned} \nabla_x \bar{\Phi}(x, \mathbf{y}) &= \Phi(\mathbf{y}/x) - \langle \nabla\Phi(\mathbf{y}/x), \mathbf{y}/x \rangle, \\ \nabla_{\mathbf{y}} \bar{\Phi}(x, \mathbf{y}) &= \nabla\Phi(\mathbf{y}/x). \end{aligned} \tag{11}$$

PROOF. The continuity follows via a straightforward limiting argument: Assume $(x^i, \mathbf{y}^i)$, $(x, \mathbf{y}) \in \bar{K}$, and $(x^i, \mathbf{y}^i) \to (x, \mathbf{y})$. If $x > 0$, then $\mathbf{y}^i/x_i\ \mathbf{y}/x \in K$ and $\mathbf{y}^i/x_i \to \mathbf{y}/x$. Since Φ is continuous, we get

$$\bar{\Phi}(x^i, \mathbf{y}^i) = \Phi(\mathbf{y}^i/x^i) \to \Phi(\mathbf{y}/x) = \bar{\Phi}(x, \mathbf{y}).$$

On the other hand, if $x = 0$, then $x^i \to 0$. Consequently,

$$|\bar{\Phi}(x^i, \mathbf{y}^i)| = |x^i \Phi(\mathbf{y}^i/x^i)| \leq x^i \max\{\Phi(\mathbf{z}): \mathbf{z} \in K\} \to 0 = \bar{\Phi}(x, \mathbf{y}).$$

Finally, the identities in (11) follow by applying the chain rule. $\square$

Assume we are given a family of nice prox-functions d_m for Δ_m, $m \in \mathbb{Z}^+$. Using this family, we recursively construct functions for treeplexes as follows:
   (i) *Basic sets*: For $Q = \Delta_m$, let $d_Q := d_m$.

(ii) *Cartesian product*: If $Q_1, \ldots, Q_k$ are treeplexes and $Q = Q_1 \times \cdots \times Q_k$, let

$$d_Q(\mathbf{x}^1, \ldots, \mathbf{x}^k) := \sum_{i=1}^{k} d_{Q_i}(\mathbf{x}^i),$$

where $d_{Q_1}, \ldots, d_{Q_k}$ are nice prox-functions for their respective treeplexes.

(iii) *Branching*: If $P \subseteq [0,1]^p$ and $R \subseteq [0,1]^r$ are treeplexes, $i \in \{1, \ldots, p\}$, and $Q = P\,\boxed{i}\,R$, let

$$d_Q(\mathbf{x}, \mathbf{y}) := d_P(\mathbf{x}) + \bar{d}_R(x_i, \mathbf{y}), \tag{12}$$

where d_P and d_R are nice prox-functions for P and R.

THEOREM 3.3.   *The functions d_Q defined above are nice prox-functions for each treeplex Q.*

To prove Theorem 3.3, it suffices to show that the properties of nice prox-functions are preserved for the Cartesian product and Branching steps. Since the Cartesian product step is straightforward, we concentrate on the Branching step as stated in the following proposition.

PROPOSITION 3.4.   *Assume $P \subseteq [0,1]^p$ and $R \subseteq [0,1]^r$ are treeplexes, $i \in \{1, \ldots, p\}$, and $Q = P\,\boxed{i}\,R$. Furthermore, assume d_P and d_R are nice prox-functions for P and R, respectively, and*

$$d_Q(\mathbf{x}, \mathbf{y}) := d_P(\mathbf{x}) + \bar{d}_R(x_i, \mathbf{y}).$$

Then

(i) *d_Q is continuous and strongly convex in Q and differentiable in the relative interior of Q.*

(ii) *d_Q^* and ∇d_Q^* are computable via the following expressions*:

$$d_Q^*(\mathbf{u}, \mathbf{v}) = d_P^*(\tilde{\mathbf{u}}), \tag{13}$$

$$\nabla d_Q^*(\mathbf{u}, \mathbf{v}) = (\nabla d_P^*(\tilde{\mathbf{u}}), \nabla_i d_P^*(\tilde{\mathbf{u}}) \cdot \nabla d_R^*(\mathbf{v})), \tag{14}$$

where

$$\tilde{u}_j = \begin{cases} u_j & \text{if } j \neq i, \\ u_i + d_R^*(\mathbf{v}) & \text{if } j = i. \end{cases}$$

PROOF.   (i) The continuity of d_Q in Q and the differentiability in the relative interior of Q follow from (12) and Proposition 3.2. Since d_Q is continuous in Q, to prove its strong convexity, from (7) it suffices to show that there exists $\sigma > 0$ such that

$$\langle \nabla d_Q(\mathbf{x}, \mathbf{y}) - \nabla d_Q(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}), (\mathbf{x}, \mathbf{y}) - (\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) \rangle \geq \sigma \|(\mathbf{x}, \mathbf{y}) - (\tilde{\mathbf{x}}, \tilde{\mathbf{y}})\|^2 \tag{15}$$

for all $(\mathbf{x}, \mathbf{y})$ and $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$ in the relative interior of Q.

Assume $(\mathbf{x}, \mathbf{y})$ and $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$ are in the relative interior of Q. Set $\mathbf{z} := \mathbf{y}/x_i$ and $\tilde{\mathbf{z}} := \tilde{\mathbf{y}}/\tilde{x}_i$. From (12), Proposition 3.2, and some elementary calculations, we get

$$\begin{aligned} \langle \nabla d_Q(\mathbf{x}, \mathbf{y}) - \nabla d_Q(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}), (\mathbf{x}, \mathbf{y}) - (\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) \rangle = {} & \langle \nabla d_P(\mathbf{x}) - \nabla d_P(\tilde{\mathbf{x}}), \mathbf{x} - \tilde{\mathbf{x}} \rangle \\ & + x_i \cdot (d_R(\mathbf{z}) - d_R(\tilde{\mathbf{z}}) + \langle \nabla d_R(\tilde{\mathbf{z}}), \tilde{\mathbf{z}} - \mathbf{z} \rangle) \\ & + \tilde{x}_i \cdot (d_R(\tilde{\mathbf{z}}) - d_R(\mathbf{z}) + \langle \nabla d_R(\mathbf{z}), \mathbf{z} - \tilde{\mathbf{z}} \rangle). \end{aligned}$$

Therefore, since d_P and d_R are strongly convex, (6) yields

$$\begin{aligned} \langle \nabla d_Q(\mathbf{x}, \mathbf{y}) - \nabla d_Q(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}), (\mathbf{x}, \mathbf{y}) - (\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) \rangle & \geq \sigma_P \|\mathbf{x} - \tilde{\mathbf{x}}\|^2 + \tfrac{1}{2}\sigma_R x_i \|\mathbf{z} - \tilde{\mathbf{z}}\|^2 + \tfrac{1}{2}\sigma_R \tilde{x}_i \|\mathbf{z} - \tilde{\mathbf{z}}\|^2 \\ & = \sigma_P \|\mathbf{x} - \tilde{\mathbf{x}}\|^2 + \sigma_R \hat{x}_i \|\mathbf{z} - \tilde{\mathbf{z}}\|^2, \end{aligned} \tag{16}$$

where $\hat{x}_i = (x_i + \tilde{x}_i)/2$ and $\sigma_P, \sigma_R > 0$ are the strong convexity parameters of d_P and d_R, respectively.

Next, we bound the right-hand side of (15). Applying the triangle inequality and using the norm-embedding assumption (10), we get

$$
\begin{aligned}
\|(\mathbf{x}, \mathbf{y}) - (\tilde{\mathbf{x}}, \tilde{\mathbf{y}})\| &\leq \|\mathbf{x} - \tilde{\mathbf{x}}\| + \|x_i \mathbf{z} - \tilde{x}_i \tilde{\mathbf{z}}\| \\
&= \|\mathbf{x} - \tilde{\mathbf{x}}\| + \|\tfrac{1}{2}(x_i + \tilde{x}_i)(\mathbf{z} - \tilde{\mathbf{z}}) + \tfrac{1}{2}(x_i - \tilde{x}_i)(\mathbf{z} + \tilde{\mathbf{z}})\| \\
&\leq \|\mathbf{x} - \tilde{\mathbf{x}}\| + \hat{x}_i \|\mathbf{z} - \tilde{\mathbf{z}}\| + \tfrac{1}{2}|x_i - \tilde{x}_i|\|\mathbf{z} + \tilde{\mathbf{z}}\|.
\end{aligned}
\tag{17}
$$

Since R is compact, the value $M := \max\{\|\mathbf{z}\| : \mathbf{z} \in R\}$ is finite. Therefore, from (17), we get

$$
\|(\mathbf{x}, \mathbf{y}) - (\tilde{\mathbf{x}}, \tilde{\mathbf{y}})\| \leq (1 + M)\|\mathbf{x} - \tilde{\mathbf{x}}\| + \hat{x}_i \|\mathbf{z} - \tilde{\mathbf{z}}\|.
$$

Now, by the Cauchy-Schwarz inequality,

$$
\|(\mathbf{x}, \mathbf{y}) - (\tilde{\mathbf{x}}, \tilde{\mathbf{y}})\|^2 \leq \left((1 + M)^2 \frac{1}{\sigma_P} + \frac{\hat{x}_i}{\sigma_R} \right)(\sigma_P \|\mathbf{x} - \tilde{\mathbf{x}}\|^2 + \sigma_R \hat{x}_i \|\mathbf{z} - \tilde{\mathbf{z}}\|^2).
$$

Since $\mathbf{x}, \tilde{\mathbf{x}} \in P \subseteq [0, 1]^p$, we get

$$
\|(\mathbf{x}, \mathbf{y}) - (\tilde{\mathbf{x}}, \tilde{\mathbf{y}})\|^2 \leq \left((1 + M)^2 \frac{1}{\sigma_P} + \frac{1}{\sigma_R} \right)(\sigma_P \|\mathbf{x} - \tilde{\mathbf{x}}\|^2 + \sigma_R \hat{x}_i \|\mathbf{z} - \tilde{\mathbf{z}}\|^2).
\tag{18}
$$

From (16) and (18), it follows that (15) holds for

$$
\sigma = \frac{1}{(1 + M)^2/\sigma_P + 1/\sigma_R} > 0.
$$

(ii) For a given vector $(\mathbf{u}, \mathbf{v}) \in \mathbb{R}^{p+r}$, we have

$$
\begin{aligned}
d_Q^*(\mathbf{u}, \mathbf{v}) &= \sup\{\langle(\mathbf{u}, \mathbf{v}), (\mathbf{x}, \mathbf{y})\rangle - d_Q(\mathbf{x}, \mathbf{y}) : (\mathbf{x}, \mathbf{y}) \in Q\} \\
&= \sup\{\langle\mathbf{u}, \mathbf{x}\rangle + \langle\mathbf{v}, \mathbf{y}\rangle - d_P(\mathbf{x}) - \bar{d}_R(x_i, \mathbf{y}) : \mathbf{x} \in P, \; \mathbf{y} \in x_i \cdot R\} \\
&= \sup\{\langle\mathbf{u}, \mathbf{x}\rangle - d_P(\mathbf{x}) + x_i \cdot (\langle\mathbf{v}, \mathbf{z}\rangle - d_R(\mathbf{z})) : \mathbf{x} \in P, \; \mathbf{z} \in R, \; x_i > 0\} \\
&= \sup\{\langle\mathbf{u}, \mathbf{x}\rangle - d_P(\mathbf{x}) + x_i \cdot d_R^*(\mathbf{z}) : \mathbf{x} \in P\} \\
&= \sup\{\langle\tilde{\mathbf{u}}, \mathbf{x}\rangle - d_P(\mathbf{x}) : \mathbf{x} \in P\} \\
&= d_P^*(\tilde{\mathbf{u}}).
\end{aligned}
\tag{19}
$$

The third and fourth steps above hold by the continuity of $\bar{d}_R$ and d_P. Hence (13) is proven. To prove (14), observe that the maximizer in the second to last step in (19) is $\bar{\mathbf{x}} = \nabla d_P^*(\tilde{\mathbf{u}})$. Next, consider two cases depending on the value of $\bar{x}_i$. If $\bar{x}_i > 0$, then the maximizer in the third step in (19) is $\bar{\mathbf{z}} = \nabla d_R^*(\tilde{\mathbf{v}})$, and, consequently, the maximizer in the first step in (19) is $(\bar{\mathbf{x}}, \bar{x}_i \cdot \bar{\mathbf{z}})$. If $\bar{x}_i = 0$, then the maximizer in the first step in (19) is $(\bar{\mathbf{x}}, \mathbf{0})$. In either case, the maximizer in the first step in (19) is $\nabla d_Q^*(\mathbf{u}, \mathbf{v}) = (\bar{\mathbf{x}}, \bar{x}_i \cdot \bar{\mathbf{z}}) = (\nabla d_P^*(\tilde{\mathbf{u}}), \nabla_i d_P^*(\tilde{\mathbf{u}}) \cdot \nabla d_R^*(\mathbf{v}))$. $\square$

REMARK 3.5. We can generalize the above construction and results to weighted versions of the prox-functions. More precisely, in the Branching step, we can define $d_Q(\mathbf{x}, \mathbf{y}) := w_P d_P(\mathbf{x}) + w_R \bar{d}_R(x_i, \mathbf{y})$ for some constants $w_P, w_R > 0$. We will elaborate on this idea to obtain prox-functions yielding better complexity guarantees for uniform treeplexes.

4. Uniform treeplexes. In this section, we derive complexity results for first-order smoothing algorithms for problem (1) in the special case when $\mathscr{X}$ and $\mathscr{Y}$ are *uniform* treeplexes. This special case of (1) covers the formulation of Nash equilibrium for instances of many interesting games. Indeed, as will be discussed in §6, uniform treeplexes naturally arise in multiround sequential games such as poker.

DEFINITION 4.1. Assume that a treeplex $Q \subseteq [0, 1]^q$, an index set $I = \{i_1, \ldots, i_b\} \subseteq \{1, \ldots, q\}$, and a positive integer k are given. Define Q_r, $r = 1, 2, \ldots,$ as follows:
  (i) $Q_1 := Q \times \cdots \times Q$ (k times).
  (ii) $Q_{r+1} := \hat{Q}_r \times \cdots \times \hat{Q}_r$ (k times), where

$$
\hat{Q}_r := Q \boxed{I} Q_r := \{(\mathbf{x}, \mathbf{y}^1, \ldots, \mathbf{y}^b) : \mathbf{x} \in Q, \; \mathbf{y}^j \in x_{i_j} \cdot Q_r, \; j = 1, \ldots, b\}.
$$

We will refer to Q_r as the r-th uniform treeplex generated by Q, I, k and will sometimes write it as $\mathscr{Q}(Q, I, k, r)$.

REMARK 4.2. Notice that the operation $\boxed{I}$ is the same as the operation $\boxed{i}$ applied b times. More precisely,

$$Q\boxed{I}Q_r = Q\boxed{i_1}Q_r\boxed{i_2}\cdots\boxed{i_b}Q_r.$$

Given a nice prox-function d_Q for Q and constants $w_r > 0$, $r = 1, 2, \ldots$, consider the following weighted version of our previous construction of prox-functions for treeplexes:

(i) For $Q_1 = Q \times \cdots \times Q$ (k times), let

$$d_{Q_1}(\mathbf{x}^1, \ldots, \mathbf{x}^k) := \sum_{j=1}^{k} d_Q(\mathbf{x}^j)$$

(ii) For $Q_{r+1} = \hat{Q}_r \times \cdots \times \hat{Q}_r$ (k times), let

$$d_{Q_{r+1}}(\mathbf{u}^1, \ldots, \mathbf{u}^k) := \sum_{j=1}^{k} d_{\hat{Q}_r}(\mathbf{u}^j),$$

where $d_{\hat{Q}_r}$ is defined as

$$d_{\hat{Q}_r}(\mathbf{x}, \mathbf{y}^1, \ldots, \mathbf{y}^b) := w_r \cdot d_Q(\mathbf{x}) + \sum_{j=1}^{b} \bar{d}_{Q_r}(x_{i_j}, \mathbf{y}).$$

We now present an explicit iteration complexity bound for a first-order smoothing algorithm for the saddle-point problem (1), when $\mathscr{X}$ and $\mathscr{Y}$ are uniform treeplexes. As in Theorem 2.2, the norm of A, $\|A\|$ is the induced operator norm of A, where the underlying norms are those associated with σ_Q and $\sigma_{\tilde{Q}}$. In particular, the result below holds for any choice of norms.

THEOREM 4.3. *Suppose* A, $\mathscr{X}$, $\mathscr{Y}$, $d_{\mathscr{X}}$, *and* $d_{\mathscr{Y}}$ *satisfy the following conditions*:

(i) $\mathscr{X} = \mathcal{Q}(Q, I, k, r) \subseteq \mathbb{R}^m$ *and* $\mathscr{Y} = \mathcal{Q}(\tilde{Q}, \tilde{I}, \tilde{k}, \tilde{r}) \subseteq \mathbb{R}^n$.

(ii) *The prox-functions* $d_{\mathscr{X}}, d_{\mathscr{Y}}$ *are constructed as above with weights* $w_j = (kM)^2(bk)^j$, $j = 1, \ldots, r-1$ *and* $\tilde{w}_j = (\tilde{k}\tilde{M})^2(\tilde{b}\tilde{k})^j$, $j = 1, \ldots, \tilde{r}-1$, *respectively, where* $b = |I|$, $\tilde{b} = |\tilde{I}|$, $M := \max\{\|\mathbf{u}\| : \mathbf{u} \in Q\}$, $\tilde{M} := \max\{\|\mathbf{u}\| : \mathbf{u} \in \tilde{Q}\}$.
Then after N *iterations the procedure from Theorem* 2.2 *yields* $(\mathbf{x}, \mathbf{y}) \in \mathscr{X} \times \mathscr{Y}$ *such that*

$$0 \le f(\mathbf{x}) - \phi(\mathbf{y}) = \max_{\mathbf{v} \in \mathscr{Y}} \langle \mathbf{v}, A\mathbf{x} \rangle - \min_{u \in \mathscr{X}} \langle \mathbf{y}, A\mathbf{u} \rangle \le \frac{4\|A\|G}{N+1}\sqrt{\frac{D_Q D_{\tilde{Q}}}{\sigma_Q \sigma_{\tilde{Q}}}}, \tag{20}$$

where $G = mn(kMr)(\tilde{k}\tilde{M}\tilde{r})$.

The crux of the proof of Theorem 4.3 is Lemma 4.4, which bounds the ratio of the maximum value to the strong convexity modulus for the prox-functions for uniform treeplexes. This ratio can be seen as a measure of the prox-function's quality. Lemma 4.4 provides an estimate of this ratio for the prox-functions d_{Q_r} constructed above, provided the weights w_r are chosen judiciously.

LEMMA 4.4. *Assume* Q *and* Q_r, $r = 1, 2, \ldots$, *are as in Definition* 4.1. *Let* σ, σ_r, D, D_r, *and* M *be defined as follows*:

$$\sigma := \text{strong convexity modulus of } d_Q, \qquad \sigma_r := \text{strong convexity modulus of } d_{Q_r},$$

$$D := \max\{d_Q(\mathbf{z}) : \mathbf{z} \in Q\}, \qquad D_r := \max\{d_{Q_r}(\mathbf{z}) : \mathbf{z} \in Q_r\},$$

$$M := \max\{\|\mathbf{z}\| : \mathbf{z} \in Q\}, \qquad M_r := \max\{\|\mathbf{z}\| : \mathbf{z} \in Q_r\}.$$

(i) *The strong convexity moduli* σ_r *of* d_{Q_r}, $r = 1, 2, \ldots$, *satisfy*

$$\sigma_{r+1} \ge \frac{1}{k(1+M_r)^2/(w_r\sigma) + bk/\sigma_r}. \tag{21}$$

(ii) *If* $w_r = (kM)^2(bk)^r$, $r = 1, 2, \ldots$, *then*

$$\frac{D_r}{\sigma_r} \le b^{2r-2}k^{2r+2}r^2M^2\frac{D}{\sigma}. \tag{22}$$

PROOF. (i) Let $\hat{\sigma}_r$ be the strong convexity modulus of $d_{\hat{Q}_r}$. From the construction of d_{Q_r}, it follows that $\sigma_{r+1} \geq \hat{\sigma}_r/k$. Hence it suffices to bound $\hat{\sigma}_r$. Proceeding as in the proof of Proposition 3.4(i), it follows that for all $\mathbf{w} = (\mathbf{x}, \mathbf{y}^1, \dots, \mathbf{y}^b)$ and $\widetilde{\mathbf{w}} = (\tilde{\mathbf{x}}, \tilde{\mathbf{y}}^1, \dots, \tilde{\mathbf{y}}^b)$ in the relative interior of $\hat{Q}_r$, we have

$$\langle \nabla d_{\hat{Q}_r}(\mathbf{w}) - \nabla d_{\hat{Q}_r}(\widetilde{\mathbf{w}}), \mathbf{w} - \widetilde{\mathbf{w}} \rangle \geq w_r \sigma \|\mathbf{x} - \tilde{\mathbf{x}}\|^2 + \sigma_r \sum_{j=1}^b \hat{x}_{i_j} \|\mathbf{z}^j - \tilde{\mathbf{z}}^j\|^2 \tag{23}$$

and

$$\|\mathbf{w} - \widetilde{\mathbf{w}}\| \leq (1 + M_r)\|\mathbf{x} - \tilde{\mathbf{x}}\| + \sum_{j=1}^b \hat{x}_{i_j} \|\mathbf{z}^j - \tilde{\mathbf{z}}^j\|, \tag{24}$$

where $\mathbf{z}^j = \mathbf{y}^j/x_{i_j}$ and $\tilde{\mathbf{z}}^j = \tilde{\mathbf{y}}^j/\tilde{x}_{i_j}$ for $j = 1, \dots, b$. Applying the Cauchy-Schwarz inequality to (24), we get

$$\|\mathbf{w} - \widetilde{\mathbf{w}}\|^2 \leq \left(\frac{(1 + M_r)^2}{w_r \sigma} + \frac{\sum_{j=1}^b \hat{x}_{i_j}}{\sigma_r} \right) \left(w_r \sigma \|\mathbf{x} - \tilde{\mathbf{x}}\|^2 + \sigma_r \sum_{j=1}^b \hat{x}_{i_j} \|\mathbf{z}^j - \tilde{\mathbf{z}}^j\|^2 \right)$$

$$\leq \left(\frac{(1 + M_r)^2}{w_r \sigma} + \frac{b}{\sigma_r} \right) \left(w_r \sigma \|\mathbf{x} - \tilde{\mathbf{x}}\|^2 + \sigma_r \sum_{j=1}^b \hat{x}_{i_j} \|\mathbf{z}^j - \tilde{\mathbf{z}}^j\|^2 \right). \tag{25}$$

From (23), (25), and the continuity of $d_{\hat{Q}_r}$, we obtain

$$\hat{\sigma}_r \geq \frac{1}{(1 + M_r)^2/(w_r \sigma) + b/\sigma_r},$$

which yields (21) since $\sigma_{r+1} \geq \hat{\sigma}_r/k$.

(ii) Let $M_r := \max\{\|\mathbf{z}\|: \mathbf{z} \in Q_r\}$. We have $M_1 \leq kM$ and $M_{r+1} \leq k(M + bM_r)$, so

$$1 + M_r \leq kM(bk)^r, \quad r = 1, 2, \dots.$$

Hence $w_r \geq (1 + M_r)^2/((bk)^r)$, and, consequently, (21) yields

$$\frac{1}{(bk)^{r+1}\sigma_{r+1}} \leq \frac{1}{b\sigma} + \frac{1}{(bk)^r \sigma_r}.$$

Therefore, since $\sigma_1 \geq \sigma/k$, it follows that

$$\frac{1}{(bk)^r \sigma_r} \leq \frac{r}{b\sigma}, \quad r = 1, 2, \dots. \tag{26}$$

On the other hand, from the construction of Q_r and d_{Q_r}, we have

$$D_1 \leq kD, \qquad D_{r+1} \leq k(w_r D + bD_r), \quad r = 1, 2, \dots,$$

so

$$D_r \leq kD\left((bk)^{r-1} + \sum_{j=1}^{r-1} w_j (bk)^{r-1-j} \right).$$

Thus

$$D_r \leq kD\left((bk)^{r-1} + \sum_{j=1}^{r-1} w_j (bk)^{r-1-j} \right)$$

$$= kD\left((bk)^{r-1} + (kM)^2 \sum_{j=1}^{r-1} (bk)^j (bk)^{r-1-j} \right)$$

$$= kD(1 + (kM)^2(r-1))(bk)^{r-1}$$

$$\leq krD(kM)^2(bk)^{r-1}. \tag{27}$$

Finally, (22) follows by putting together (26) and (27). $\square$

PROOF OF THEOREM 4.3. Since $\mathscr{X} = \mathcal{Q}(Q, I, k, r) \subseteq \mathbb{R}^m$, Lemma 4.4 yields

$$\frac{D_{\mathscr{X}}}{\sigma_{\mathscr{X}}} \leq b^{2r-2} k^{2r+2} r^2 M^2 \frac{D_Q}{\sigma_Q}.$$

In addition, a simple induction argument shows the dimension m of $\mathscr{X} = \mathcal{Q}(Q, I, k, r)$ satisfies $m = kq((bk)^r - 1)/(bk - 1)$. Therefore

$$\frac{D_{\mathscr{X}}}{\sigma_{\mathscr{X}}} \leq m^2 k^2 r^2 M^2 \frac{D_Q}{\sigma_Q}. \tag{28}$$

Similarly,

$$\frac{D_{\mathscr{Y}}}{\sigma_{\mathscr{Y}}} \leq n^2 \tilde{k}^2 \tilde{r}^2 \tilde{M}^2 \frac{D_{\tilde{Q}}}{\sigma_{\tilde{Q}}}. \tag{29}$$

The iteration bound (20) now follows from (8), (28), and (29). $\square$

For the special case when the norm in $\mathbb{R}^q$ and each $\mathbb{R}^{q_r}$ is the Euclidean norm, we can sharpen the bound in Lemma 4.4, and thus also the bound in Theorem 4.3.

LEMMA 4.5. *Assume b, M, D, D_r, σ, and σ_r are as in Lemma 4.4, and the norm in $\mathbb{R}^q$ and each $\mathbb{R}^{q_r}$ is the Euclidean norm. If $w_r = kM^2 k^r$, $r = 1, 2, \ldots$, then*

$$\frac{D_r}{\sigma_r} \leq b^{2r-2} k^{r+1} r^2 M^2 \frac{D}{\sigma}. \tag{30}$$

PROOF. For the Euclidean norm, we have $\sigma_{r+1} = \hat{\sigma}_r$, where $\hat{\sigma}_r$ is the strong convexity modulus of $d_{\hat{Q}_r}$. Next, we proceed to bound $\hat{\sigma}_r$ as in the proof of Lemma 4.4. For all $\mathbf{w} = (\mathbf{x}, \mathbf{y}^1, \ldots, \mathbf{y}^b)$ and $\tilde{\mathbf{w}} = (\tilde{\mathbf{x}}, \tilde{\mathbf{y}}^1, \ldots, \tilde{\mathbf{y}}^b)$ in the relative interior of $\hat{Q}_r$, the inequality (23) holds. Next, instead of (24), we can use

$$\|\mathbf{w} - \tilde{\mathbf{w}}\|^2 = \|\mathbf{x} - \tilde{\mathbf{x}}\|^2 + \sum_{j=1}^{b} \|x_{i_j} \mathbf{z}^j - \tilde{x}_{i_j} \tilde{\mathbf{z}}^j\|^2$$

$$\leq \|\mathbf{x} - \tilde{\mathbf{x}}\|^2 + \sum_{j=1}^{b} (|x_{i_j} - \tilde{x}_{i_j}| M_r + \hat{x}_{i_j} \|\mathbf{z}^j - \tilde{\mathbf{z}}^j\|)^2.$$

Hence, by the Cauchy-Schwarz inequality, we get

$$\|\mathbf{w} - \tilde{\mathbf{w}}\|^2 \leq \|\mathbf{x} - \tilde{\mathbf{x}}\|^2 + \left(\frac{M_r^2}{w_r \sigma} + \frac{b}{\sigma_r} \right) \left(w_r \sigma \|\mathbf{x} - \tilde{\mathbf{x}}\|^2 + \sigma_r \sum_{j=1}^{b} \hat{x}_{i_j} \|\mathbf{z}^j - \tilde{\mathbf{z}}^j\|^2 \right)$$

$$\leq \left(\frac{(1 + M_r^2)}{w_r \sigma} + \frac{b}{\sigma_r} \right) \left(w_r \sigma \|\mathbf{x} - \tilde{\mathbf{x}}\|^2 + \sigma_r \sum_{j=1}^{b} \hat{x}_{i_j} \|\mathbf{z}^j - \tilde{\mathbf{z}}^j\|^2 \right). \tag{31}$$

Thus the bound in Lemma 4.4 can be sharpened to

$$\sigma_{r+1} = \hat{\sigma}_r \geq \frac{1}{(1 + M_r^2)/(w_r \sigma) + b/\sigma_r}. \tag{32}$$

Furthermore, in this case, $M_1^2 = kM^2$ and $M_{r+1}^2 \leq k(M^2 + bM_r^2)$, which implies

$$1 + M_r^2 \leq kM^2 (bk)^r.$$

Hence $w_r \geq (1 + M_r^2)/b^r$, and, consequently, (32) yields

$$\frac{1}{b^{r+1} \sigma_{r+1}} \leq \frac{1}{b\sigma} + \frac{1}{b^r \sigma_r}.$$

Therefore, since $\sigma_1 = \sigma$, it follows that

$$\frac{1}{b^r \sigma_r} \leq \frac{r}{b\sigma}, \quad r = 1, 2, \ldots. \tag{33}$$

On the other hand, since $D_1 = kD$ and $D_{r+1} \leq k(w_r D + bD_r)$, it follows that

$$
\begin{aligned}
D_r &\leq kD\left((bk)^{r-1} + \sum_{j=1}^{r-1} w_j (bk)^{r-1-j} \right) \\
&= kD\left((bk)^{r-1} + kM^2 \sum_{j=1}^{r-1} k^j (bk)^{r-1-j} \right) \\
&\leq kD(1 + kM^2(r-1))(bk)^{r-1} \\
&\leq k^2 r D M^2 (bk)^{r-1}.
\end{aligned}
\tag{34}
$$

Finally, (30) follows by putting together (33) and (34). $\quad\square$

5. Implementation. In this section, we describe an implementation to solve (1) based on Nesterov's [16] *excessive gap technique* (EGT) and the prox-functions constructed in this paper. We present Nesterov's [16] algorithm specialized for problem (1). We also give a complexity analysis of each iteration of this algorithm when applied to games with uniform treeplexes, and describe two heuristics that were incorporated in our implementation.

5.1. Nesterov's [16] EGT. Assume $d_{\mathscr{X}}$ and $d_{\mathscr{Y}}$ are nice prox-functions for $\mathscr{X}$ and $\mathscr{Y}$, respectively. For $\mu_{\mathscr{X}}$, $\mu_{\mathscr{Y}} > 0$, consider the pair of problems:

$$
f_{\mu_{\mathscr{Y}}}(\mathbf{x}) := \max\{\langle \mathbf{y}, A\mathbf{x} \rangle - \mu_{\mathscr{Y}} d_{\mathscr{Y}}(\mathbf{y}) \colon \mathbf{y} \in \mathscr{Y}\}, \qquad \phi_{\mu_{\mathscr{X}}}(\mathbf{y}) := \min\{\langle \mathbf{y}, A\mathbf{x} \rangle + \mu_{\mathscr{X}} d_Q(\mathbf{x}) \colon \mathbf{x} \in \mathscr{X}\}.
$$

Algorithm 3 below, because of Nesterov [16, §5], generates iterates $(\mathbf{x}^k, \mathbf{y}^k, \mu_{\mathscr{X}}^k, \mu_{\mathscr{Y}}^k)$ with $\mu_{\mathscr{X}}^k$, $\mu_{\mathscr{Y}}^k$ decreasing to zero and such that the following *excessive gap condition* is satisfied at each iteration:

$$
f_{\mu_{\mathscr{Y}}}(\mathbf{x}) \leq \phi_{\mu_{\mathscr{X}}}(\mathbf{y}).
\tag{35}
$$

Notice that $f(\mathbf{x}) \geq \phi(\mathbf{y})$ for all $\mathbf{x} \in \mathscr{X}$, $\mathbf{y} \in \mathscr{Y}$. Thus if $(\mathbf{x}, \mathbf{y}, \mu_{\mathscr{X}}, \mu_{\mathscr{Y}})$ satisfy the excessive gap condition (35) and $\mathbf{x} \in \mathscr{X}$, $\mathbf{y} \in \mathscr{Y}$, then

$$
0 \leq \phi(\mathbf{y}) - f(\mathbf{x}) \leq \mu_{\mathscr{X}} D_{\mathscr{X}} + \mu_{\mathscr{Y}} D_{\mathscr{Y}}.
\tag{36}
$$

(See Nesterov [16, Lemma 3.1].)

Consequently, if the iterates $(\mathbf{x}^k, \mathbf{y}^k, \mu_{\mathscr{X}}^k, \mu_{\mathscr{Y}}^k)$ satisfy (35), then $f(\mathbf{x}^k) \approx \phi(\mathbf{y}^k)$ when $\mu_{\mathscr{X}}^k$ and $\mu_{\mathscr{Y}}^k$ are small.

The building blocks of our Algorithm 3 are the procedures initial and shrink defined next.

By Lemma 5.1 of Nesterov [16], the following procedure initial finds a starting point $(\mu_{\mathscr{X}}^0, \mu_{\mathscr{Y}}^0, \mathbf{x}^0, \mathbf{y}^0)$ that satisfies the excessive gap condition (35).

Algorithm 1 (initial($A, d_{\mathscr{X}}, d_{\mathscr{Y}}$))

  (i) $\mu_{\mathscr{X}}^0 := \mu_{\mathscr{Y}}^0 := \dfrac{\|A\|}{\sqrt{\sigma_{\mathscr{X}} \sigma_{\mathscr{Y}}}}$

  (ii) $\hat{\mathbf{x}} := \nabla d_{\mathscr{X}}^*(\mathbf{0})$

  (iii) $\mathbf{y}^0 := \nabla d_{\mathscr{Y}}^*\left(\dfrac{1}{\mu_{\mathscr{Y}}^0} A\hat{\mathbf{x}} \right)$

  (iv) $\mathbf{x}^0 := \nabla d_{\mathscr{X}}^*\left(\nabla d_{\mathscr{X}}(\hat{\mathbf{x}}) + \dfrac{1}{\mu_{\mathscr{X}}^0} A^{\mathrm{T}} \mathbf{y}^0 \right)$

  (v) Return $(\mu_{\mathscr{X}}^0, \mu_{\mathscr{Y}}^0, \mathbf{x}^0, \mathbf{y}^0)$

The following procedure shrink enables us to reduce $\mu_{\mathscr{X}}$ and $\mu_{\mathscr{Y}}$, while maintaining (35).

Algorithm 2 (shrink($A, \mu_{\mathscr{X}}, \mu_{\mathscr{Y}}, \tau, \mathbf{x}, \mathbf{y}, d_{\mathscr{X}}, d_{\mathscr{Y}}$))

  (i) $\check{\mathbf{x}} := \nabla d_{\mathscr{X}}^*\left(-\dfrac{1}{\mu_{\mathscr{X}}} A^{\mathrm{T}} \mathbf{y} \right)$

  (ii) $\hat{\mathbf{x}} := (1 - \tau)\mathbf{x} + \tau\check{\mathbf{x}}$

  (iii) $\hat{\mathbf{y}} := \nabla d_{\mathscr{Y}}^*\left(\dfrac{1}{\mu_{\mathscr{Y}}} A\hat{\mathbf{x}} \right)$

  (iv) $\tilde{\mathbf{x}} := \nabla d_{\mathscr{X}}^*\left(\nabla d_{\mathscr{X}}(\check{\mathbf{x}}) - \dfrac{\tau}{(1-\tau)\mu_{\mathscr{X}}} A^{\mathrm{T}} \hat{\mathbf{y}} \right)$

(v) $\mathbf{y}^+ := (1 - \tau)\mathbf{y} + \tau\hat{\mathbf{y}}$

(vi) $\mathbf{x}^+ := (1 - \tau)\mathbf{x} + \tau\tilde{\mathbf{x}}$

(vii) $\mu_{\mathscr{X}}^+ := (1 - \tau)\mu_{\mathscr{X}}$

(viii) Return $(\mu_{\mathscr{X}}^+, \mathbf{x}^+, \mathbf{y}^+)$

By Theorem 5.2 of Nesterov [16], if the input $(\mu_{\mathscr{X}}, \mu_{\mathscr{Y}}, \mathbf{x}, \mathbf{y})$ to shrink satisfies (35), then so does $(\mu_{\mathscr{X}}^+, \mu_{\mathscr{Y}}, \mathbf{x}^+, \mathbf{y}^+)$, as long as τ satisfies $\tau^2/(1 - \tau) \leq \mu_{\mathscr{X}}\mu_{\mathscr{Y}}\sigma_{\mathscr{X}}\sigma_{\mathscr{Y}}/\|A\|^2$.

We are now ready to describe Nesterov's [16] EGT specialized to (1).

Algorithm 3 ($\texttt{EGT}(A, d_{\mathscr{X}}, d_{\mathscr{Y}})$)

(i) $(\mu_{\mathscr{X}}^0, \mu_{\mathscr{Y}}^0, \mathbf{x}^0, \mathbf{y}^0) = \texttt{initial}(A, d_{\mathscr{X}}, d_{\mathscr{Y}})$

(ii) For $k = 0, 1, \ldots,$

    (a) $\tau := \dfrac{2}{k+3}$

    (b) If k is even,   // shrink $\mu_{\mathscr{X}}$

        i. $(\mu_{\mathscr{X}}^{k+1}, \mathbf{x}^{k+1}, \mathbf{y}^{k+1}) := \texttt{shrink}(A, \mu_{\mathscr{X}}^k, \mu_{\mathscr{Y}}^k, \tau, \mathbf{x}^k, \mathbf{y}^k, d_{\mathscr{X}}, d_{\mathscr{Y}})$

        ii. $\mu_{\mathscr{Y}}^{k+1} := \mu_{\mathscr{Y}}^k$

    (c) If k is odd,   // shrink $\mu_{\mathscr{Y}}$

        i. $(\mu_{\mathscr{Y}}^{k+1}, \mathbf{y}^{k+1}, \mathbf{x}^{k+1}) := \texttt{shrink}(-A^{\mathrm{T}}, \mu_{\mathscr{Y}}^k, \mu_{\mathscr{X}}^k, \tau, \mathbf{y}^k, \mathbf{x}^k, d_{\mathscr{Y}}, d_{\mathscr{X}})$

        ii. $\mu_{\mathscr{X}}^{k+1} := \mu_{\mathscr{X}}^k$

By Nesterov [16, Theorem 5.2], the iterates generated by procedure EGT satisfy (35). In addition, by Nesterov [16, Theorem 6.3], after N iterations, Algorithm EGT yields points $\mathbf{x}^N \in Q_{\mathscr{X}}$ and $\mathbf{y}^N \in Q_{\mathscr{Y}}$ with

$$0 \leq \max_{y \in Q_{\mathscr{Y}}}\langle A\mathbf{x}^N, \mathbf{y}\rangle - \min_{x \in Q_{\mathscr{X}}}\langle A\mathbf{x}, \mathbf{y}^N\rangle \leq \frac{4\|A\|}{N}\sqrt{\frac{D_{\mathscr{X}}D_{\mathscr{Y}}}{\sigma_{\mathscr{X}}\sigma_{\mathscr{Y}}}}. \tag{37}$$

5.2. Complexity of each EGT iteration. We next give a complexity bound on the number of arithmetic operations performed in each EGT iteration. We provide our estimate in terms of the size of the *game tree* in the *extensive form* representation of the sequential game. The extensive form is a full description of the game given by a tree whose nodes correspond to the possible states of the game, branches that correspond to players' moves, payoffs at the tree's leaves, and information sets. For a detailed exposition on the extensive form representation, see, e.g., Osborne and Rubinstein [18].

We shall refer to the number of leaves in the game tree as the *size of the game tree*. We show next that for games with uniform treeplexes, the total number of basic arithmetic operations in each EGT iteration is linear in the size of the game tree. To that end, notice that aside from negligible updates, two consecutive iterations in the EGT algorithm require the following operations:

(i) three matrix-vector products of the form $A\mathbf{x}$ and three of the form $A^{\mathrm{T}}\mathbf{y}$ for some $\mathbf{x}$ and $\mathbf{y}$,

(ii) one calculation of the form $\nabla d_{\mathscr{X}}(\mathbf{x})$ and one of the form $\nabla d_{\mathscr{Y}}(\mathbf{y})$ for some $\mathbf{x}$ and $\mathbf{y}$,

(iii) three calculations of the form $\nabla d_{\mathscr{X}}^*(\mathbf{u})$ and three of the form $\nabla d_{\mathscr{Y}}^*(\mathbf{v})$ for some $\mathbf{u}$ and $\mathbf{v}$.

Hence it suffices to show that each of these operations requires a number of basic arithmetic operations that is linear in the size of the game tree.

Let $\texttt{flops}(\langle expression\rangle)$ denote the number of arithmetic operations needed in the calculation of $\langle expression\rangle$. We next estimate this number for each of the calculations in (i), (ii), and (iii) above.

For (i), if the payoff matrix A is represented in explicit sparse form, then $\texttt{flops}(A\mathbf{x})$ and $\texttt{flops}(A^{\mathrm{T}}\mathbf{y})$ are less than or equal to twice the number of nonzero entries in A, because each of these calculations requires one scalar multiplication and at most one addition for each nonzero in A. Since the number of nonzero entries in A is bounded by the number of leaves in the game tree (von Stengel [21, 22]), it follows that $\texttt{flops}(A\mathbf{x})$ and $\texttt{flops}(A^{\mathrm{T}}\mathbf{y})$ are linear in the size of the game tree.

For the calculations in (ii), assume $\mathscr{X} = \mathbb{Q}(Q, I, k, r) \subseteq \mathbb{R}^m$ and $\mathscr{Y} = \mathbb{Q}(\tilde{Q}, \tilde{I}, \tilde{k}, \tilde{r}) \subseteq \mathbb{R}^n$. The construction of the uniform treeplex $\mathbb{Q}(Q, I, k, r)$ and a straightforward induction argument shows that for generic $\mathbf{x} \in \mathscr{X}$, $\mathbf{z} \in Q$,

$$\texttt{flops}(\nabla d_{\mathscr{X}}(\mathbf{x})) = \frac{(bk)^r - 1}{bk - 1} \cdot k \cdot \texttt{flops}(\nabla d_Q(\mathbf{z})) + m \leq m \cdot (\texttt{flops}(\nabla d_Q(\mathbf{z})) + 1).$$

Likewise, for generic $\mathbf{y} \in \mathscr{Y}$, $\mathbf{w} \in \tilde{Q}$,

$$\texttt{flops}(\nabla d_{\mathscr{Y}}(\mathbf{y})) \leq n \cdot (\texttt{flops}(\nabla d_{\tilde{Q}}(\mathbf{w})) + 1).$$

Since both m and n are smaller than the size of the game tree (von Stengel [21, 22]), it follows that flops($\nabla d_{\mathscr{X}}(\mathbf{x})$) and flops($\nabla d_{\mathscr{Y}}(\mathbf{y})$) are sublinear in the size of the game tree.

Finally, for the calculations in (iii), again assume $\mathscr{X} = \mathcal{Q}(Q, I, k, r) \subseteq \mathbb{R}^m$ and $\mathscr{Y} = \mathcal{Q}(\tilde{Q}, \tilde{I}, \tilde{k}, \tilde{r}) \subseteq \mathbb{R}^n$. An inductive argument similar to those in §4 shows that for generic $\mathbf{u} \in \mathbb{R}^m$, $\mathbf{s} \in \mathbb{R}^q$

$$\text{flops}(\nabla d_{\mathscr{X}}^*(\mathbf{u})) \leq m \cdot (\text{flops}(\nabla d_Q^*(\mathbf{s})) + 1),$$

and for generic $\mathbf{v} \in \mathbb{R}^n$, $\mathbf{t} \in \mathbb{R}^{\tilde{q}}$

$$\text{flops}(\nabla d_{\mathscr{Y}}^*(\mathbf{v})) \leq n \cdot (\text{flops}(\nabla d_{\tilde{Q}}^*(\mathbf{t})) + 1).$$

Thus both flops($\nabla d_{\mathscr{X}}^*(\mathbf{u})$) and flops($\nabla d_{\mathscr{Y}}^*(\mathbf{v})$) are sublinear in the size of the game tree.

Consequently, the overall number of arithmetic operations in each iteration of the EGT algorithm is bounded by a small factor of the size of the game tree. Furthermore, the matrix-vector multiplications $A\mathbf{x}$ and $A^T\mathbf{y}$ dominate the total number of arithmetic operations.

5.3. Heuristics. Algorithm EGT has worst-case iteration-complexity $\mathcal{O}(1/\epsilon)$ and already scales to problems much larger than is possible to solve using state-of-the-art linear programming solvers (as we demonstrate in the experiments later in this paper). In this section, we introduce two heuristics for further improving the speed of the algorithm, while retaining the guaranteed worst-case iteration-complexity $\mathcal{O}(1/\epsilon)$. The heuristics attempt to decrease $\mu_{\mathscr{X}}$ and $\mu_{\mathscr{Y}}$ faster than prescribed by the EGT algorithm, while maintaining the excessive gap condition (35). This leads to overall faster convergence in practice, as our experiments will show.

5.3.1. Heuristic 1: Aggressive μ reduction. The first heuristic is based on the following observation: although the value $\tau = 2/(k + 3)$ computed in step 2(a) of Algorithm EGT guarantees the excessive gap condition (35), this is potentially an overly conservative value. 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 (35) a priori, we are required to do a posterior verification, which occasionally necessitates an adjustment in the parameter τ. To check (35), we need to compute the values of $f_{\mu_{\mathscr{Y}}}$ and $\phi_{\mu_{\mathscr{X}}}$. Observe that

$$\phi_{\mu_{\mathscr{X}}}(\mathbf{y}) = -\mu_{\mathscr{X}} d_{\mathscr{X}}^*\left(-\frac{1}{\mu_{\mathscr{X}}} A^T \mathbf{y}\right)$$

and

$$f_{\mu_{\mathscr{Y}}}(\mathbf{x}) = \mu_{\mathscr{Y}} d_{\mathscr{Y}}^*\left(\frac{1}{\mu_{\mathscr{Y}}} A\mathbf{x}\right).$$

Therefore, both $f_{\mu_{\mathscr{Y}}}$ and $\phi_{\mu_{\mathscr{X}}}$ are easily computable since $d_{\mathscr{X}}, d_{\mathscr{Y}}$ are nice prox-functions by construction.

To incorporate Heuristic 1 in Algorithm EGT, we extend the procedure shrink as follows.

Algorithm 4 (decrease($A, \mu_{\mathscr{X}}, \mu_{\mathscr{Y}}, \tau, \mathbf{x}, \mathbf{y}, d_{\mathscr{X}}, d_{\mathscr{Y}}$))

(i) $(\mu_{\mathscr{X}}^+, \mathbf{x}^+, \mathbf{y}^+) := \text{shrink}(A, \mu_{\mathscr{X}}, \mu_{\mathscr{Y}}, \tau, \mathbf{x}, \mathbf{y}, d_{\mathscr{X}}, d_{\mathscr{Y}})$

(ii) While $-\mu_{\mathscr{X}}^+ d_{\mathscr{X}}^*\left(-\dfrac{1}{\mu_{\mathscr{X}}^+} A^T \mathbf{y}^+\right) < \mu_{\mathscr{Y}} d_{\mathscr{Y}}^*\left(\dfrac{1}{\mu_{\mathscr{Y}}} A\mathbf{x}^+\right),$ // τ is too big

   (a) $\tau := \tau/2$

   (b) $(\mu_{\mathscr{X}}^+, \mathbf{x}^+, \mathbf{y}^+) := \text{shrink}(A, \mu_{\mathscr{X}}, \mu_{\mathscr{Y}}, \tau, \mathbf{x}, \mathbf{y}, d_{\mathscr{X}}, d_{\mathscr{Y}})$

(iii) Return $(\mu_{\mathscr{X}}^+, \mathbf{x}^+, \mathbf{y}^+, \tau)$

By Theorem 4.1 of Nesterov [16], when the input $(\mu_{\mathscr{X}}, \mu_{\mathscr{Y}}, \mathbf{x}, \mathbf{y})$ to decrease satisfies (35), the procedure decrease will halt.

5.3.2. Heuristic 2: Balancing and reduction of $\mu_{\mathscr{X}}$ and $\mu_{\mathscr{Y}}$. Our second heuristic is motivated by the observation that after several calls to the decrease procedure, one of $\mu_{\mathscr{X}}$ and $\mu_{\mathscr{Y}}$ may be much smaller than the other. This imbalance is undesirable because the larger one contributes the most to the worst-case bound given by (36). 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 $\mu_{\mathscr{X}}$ and $\mu_{\mathscr{Y}}$.

We also observed that after such balancing, the values of $\mu_{\mathscr{X}}$ and $\mu_{\mathscr{Y}}$ can sometimes be further reduced without violating the excessive gap condition (35). We thus include a final reduction step in the balancing heuristic.

This balancing and reduction heuristic is incorporated via the following procedure. (We chose the parameter values 0.9 and 1.5 based on some initial experimentation.)

Algorithm 5 ($\mathtt{balance}(A, \mu_{\mathscr{X}}, \mu_{\mathscr{Y}}, \tau, \mathbf{x}, \mathbf{y}, d_{\mathscr{X}}, d_{\mathscr{Y}})$)

    (i) While $\mu_{\mathscr{X}} > 1.5\mu_{\mathscr{Y}}$,   // shrink $\mu_{\mathscr{X}}$
       $(\mu_{\mathscr{X}}, \mathbf{x}, \mathbf{y}, \tau) := \mathtt{decrease}(A, \mu_{\mathscr{X}}, \mu_{\mathscr{Y}}, \tau, \mathbf{x}, \mathbf{y}, d_{\mathscr{X}}, d_{\mathscr{Y}})$

    (ii) While $\mu_{\mathscr{Y}} > 1.5\mu_{\mathscr{X}}$,   // shrink $\mu_{\mathscr{Y}}$
       $(\mu_{\mathscr{Y}}, \mathbf{y}, \mathbf{x}, \tau) := \mathtt{decrease}(-A^{\mathrm{T}}, \mu_{\mathscr{Y}}, \mu_{\mathscr{X}}, \tau, \mathbf{y}, \mathbf{x}, d_{\mathscr{Y}}, d_{\mathscr{X}})$

    (iii) While $0.9\mu_{\mathscr{Y}} d_{\mathscr{Y}}^* \left(\dfrac{1}{0.9\mu_{\mathscr{Y}}} A\mathbf{x} \right) \leq -0.9\mu_{\mathscr{X}} d_{\mathscr{X}}^* \left(-\dfrac{1}{0.9\mu_{\mathscr{X}}} A^{\mathrm{T}}\mathbf{y} \right),$
       // decrease $\mu_{\mathscr{X}}$ and $\mu_{\mathscr{Y}}$ if possible
      (a) $\mu_{\mathscr{X}} := 0.9\mu_{\mathscr{X}}$
      (b) $\mu_{\mathscr{Y}} := 0.9\mu_{\mathscr{Y}}$

    (iv) Return $(\mu_{\mathscr{X}}, \mu_{\mathscr{Y}}, \mathbf{x}, \mathbf{y}, \tau)$

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

Algorithm 6 (EGT-2)

    (i) $(\mu_{\mathscr{X}}^0, \mu_{\mathscr{Y}}^0, \mathbf{x}^0, \mathbf{y}^0) = \mathtt{initial}(A, d_{\mathscr{X}}, d_{\mathscr{Y}})$
    (ii) $\tau := 0.5$
    (iii) For $k = 0, 1, \ldots,$
      (a) If k is even, // shrink $\mu_{\mathscr{X}}$
        i. $(\mu_{\mathscr{X}}^{k+1}, \mathbf{x}^{k+1}, \mathbf{y}^{k+1}, \tau) := \mathtt{decrease}(A, \mu_{\mathscr{X}}^k, \mu_{\mathscr{Y}}^k, \tau, \mathbf{x}^k, \mathbf{y}^k, d_{\mathscr{X}}, d_{\mathscr{Y}})$
        ii. $\mu_{\mathscr{Y}}^{k+1} = \mu_{\mathscr{Y}}^k$
      (b) If k is odd, // shrink $\mu_{\mathscr{Y}}$
        i. $(\mu_{\mathscr{Y}}^{k+1}, \mathbf{y}^{k+1}, \mathbf{x}^{k+1}, \tau) := \mathtt{decrease}(-A^{\mathrm{T}}, \mu_{\mathscr{Y}}^k, \mu_{\mathscr{X}}^k, \tau, \mathbf{y}^k, \mathbf{x}^k, d_{\mathscr{Y}}, d_{\mathscr{X}})$
        ii. $\mu_{\mathscr{X}}^{k+1} = \mu_{\mathscr{X}}^k$
      (c) If $k(\mathrm{mod})100 = 0$, // balance and reduce
        $(\mu_{\mathscr{X}}^k, \mu_{\mathscr{Y}}^k, \mathbf{x}^k, \mathbf{y}^k, \tau) := \mathtt{balance}(A, \mu_{\mathscr{X}}^k, \mu_{\mathscr{Y}}^k, \tau, \mathbf{x}^k, \mathbf{y}^k, d_{\mathscr{X}}, d_{\mathscr{Y}})$

6. Computational results. We implemented Algorithm EGT-2 in C++ and ran the computational experiments on an IBM eServer p5 570 with 128 GBs of RAM and four 1.65 GHz processors. We next report some computational experiments as well as an interesting application to the design of poker-playing programs.

6.1. Experimental setup. We tested the algorithm on five abstractions of poker games ranging from relatively small to very large. An *abstraction* of a game is a smaller game that captures some of the main features of the original game (Billings et al. [2]; Gilpin and Sandholm [4, 5]; Shi and Littman [20]). The approach of abstracting a game and then solving for the equilibrium of the abstracted game is a practical way of constructing good strategies for the original game (Billings et al. [2], Gilpin and Sandholm [4, 5]; Gilpin et al. [6, 7]), and is the state-of-the-art approach to generating poker-playing programs.

We chose these 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). Table 1 provides the sizes of the test instances. The first three instances, 10k, 160k, and RI, are abstractions of Rhode Island Hold'em poker (Shi and Littman [20]) computed using the *GameShrink* automated abstraction algorithm (Gilpin and Sandholm [5]). The first two instances are lossy (nonequilibrium preserving) abstractions, while the RI instance is a lossless abstraction. The Texas and GS4 instances are lossy abstractions of limit Texas Hold'em poker (Gilpin [3], Gilpin et al. [6]).

Table 2 provides the average time per EGT iteration of our implementation for each of the test problems both with and without the heuristics.

TABLE 1. Problem sizes (when formulated as a linear program) for the instances used in our experiments.

Name	Rows	Columns	Nonzero entries
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
GS4	299,477,082	299,477,102	4,105,365,178,571

TABLE 2.   Average CPU time per EGT iteration for the instances used in our experiments.

Name	EGT with heuristics (time in secs)	EGT without heuristics (time in secs)
10k	0.10	0.10
160k	1.28	1.20
RI	7.65	6.53
Texas	2,400	1,420
GS4	42,400	28,000

Because of the enormous size of the GS4 instance, we do not include it in the experiments that compare better and worse techniques within our algorithm. Instead, we use the four smaller instances to find a good configuration of the algorithm, and we use that configuration to tackle the GS4 instance. We then report on how well the resulting strategies on the GS4 instance did in the Association for the Advancement of Artificial Intelligence AAAI-08 Computer Poker Competition.

Previously, the most effective algorithms for solving sequential games of imperfect information were based on IPMs applied to the linear programming formulation of the problem (Billings et al. [2], Gilpin and Sandholm [5]). It seems desirable to test our algorithm against state-of-the-art implementations of such methods. However, this is not particularly relevant in the context of the problems we are solving. For example, solving the relatively small game of Rhode Island Hold'em poker required 25 GB RAM using CPLEX's IPM. The instance GS4 is more than 200 times larger. Simply *representing* such a problem in the explicit representation required by CPLEX and other interior-point solvers would require more than 80,000 GB RAM. The memory needed for the necessary data structures, such as storing the Cholesky factorization, would increase this further. Such a requirement is far beyond the capability of current hardware. Thus, it is not even possible to compare the runtime performance of our algorithm with linear programming approaches.

6.2. Experimental comparison of prox-functions.   Our first experiment compared the relative performance of the prox-functions induced by the entropy and Euclidean prox-functions described in Example 1 earlier in this paper. Figure 1 shows the results. (Heuristics 1 and 2, described above, and the memory-saving technique
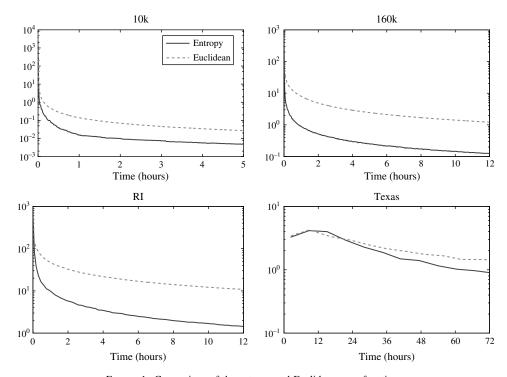

FIGURE 1. Comparison of the entropy and Euclidean prox-functions.
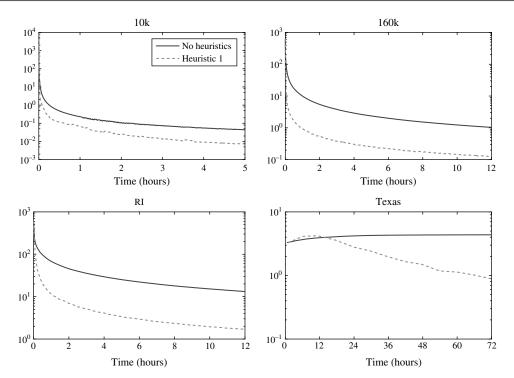*Note.* The value axis is the gap ϵ (Equation (2)).

FIGURE 2. Experimental evaluation of Heuristic 1.

Note. The value axis is the gap ϵ (Equation (2)).

described later, were enabled in this experiment.) In all of the figures, the units of the vertical axis are the number of chips in the corresponding poker games.

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

6.3. Experimental comparison of the heuristics.   Figure 2 demonstrates the impact of applying Heuristic 1: *Aggressive μ reduction.* (For this experiment, Heuristic 2 was not used. The memory-saving technique, also described later, was used.) On all four instances, Heuristic 1 reduced the gap significantly. On the larger instances, this reduction was an order of magnitude.

Figure 3 demonstrates the impact of applying Heuristic 2: *Balancing and reduction of $\mu_{\mathscr{X}}$ and $\mu_{\mathscr{Y}}$.* Because Heuristic 2 is somewhat expensive to apply, 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. For example, when the figure states "10 iterations," that means that the heuristic is run once every 10 iterations. In this experiment, Heuristic 1 was turned off, but the memory-saving technique, described later, was used.) Figure 3 shows that it is always effective to use Heuristic 2, although the frequency at which it should be applied varies depending on the instance.

6.4. Application to Texas Hold'em poker.   Poker is a game involving elements of chance, imperfect information, and counterspeculation. Game-theoretic optimal strategies are far from straightforward, often necessitating such tactics as bluffing and slow playing. For these reasons, and others, poker has been identified as an important challenge problem for the field of artificial intelligence (Billings et al. [1]). Just as the development of a computer program capable of beating the world's best human chess player was once seen as an important milestone, the development of a poker-playing program capable of beating the best humans is now seen as an equally important milestone.

The prox-function construction described in §3 has been instrumental in the development of some recent programs for playing Texas Hold'em poker. An important difference between different variants of Texas Hold'em is the *betting structure*. Two common betting structures are *limit*, in which players may bet a fixed amount, and *no-limit*, in which players may bet any number of their chips. Our equilibrium-finding algorithm computed the strategies for both *GS*3 (Gilpin et al. [6]) and *Tartanian* (Gilpin et al. [7]), two programs that play limit and no-limit Texas Hold'em, respectively.

In 2008, the AAAI held the third annual Computer Poker Competition, where computer programs submitted by teams worldwide compete against each other. *GS*4-*Beta* (a subsequent version of *GS*3) placed first (out of
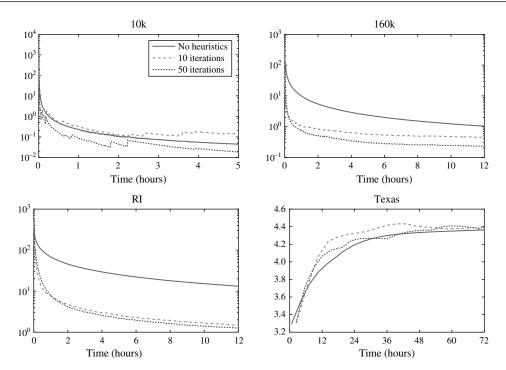
FIGURE 3. Heuristic 2 applied at different intervals.

Note. The value axis is the gap ϵ (Equation (2)).

nine) in the Limit Bankroll competition and *Tartanian* placed third (out of four) in the no-limit competition. (*Tartanian* actually had the highest winning rate in the competition, but because of the winner determination rule for the competition, it only got third place.) This is particularly impressive given the small amount of poker-specific knowledge that was incorporated into those programs. They instead depend on an equilibrium analysis conducted by our algorithm (which, in turn, relies on our prox-function construction) for determining their strategies. As the developers of *GS*3 and *Tartanian* point out, it is currently not feasible to solve their models using off-the-shelf linear programming solvers.

The approach used for constructing the above players is based on algorithmically creating *lossy* abstractions of the original game (Gilpin [3]; Gilpin et al. [6, 7]). These abstractions are smaller sequential games that attempt to preserve the strategic properties of the original game. The abstracted game is then solved for an ϵ-equilibrium using the algorithm discussed in this paper. The larger the abstracted game (i.e., the finer the abstraction), the better the quality of the strategies generally is. The approach of automated abstraction followed by equilibrium finding was first used in Texas Hold'em in Gilpin and Sandholm [4], and is nowadays used by basically all of the competitive poker-playing programs.

For the limit competition, our implementation of the EGT algorithm solved an abstracted game whose payoff matrix was $10^8 \times 10^8$. For the no-limit competition, our algorithm solved a game with payoff matrix of size $10^7 \times 10^7$. The uniform treeplexes introduced in §4 provide a perfect framework for modeling limit Texas Hold'em poker. For this game, the treeplex $Q_{\mathscr{K}}$ for the first player is a uniform treeplex. The "basic" treeplex $Q \subseteq [0, 1]^{14}$ has the linear description $Q = \{\mathbf{x} \in [0, 1]^{14} : E\mathbf{x} = \mathbf{e}\}$, where

$$
E := \begin{bmatrix}
1 & 1 & 1 & & & & & & & & & & & \\
& -1 & & 1 & 1 & 1 & & & & & & & & \\
& & & & -1 & 1 & 1 & 1 & & & & & & \\
& -1 & & & & & & & 1 & 1 & 1 & & & \\
& & & & & & & & & -1 & 1 & 1 &
\end{bmatrix}, \qquad
\mathbf{e} := \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.
$$

The 14 columns of E represent the possible sequence of actions that the first player can take during each betting round of the game. Each row in E encodes a simplex over three actions: *fold*, *call*, and *raise*. (The last row only allows fold and call.) The set $I = \{2, 3, 5, 6, 8, 9, 11, 12, 14\}$ indexes the sequences that do not end with a fold. Texas Hold'em is played in four rounds so $r = 4$. Finally, the value of k depends on the quality of the

abstraction. The abstractions in Gilpin et al. [6] range from $k = 6$ to $k = 40$ (the k is actually different in each round). The treeplex $Q_{\mathcal{Y}}$ for the second player is also a uniform treeplex with similar characteristics.

6.5. Memory requirements. One particularly attractive feature of the EGT algorithm is the fact that the only operation performed on the matrix A is a matrix-vector product. As a consequence, we can exploit the problem structure to store only an *implicit* representation of the payoff matrix A. This implicit representation relies on a certain type of decomposition that is present in poker games as well as in the more general class of *games with ordered signals* (Gilpin [3], Gilpin and Sandholm [5]). For example, the betting sequences that can occur in most poker games are independent of the cards that are dealt. We can decompose the payoff matrix based on these two aspects.[1]

For ease of exposition, we explain the concise representation in the context of Rhode Island Hold'em poker (Shi and Littman [20]), although the general technique applies much more broadly (and we use it in our Texas Hold'em games as well). The payoff matrix A can be written as

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

where

$$
\begin{aligned}
A_1 &= F_1 \otimes B_1, \\
A_2 &= F_2 \otimes B_2, \qquad \text{and} \\
A_3 &= F_3 \otimes B_3 + S \otimes W
\end{aligned}
\tag{38}
$$

for much smaller matrices F_i, B_i, S, and W. The matrices F_i correspond to sequences of moves in round i that end with a fold, and S corresponds to the sequences in round 3 that end in a showdown. The matrices B_i encode the betting structures in round i, while W encodes the win/lose/draw information determined by poker hand ranks. The symbol $\otimes$ in (38) denotes the *Kronecker product*. Recall that the Kronecker product of two matrices, $B \in \mathbb{R}^{m \times n}$ and $C \in \mathbb{R}^{p \times q}$, is

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

Given the above concise representation of A, computing $\mathbf{x} \mapsto A\mathbf{x}$ and $\mathbf{y} \mapsto A^{\mathrm{T}}\mathbf{y}$ 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_1 and F_2 matrices are 10×10 and 70×70, respectively. The dimension of the F_3 and S matrices are 490×490. The dimensions of B_1, B_2, and B_3 are 13×13, 205×205, and $1{,}774 \times 1{,}774$, respectively. By contrast, the matrix A is $883{,}741 \times 883{,}741$. Furthermore, the matrices F_i, B_i, S, and W are themselves sparse, which allows us to use the compressed row storage data structure that only stores nonzero entries.

Table 3 clearly demonstrates the extremely low memory requirements of the EGT algorithms when using our memory-saving technique. Most notably, on the GS4 instance, both of the CPLEX algorithms (simplex and interior point) require more than 80,000 GB simply to *represent* the problem. In contrast, using the decomposed payoff matrix representation, the EGT algorithms require only 43.96 GB. Furthermore, to solve the problem, both the simplex and interior-point algorithms would require additional memory for their internal data structures. Therefore the EGT family of algorithms with our memory-saving techniques is a significant improvement over the state-of-the-art for large-scale problems.

The memory use for the CPLEX simplex algorithm reported in Table 3 is the memory used after 10 minutes of execution (except for the Texas and GS4 instances, which could not run at all using either CPLEX algorithm). 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 use during execution.

Although the results presented in Table 3 are for CPLEX, they apply to any algorithm that requires an explicit representation of the constraint matrix of the linear program. Because the only matrix operation needed by our algorithm is a matrix-vector product, we are able to use an implicit representation of the constraint matrix, as discussed above.

---

[1] The fact that possible betting sequences are independent of cards has also been exploited by automated abstraction algorithms, but in a totally different way (Gilpin and Sandholm [5]).

TABLE 3. Memory footprint in gigabytes of CPLEX IPM, CPLEX simplex, and our EGT algorithms.

Name	CPLEX IPM	CPLEX simplex	EGT
10k	0.082 GB	>0.051 GB	0.012 GB
160k	2.250 GB	>0.664 GB	0.035 GB
RI	25.200 GB	>3.450 GB	0.150 GB
Texas	>458 GB	>458 GB	2.490 GB
GS4	>80,000 GB	>80,000 GB	43.96 GB

6.6. Speedup from parallelizing the matrix-vector product. Beyond our time-saving heuristics discussed earlier in this paper, we further reduce the time requirements of the matrix-vector product by parallelization. We 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 4. 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 scales nearly linearly, decreasing by a factor of 3.69 when using four CPUs.

7. Conclusions and future research. We developed first-order algorithms to approximate Nash equilibria of two-person zero-sum sequential games by applying Nesterov's [16, 17] smoothing technique to the saddle-point formulation (1) of the Nash equilibrium problem. The heart of our approach is a construction of nice prox-functions for the treeplex polytopes in the saddle-point formulation.

We implemented an algorithm based on our prox-functions and Nesterov's [16] EGT. We included two novel heuristics that improve the algorithm's speed of convergence considerably. Experiments show that the algorithm based on the entropy-induced prox-function is faster than the algorithm based on the Euclidean-induced prox-function. For poker games and similar games, we introduced a decomposed matrix representation that reduces storage requirements drastically. Our techniques enable us to solve to near-equilibrium games that are more than four orders of magnitude larger than the largest addressable previously. We also showed near-perfect speedup from parallelization, which makes our algorithms particularly appropriate for modern multicore architectures.

In contrast to a direct first-order approach to solve the linear programming formulation of (1) such as that proposed in Lan et al. [12], our approach automatically yields algorithms that generate feasible strategies $\mathbf{x} \in \mathscr{X}$, $\mathbf{y} \in \mathscr{Y}$ throughout execution. This is of crucial importance because points that violate the constraints defining the treeplexes $\mathscr{X}, \mathscr{Y}$, even slightly, are typically meaningless strategies. In particular, unlike the iterates generated by our algorithm, the iterates generated by an infeasible algorithm would typically not yield approximate equilibria. Furthermore, the linear programming formulation of (1) increases the dimension of the problem substantially since it requires a new variable for each constraint in the description of the treeplexes $\mathscr{X}, \mathscr{Y}$.

In addition to our first-order smoothing approach to problem (1), it is conceivable that specialized versions of other algorithmic approaches may also lead to effective algorithms for solving the saddle-point problem (1). For example, a specialized interior-point algorithm could use an appropriately designed iterative method to solve the system of equations at each main iteration. No such approach has been successfully developed so far.

Another approach we plan to investigate is the use of *stochastic sampling* for approximating the objective function. This has already been studied in the context of matrix games (Juditsky et al. [10]), although that approach was based on a different optimization algorithm. For large-scale instances, it is quite expensive to evaluate the matrix-vector product in the objective function (and in the gradient computations). Speeding up these operations, in conjunction with strong convergence guarantees, could have a significant impact in practice. These interesting alternative algorithmic approaches will be the subject of future research.

TABLE 4. Effect of parallelization for the Texas instance.

CPUs	Matrix-vector product		EGT iteration	
	Time (secs)	Speedup	Time (secs)	Speedup
1	278	1.00x	1,420	1.00x
2	140	1.98x	730	1.94x
3	93	2.98x	490	2.89x
4	69	4.00x	384	3.69x

Soon after we published our EGT-2 algorithm in a conference (WINE-07), an algorithm based on a totally different paradigm (regret minimization at each information set of the game), but comparable performance, was published in a conference (Zinkevich et al. [23]). It would be interesting to conduct direct scalability comparisons of the two algorithms in the future. For one, we expect that our algorithm exhibits better parallelization; no convergence guarantees have been proven for the other algorithm if it is used in parallel mode where regrets on information sets are updated in parallel.

References

[1] Billings, D., L. Peña, J. Schaeffer, D. Szafron. 2002. The challenge of poker. *Artificial Intelligence* **134**(1–2, Special Issue on Games, Computers and Artificial Intelligence) 201–240.

[2] Billings, D., N. Burch, A. Davidson, R. Holte, J. Schaeffer, T. Schauenberg, D. Szafron. 2003. Approximating game-theoretic optimal strategies for full-scale poker. *Proc. 18th Internat. Joint Conf. Artificial Intelligence* (*IJCAI*), *Acapulco, Mexico.*

[3] Gilpin, A. 2009. Algorithms for abstracting and solving imperfect information games. Doctoral dissertation, Computer Science Department, Carnegie Mellon University, Pittsburgh.

[4] Gilpin, A., T. Sandholm. 2006. A competitive Texas Hold'em poker player via automated abstraction and real-time equilibrium computation. *Proc. National Conf. Artificial Intelligence* (*AAAI*), *Boston.*

[5] Gilpin, A., T. Sandholm. 2007. Lossless abstraction method for sequential games of imperfect information. *J. ACM* **54**(5) Article 25.

[6] Gilpin, A., T. Sandholm, T. B. Sørensen. 2007. Potential-aware automated abstraction of sequential games, and holistic equilibrium analysis of Texas Hold'em poker. *Proc. National Conf. Artificial Intelligence* (*AAAI*), *Vancouver.*

[7] Gilpin, A., T. Sandholm, T. B. Sørensen. 2008. A heads-up no-limit Texas Hold'em poker player: Discretized betting models and automatically generated equilibrium-finding programs. *Internat. Conf. Autonomous Agents and Multi-Agent Systems* (*AAMAS*), *Estoril, Portugal.*

[8] Goffin, J.-L. 1977. On the convergence rate of subgradient optimization methods. *Math. Programming* **13**(1) 329–347.

[9] Hirriart-Urruty, J., C. Lemaréchal. 2001. *Fundamentals of Convex Analysis*. Springer-Verlag, Berlin.

[10] Juditsky, A., G. Lan, A. Nemirovski, A. Shapiro. 2009. Stochastic approximation approach to stochastic programming. *SIAM J. Optim.* **19**(4) 1574–1609.

[11] Koller, D., N. Megiddo, B. von Stengel. 1996. Efficient computation of equilibria for extensive two-person games. *Games Econom. Behavior* **14**(2) 247–259.

[12] Lan, G., Z. Lu, R. Monteiro. 2010. Primal-dual first-order methods with $O(1/\epsilon)$ iteration-complexity for cone programming. *Math. Programming.* Forthcoming.

[13] Nemirovski, A. 2004. Prox-method with rate of convergence $O(1/t)$ for variational inequalities with Lipschitz-continuous monotone operators and smooth convex-concave saddle point problems. *SIAM J. Optim.* **15**(1) 229–251.

[14] Nesterov, Y. 1983. A method for unconstrained convex minimization problem with rate of convergence $O(1/k^2)$. *Doklady AN SSSR* (In Russian). (English translation. *Soviet Math. Dokl.*) **269** 543–547.

[15] Nesterov, Y. 2004. Introductory lectures on convex optimization: A basic course. *Applied Optimization*. Kluwer Academic Publishers, Boston.

[16] Nesterov, Y. 2005. Excessive gap technique in nonsmooth convex minimization. *SIAM J. Optim.* **16**(1) 235–249.

[17] Nesterov, Y. 2005. Smooth minimization of non-smooth functions. *Math. Programming* **103**(1) 127–152.

[18] Osborne, M., A. Rubinstein. 1994. *A Course in Game Theory*. MIT Press, Cambridge, MA.

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

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

[21] von Stengel, B. 1996. Efficient computation of behavior strategies. *Games and Econom. Behavior* **14** 220–246.

[22] von Stengel, B. 2007. Equilibrium computation for games in strategic and extensive form. N. Nisan, T. Roughgarden, E. Tardos, V. V. Vazirani, eds. *Algorithmic Game Theory*. Cambridge University Press, Cambridge, UK, 53–78.

[23] Zinkevich, M., M. Bowling, M. Johanson, C. Piccione. 2007. Regret minimization in games with incomplete information. *Annual Conf. Neural Inform. Processing Systems* (*NIPS*), *Vancouver.*