An Empirical Study on Computing Equilibria in Polymatrix Games

Argyrios Deligkas
University of Liverpool, UK

John Fearnley
University of Liverpool, UK

Tobenna Peter Igwe
University of Liverpool, UK

Rahul Savani
University of Liverpool, UK

ABSTRACT

The Nash equilibrium is an important benchmark for behaviour in systems of strategic autonomous agents. Polymatrix games are a succinct and expressive representation of multiplayer games that model pairwise interactions between players. The empirical performance of algorithms to solve these games has received little attention, despite their wide-ranging applications. In this paper we carry out a comprehensive empirical study of two prominent algorithms for computing a sample equilibrium in these games, Lemke's algorithm that computes an exact equilibrium, and a gradient descent method that computes an approximate equilibrium. Our study covers games arising from a number of interesting applications. We find that Lemke's algorithm can compute exact equilibria in relatively large games in a reasonable amount of time. If we are willing to accept (high-quality) approximate equilibria, then we can deal with much larger games using the descent method. We also report on which games are most challenging for each of the algorithms.

General Terms

Algorithms, Economics

Keywords

Game Theory; Nash Equilibrium; Approximate Equilibria; Polymatrix Games; Auctions; Bayesian Two-Player Games; Lemke's Algorithm; Gradient Descent

1. INTRODUCTION

In multiagent systems it is often the case that autonomous agents interact with each other, but do not necessarily have the same objectives or goals. This situation can be described as a game played between the agents, and the tools from game theory can be used to analyse the possible outcomes. In particular, the concept of a Nash equilibrium [37] describes a stable situation in which no agent can increase its reward by changing its behaviour. Therefore, to gain insight into the possible behaviours that a system of rational agents will produce, one can compute the Nash equilibria of the game that is played between the agents.

Appears in: *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016), J. Thangarajah, K. Tuyls, C. Jonker, S. Marsella (eds.), May 9–13, 2016, Singapore.*

Games are often represented in *strategic-form*, where for each possible combination of strategy choices, a numerical payoff is specified for each of the players. However, the size of this representation grows *exponentially* in the number of players. For example, for a game with n players who can each choose between 2 strategies, $n \cdot 2^n$ payoffs must be specified. Hence, the strategic-form is typically unsuitable for the types of games that arise from multiagent systems.

Many realistic scenarios do not need the flexibility that strategic-form games provide. In particular, it is often the case that only the *pairwise* interactions between players are important. In this paper we study *polymatrix games* which model this. In these games the interaction between the players is specified as a graph. Each player plays an independent two-player game against each other player that he is connected to, and the same strategy must be played in all of his games. A player's payoff is then the sum of the payoffs from each of the games. Crucially, the representations of these games grow *quadratically* in the number of players, which makes them more suitable for representing the large multiagent systems that arise from real-world scenarios.

While there has been a large amount of theoretical work on polymatrix games [14, 15, 17, 24, 26, 28], the practical aspects of computing equilibria in polymatrix games have yet to be studied. In this paper, we provide an empirical study of two prominent methods for computing equilibria in these games. Firstly, we study *Lemke's algorithm*. The Lemke-Howson algorithm is a famous algorithm for finding Nash equilibria in bimatrix games [35], and Lemke's algorithm is a more general technique for solving *linear complementarity problems* (LCPs). Miller and Zucker [36] have shown that the problem of finding a Nash equilibrium in a polymatrix game can be reduced to the problem of solving an LCP that can then be tackled by Lemke's algorithm.

Secondly, we study a method for finding *approximate* equilibria in polymatrix games. In contrast to a Nash equilibrium, where no player has an incentive to deviate, an approximate equilibrium allows the players to have a positive, but small, incentive to deviate from their current strategies. Approximate equilibria have received a large amount of interest in theoretical work [9, 11, 18, 22, 30, 45], because the problem of finding an exact Nash equilibrium in a polymatrix game, even for only two players, is known to be PPAD-complete (which implies that there is unlikely to be a polynomial-time algorithm for this problem). From a practical point of view, it is reasonable to use sufficiently accurate approximate equilibria to study real-world systems, because often there is a non-negative cost to changing strategy, which could deter an agent from deviating even if doing so would lead to a small increase in payoff. Also, if the game is derived from real data, then any uncertainty in the actual payoffs of the underlying situation means that agents may be perfectly happy in the real world, even if the game model says that they can gain a small amount through deviation.

We study a recently proposed gradient descent-like algorithm for finding approximate equilibria in polymatrix games [22], which is the only known approximation technique for (general) polymatrix games. It generalizes the algorithm of Tsaknakis and Spirakis (TS) for bimatrix games [45]. A recent study found that the TS algorithm typically finds high-quality approximate equilibria in practice [25], much better that its theoretical worst-case performance.

Our contribution. We provide a thorough empirical study of finding exact and approximate Nash equilibria in polymatrix games. We develop an extensive library of game classes that cover a number of applications of polymatrix games including cooperation games, strictly competitive games, and group-wise zero-sum games. We also study *Bayesian two-player games*, which can be modelled as polymatrix games. In particular, we focus on various forms of *Bayesian auctions* (e.g., item bidding combinatorial auctions) and Bayesian variants of Colonel Blotto games, which have applications to task allocation and resource allocation problems between agents [44]. All of our algorithm implementations and game generators are open source and publicly available[1], so that any new algorithms developed for polymatrix games can be tested against our test suite.

We study Lemke's algorithm and the descent method on all of the problems that we consider. In total we applied Lemke's algorithm to 188,000 instances using 26 months of CPU time, while we applied descent to 213,000 instances using 2.7 months of CPU time. We found that Lemke's algorithm can compute exact equilibria in relatively large games in a reasonable amount of time, though the descent method is much more scalable and can be used to compute approximate equilibria for instances that are an order of magnitude bigger. Moreover, in contrast to its theoretical worst-case performance guarantee, the descent method typically finds very high quality approximate equilibria.

Related work. The problem of equilibrium computation has received much attention from the theoretical point of view. Firstly, it was proven that computing an exact Nash equilibrium is PPAD-complete [14, 17], even for games with only two players. While the class NP captures decision problems, the complexity class PPAD captures problems where it is *known* that a solution exists. It is assumed that it is unlikely that there exists a polynomial time algorithm for PPAD-complete problems. For this reason a line of work that studies approximate notions of Nash equilibria has arisen [9, 11, 18, 30, 45]. Specifically for polymatrix games, there is the recent descent procedure studied in this paper [22], and a recent QPTAS for polymatrix games on trees [6].

There are empirical studies on equilibrium computation both for exact equilibria [4, 5, 25, 40, 43] and approximate equilibria [25], but none of them focused on polymatrix games. Instead, these studies mainly focused on games created by GAMUT [38], the most famous suite of game generators. GAMUT has a generator for some simple polymatrix games, but it converts them to strategic-form games which blows up the representation exponentially.

Polymatrix games have received a lot of attention recently. Computing a Nash equilibrium in a polymatrix game is PPAD-hard even when all the bimatrix games are either zero-sum or coordination games [12]. Recently, it was proven that there is a constant $\epsilon > 0$ such that it is PPAD-hard to compute an ϵ-Nash equilibrium of a polymatrix game [42]. Govindan and Wilson proposed a (non-polynomial-time) algorithm for computing equilibria of an n-player strategic-form game, by approximating the game with a sequence of polymatrix games [28]. Later, they presented a (non-polynomial) reduction that reduces n-player games to polymatrix games while preserving approximate Nash equilibria [29].

Many papers have derived bounds on the Price of Anarchy [8, 27, 41] in item bidding auctions [16]. Only recently Cai and Papadimitriou [13] and Dobzinski, Fu and Kleinberg [23] studied the question of the complexity of the equilibrium computation problem in this setting. Blotto games are a basic model of resource allocation, and have therefore been studied in the agents community [2, 39]. There have also been several papers that study Blotto games with incomplete information as well, see for example [1, 33].

Polymatrix games are examples of *graphical games*, which are succinct representations of games where interactions between players are encoded in a graph. A related succinct representation is that of Action Graph Games (AGGs); introduced by Bhat and Leyton-Brown, AGGs capture local dependencies as in graphical games, and partial indifference to other agents' identities as in anonymous games [7, 20, 32].

2. PRELIMINARIES

Bimatrix games. A bimatrix game is a pair (R, C) of two $n \times n$ matrices: R gives payoffs for the *row* player, and C gives the payoffs for the *column* player. Each player has n *pure* strategies. To play the game, both players simultaneously select a pure strategy: the row player selects a row i, and the column player selects a column j. The row player then receives $R_{i,j}$, and the column player $C_{i,j}$.

A *mixed strategy* is a probability distribution over $[n]$. We denote a row player mixed strategy as a vector $\mathbf{x}$ of length n, such that $\mathbf{x}_i$ is the probability assigned to row i. Mixed strategies of the column player are defined symmetrically. If $\mathbf{x}$ and $\mathbf{y}$ are mixed strategies for the row and the column player, respectively, then the expected payoff for the row player under the strategy profile $(\mathbf{x}, \mathbf{y})$ is given by $\mathbf{x}^T R \mathbf{y}$ and for the column player by $\mathbf{x}^T C \mathbf{y}$.

Polymatrix games. An n-player polymatrix game is defined by an n-vertex graph. Each vertex represents a player. Each edge e corresponds to a bimatrix game that will be played by the players that e connects. Hence, a player with degree d plays d bimatrix games. More precisely, each player picks a strategy x_i and plays that strategy in *all* of the bimatrix games that he is involved in. His expected payoff is given by the sum of the expected payoffs that he obtains over all the bimatrix games that he participates in. We use $\mathbf{x} = (x_1, \ldots, x_n)$ to denote a strategy profile of an n-player game, where x_i denotes the mixed strategy of player $i \in [n]$.

Solution concepts. The standard solution concept for strategic-form games is the *Nash equilibrium* (NE). A relaxed version of this concept is the approximate NE, or ϵ-NE. Intuitively, a strategy profile is an ϵ-NE in an n-player game,

[1] http://polymatrix-games.github.io/

if no player can increase his utility more than ϵ by unilaterally changing his strategy. To put it formally, let $\mathbf{x}$ denote a strategy profile for the players and let $u_i(z, \mathbf{x}_{-i})$ denote the utility of player i when he plays the strategy z and the rest of the players play according to $\mathbf{x}$. We say that $\mathbf{x}$ is an ϵ-NE if for every player i it holds that $u_i(x_i, \mathbf{x}_{-i}) \geq u_i(z, \mathbf{x}_{-i}) - \epsilon$ for all possible z. If $\epsilon = 0$, we have an exact Nash equilibrium.

3. ALGORITHMS

Lemke's algorithm is a complementary pivoting algorithm for the Linear Complementarity Problem (LCP) [34]. Miller and Zucker have shown that finding a Nash equilibrium in a polymatrix game can be reduced in polynomial time to a Lemke-solvable LCP [36]. So, we first turn the polymatrix game into an LCP, and then apply Lemke's algorithm. We shall refer to this algorithm as LEMKE.

One drawback is that the reduction assumes a complete interaction graph even if the actual interaction graph is not complete (by padding with all-zero payoff bimatrix games). Hence, for sparse polymatrix games the reduction introduces a significant blowup, which affects the performance of the algorithm. To make this blowup clear, in our results we report the number of payoffs in the original polymatrix game and the number of matrix entries in the resulting LCP.

Descent is a gradient descent-like algorithm proposed in [22]. It tries to minimize the *regret* that a player suffers, which is the difference between the best-response utility and the actual utility he gets. The algorithm starts from an arbitrary strategy profile $\mathbf{x}$ and in each iteration it computes a new profile in which the maximum regret (over the players) has been reduced. The algorithm takes a parameter δ that controls how accurate the resulting approximate Nash equilibrium is. The theoretical results state that the algorithm finds a $(0.5 + \delta)$-NE after $O(\delta^{-2})$ iterations (for a fixed size game). We test the cases where δ is either 0.1 or 0.001, and we provide full results for both cases. We shall refer to this algorithm as DESCENT in our results.

For the strategy profile $\mathbf{x}$, the algorithm first computes a direction vector $(\mathbf{x}' - \mathbf{x})$, and then moves a certain distance in that direction. In other words, the algorithm moves to a new strategy profile $\mathbf{x} + \alpha(\mathbf{x}' - \mathbf{x})$, where α is some constant in the range $(0, 1]$. The theoretical analysis in [22] uses $\alpha = \frac{\delta}{\delta+2}$ in the proof of polynomial-time convergence. In practice we found that using larger step sizes greatly improves the running time. Hence, we adopt a *line search* technique, which we adapt from the two-player setting [46]. It checks a number of equally spaced values for α in $[0, 1]$, and selects the best improvement that is found. We also include $\alpha = \frac{\delta}{\delta+2}$ as an extra value in this check, so that the theoretical worst-case running time is unchanged. In our results, we check 201 different points for α in each iteration. For a justification of the reasonableness of this choice, see Appendix A in the full version of this paper [21].

4. GAME CLASSES

Bayesian Auctions

Combinatorial auctions. In a combinatorial auction, m items are auctioned to n bidders. Each bidder has a valuation function that assigns a non-negative real number to every subset of the items. Notice that in this setting, in general, the size required to represent the valuation function is exponential in m. In combinatorial auctions with *item bidding*, each player bids for every item separately and all the items are auctioned simultaneously. A bidder wins an item if he submitted the highest bid for that item. The bidders pay according to a predefined payment rule. We study three popular payment rules. In a *first price* auction the winner of an item has to pay his bid for that item, in a *second price* auction the winner of an item has to pay the second highest bid submitted for the item, and in an *all pay* auction every bidder has to pay his bid *irrespective* of whether he won the item or not. If more than one bidder has the highest bid for some item, we resolve this tie according to a predefined publicly known rule. We study two tie-breaking rules: either we always favor one of the players, or we choose the winner for each item independently uniformly at random.

We say that a combinatorial auction allows *overbidding* if a bidder is allowed to make a bid for a item greater than his value for it. A common assumption in the literature is that overbidding is not allowed, since allowing overbidding leads to the existence of trivial equilibria. Therefore, in our experiments we do not allow overbidding.

In a *Bayesian* combinatorial auction the valuation function for every player is chosen according to a commonly known joint probability distribution, which in this paper is always discrete. The different valuation functions that may be drawn for a player are known as his *types*.

Item bidding auctions. We create two-bidder Bayesian item bidding combinatorial auctions with 2 to 4 items for sale and 2 to 5 different types per player. Each player's type (valuation function) is chosen uniformly at random. We study several well known valuation functions:

- *Additive*: the value of each bundle of items is the sum of the values of the items contained in the bundle.
- *Budget additive*: the value of each bundle is the minimum of a budget parameter and the sum of the values of the items contained in the bundle.
- *Single minded*: each bidder has positive value for a specific bundle of items (and the same value for any other bundle of items containing that bundle) and zero otherwise.
- *Unit demand*: the value of each bundle is the maximum value the bidder has for any single item contained in the bundle.
- *AND-OR*: the first bidder has positive value only for the grand bundle of items and zero otherwise, while the second one has a unit demand valuation.

To create the valuations, we set a maximum value $M \in \mathbb{N}$ that a player can have for any item, and a minimum value $m \in \mathbb{N}$ such that in every valuation there must be an item with a value of at least m. Bids are restricted to $\mathbb{N}$, so M and m define the number of pure strategies a player has, and consequently the size of the game. For a player with a single-minded valuation function, we choose a random subset of items and a random value for that subset in the range $[m, M]$. For additive and unit demand valuations, we randomly select a value for each item from the set of allowed valuations. The same procedure is extended for budget additive bidders: first we draw values for items as for additive bidders; then we draw the budget as a random integer in $[M, N]$ where N is the sum of the values for the items.

Multi-unit auctions. In these auctions all the items being sold are identical. When there are n items for sale, a valuation is given by an n-tuple $(v_1, \ldots, v_n)$, where v_j represents the player's marginal value for receiving a j-th copy of the item. Hence, the valuation for a bidder when he wins k items is the sum of the values v_1 up to v_k.

Again we study the three most common payment rules: the first price rule, a.k.a. the *discriminatory auction*, where a player that won k items has to pay the sum of his k highest bids; the second price rule, a.k.a. the *uniform-price auction*, where the price for every item is the market-clearing price, i.e., the highest losing bid; and the all-pay rule, where a player has to pay the sum of his bids.

We consider two well known valuation functions: additive, where $v_j = v_1$ for all $j > 1$, and submodular, where $v_j \geq v_{j+1}$ for all $j \in [n-1]$. We create games with 2 to 4 items and 2 to 5 different types per player. The sampling of non-additive sub-modular valuation functions is not described here; we refer the reader to the source code for further details[2].

Other Bayesian Two-player Games

A two-player Bayesian game is played between a row player and a column player. Each player has a set of possible types, and at the start of the game, each player is assigned a type according to a publicly known joint probability distribution. Each player learns his type, but not the type of the other player. Rosenthal and Howson showed that the problem of finding an exact equilibrium in a two-player Bayesian game can be reduced to finding an exact equilibrium in a polymatrix game [31], and this was extended to approximate equilibria in [22]. The underlying graph in the resulting polymatrix game is a complete bipartite graph where the vertices of each side represent the types of a player. More specifically, if the row player has n types and the column player has m types, the corresponding polymatrix game has $n + m$ vertices and the payoff matrix for edge (uv) corresponds to the payoff matrix of the Bayesian game where the row player has type u and the column player has type v. We study the following Bayesian two-player games.

Colonel Blotto games. In Colonel Blotto games, each player has a number of soldiers m_1, m_2 that are simultaneously assigned to n hills. Each player has a value for each hill that he receives if he assigns strictly more soldiers to the hill than his opponent and any ties are resolved by choosing a winner uniformly at random. The value that a player has for a hill is generated independently uniformly at random and the payoff a player gets under a strategy profile is given by the sum of the value of the hills won by that player. We consider games with 3 and 4 hills and 3 to 15 soldiers per player. We study two different Bayesian parameters: the valuations of the players over the hills and the number of soldiers that each player has. In the game we looked at, only one of these two parameters was used (i.e., for the other there was complete information). For both cases we study games with 2 to 4 types per player. When the types correspond to different valuations for hills, for every type, the valuations for each hill were drawn from independent uniform distributions on $[0, 1]$. When the types correspond to the number of soldiers, we drew independently from $\{3, \ldots, 15\}$ for each player and each type.

[2] `http://polymatrix-games.github.io/`

Adjusted Winner games. The adjusted winner procedure fractionally allocates a set of n divisible items to two players [10]. Under the procedure, $n - 1$ items stay whole and at most one is split between the players. Each player has a non-negative value for each item, and these values sum to 1. Both players have additive valuations over bundles of items. For a split item that a player has value v for, if the player receives $w \in [0, 1]$ of the item, he gets gets $w \cdot v$ value from this part of the item.

The players simultaneously assign m points to the items. Suppose player 1 assigns α_i points for the items $i = 1, \ldots, n$ with $\sum_i \alpha_i = m$, and similarly player 2's assignment is $(\beta_1, \ldots, \beta_n)$. The procedure starts with an initial allocation in which each item goes to one of the players that assigned most points to it. If the players get equal utilities it stops. Otherwise it next determines which player gets higher utility under this allocation, say player 1. Next it finds the item i that is currently allocated to player 1 and has the smallest ratio α_i/β_i. If possible it splits this item in a such a way as to equalize the total utilities of the two players, or, if not, completely reallocates this item to player 2, and repeats this step until the utilities of the two players are equalized. Thus, at most one item is actually split.

We create Bayesian games where the players' types are different valuations for the items. We study the cases of 2 to 4 items and between 3 and 15 points for the players to assign. Independently for every type, the valuations for each item were drawn from independent uniform distributions on $[0, 1]$, and then normalized to sum up to 1.

Multi-player Polymatrix Games

We study several types of game. For each, we study a range of underlying graphs: complete graphs, cycles, stars, and grid graphs. In each case the entries of the payoff matrices are drawn from independent uniform distributions on $[0, 1]$.

Net coordination games. In these games, every edge e corresponds to a coordination bimatrix game (A_e, A_e). These games posses a pure NE, which is PLS-complete to compute. The complexity of finding a (possibly non-pure) exact equilibrium is in $\texttt{PLS} \cap \texttt{PPAD}$ [12].

Coordination/zero-sum games. Here each edge is either a coordination or zero-sum game, i.e., on edge e the bimatrix game is (A_e, A_e), or $(A_e, -A_e)$. These games are PPAD-complete [12] to solve. We create games having a proportion p of coordination games for $p \in \{0, 0.25, 0.5, 0.75, 1\}$. We study how p affects the running time of the algorithms.

Group-wise zero-sum games. The players are partitioned into groups so that the edges going between groups are zero-sum while those within the same group are coordination games. In other words, players inside a group are "friends" who want to coordinate their actions, while players in different groups are competitors. These games are PPAD-complete [12] to solve even with 3 groups. We create games with 2 and 5 groups, all played on complete graphs. In every case, each group is approximately the same size, and each player is assigned to a group at random.

Strictly competitive games. A bimatrix game is strictly competitive if for every pair of mixed strategy profiles s and s' we have that: if the payoff of one player is better in s than in s', then the payoff of the other player is worse in

s than in s'. We study polymatrix games with strictly competitive games on the edges, which are PPAD-complete [12].

Weighted cooperation games. The unweighted version of these games was introduced in [3]. Each player chooses a colour from a set of available colours. The payoff of a player is the number of neighbours who choose the same colour. These games have a pure NE that can be computed in polynomial time. We study the more general case where each edge has a positive weight. The complexity for the weighted case is unknown [19]. We create games where all players have the same number of available colors k, where k is in $\{15, \ldots, 45\}$. For every player, his available colors are chosen uniformly at random from all k-sized subsets from a universe of colors of size either $2k$ or $5k$.

5. EXPERIMENTAL SETUP

The algorithms and game generators were implemented in C. The CPLEX library was used for solving LPs in the implementation of DESCENT. Our implementation of LEMKE uses integer pivoting in exact arithmetic using the GMP library; we were unable to produce a numerically stable floating point implementation (generally our attempts would start to fail on LCP instances of dimension 60). All experiments had a time-out of 10 minutes. In our results, the average runtime of the algorithms, as well as the approximation guarantee found, include the instances which timed out. The experiments used a cluster of 8 machines with Intel Core i7-2600 CPU's clocked at 3.40GHz and 16GB of memory, running Scientific Linux 6.6 with kernel version 2.6.32.

6. RESULTS

We ran both LEMKE and DESCENT on all of our input instances. Table 1 shows the results for auctions, Table 2 shows the results for other Bayesian two-player games, and Table 5 shows the results for multi-player polymatrix games. While we tested games of many different sizes, for the purposes of exposition, the tables display the largest instances that LEMKE can solve without timing out.

One general feature of our results is that DESCENT is much faster than LEMKE. To illustrate this, Figure 1 shows the performance of the two algorithms on the three types of additive item-bidding auctions included in the study. It can be seen that on the hard instances (first price and all pay), LEMKE starts to struggle when there are around 5 million payoffs in the game, whereas even the slower and more accurate of the two DESCENT variants ($\delta = 0.001$) can handle games with 30 million payoffs in under a minute. Indeed, a runtime regression for Bayesian Blotto games found that LEMKE has roughly quadratic running time (with an R^2 of 0.75 for the regression), while DESCENT has roughly linear running time (with R^2s of 0.88 and 0.96 for the δ values of 0.1 and 0.001 respectively).

However, good runtime performance for DESCENT would be of limited value if it only found poor quality approximate equilibria. Fortunately, our results show that this is not the case. In almost all experiments DESCENT found high quality approximate equilibria. The variant with $\delta = 0.1$ typically found an ϵ-NE with $\epsilon \leq 0.05$, while the variant with $\delta = 0.001$ typically found an ϵ-NE with $\epsilon \leq 0.002$.

Figure 2 shows a box and whisker plot for the quality of approximate equilibrium found by accurate DESCENT variant. It can be seen that even the worst performance of the algorithm is relatively good for several classes of game. The overall worst approximate equilibrium was a 0.1065-NE that was found on a weighted cooperation game. While this is far larger than the average performance, it is still much better than best-known theoretical upper bound of 0.5.

We now make more detailed observations about the specific classes of games that we tested. For auctions, one interesting observation is that on certain classes of auctions LEMKE will often find a pure Nash equilibrium. This is shown in the "% Pure column" of Table 1. This phenomenon is particularly prevalent for second-price auctions, where in some cases we found that LEMKE always finds a pure, and in these cases it does so in a very small amount of time.

We also found that the tie-breaking rule used in the auction can have a huge impact on the time that LEMKE takes to find an exact equilibrium. Table 3 shows the performance of the algorithm on otherwise identical auctions with different tie-breaking rules. It can be seen that resolving ties deterministically makes the game much easier to solve than resolving ties randomly.

Finally, we discuss the results in Table 4 for DESCENT with and without line search. It can be seen that, without line search, the DESCENT algorithm with $\delta = 0.001$ is often slower than LEMKE, and that using line search greatly speeds it up (and so in our other results we always used line search). Interestingly, the line-search variant of the algorithm also finds better quality approximate equilibria; it would be interesting to understand why.

7. CONCLUSIONS

In this paper we extensively studied the performance of two algorithms for computing a sample equilibria of polymatrix games. Both algorithms produce good results for most of the test instances, even though many were drawn from theoretically hard classes. More specifically, we saw that combinatorial auctions with two bidders are relatively easy to solve. This raises the natural question whether we can derive efficient algorithms for auctions with two, or a constant number of players. Furthermore, we saw that tie resolution significantly affects the difficulty of the auctions (see [13] for a discussion of this issue in a theoretical context).

In all of our experiments DESCENT produced ϵ-NE far better than the (best-known) 0.5 theoretical worst-case guarantee, which is not known to be tight. So it would be interesting to understand if this good performance is due to the nature of the games we studied or if there is a better theoretical analysis. In [25], a genetic algorithm was used to construct a bimatrix game for which the Tsaknakis and Spirakis (TS) algorithm computes an 0.3393-NE, which shows the analysis of the TS algorithm is essentially tight. Since bimatrix games are a special case of polymatrix games, this gives a lower bound of 0.03393 for the best-possible approximation guarantee for DESCENT in polymatrix games. Can a better lower bound, closer to the 0.5 upper bound, be found? We believe that it should be easier to construct a bad game for the DESCENT algorithm compared to the TS algorithm, because DESCENT computes a single strategy profile, whereas TS computes three profiles (one by descent, and then two further profiles are derived from that one) and then chooses the best one.

Acknowledgements. The last author is supported by EPSRC grant EP/L011018/1.

Games				Lemke			Descent 0.1 LS		Descent 0.001 LS	
	Valuation	Avg. Size	Auc	Time	% Timeout	% Pure	Time	ϵ	Time	ϵ
Itembidding	Additive	623990	FP	35.005	0.0	0.0	0.200	1.133e-02	1.287	2.630e-04
			SP	0.764	0.0	100.0	0.233	7.103e-03	0.232	2.312e-05
			AP	214.049	12.0	0.0	0.289	6.163e-03	1.610	1.943e-04
	Unit	650417	FP	46.351	0.0	34.0	0.199	6.249e-02	3.336	3.063e-02
			SP	203.107	19.0	58.0	0.459	1.442e-02	1.831	2.647e-04
			AP	14.709	0.0	21.0	0.168	4.182e-02	2.757	1.113e-02
	AndOr	519055	FP	280.322	23.0	0.0	0.194	1.927e-02	1.732	1.594e-04
			SP	1.269	0.0	100.0	0.279	7.026e-03	0.595	1.019e-04
			AP	166.011	9.1	0.0	0.171	8.662e-03	1.328	2.842e-04
	Budget	647583	FP	100.444	5.0	3.16	0.206	3.055e-02	2.003	4.064e-03
			SP	9.438	1.0	84.85	0.348	2.543e-02	0.915	2.186e-04
			AP	248.739	24.0	0.0	0.271	1.990e-02	2.199	2.233e-03
	SingleMinded	606511	FP	82.166	7.37	5.0	0.139	2.775e-02	1.551	1.822e-03
			SP	110.341	17.0	48.19	0.475	1.045e-02	1.359	1.669e-04
			AP	59.942	3.0	1.0	0.162	1.856e-02	1.544	1.038e-03
Multiunit	Additive	836465	D	9.504	0.0	30.0	0.295	1.452e-02	1.481	4.411e-04
			U	0.954	0.0	100.0	0.380	1.321e-02	1.870	3.579e-04
			AP	512.564	64.0	0.0	0.417	4.080e-03	2.182	3.170e-04
	SubModular	878491	D	29.054	0.0	5.0	0.270	1.326e-02	1.954	3.209e-04
			U	5.818	0.0	83.0	0.272	2.636e-02	1.741	8.645e-04
			AP	210.290	12.0	0.0	0.323	1.115e-02	2.192	3.273e-04

Table 1: Results for item bidding and multi-unit auctions with 3 items and 3 types per player. Ties are broken by favouring the second player for item bidding, and by allocating the item uniformly at random in the multi-unit case. For Lemke, we report the average running time, the percentage of instances that exceeded our timeout of 10 minutes, and the percentage of instances for which the algorithm finds a pure equilibrium. For Descent, we report the average running time and the approximation quality of the approximate equilibrium that was found.

Games			Lemke		Descent 0.1 LS		Descent 0.001 LS		
Game	# Types	# Points/Troops	Time	% Timeout	Time	ϵ	Time	ϵ	% Timeout
AdjWinner	30	5	281.407	21.0	0.939	4.450e-02	3.065	7.469e-03	0.0
AdjWinner	3	60	233.963	10.0	138.584	2.683e-02	220.254	1.827e-03	10.0
Blotto	3	8	71.814	0.0	0.032	9.181e-03	0.480	5.281e-04	0.0
Blotto	3	10	382.497	23.0	0.063	8.859e-03	0.845	5.408e-04	0.0
Blotto	3	12	573.663	91.0	0.118	8.590e-03	1.392	6.268e-04	0.0

Table 2: Results for Adjusted Winner and Blotto games. The two rows for Adjusted Winner show similar running times but actually correspond to very different input sizes, with the second row corresponding to much larger games. The underlying reason is that the number of players in the polymatrix game (i.e., number of types) affects the running time much more that the number of actions (i.e., number of items/troops). Also see Appendix B in the full version of this paper [21].

Auc	Player 1		Random	
	Time	% Timeout	Time	% Timeout
FP	63.411	0.0	408.223	43.0
SP	3.663	0.0	39.727	3.0
AP	100.949	4.0	248.665	19.0

Table 3: Results for Lemke showing the impact of the tie-breaking rule. We report on first price (FP), second price (SP) and all-pay (AP) auctions with budget additive valuations, 3 items, and 5 types per player. In the first two columns, all tied items are allocated to player 1, while in the last two, tied items are allocated uniformly at random.

	Lemke	Descent 0.001		Descent LS 0.001	
	Time	Time	ϵ	Time	ϵ
FP	229.4	508.0	5.4e-03	4.894	6.768e-04
SP	1.6	470.8	3.7e-03	0.491	1.013e-05
AP	547.3	496.0	6.5e-03	5.551	3.283e-04

Table 4: Results showing the impact of line search for Descent. We report results for first price (FP), second price (SP), and all-pay (AP) auctions for additive bidders. Lemke timed out on 13% and 44.5% of FP and AP auctions respectively, while Descent without line search times out on 61.5%, 56 % and 94% of instances on the respective auctions.

Games					Lemke		Descent 0.1 LS		Descent 0.001 LS		
Game	Graph	# Payoff	LCP	p	Time	% T	Time	ϵ	Time	ϵ	% T
Coord-Zero	Complete	26010	32400	0	1.270	0.0	0.034	2.103e-02	0.760	9.951e-04	0.0
				0.25	63.407	4.0	0.033	2.115e-02	0.748	1.026e-03	0.0
				0.5	337.443	45.0	0.034	1.859e-02	0.750	1.070e-03	0.0
				0.75	522.207	74.0	0.033	1.604e-02	0.725	1.076e-03	0.0
				1	116.354	0.0	0.034	4.844e-03	0.598	5.087e-04	0.0
	Cycle	25920	136900	0	18.430	2.0	0.103	3.352e-02	3.612	1.093e-03	0.0
				0.25	184.451	21.0	0.103	3.157e-02	3.534	1.167e-03	0.0
				0.5	412.947	55.0	0.105	2.859e-02	3.430	1.136e-03	0.0
				0.75	593.414	96.0	0.103	2.626e-02	3.206	1.121e-03	0.0
				1	600.097	100.0	0.107	1.906e-02	2.712	6.557e-04	0.0
	Grid	26136	93636	0	35.447	3.0	0.072	3.143e-02	2.257	1.023e-03	0.0
				0.25	260.455	35.0	0.069	3.239e-02	2.233	1.137e-03	0.0
				0.5	451.699	61.0	0.072	2.955e-02	2.254	1.170e-03	0.0
				0.75	552.286	82.0	0.072	2.786e-02	2.106	1.186e-03	0.0
				1	599.349	99.0	0.070	2.159e-02	1.802	6.489e-04	0.0
	Tree	25992	152100	0	0.276	0.0	0.060	1.012e-02	0.818	1.175e-03	0.0
				0.25	0.542	0.0	0.062	1.997e-02	0.806	1.220e-03	0.0
				0.5	73.443	5.0	0.062	2.139e-02	0.814	1.246e-03	0.0
				0.75	165.418	4.0	0.061	2.150e-02	0.796	1.084e-03	0.0
				1	162.420	0.0	0.063	1.469e-03	0.778	7.686e-04	0.0
Group Zero	Complete	20250	25600	2	368.032	36.0	0.025	1.976e-02	0.564	1.093e-03	0.0
				3	495.919	66.0	0.025	1.762e-02	0.550	1.129e-03	0.0
				5	435.207	29.0	0.025	1.308e-02	0.525	9.926e-04	0.0
		26010	32400	2	438.650	59.0	0.034	1.855e-02	0.760	1.068e-03	0.0
				3	576.439	91.0	0.034	1.583e-02	0.731	1.120e-03	0.0
				5	582.924	88.0	0.033	1.186e-02	0.677	9.738e-04	0.0
		36000	44100	2	545.997	84.0	0.052	1.564e-02	1.073	1.049e-03	0.0
				3	598.616	99.0	0.051	1.396e-02	1.037	1.110e-03	0.0
				5	600.088	100.0	0.051	1.101e-02	0.969	9.721e-04	0.0
Strict	Complete	20250	25600	5	356.009	17.0	0.024	1.878e-02	0.552	1.054e-03	0.0
	Cycle	20480	108900	5	580.891	85.0	0.087	1.729e-02	2.102	1.068e-03	0.0
	Grid	20184	72900	5	551.795	77.0	0.066	1.612e-02	1.428	1.108e-03	0.0
	Tree	20808	122500	5	79.560	0.0	0.048	2.571e-03	0.664	8.111e-04	0.0
Weighted Cooperation	Complete	995000	1440000	2	194.233	8.0	8.455	1.446e-02	86.116	9.603e-04	0.0
				3	410.118	38.0	6.551	1.909e-02	73.485	1.047e-03	0.0
				5	552.583	81.0	4.957	2.585e-02	64.676	1.130e-03	0.0
	Cycle	17500	4410000	2	103.403	0.0	0.227	1.334e-01	577.984	3.094e-02	73.0
				3	90.062	0.0	0.172	1.412e-01	529.581	4.199e-02	48.0
				5	78.883	0.0	0.156	1.427e-01	438.707	3.950e-02	28.0
	Grid	27200	3006756	2	116.157	0.0	0.864	8.298e-02	275.839	5.461e-03	1.0
				3	81.933	0.0	0.464	1.110e-01	480.608	1.368e-02	15.0
				5	58.054	0.0	0.131	1.384e-01	358.662	3.028e-02	2.0
	Tree	24950	9000000	2	240.215	0.0	0.750	0.000e+00	2.768	3.253e-04	0.0
				3	220.510	0.0	0.709	0.000e+00	2.533	1.194e-04	0.0
				5	204.919	0.0	0.653	0.000e+00	2.345	8.947e-05	0.0

Table 5: Results for multi-player (non-Bayesian) polymatrix games. The underlying graphs are complete graphs, cycles, grids and star graphs. $\%T$ is the proportion of the timed out instances. On Cooperation-Zerosum games, the value of p represents the proportion of games which are coordination games, for group zero-sum games, it represents the number of groups, and for weighted cooperation games, it represents the multiplier dictating the total number of colours available, i.e. if there are k colours per player, then there are $k \cdot p$ total colours available.

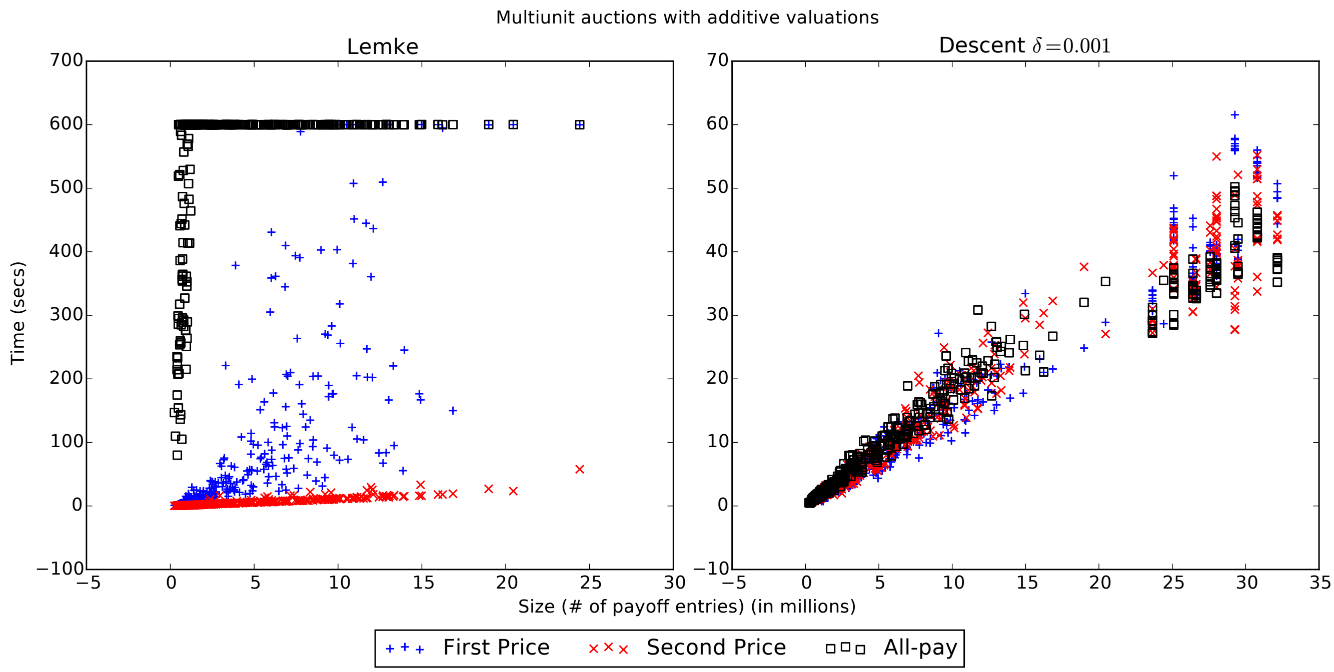

Figure 1: Plots showing the performance of the algorithms on multi-unit auctions with first price, second price, and all-pay payment rules. The left plot shows the performance of LEMKE's algorithm. It can clearly be seen that the allocation rule impacts the performance of the algorithm. The right chart right shows the performance of DESCENT with $\delta = 0.001$. The y-axis scales on the two charts are not equal: DESCENT is much faster than LEMKE.

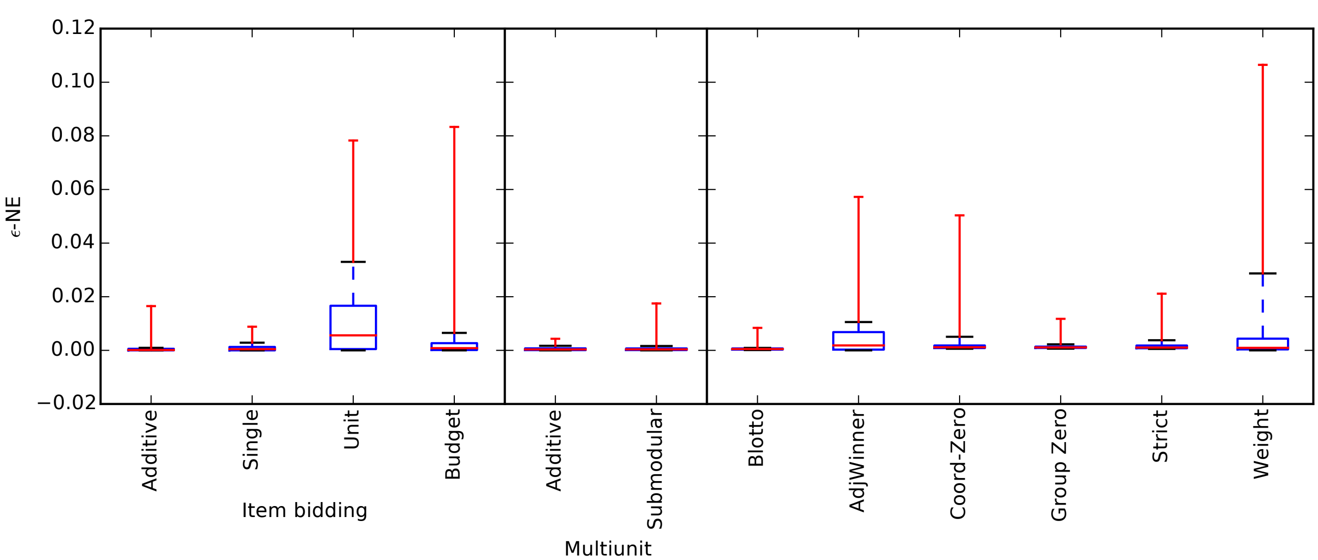

Figure 2: Box and whisker plots showing the approximation quality of the approximate equilibria found by DESCENT with $\delta = 0.001$. The results show that DESCENT almost always finds a high quality approximate equilibrium. It can be seen that on many classes of games, even the worst approximation quality over all test cases is very good.

REFERENCES

[1] T. Adamo and A. Matros. A Blotto game with incomplete information. *Economics Letters*, 105(1):100 – 102, 2009.

[2] A. Ahmadi, M. Hajiaghayi, B. Lucier, H. Mahini, and S. Seddighin. From duels to battlefields: Computing equilibria of Blotto and other games. In *Proc. of AAAI*, 2016.

[3] K. R. Apt, M. Rahn, G. Schäfer, and S. Simon. Coordination games on graphs. *CoRR*, abs/1501.07388, 2015.

[4] C. Audet, S. Belhaiza, and P. Hansen. Enumeration of all the extreme equilibria in game theory: Bimatrix and polymatrix games. *Journal of Optimization Theory and Applications*, 129(3):349–372, 2006.

[5] D. Avis, G. Rosenberg, R. Savani, and B. von Stengel. Enumeration of Nash equilibria for two-player games. *Economic Theory*, 42(1):9–37, 2010.

[6] S. Barman, K. Ligett, and G. Piliouras. Approximating nash equilibria in tree polymatrix games. In *Proc. of SAGT*, pages 285–296, 2015.

[7] N. A. R. Bhat and K. Leyton-Brown. Computing Nash equilibria of action-graph games. In *Proc. of UAI*, pages 35–42, 2004.

[8] K. Bhawalkar and T. Roughgarden. Welfare guarantees for combinatorial auctions with item bidding. In *Proc. of SODA*, pages 700–709, 2011.

[9] H. Bosse, J. Byrka, and E. Markakis. New algorithms for approximate Nash equilibria in bimatrix games. *Theoretical Computer Science*, 411(1):164–173, 2010.

[10] S. J. Brams. Fair division. In *Computational Complexity*, pages 1073–1080. Springer, 2012.

[11] P. Briest, P. W. Goldberg, and H. Röglin. Approximate equilibria in games with few players. *CoRR*, abs/0804.4524, 2008.

[12] Y. Cai and C. Daskalakis. On minmax theorems for multiplayer games. In *Proc. of SODA*, pages 217–234, 2011.

[13] Y. Cai and C. H. Papadimitriou. Simultaneous bayesian auctions and computational complexity. In *Proc. of EC*, pages 895–910, 2014.

[14] X. Chen, X. Deng, and S.-H. Teng. Settling the complexity of computing two-player Nash equilibria. *Journal of the ACM*, 56(3):14:1–14:57, 2009.

[15] X. Chen, D. Paparas, and M. Yannakakis. The complexity of non-monotone markets. In *Proc. of STOC*, pages 181–190, 2013.

[16] G. Christodoulou, A. Kovács, and M. Schapira. Bayesian combinatorial auctions. In *Proc. of ICALP*, pages 820–832, 2008.

[17] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.

[18] C. Daskalakis, A. Mehta, and C. H. Papadimitriou. A note on approximate Nash equilibria. *Theoretical Computer Science*, 410(17):1581–1588, 2009.

[19] C. Daskalakis and C. H. Papadimitriou. Continuous local search. In *Proc. of SODA*, pages 790–804, 2011.

[20] C. Daskalakis, G. Schoenebeck, G. Valiant, and P. Valiant. On the complexity of Nash equilibria of action-graph games. In *Proc. of SODA*, pages 710–719, 2009.

[21] A. Deligkas, J. Fearnley, T. P. Igwe, and R. Savani. An Empirical Study on Computing Equilibria in Polymatrix Games. *CoRR*, abs/1602.06865, 2016.

[22] A. Deligkas, J. Fearnley, R. Savani, and P. Spirakis. Computing approximate Nash equilibria in polymatrix games. *Algorithmica*, pages 1–28, 2015.

[23] S. Dobzinski, H. Fu, and R. D. Kleinberg. On the complexity of computing an equilibrium in combinatorial auctions. In *Proc. of SODA*, pages 110–122, 2015.

[24] K. Etessami and M. Yannakakis. On the complexity of Nash equilibria and other fixed points. *SIAM J. Comput.*, 39(6):2531–2597, 2010.

[25] J. Fearnley, T. P. Igwe, and R. Savani. An empirical study of finding approximate equilibria in bimatrix games. In *Proc. of SEA*, pages 339–351, 2015.

[26] U. Feige and I. Talgam-Cohen. A direct reduction from k-player to 2-player approximate Nash equilibrium. In *Proc. of SAGT*, pages 138–149, 2010.

[27] M. Feldman, H. Fu, N. Gravin, and B. Lucier. Simultaneous auctions are (almost) efficient. In *Proc. of STOC*, pages 201–210, 2013.

[28] S. Govindan and R. Wilson. Computing Nash equilibria by iterated polymatrix approximation. *Journal of Economic Dynamics and Control*, 28(7):1229–1241, 2004.

[29] S. Govindan and R. Wilson. A decomposition algorithm for n-player games. *Economic Theory*, 42(1):97–117, 2010.

[30] S. Hémon and M. Santha. Approximate Nash equilibria for multi-player games. In *Proc. of SAGT*, pages 267–278, 2008.

[31] J. Howson, Joseph T. and R. W. Rosenthal. Bayesian equilibria of finite two-person games with incomplete information. *Management Science*, 21(3):pp. 313–315, 1974.

[32] A. X. Jiang, K. Leyton-Brown, and N. A. R. Bhat. Action-graph games. *Games and Economic Behavior*, 71(1):141–173, 2011.

[33] D. Kovenock and B. Roberson. A Blotto game with multi-dimensional incomplete information. *Economics Letters*, 113(3):273 – 275, 2011.

[34] C. E. Lemke. Bimatrix equilibrium points and mathematical programming. *Management Science*, 11(7):pp. 681–689, 1965.

[35] C. E. Lemke and J. Howson, J. T. Equilibrium points of bimatrix games. *Journal of the Society for Industrial and Applied Mathematics*, 12(2):pp. 413–423, 1964.

[36] D. A. Miller and S. W. Zucker. Copositive-plus lemke algorithm solves polymatrix games. *Operations Research Letters*, 10(5):285 – 290, 1991.

[37] J. Nash. Non-cooperative games. *The Annals of Mathematics*, 54(2):286–295, 1951.

[38] E. Nudelman, J. Wortman, Y. Shoham, and K. Leyton-Brown. Run the gamut: A comprehensive approach to evaluating game-theoretic algorithms. In *Proc. of AAMAS*, pages 880–887, 2004.

[39] G. Polevoy, S. Trajanovski, and M. de Weerdt. Nash

equilibria in shared effort games. In *Proc. of AAMAS*, pages 861–868, 2014.

[40] R. Porter, E. Nudelman, and Y. Shoham. Simple search methods for finding a Nash equilibrium. In *Proc. of AAAI*, pages 664–669, 2004.

[41] T. Roughgarden. The price of anarchy in games of incomplete information. In *Proc. of EC*, pages 862–879, 2012.

[42] A. Rubinstein. Inapproximability of Nash equilibrium. In *Proc. of STOC*, pages 409–418, 2015.

[43] T. Sandholm, A. Gilpin, and V. Conitzer. Mixed-integer programming methods for finding Nash equilibria. In *Proc. of AAAI*, pages 495–501, 2005.

[44] Y. Shoham and K. Leyton-Brown. Protocols for multiagent resource allocation: Auctions. In *Multiagent Systems*, pages 315–366. Cambridge University Press, 2008. Cambridge Books Online.

[45] H. Tsaknakis and P. G. Spirakis. An optimization approach for approximate Nash equilibria. *Internet Mathematics*, 5(4):365–382, 2008.

[46] H. Tsaknakis, P. G. Spirakis, and D. Kanoulas. Performance evaluation of a descent algorithm for bi-matrix games. In *Proc. of WINE*, pages 222–230, 2008.