A Fast Bundle-based Anytime Algorithm for Poker and other Convex Games

H. Brendan McMahan*
Google, Inc.
4720 Forbes Avenue
Pittsburgh, PA 15213

Geoffrey J. Gordon†
Computer Science Dept.
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Convex games are a natural generalization of matrix (normal-form) games that can compactly model many strategic interactions with interesting structure. We present a new anytime algorithm for such games that leverages fast best-response oracles for both players to build a model of the overall game. This model is used to identify search directions; the algorithm then does an exact minimization in this direction via a specialized line search. We test the algorithm on a simplified version of Texas Hold'em poker represented as an extensive-form game. Our algorithm approximated the exact value of this game within $0.20 (the maximum pot size is $310.00) in a little over 2 hours, using less than 1.5GB of memory; finding a solution with comparable bounds using a state-of-the-art interior-point linear programming algorithm took over 4 days and 25GB of memory.

1 INTRODUCTION

Convex games generalize zero-sum matrix games by allowing arbitrary convex sets in place of probability simplices. This very general framework can compactly represent large games with interesting structure. For example, extensive-form games (EFGs), which model sequential decisions on a tree with random nodes and partial observability, can be represented in this way; but, so can games with other kinds of structure, including path-planning games with uncertain outcomes and adversary controlled costs, routing problems with adversary-controlled demands, and other optimization problems like computing minimax expected-size confidence regions.

*mcmahan@google.com
†ggordon@cs.cmu.edu

The representational power of convex games makes algorithms for their solution particularly important. It was shown by Koller et al. [1994] that polyhedral convex games can be solved via linear programming (their work focuses on EFGs, but the formulation holds for general convex games). Since that seminal result, the reduction to linear programming has been the state-of-the-art for solving this class of problems. For example, sophisticated game-abstraction techniques combined with linear programming only recently allowed for the exact solution of Rhode Island Hold'em poker, a simplified version of the standard game of heads up, limit Texas Hold'em. Even after the application of the equilibria-preserving abstraction, solving the corresponding linear program exactly took over 7 days of CPU time and 25 GB of memory [Gilpin and Sandholm, 2005].

Convex games often have significant structure that is not exploited by general-purpose linear programming algorithms. One way such structure can be exploited is through fast algorithms for calculating a best response strategy to a fixed strategy of the opponent. The classic fictitious play algorithm takes advantage of such oracles, and demonstrates remarkably good performance on Rhode Island Hold'em. In this paper, we develop a new algorithm for solving convex games that also uses best-response oracles and outperforms fictitious play.

This algorithm, which we call the double oracle bundle algorithm, builds up a collection of strategies (called a bundle) for each player. On each iteration it solves an approximate game where each player is only allowed to randomize among the strategies contained in his bundle. Given the optimal mixed strategies for this restricted game, it calls the oracles to find best responses for each player in the full game, and then adds these responses to the bundles to improve the approximation. The double oracle bundle method is related to the family of cutting plane and bundle algorithms for non-smooth optimization, and to Benders' decomposition in the case of polyhedra [Hiriart-Urruty

and Lemaréchal, 1993]. However, the direct application of those techniques to convex games yields algorithms that only take advantage of the best response oracle for one of the players, not both.

In the next section we review matrix games and convex games. Section 3 recalls the fictitious play algorithm and the types of best-response algorithms available to us. Section 4 reviews extensive-form games as this class of convex games has received the most attention in the literature and also because we draw our experimental testbed problems from this class. Section 5 presents our algorithm, beginning with an intuitively straightforward version and then proceeding to our full algorithm which addresses some deficiencies of the simplified version. Finally, we present experimental results in Section 6, demonstrating dramatic improvements over both fictitious play and commercial linear programming software.

2 MATRIX AND CONVEX GAMES

A zero-sum matrix (normal-form) game is played by two players, player row with strategies $R = \{1, \ldots, m\}$ and player column with strategies $C = \{1, \ldots, n\}$. A $m \times n$ matrix M specifies the payoffs, so that if row plays strategy $i \in R$ and column plays $j \in C$, the payment from row to column is the $(i, j)th$ entry of M, denoted $M(i, j)$. The players select their strategies simultaneously, without knowledge of the other player's choice.

We use $\Delta(\cdot)$ to denote the probability simplex over a finite set, so for example $\Delta(R) = \{x \in R^m \mid \sum_{i=1}^{m} x(i) = 1 \text{ and } x(i) \geq 0\}$. A mixed strategy is an element $x \in \Delta(R)$ for the row player or $y \in \Delta(C)$ for the column player, corresponding to a distribution over the rows or columns, respectively. If the players select mixed strategies x and y, it can be shown that the expected payoff $V(x, y)$ from row to column is given by the bilinear form $x^T My$. A solution to the game is a minimax equilibrium (x^*, y^*), a pair of strategies such that neither player has an incentive to play differently given that the other player plays their strategy from the pair. The minimax theorem says that if the players are allowed to select mixed strategies, there is no advantage to playing first or second:

$$\min_{x \in \Delta(R)} \max_{y \in \Delta(C)} x^T My = \max_{y \in \Delta(C)} \min_{x \in \Delta(R)} x^T My.$$

Such an (x^*, y^*) can be found via linear programming. The (minimax) value of the game is $v^* = V(x^*, y^*)$.

An ϵ-approximate minimax equilibrium for a matrix game is a pair of strategies (x', y') where neither player can gain more than ϵ value by switching to some other strategy. Formally,

$$V(x', y') \leq \min_{x \in \Delta(R)} V(x, y') + \epsilon$$
$$V(x', y') \geq \max_{y \in \Delta(C)} V(x', y) - \epsilon.$$

If $\epsilon = 0$ we have an exact minimax equilibrium.

Two-player zero-sum bilinear-payoff convex games are a natural generalization of matrix games; we will simply refer to this class as "convex games" or CGs for the sequel. This formulation was first introduced by Dresher and Karlin [1953], but convex games have received remarkably little treatment in the literature considering the generality of the framework. One of the goals of this paper is to highlight several interesting special cases of convex games, and suggest that the class deserves much greater attention.

Convex games allow arbitrary convex sets in place of the probability simplices $\Delta(R)$ and $\Delta(C)$ of matrix games. A convex game is specified by a tuple (X, Y, M) where $X \subseteq \mathbb{R}^m$ and $Y \subseteq \mathbb{R}^n$ are the strategy sets for the two players, whom we will name x and y, and M is a $m \times n$ payoff matrix. Again both players select strategies simultaneously, and the payoff from x to y given as $V(x, y) = x^T My$. The concepts of equilibria and ϵ-approximate equilibria naturally generalize to convex games, and it can be shown that the minimax theorem still holds.[1] A polyhedron is a convex set defined by a finite number of linear equality and inequality constraints, and a convex game is polyhedral if X and Y are polyhedra. Polyhedral convex games can be solved in polynomial time via linear programming, following Koller et al. [1994]. The ability to represent arbitrary convex strategy sets lets us take advantage of structure in many types of games, yielding exponentially smaller representations. Here we give four examples to briefly illustrate this point:

In **cost-paired Markov decision process games**, each player selects a stochastic policy in an MDP, and their choice determines the costs in the opponent's MDP. The set of strategies (stochastic policies in the MDPs) for each player has a concise[2] representation as a polyhedron, but there are exponentially many deterministic policies and so the corresponding matrix game is exponential in both rows and columns [McMahan et al., 2003, McMahan, 2006].

The well-studied problem of computing an **optimal oblivious routing** can be expressed as convex game where one player picks a routing in a network and the other picks traffic demands on source-sink pairs.

[1] Some mild technical assumptions are required.

[2] That is, the size of the representation of the constraints is polynomial in the size of the representation of the problem.

$x^{\mathsf{cntr}} \leftarrow$ any strategy in X
$y^{\mathsf{cntr}} \leftarrow$ any strategy in Y
$\text{lb} \leftarrow -\infty$ $\quad\quad$ $\text{ub} \leftarrow \infty$
$t \leftarrow 0$
while ($(\text{ub} - \text{lb}) > \epsilon$)
$t \leftarrow t + 1$
$x^{\mathsf{srch}} \leftarrow \text{BR}_{\mathsf{x}}(My^{\mathsf{cntr}})$ $\quad\quad$ $y^{\mathsf{srch}} \leftarrow \text{BR}_{\mathsf{y}}((x^{\mathsf{cntr}})^T M)$
$v_x = V(x^{\mathsf{cntr}}, y^{\mathsf{srch}})$ $\quad\quad$ $v_y = V(x^{\mathsf{srch}}, y^{\mathsf{cntr}})$
$\text{lb} \leftarrow \max(\text{lb}, v_y)$ $\quad\quad$ $\text{ub} \leftarrow \min(\text{ub}, v_x)$
$x^{\mathsf{cntr}} \leftarrow \frac{t}{t+1}x^{\mathsf{cntr}} + \frac{1}{t+1}x^{\mathsf{srch}}$ $\quad\quad$ $y^{\mathsf{cntr}} \leftarrow \frac{t}{t+1}y^{\mathsf{cntr}} + \frac{1}{t+1}y^{\mathsf{srch}}$
end
return $(x^{\mathsf{cntr}}, y^{\mathsf{cntr}})$ corresponding to ub and lb, respectively

Figure 1: Fictitious Play

There are exponentially many deterministic routings (pure strategies in the matrix game), but again there is a concise representation of the set of strategies as a polyhedron. The details of expressing this problem as a convex game follow from work by Azar et al. [2003], though they did not connect their work to the convex game model. The observation that optimal oblivious routing is a convex game is new (see McMahan [2006] for details), and the algorithms presented here may be of practical interest for that problem.

As previously mentioned, **extensive-form games** can be transformed to convex games. While there are typically exponentially many (in the size of the game tree) pure strategies for an EFG, the set of behavioral strategies can be represented concisely as a convex set of achievable sequence weight vectors.

Bryan et al. [2007] recently showed that the statistical problem of computing **minimax expected-size confidence regions** can be formulated as a convex game, played by nature (who picks the true parameter values) and a statistician (who picks a confidence procedure). The requirement that the statistician must pick a valid confidence procedure can be expressed via a concise set of linear constraints, but the matrix-game formulation is exponential in size. As all these examples demonstrate, fast algorithms for CGs have broad applicability.

3 BEST RESPONSES

Suppose player y fixes a strategy $y \in Y$. Then, letting $c = My$ (think of c as a cost vector), the best response problem is to compute:

$$\min_{x \in X} \quad c \cdot x \tag{1}$$

If X is a polyhedron, then this is just a standard linear program. But, in many cases (including all the examples from the previous section) much faster special-purpose algorithms are available for solving Equation (1). For example, in the case of cost-paired MDP games, solving Equation (1) is exactly the problem of planning in an MDP with known costs. For the remainder of this paper, we assume we have efficient algorithms (best response oracles) $\text{BR}_{\mathsf{x}} : \mathbb{R}^m \to \mathbb{R}^m$ and $\text{BR}_{\mathsf{y}} : \mathbb{R}^n \to \mathbb{R}^n$ for solving Equation (1). We generally view these oracles as functions from cost vectors (rather than opponent strategies) to strategies, so for example $x = \text{BR}_{\mathsf{x}}(My)$ is a best response for x to the strategy y. We choose this notation because the matrix-vector multiplications with M are often a dominating computational cost, and so explicitly tracking such multiplications is important.

It is natural to look for algorithms for solving the overall game that can exploit these special-purpose best response oracles. One simple, well-studied algorithm that accomplishes this is fictitious play: the algorithm simulates two players repeatedly playing the convex game G. Each time G is played, each player chooses to play a best-response to the average of all her opponent's previous plays.[3] While no guarantees can be made about the performance of each of these players in the simulation, the average over their past plays eventually converges to a minimax equilibrium. For a recent treatment of fictitious play, see [Leslie and Collins, 2006]. Pseudo-code for this simple algorithm is given in Figure 1. Note that each call to the best-response oracles generates an upper or lower bound for the minimax value v^* of G: if x (the min player) plays x^{cntr} in G, then the max player y can do no better than playing $y^{\mathsf{srch}} = \text{BR}_{\mathsf{y}}((x^{\mathsf{cntr}})^T M)$, and so we conclude $v^* \leq V(x^{\mathsf{cntr}}, y^{\mathsf{srch}})$. A similar argument holds for calls to BR_{x}. Synchronous FP executes the commands as given; Asynchronous FP does all the updates for x first (the left column in the loop), and then all the updates for y. The sequence of bounds corresponding to $(x_t^{\mathsf{cntr}}, y_t^{\mathsf{cntr}})$ need not be improving monotonically, so we use the max and min to guarantee the sequence is monotonic. An implementation can then track the corresponding argmax and argmin strategies, and return these if the algorithm is interrupted and asked to produce a solution in an anytime fashion; this pair of strategies forms a $(ub - lb)$-approximate minimax equilibrium.

This anytime performance can be particularly important when considering very large games where abstractions must be introduced to make any solution possi-

[3] Because the sets X and Y are convex, this average is also a valid strategy, and hence we can compute a best response to it.

ble. For example, there has been much recent work on abstraction for extensive-form games, and poker in particular [Billings et al., 2003, Gilpin and Sandholm, 2006]. In such applications, approximately solving a larger (less abstracted) game may be preferable to exactly solving a much more coarsely abstracted version.

There is a close connection between fictitious play (especially smooth versions of fictitious play) and running a pair of no-regret algorithms in self-play, one for each player. For example, the algorithms of Kalai and Vempala [2003] and Gordon [2005] can be be used in self-play in the same general form as Figure 1; however, the best-response oracle is replaced with a special-purpose oracle that at the intuitive level introduces additional smoothing. The regret bounds for such algorithms immediately give both convergence-rate guarantees as well as performance guarantees for agents actually playing a repeated game.

4 EXTENSIVE-FORM GAMES

In this section we briefly review extensive-form games with the aim of connecting known results to our notation and perspective. Two-player, zero-sum extensive-form games can model competitive strategic interactions that involve a sequence of decisions and random events. The game is specified via a game tree, where at each node either one of the players selects an action (corresponding to a successor of the current node) or nature picks a random successor according to a fixed probability distribution. Partial observability in the game is modeled via information sets: an information set is a subset of a player's nodes that are indistinguishable to the player. That is, each player's policy is only allowed to be a function of his observed information set, not the exact node in the game tree. (Necessarily, all nodes in an information set must have an equal number of successors.)

We only consider games with *perfect recall*, which ensures each player's information sets form a tree. This implies that all of a player's past actions and observations can be inferred given the current information set. With perfect recall, it is sufficient to consider only behavior policies, that is, policies which simply specify a probability distribution over actions at each information set.

The key results for extensive-form games that pertain to our work are the fact that extensive-form games can be transformed to convex games, and that computing best responses for extensive-form games is very fast.

The transformation of an extensive-form game to a convex game (X, Y, M) is via the sequence weight representation of strategies: the strategy set X has one dimension for each possible sequence of (information set, action) pairs for player x. For a given $x \in X$, the value x_i can be interpreted as the probability that the ith possible sequence occurs, conditioned on the other player and nature deterministically taking actions compatible with this sequence. The perfect recall assumption leads to a concise representation of the sets of valid sequence weight vectors as polyhedra X and Y. The payoff matrix entry $M(i, j)$ encodes both nature's contribution to the probability that sequence i occurs for x and sequence j occurs for y as well as the expected payoff given that both of these sequences occur.

The tree structure of information sets also leads to fast best response algorithms. If we fix an opponent strategy y, the vector My assigns a cost to each edge which corresponds to an action in the tree of player x's information sets. Values associated with each information set can then be computed via dynamic programming, working from the leaves backward to the root; any behavior policy that is greedy with respect to these values is a best response to y. Computing these values and reading off a best response takes time $\mathcal{O}(m)$. For a more detailed treatment of the best-response problem in extensive-form games, see [Koller and Megiddo, 1992]; for the transformation of extensive-form games to convex games see [Koller et al., 1994].

$\mathcal{B}_\mathsf{x} \leftarrow \{x_0\}$ $\quad$ $\mathcal{B}_\mathsf{y} \leftarrow \{y_0\}$
$\tilde{M}_{x_0,y_0} \leftarrow (x_0)^T M y_0$
$\mathrm{lb} \leftarrow -\infty$ $\quad$ $\mathrm{ub} \leftarrow \infty$
$t \leftarrow 0$
while $((\mathrm{ub} - \mathrm{lb}) > \epsilon)$
  $(p, q) \leftarrow$ **solveMatrixGame**$(\tilde{M})$
  $x^{\mathsf{mix}} \leftarrow \sum_{x \in \mathcal{B}_\mathsf{x}} p(x)\, x$ $\quad$ $y^{\mathsf{mix}} \leftarrow \sum_{y \in \mathcal{B}_\mathsf{y}} q(y)\, y$
  $x^{\mathsf{srch}} \leftarrow \mathrm{BR}_\mathsf{x}(My^{\mathsf{mix}})$ $\quad$ $y^{\mathsf{srch}} \leftarrow \mathrm{BR}_\mathsf{y}((x^{\mathsf{mix}})^T M)$
  $v_x = V(x^{\mathsf{mix}}, y^{\mathsf{srch}})$ $\quad$ $v_y = V(x^{\mathsf{srch}}, y^{\mathsf{mix}})$
  $\mathrm{lb} \leftarrow \max(\mathrm{lb}, v_y)$ $\quad$ $\mathrm{ub} \leftarrow \min(\mathrm{ub}, v_x)$
  $\mathcal{B}_\mathsf{x} \leftarrow \mathcal{B}_\mathsf{x} \cup \{x^{\mathsf{srch}}\}$ $\quad$ $\mathcal{B}_\mathsf{y} \leftarrow \mathcal{B}_\mathsf{y} \cup \{y^{\mathsf{srch}}\}$
  $(\forall y \in \mathcal{B}_\mathsf{y})\ \tilde{M}_{x^{\mathsf{srch}},y} \leftarrow (x^{\mathsf{srch}})^T My$
  $(\forall x \in \mathcal{B}_\mathsf{x})\ \tilde{M}_{x,y^{\mathsf{srch}}} \leftarrow x^T My^{\mathsf{srch}}$
  $t \leftarrow t + 1$
end
return best $(x^{\mathsf{mix}}, y^{\mathsf{mix}})$

Figure 2: Basic Double-Oracle Algorithm

5 BUNDLE-BASED ALGORITHMS

McMahan et al. [2003] introduced the double-oracle algorithm for the specific problem of planning in MDPs with adversary-controlled costs. In this section, we

show how the double-oracle algorithm can be extended to arbitrary convex games, and introduce the Double Oracle Bundle Algorithm (DOBA). The original double-oracle algorithm (the "basic" algorithm) may require an amount of memory exponential in the problem size, making it of only theoretical interest for solving large extensive-form games like Rhode Island Hold'em; DOBA builds on the double-oracle algorithm by providing a way to bound memory use.

The principal intuition of the double-oracle algorithm is to use our best-response oracles to build up an approximate version of the full convex game. Let $G = (X, Y, M)$ be the game we wish to solve. Our approximate game $\tilde{G}$ will also be convex, given by $(\tilde{X}, \tilde{Y}, M)$, where $\tilde{X} \subseteq X$ will be constructed from a set of best responses for x to various y strategies, and analogously for $\tilde{Y} \subseteq Y$. It should be clear that, if $\tilde{X}$ approaches X and $\tilde{Y}$ approaches Y, the approximate game $\tilde{G}$ becomes more and more similar to G. Of course, we hope that the approximation becomes good before $\tilde{X}$ and $\tilde{Y}$ become intractably large. The key difference between the double-oracle algorithm and DOBA is that the latter explicitly manages the complexity of $\tilde{X}$ and $\tilde{Y}$ while still guaranteeing convergence to a solution to the overall game G.

In both algorithms, we maintain a *finite* set of strategies for each player, $\mathcal{B}_\mathsf{x} \subseteq X$ and $\mathcal{B}_\mathsf{y} \subseteq Y$ respectively; we will call these sets bundles. We denote the convex hull of a finite set such as $\mathcal{B}_\mathsf{x}$ by

$$H(\mathcal{B}_\mathsf{x}) = \left\{ \sum_{x \in \mathcal{B}_\mathsf{x}} p(x)x \mid p \in \Delta(\mathcal{B}_\mathsf{x}) \right\}.$$

Note that $H(\mathcal{B}_\mathsf{x})$ is a convex subset of X, and similarly $H(\mathcal{B}_\mathsf{y})$ is a convex subset of Y. Letting $\tilde{X} = H(\mathcal{B}_\mathsf{x})$ and $\tilde{Y} = H(\mathcal{B}_\mathsf{y})$, we have the game $\tilde{G} = (\tilde{X}, \tilde{Y}, M)$ which we will use as a model of G.

To interpret $\tilde{G}$, we define a matrix game $\tilde{M}$ which has strategy sets $\mathcal{B}_\mathsf{x}$ for row and $\mathcal{B}_\mathsf{y}$ for column, with the $|\mathcal{B}_\mathsf{x}| \times |\mathcal{B}_\mathsf{y}|$ payoff matrix given by $\tilde{M}(x, y) = x^T M y$. The game $\tilde{G}$ is equivalent to $\tilde{M}$, in that (p, q) is a solution to $\tilde{M}$ if and only if $x^{\mathsf{mix}} = \sum_{x \in \mathcal{B}_\mathsf{x}} p(x)x$ and $y^{\mathsf{mix}} = \sum_{y \in \mathcal{B}_\mathsf{y}} q(y)y$ form a solution to $\tilde{G}$.

We will move back and forth between the two equivalent representations $\tilde{G}$ and $\tilde{M}$. For interpretation we will use $\tilde{G}$, since its relationship to G is more clear. But for computation we will work with $\tilde{M}$, since its size is independent of m and n (it is $|\mathcal{B}_\mathsf{x}| \times |\mathcal{B}_\mathsf{y}|$). This last fact is critical for large games: for example, in Rhode Island Hold'em, m and n are both approximately 1×10^6, while we fix $|\mathcal{B}_\mathsf{x}|$ and $|\mathcal{B}_\mathsf{y}|$ at, say, 55.

Both algorithms build up the model game $\tilde{M}$ in an intuitive way using the best response oracles: we initialize the bundles with one or more arbitrarily chosen strategies[4] for each player. Given the current bundles $\mathcal{B}_\mathsf{x}$ and $\mathcal{B}_\mathsf{y}$, we solve the corresponding matrix game $\tilde{M}$, producing a mixed strategy (p, q). We then compute the corresponding strategies $x^{\mathsf{mix}} \in X$ and $y^{\mathsf{mix}} \in Y$; $(x^{\mathsf{mix}}, y^{\mathsf{mix}})$ is a valid strategy pair in either $\tilde{G}$ or G, and so we can use our oracles to generate best responses $\mathrm{BR}_\mathsf{y}((x^{\mathsf{mix}})^T M)$ and $\mathrm{BR}_\mathsf{x}(M y^{\mathsf{mix}})$. We add these new strategies to the bundle, and also use the fact that they are best responses to update upper and lower bounds on the value of the game G. This process defines the basic double-oracle algorithm (Figure 2).

On each iteration, the size of each bundle increases by one, as does each dimension of the matrix game $\tilde{M}$. This is the principal weakness of the basic algorithm: the cost of each iteration and the size of the bundles grow, making it infeasible to run an arbitrary number of iterations. While McMahan et al. [2003] demonstrated that for some problems only a few iterations of this algorithm can produce very good solutions, in general this will not be the case. In particular, for Rhode Island Hold'em, storing each strategy in the bundle requires about 7MB of memory, and so physical memory rapidly limits the size of the bundles and hence the number of iterations. To address this issue, DOBA introduces an aggregation and pruning scheme that allows it to maintain a constant bundle size.

A second deficiency of the basic algorithm is that inaccuracies in the model $\tilde{G}$ can lead to solutions $(x^{\mathsf{mix}}, y^{\mathsf{mix}})$ that perform poorly in the true game G. However, the direction from the current best pair of strategies (x^*, y^*) towards $(x^{\mathsf{mix}}, y^{\mathsf{mix}})$ usually provides a good direction of improvement. To exploit this fact, we introduce a fast line search procedure that efficiently solves this 1-dimensional optimization problem. Our final algorithm is given in Figure 3; in the following sections we will outline the principle differences from the basic version.

Aggregation Our aggregation and pruning scheme has two components. First, we insert the minimax strategies x^{cntr} and y^{cntr} into the bundles. This has no effect on the convex hulls of the bundles if we never remove strategies, but since we will be discarding strategies, adding the mixtures is useful: in this way even if we throw out some strategies that support x^{cntr}, we may still keep $x^{\mathsf{cntr}} \in H(\mathcal{B}_\mathsf{x})$ by explicitly placing $x^{\mathsf{cntr}} \in \mathcal{B}_\mathsf{x}$.

In order to determine which strategies to discard, each time we solve $\tilde{M}$ we use the mixed strategies to up-

[4] These could be the uniform random strategy (this is how we initialize for our experiments), but there is the opportunity to increase performance by seeding the algorithm with a collection of expert-generated strategies.

while $((\text{ub} - \text{lb}) > \epsilon)$
  $(p, q) \leftarrow$ **solveMatrixGame**$(\tilde{M})$
  update strategy weights
  $x^{\mathsf{mix}} \leftarrow \sum_{x \in \mathcal{B}_{\mathsf{x}}} p(x)x \qquad y^{\mathsf{mix}} \leftarrow \sum_{y \in \mathcal{B}_{\mathsf{y}}} q(y)y$
  updateCenter(x)
  $\text{ub} \leftarrow \min\left(\text{ub}, V(x^{\mathsf{cntr}}, \text{BR}_{\mathsf{y}}((x^{\mathsf{cntr}})^T M))\right)$
  updateCenter(y)
  $\text{lb} \leftarrow \max\left(\text{lb}, V(\text{BR}_{\mathsf{x}}(My^{\mathsf{cntr}}), y^{\mathsf{cntr}})\right)$
  if ($\mathcal{B}_{\mathsf{x}}$ or $\mathcal{B}_{\mathsf{y}}$ are too big) do aggregation
  update(ϕ)
  $t \leftarrow t + 1$
end
updateCenter(x):
  $x^{\mathsf{srch}} \leftarrow$ **search**$(\text{BR}_{\mathsf{x}}(My^{\mathsf{cntr}}), x^{\mathsf{mix}}, [0, 1 - \phi])$
  $\mathsf{fpstep} \leftarrow 1/(t + 1)$
  $\alpha \leftarrow \phi \cdot \mathsf{fpstep}$
  $\beta \leftarrow \mathsf{fpstep} + (1 - \phi)(1 - \mathsf{fpstep})$
  $x^{\mathsf{cntr}} \leftarrow$ **search**$(x^{\mathsf{cntr}}, x^{\mathsf{srch}}, [\alpha, \beta])$
  add $\{x^{\mathsf{mix}}, \text{BR}_{\mathsf{x}}(My^{\mathsf{mix}}), x^{\mathsf{cntr}}, \text{BR}_{\mathsf{x}}(My^{\mathsf{cntr}})\}$ to $\mathcal{B}^*_{\mathsf{x}}$)
end

Figure 3: DOBA: the double oracle bundle algorithm with line search, aggregation, and convergence guarantees. Initialization and updates to $\tilde{M}$ (performed on line $(*)$) are similar to those in the basic algorithm.

date a weight $w(x)$ or $w(y)$ associated with each strategy in the bundle: this weight is a discounted average of the probabilities placed on x or y by past solutions to $\tilde{M}$. Each iteration, we choose to remove the strategies with the smallest weights; we then add to the bundle an aggregate of the removed strategies, with each removed strategy weighted proportionally to $w(x)$ or $w(y)$. That is, if we remove $x_1, \ldots, x_k$ and if $W = \sum_{i=1}^{k} w(x_i)$, then the aggregate strategy is given by

$$x_{\text{aggr}} = \sum_{i=1}^{k} \frac{w(x_i)}{W} x_i.$$

To keep the bundle size constant, we remove five strategies on each step: one each to to make room for the four strategies added in line (*) in Figure 3, and one to make room for the aggregated strategy x_{aggr}.

Line Search For extensive-form games, it takes time $\mathcal{O}(m)$ to run the oracle $\text{BR}_{\mathsf{x}}(c_y)$ for a fixed cost vector $c_y = My$, but the cost of the multiplication to compute c_y is $\mathcal{O}(nm)$. While the matrix M may be sparse, multiplications with M will still typically be slower than best response calls by a considerable constant; for example, this constant is around 20 for Rhode Island Hold'em.

In this section we show how we can take advantage of the relative speed of evaluating best responses for fixed cost vectors. Consider a restricted convex game with $\mathcal{B}_{\mathsf{x}} = \{x_1, x_2\}$ and $\tilde{X} = H(\mathcal{B}_{\mathsf{x}})$ but $\tilde{Y} = Y$. That is, x has exactly two strategies, while y has full access to his strategy set. We show that we can solve the corresponding restricted game $(H(\mathcal{B}_{\mathsf{x}}), Y, M)$ efficiently via a line search. The key is that x's choice of a probability distribution over $\mathcal{B}_{\mathsf{x}}$ only has a single degree of freedom. Using θ to represent this free variable, we can write the problem of solving this game as:

$$\min_{\theta \in [0,1]} \max_{y \in Y} \left((1 - \theta)x_1 + \theta x_2\right)^T My \tag{2}$$

For simplicity, we write $x(\theta) = ((1 - \theta)x_1 + \theta x_2)$. Then, define the function $f : \mathbb{R} \to \mathbb{R}$ by

$$f(\theta) = \max_{y \in Y} x(\theta)^T My \tag{3}$$

and so solving Equation (2) is equivalent to solving $\min_{\theta \in [0,1]} f(\theta)$. Since f is a piecewise maximum over a set of affine functions, one for each $y \in Y$, it is convex. We can minimize such a function via an exact line search if we can evaluate f at all $\theta \in [0, 1]$ and also compute a subgradient to f at each θ. The best response oracle BR_{y} can be used to accomplish both these tasks.

For a fixed θ, we can find a y that achieves the maximum in Equation (3) by computing $y = \text{BR}_{\mathsf{y}}(x(\theta)^T M)$, so that $f(\theta) = V(x(\theta), y)$. Further, y corresponds to the linear function $x(\theta)^T My$ which gives a lower bound on f and is tight at θ, so the slope of f_y is a subgradient of f at θ. This can easily be calculated as $(x_2 - x_1)^T My$.

Directly implementing this approach requires a multiplication with M on each iteration, but we can avoid these multiplications by pre-computing (or caching) $c_1 = x_1^T M$ and $c_2 = x_2^T M$. Define $c(\theta) = (1 - \theta)c_1 + \theta c_2$. For a fixed θ, we can evaluating $c(\theta)$ in $\mathcal{O}(m)$ time. After computing $y = \text{BR}_{\mathsf{y}}(c(\theta))$, we calculate $f(\theta)$ as $c(\theta) \cdot y$. Using the same y, we can calculate the necessary subgradient as $(c_2 - c_1) \cdot y$. Thus, each iteration of the line search can be completed in $\mathcal{O}(m)$ time.

Convergence Guarantees via Fictitious Play Fictitious play (or a no-regret algorithm in self-play) maintains *centers* x^{cntr} and y^{cntr}, estimates of the minimax optimal mixed strategies. On each iteration FP updates these centers in the *search direction*, $x^{\mathsf{srch}} = \text{BR}_{\mathsf{x}}(y^{\mathsf{cntr}})$ and $y^{\mathsf{srch}} = \text{BR}_{\mathsf{y}}(x^{\mathsf{cntr}})$. DOBA has a similar structure: it maintains a center for each player, and on each iteration updates these centers towards a search direction. The algorithm maintains a parameter ϕ (the fictitious play fraction), so that when $\phi = 0$

the algorithm runs in an unrestricted fashion, while if $\phi = 1$, the algorithm is exactly FP.

The selection of the search direction and update of the center occurs in the **updateCenter**(x) method of Figure 3; **updateCenter**(y) is identical, but with the roles of x and y switched. The best response to the opponent's current center is one possible search direction; the solution to the model game $\tilde{M}$ provides another. DOBA does a line search between these two possibilities in order to choose its search direction; however, at least ϕ weight is required to be on the best response to the opponent's center,[5] so that when $\phi = 1$ DOBA uses the same search direction as FP. This is accomplished via the call to $\textbf{search}(\text{BR}_{\mathsf{x}}(My^{\mathsf{cntr}}), x^{\mathsf{mix}}, [0, 1-\phi])$.

Similarly, we update the center by a line search from x^{cntr} towards x^{srch}, but we constrain the interval of the search to linearly interpolate from $[0, 1]$ when $\phi = 0$, to $[1/(t+1), 1/(t+1)]$ when $\phi = 1$. The constants α and β in the call to $\textbf{search}(x^{\mathsf{cntr}}, x^{\mathsf{srch}}, [\alpha, \beta])$ accomplish this interpolation; when $\phi = 1$, we have the fixed step-size $1/(t+1)$ of fictitious play.

We can insure convergence of DOBA by updating ϕ based on the rate of change of $(\text{ub} - \text{lb})$ so that if the rate drops lower than that expected of FP, ϕ eventually goes to 1, and DOBA effectively becomes FP. We ran experiments with several simple methods for updating ϕ; generally these had little impact of the runtime of the algorithm. To avoid conflating the impact of the ϕ updating scheme with the performance of our unconstrained approach, we present experimental results with ϕ fixed at 0.

6 EXPERIMENTAL RESULTS

We tested DOBA and FP on two poker games represented as EFGs: exactly abstracted Rhode Island Hold'em (RIH), and approximately abstracted[6] Texas Hold'em (TH). For an introduction to two-player (heads up) limit Texas Hold'em, see, for example, Billings et al. [2003]. RIH is a restricted version of TH: it is played with a full deck of 52 cards, but each player receives only a single face-down hole card, and there are only two community cards. There are three rounds of betting, with up to three raises per betting round.

[5]It is probably better to directly constrain the value of x^{srch} against y^{cntr}, say requiring that x^{srch} is a $(1-\phi)$-approximate best response, for example. Or, perhaps even better, this constraint could be embedded directly into the linear program for solving the small matrix game $\tilde{M}$. We have not yet run experiments to test these ideas.

[6]This instance only models the first three rounds of betting, and uses other approximation techniques from Gilpin and Sandholm [2006].

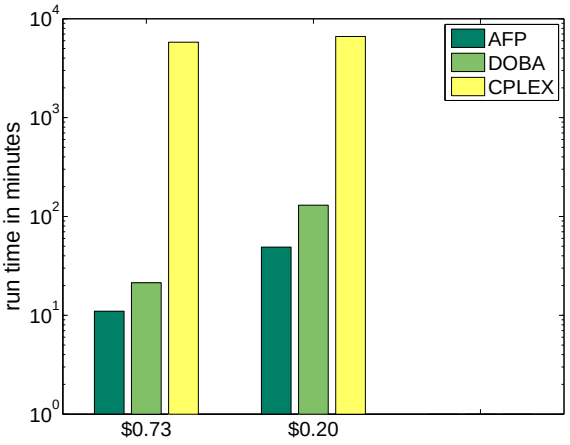

Figure 4: Runtimes for asynchronous FP (AFP), DOBA (with $|\mathcal{B}| = 55$), and CPLEX to produce $\epsilon = \$0.73$ and $\epsilon = \$0.20$ approximate minimax solutions. Both AFP and DOBA outperform CPLEX by several orders of magnitude (note log scale). CPLEX was run on a different machine than FP and DOBA, so the comparison is approximate.

Unabstracted Rhode Island Hold'em has a game tree with 3.1 billion nodes, which is still too large to solve directly. However, by applying a novel exact abstraction technique (the GameShrink algorithm), Gilpin and Sandholm [2005] were able to produce an equivalent but much smaller EFG: it has 50×10^6 non-zeros in the payoff and sequence constraint matrices, with dimensions $m = n = 883,741$, taking about 600MB of memory to store. A solution to this game can be converted to a payoff-equivalent strategy for the unabstracted game. This version of RIH has \$5.00 antes and a maximum pot size of \$310.00. The uniform random strategy, from which we started both our algorithm and fictitious play, loses approximately \$290.00 per game. The minimax value of the game is $-\$0.64$; the value is negative because player x (the minimizing player) bets second, and thus gains a small advantage based on the information revealed by the first player's initial bet.

Gilpin and Sandholm [2005] used the CPLEX commercial linear programming package to solve RIH via the barrier method in about 7.5 days, using 25 GB of memory; achieving an $\epsilon = \$0.20$ approximate minimax solution took over 4.5 days. DOBA produced a solution of that quality in 130 minutes, using less than 1.5GB of memory; to our surprise, asynchronous FP (AFP) was even faster, needing only 50 minutes; see Figure 4. TH is larger than RIH (130×10^6 non-zeros), but has lower dimensionality ($m = n = 236{,}416$). The instance we used has a small blind (similar to an ante) of \$0.50, and a big blind of \$1.00. For this problem, DOBA signifi-

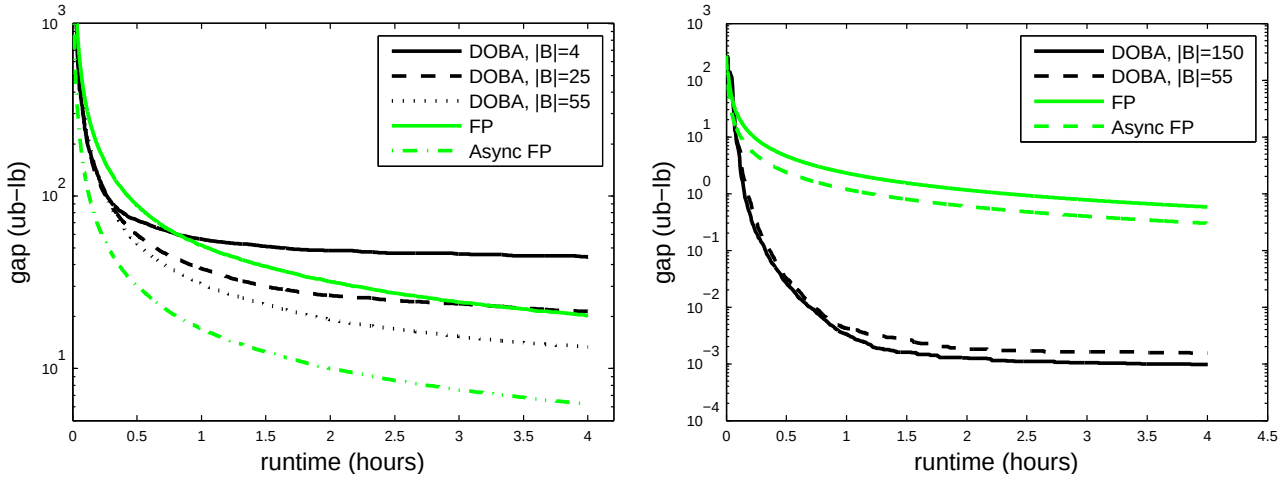

Figure 5: Algorithm runtime vs. approximation error for **RIH** (left) and **TH** (right). The Y axis is a log scale on ϵ for the best approximate solution the algorithms can return at a given time, in units of \$0.01. Note that DOBA outperforms both versions of FP by several orders of magnitude for TH.

cantly outperformed asynchronous fictitious play: FP bounded the value of the game in $[-\$0.028, -\$0.046]$ in a 2 hour run, while DOBA achieved better bounds in less than 6 minutes. Figure 5 compares the anytime performance of DOBA and fictitious play on both TH and RIH. For more details, see McMahan [2006].

7 CONCLUSIONS

This work demonstrates that a variety of research problems of current interest can be caste as convex games, and demonstrates a powerful new algorithm for finding approximate equilibria in these games. There is great promise in both mapping additional optimization problems to the convex game framework as well as continuing the development of algorithms for the solution of convex games.

Acknowledgments

Special thanks to Andrew Gilpin who graciously provided us with the sequence-weight representations for the poker problems used in the experiments and shared data from experiments using CPLEX to solve the same instances. This work was supported in part by DARPA CSSP HR0011-06-1-0023.

References

Y. Azar, E. Cohen, A. Fiat, H. Kaplan, and H. Räcke. Optimal oblivious routing in polynomial time. In *Proc. of ACM Symposium on the Theory of Computation*, 2003.

D. Billings, N. Burch, A. Davidson, R. Holte, J. Schaeffer, T. Schauenberg, and D. Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In *IJCAI*, 2003.

Brent Bryan, H. Brendan McMahan, Chad M. Schafer, and Jeff Schneider. Efficiently computing minimax expected-size confidence regions. Under review, 2007.

M. Dresher and S. Karlin. Solutions of convex games as fixed-points. In H. W. Kuhn and A.W. Tucker, editors, *Contributions To The Theory of Games: Volume 2*, number 28 in Annals of Mathematics Studies, pages 75–86. Princeton University Press, 1953.

Andrew Gilpin and Tuomas Sandholm. Optimal rhode island hold'em poker. In *AAAI*, pages 1684–1685, 2005.

Andrew Gilpin and Tuomas Sandholm. A texas hold'em poker player based on automated abstraction and real-time equilibrium computation. In *AAMAS 06*, 2006.

Geoffrey J. Gordon. No-regret algorithms for structured prediction problems. Technical Report CMU-CALD-05-112, Carnegie Mellon University, 2005.

J.-B. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization Algorithms*. Springer Verlag, Heidelberg, 1993. Two volumes.

Adam Kalai and Santosh Vempala. Efficient algorithms for online optimization. In *COLT*, 2003.

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

Daphne Koller, Nimrod Megiddo, and Bernhard von Stengel. Fast algorithms for finding randomized strategies in game trees. In *STOC*, 1994.

David S. Leslie and E.J. Collins. Generalised weakened fictitious play. *Games and Economic Behavior*, 56(2): 285–298, August 2006.

H. Brendan McMahan, Geoffrey J. Gordon, and Avrim Blum. Planning in the presence of cost functions controlled by an adversary. In *ICML 2003*, 2003.

Hugh Brendan McMahan. *Robust Planning in Domains with Stochastic Outcomes, Adversaries, and Partial Observability*. PhD thesis, Carnegie Mellon University, December 2006.