Exclusion Method for Finding Nash Equilibrium in Multiplayer Games

Kimmo Berg
Aalto University School of Science
Dept. of Mathematics and Systems Analysis
kimmo.berg@aalto.fi

Tuomas Sandholm
Carnegie Mellon University
Computer Science Department
sandholm@cs.cmu.edu

Abstract

We present a complete algorithm for finding an ϵ-Nash equilibrium, for arbitrarily small ϵ, in games with more than two players. The method improves the best-known upper bound with respect to the number of players n, and it is the first implemented algorithm, to our knowledge, that manages to solve all instances. The main components of our tree-search-based method are a node-selection strategy, an exclusion oracle, and a subdivision scheme. The node-selection strategy determines the next region (of the strategy profile probability vector space) to be explored—based on the region's size and an estimate of whether the region contains an equilibrium. The exclusion oracle provides a provably correct sufficient condition for there not to exist an equilibrium in the region. The subdivision scheme determines how the region is split if it cannot be excluded. Unlike the well-known incomplete methods, our method does not need to proceed locally, which avoids it getting stuck in a local minimum—in the space of players' regrets—that may be far from any actual equilibrium. The run time grows rapidly with the game size; this reflects the dimensionality of this difficult problem. That suggests a hybrid scheme where one of the relatively fast prior incomplete algorithms is run, and if it fails to find an equilibrium, then our method is used.

Introduction

Unlike in single-agent settings, in games, the best action for a player may depend on what actions the other player(s) choose. Solution concepts from noncooperative game theory are sound definitions of rationality in such settings. However, to operationalize those concepts, one needs to develop algorithms for finding solutions that satisfy the definition of the concept.

The seminal solution concept in noncooperative game theory is *Nash equilibrium* (Nash 1950). It plays an important role in analyzing game-theoretic situations, and has transformed economics and other fields of science.

Finding a Nash equilibrium is an important, interesting, and well-studied problem (Herings and Peeters 2010; Daskalakis, Goldberg, and Papadimitriou 2009). Finding (even an approximate) Nash equilibrium in a two-player general-sum game or in a multiplayer game is PPAD-complete (Chen, Deng, and Teng 2009; Daskalakis 2013; Rubinstein 2015; 2016). Furthermore, the multiplayer games are FIXP-complete (Etessami and Yannakakis 2010) and the query complexity has been examined in Babichenko (2014). In contrast, a correlated equilibrium can be computed efficiently (Papadimitriou and Roughgarden 2005; Jiang and Leyton-Brown 2011). In larger games, it is not only the computation time that matters but also the memory requirements of storing the players' payoffs, which grow exponentially in the number of players in normal-form games; classes of games have been introduced where the payoffs can be compactly represented (Jiang, Leyton-Brown, and Bhat 2011). Two-player zero-sum games can be solved in polynomial time.

Multiplayer games differ significantly from two-player games. In a two-player game, the set of equilibria can be defined by, and found using, linear equations (with binary and continuous variables (Sandholm, Gilpin, and Conitzer 2005)). In contrast, modeling equilibria in multiplayer games involves nonlinear polynomial equations.

Finding a Nash equilibrium in multiplayer games is known to be more challenging in practice than in two-player games. It has also received much less attention. That said, several techniques have been proposed:

- *Homotopy* (path-following) methods (Herings and van den Elzen 2002; Govindan and Wilson 2003; 2004; McKelvey and Palfrey 1995; Turocy 2005) find an equilibrium in an easy artificial game first and then transform that problem continuously to the original problem while tracking how the equilibrium changes during this process. The transformation means that a nonlinear homotopy path is traced numerically.
- *Polynomial equation solving and support enumeration* methods (Porter, Nudelman, and Shoham 2008; Lipton and Markakis 2004) are based on solving a system of nonlinear polynomial equations that define the equilibrium, and may enumerate and go through all the possible supports of the players' mixed strategies.
- *Function minimization* methods (Sandholm, Gilpin, and Conitzer 2005; Chatterjee 2009; Boryczka and Juszczuk 2013; Buttler and Akchurina 2013) provide optimization formulations and algorithms for finding a Nash equilibrium. The methods may use a mixed-integer programming

formulation, where some of the variables are binary and they indicate whether a given pure strategy is in the support of the mixed-strategy equilibrium. The algorithms are typically based on tree search, and they branch on the binary variables. That is in stark contrast to the tree-search-based algorithm that we develop, which is based on splitting the *continuous* probability space.

- *Simplicial subdivision* methods (van der Laan, Talman, and van der Heyden 1987) construct a triangulated mesh for the search space and try to find a completely labeled simplex in the mesh. The triangulation is adaptively refined and the algorithm can be restarted from the previously found simplex. The algorithm moves locally from one simplex to another, which may require a long path in a highly-refined mesh. A Nash equilibrium need not be close to the found simplex on any given triangulation.
- *Uniform-strategy enumeration* methods (Lipton, Markakis, and Mehta 2003; Hémon, de Rougemont, and Santha 2008; Babichenko, Barman, and Peretz 2014) search exhaustively for an approximate equilibrium over all k-uniform strategies, that is, strategies where all the probabilities are integer multiples of $1/k$, for a given k.

Some of the above methods require drastic modifications when moving from the two-player setting to multiple players. For example, the homotopy method of Herings and van den Elzen (2002) uses piecewise linearization to handle the nonlinearities in the tracing procedure, and Ganzfried and Sandholm (2010) generalize the mixed-integer program formulation of Sandholm et al. (2005) using piecewise linearization.

The homotopy methods that use the global Newton method do not converge globally (McKelvey and McLennan 1996). Govindan and Wilson (2004) observe that the iterated polymatrix approximation method typically converges globally but is not failsafe and may get stuck in some games. They find that the problem with homotopy methods is that they need to traverse nonlinear paths and require many small steps in order to obtain reasonable accuracy. They also observe that the homotopy path may have many twists and reversals. Goldberg et al. (2013) construct examples where homotopy methods will not only need an exponential number of pivots but also an exponential number of direction reversals. Herings and van den Elzen (2002) and Herings and Peeters (2010) present a globally convergent homotopy method but note that the triangulations must have very refined mesh and the homotopy path must be traced numerically.

In principle, one can find all equilibria since the nonlinear equations are polynomials (Herings and Peeters 2005; Datta 2010). The idea is to enumerate all supports, solve all roots of the polynomial equations, and select the solutions that correspond to probability distributions (Turocy 2008). The methods of finding all equilibria are probabilistic, that is, they will find all solutions with given probability when they are run for at least some amount of time (which depends on the probability) (Herings and Peeters 2005). There are exponentially many supports in the game and there can be exponentially many equilibria (McLennan 2005; von Stengel 2012). Moreover, the homotopy methods (global Newton, tracing procedure, or quantal response method) are not guaranteed to find all equilibria (Turocy 2010).

Lipton et al. (2003) showed that any Nash equilibrium can be approximated with some ϵ-Nash where the players use supports with a small number of pure strategies; see also Hemon et al. (2008). Babichenko et al. (2014) improved the bounds and presented a complete method for finding an ϵ-Nash equilibrium in multiplayer games that has so far the best-known upper bound with respect to the number of strategies m and the number of player n. Their bound $m^{\log m}$ is tight up to a constant if it takes exponential time to solve PPAD-hard problems (Rubinstein 2016). The method enumerates all k-uniform strategy profiles among which an ϵ-Nash equilibrium must exist. The k-uniform strategies are mixed strategies that assign to each pure strategy a rational probability with denominator k. The main advantage of using k-uniform strategies is that this strategy space is relatively small compared to the original strategy space. Lipton and Markakis (2004) propose a method that has the best-known upper bound with respect to the accuracy ϵ. However, none of those methods have been implemented, as far as we know. We implement a method based on uniform strategies and make a comparison with our method. We also test all the GAMBIT multiplayer algorithms and find that our exclusion method is the first implemented algorithm that can find approximate equilibrium in all instances.

We present a complete tree-search-based method that improves the best-known upper bound with respect to the number of players n; see the 2-page early version published as Berg and Sandholm (2016). We improve the bounds of Babichenko et al. (2014) in n and ϵ, but our method is worse in the number of actions m. Moreover, our method is better in n compared to Lipton and Markakis (2004), equal in m, and worse in ϵ. Babichenko et al. show that the query complexity of the problem is exponential in n (as is our run-time bound),if $\epsilon = O(1/n)$; so, under that condition, our run time is optimal in n.

Our method divides the search space into smaller regions and examines whether an equilibrium cannot exist in the region. The regions are explored in an order given by a ranking function. This means that the regions are not examined locally, as in the simplicial subdivision method; this prevents our algorithm from getting stuck in local minima—in the space of players' regrets—that may be far away from any actual Nash equilibrium. The regions are either excluded by the oracle or further subdivided into smaller regions. The proposed oracle never excludes a region that contains an equilibrium. Moreover, any point with positive regret can be excluded when the surrounding region is small enough. Our method keeps removing regions based on points with positive regret, and is guaranteed to find an ϵ-Nash equilibrium, for arbitrarily small ϵ, in finite time that depends on ϵ.

Normal-form games

A normal-form game can be defined as a tuple $G = (N, A, u)$, where $N = \{1, \ldots, n\}$ is the set of players and $A_i = \{a_1, \ldots, a_m\}$ is the set of pure actions for player i. For simplicity, we assume that all the players have the same number of actions m. The function $u_i : A \mapsto \mathbf{R}$ gives player i's payoff, where the set of pure action profiles is $A = A_1 \times \ldots \times A_n$.

The players may use mixed strategies, that is, randomize among the pure actions. Each player i assigns a probability $p_i(a_j) \geq 0$ for each pure action $a_j \in A_i$ such that $\sum_{k=1}^{m} p_i(a_k) = 1$. The mixed strategy profile is denoted by $p = (p_1, \ldots, p_n)$. For a given strategy p, the players' payoffs are $u_i(p) = \sum_{a \in A} p(a) u_i(a)$, where $p(a) = \prod_{i \in N} p_i(a_i)$. Player i's opponents' strategies are denoted by $p_{-i} = (p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n)$.

A strategy p^* is a Nash equilibrium if $u_i(a_j, p^*_{-i}) \leq u_i(p^*_i, p^*_{-i})$ for all $a_j \in A_i$ and $i \in N$. This means that no player has incentive to deviate when the other players stick to their equilibrium strategies. At least one Nash equilibrium always exists in these finite games (Nash 1950).

For each strategy p, we can define player i's regret of action a_j as $r_i(a_j, p) = u_i(a_j, p_{-i}) - u_i(p)$. The regret of player i is then $r_i(p) = \max_{a_j \in A_i} r_i(a_j, p)$. Moreover, the regret in the game is

$$r(p) = \max_{i \in N} r_i(p). \tag{1}$$

A strategy p^* is a Nash equilibrium if the players have no regret when playing their strategies, that is, $r(p^*) = 0$. A strategy p^* is an ϵ-Nash equilibrium if $r(p^*) \leq \epsilon$. Thus, the computation of an equilibrium can be formulated as finding a root of the regret function. In general, the regret function is continuous, nonlinear, and piecewise polynomial in the players' probabilities. The regret function $r(p)$ need not be differentiable, but each component $r_i(a_j, p)$ is differentiable. We reduce the dimension of the search space by noting that the probabilities sum up to one for each player. The reduced search space of player i is $p'_i = (p_i(a_1), \ldots, p_i(a_{m-1}))$ and $p_i(a_m) = 1 - \sum_{k=1}^{m-1} p_i(a_k)$ for each $i \in N$. Thus, the variables of the reduced search space are $p' = (p'_1, \ldots, p'_n)$. The reduced search space is denoted by P' and its dimension is $d' = dim(P') = (m-1)n$.

Tree-search-based method

We propose a tree-search-based method for finding a Nash equilibrium in multiplayer games. It splits the search space P' into smaller regions, until a solution is found, that is, when the regrets are small enough for all players. We denote the regions by $R_k, k \in K$, where K indexes the regions. The method either starts with the whole search space ($R_1 = P'$) or a collection of regions that cover the space ($P' \subseteq \cup R_k$). The regions are parts that do not overlap ($R_j \cap R_k = \emptyset$ for all $j, k \in K$, $j \neq k$). The three main components of the method are the following.

Region selection (node-selection strategy)

An important choice in the method is the order in which the regions are examined. In general, there can be an arbitrary function that computes various measures from each region and ranks them based on these measures. An ideal function would select a region that has high probability of containing a Nash equilibrium, and is easy to search through (e.g., small). The function should be fast to compute. We propose a ranking function $g(R, p^0)$ that uses the size of the region $d(R)$, the regret, and its gradient evaluated at one point $p^0 \in R$:

$$g(R, p^0) = \max_{i \in N} r_i(p^0)/(d(R) \cdot M_i(p^0)), \tag{2}$$

where

$$M_i(p^0) = \max_{a_j \in A_i^*(p^0)} \|\nabla r_i(a_j, p^0)\|_\infty \tag{3}$$

is the maximum derivative at p^0 of player i's regret over all the variables in p', $d(R) = \max_{q \in R} \|q - p^0\|$, and $A_i^*(p) = \{a_j \text{ s.t. } r_i(a_j, p) = \max_{a_k \in A_i} r_i(a_k, p)\}$. The norm $\|\cdot\|$ is the Euclidean norm throughout the paper. Note that the region with a lower value of g is better. The following example demonstrates what $M_i(p^0)$ consists of.

Example 1. *Let us examine a two-player game with two actions. Let the payoffs of player 1 be given in matrix A, where A_{ij} contains the payoff when player 1 plays i-th action and player 2 j-th action. The players' probabilities of playing their first actions are given by $p' = (p_1, p_2)$. We have $r_1(a_1, p) = (1 - p_1)[(A_{11} - A_{21})p_2 + (A_{12} - A_{22})(1 - p_2)]$. Now, $\nabla r_1(a_1, p) = (\frac{\partial r_1}{\partial p_1}(a_1, p), \frac{\partial r_1}{\partial p_2}(a_1, p))$ and $\frac{\partial r_1}{\partial p_1}(a_1, p) = -[(A_{11} - A_{21})p_2 + (A_{12} - A_{22})(1 - p_2)]$.*

In general, $r_i(a_i, p)$'s are polynomials of order n and $\nabla r_1(a_1, p)$'s are polynomials of order $n - 1$. Also, $\nabla r_1(a_1, p)$ is of dimension $d' = (m-1)n$ and $M_i(p^0)$ is maximum over $d'm = (m-1)mn$ components.

Exclusion oracle

The oracle tells if a Nash equilibrium cannot be in the region. If this is the case, the region can be excluded, and otherwise the region is subdivided further. For example, the exclusion oracle may compute the regret values of Eq. (1) in multiple points in the region and fit a piecewise linear model to these values. The model can be used to estimate how small regret values can be obtained in the region and where the minimum value for the regret is located. The minimum point may be used in determining how the region is subdivided. For example, the point may be added as a new vertex and the new regions can be formed by using some triangulation scheme.

In this paper, we compute the regret and its gradient only at a single point in the middle of the region for the oracle. This point is chosen so that it minimizes the maximum distance to any other point in the region. We will show later on how this point is determined given the subdivision scheme that we use. Beside the regret value, we determine a global upper bound for the gradient of the regret:

$$M_i^* = \max_{p \in P'} \max_{a_j \in A_i^*(p)} \|\nabla r_i(a_j, p)\|_\infty.$$

Note that this constant is determined by maximization over the continuous probability space P' over the maximum of nonlinear polynomials. We do not know how to determine this in practice and use $M_i^* = 2(\max_{a \in A} u_i(a) - \min_{a \in A} u_i(a))$, $i \in N$, which can be shown to be a global upper bound. This maintains the correctness of the oracle.

If the regret value in the middle point p^0 is large enough, a Nash equilibrium (i.e., the root of the regret function) cannot be in the region if the region and the global constant are small enough. The following result shows that any region containing a point with positive regret can be excluded if the region is small enough.

Theorem 1. *A p^0-centered ball with radius s cannot contain 0-Nash if*

$$r_i(p^0) > s \cdot M_i^*, \text{ for some } i \in N. \qquad (4)$$

Since M_i^* is bounded, there is always some sufficiently small region such that any point $p^0 \in P'$ with $r(p^0) \geq \epsilon$ can be excluded. The following result shows how dense grid (small enough diameter d of the region) is needed to exclude all the points that have large enough regret values.

Theorem 2. *If $r_i(p^0) \geq \epsilon$, for some $i \in N$, then the diameter $d < \frac{\epsilon}{2M_i^*}$ is small enough to exclude p^0.*

Proof. Pick any point $\hat{p}$ such that $\|\hat{p} - p^0\| < \frac{\epsilon}{2M_i^*}$. By Theorem 1, $\hat{p}$-centered ball with radius $\frac{\epsilon}{2M_i^*}$ cannot contain 0-Nash, and p^0 belongs to this ball. □

Note that the oracle could exclude more if the maximum derivative were computed over just the region in question. This way the upper bound for the region would be smaller than the derivative that is computed over the whole search space. As is computing the global bound, it is difficult to compute the bound over the region.

The following example illustrates the idea of exclusion based on the gradient.

Example 2. *We are trying to find a root $r(x) = 0$ for a differentiable multivariate function $r : \mathbf{R}^2 \mapsto \mathbf{R}$. Assume that we have evaluated the regret at $r((0.2, 0.2)) = 0.3$ and we know that the gradient $\|\nabla r(x)\| \leq 2$ for all x. Then by the mean value theorem, $|r(y) - r(x)| \leq M\|x - y\|$, where $M = \max \|\nabla r(z)\|$, where $z = \lambda x + (1-\lambda)y$ and $\lambda \in [0, 1]$. Thus, we have that $r(z) > 0$ for all $\|z - (0.2, 0.2)\| < 0.3/2$. The root of $r(x)$, i.e., a Nash equilibrium, cannot be inside the $(0.2, 0.2)$-centered ball of radius 0.15.*

Instead of selecting a region to be examined we could select a point and determine a ball around the point where a Nash equilibrium cannot be. Then we could exclude this ball from the search space. A possible problem with this idea is that we need to keep track of all the points and the radii, and select the new point so that we would find a Nash equilibrium in the remaining space. Thus, an efficient coding would be required for storing in memory all the excluded balls, how they intersect, and how to keep track of the regions that still need to be examined. Moreover, as the selected point gets closer to a Nash equilibrium, the regret values and the excluded balls get smaller—in fact, infinitesimal. This means that the method would slow down without bound as it approaches an equilibrium.

Subdivision of the region

There are many ways how the search space can be subdivided. For example, we may use any full-dimensional polytopes of dimension $d' = dim(P)$ in the search space. The simplest possible polytope in any given space is a simplex, which has the smallest number of vertices (i.e., $d' + 1$ vertices) over all such polytopes. For example, we can form a simplex in a normal-form game by selecting any $d'+1$ points in P' such that their convex hull makes a full-dimensional object. Moreover, the simplices can be subdivided in many ways: a new point may be added anywhere within the simplex and any triangulation scheme may be used in forming the new simplices. For example, we may make $d' + 1$ copies of the simplex and replace a different vertex by the new point in each of the new simplices. A possible problem with this procedure is that all the vertices need to be stored in memory. Also, it may not be easy to compute the point inside the simplex so that it would minimize the maximum distance within the simplex, and it may take time to compute the distance $d(R)$ for the region. Furthermore, it is difficult to tell what is a good way to triangulate the search space.

Instead of using simplices, we may use polytopes with more vertices and bigger volume. Since we are excluding regions that cannot contain a Nash equilibrium, we want to exclude regions that are as large as possible.

In this paper, we use hyperrectangles (i.e., boxes) which makes the memory requirement linear in dimension. We only need to store the minimum and the maximum values for each dimension and region. It is also easy to compute the middle point p^0 in the hyperrectangle and the maximum distance d to the corner points with the Pythagorean theorem:

$$p_x^0 = (u_x + l_x)/2,$$

$$d = 1/2 \cdot \sqrt{\sum_{x \in p'} (u_x - l_x)^2},$$

where l_x and u_x are the lower and upper bounds of the hyperrectangle in dimension x. The problem with this choice is that the original search space is not a hyperrectangle but a simplex. Thus, we exclude the regions that are completely outside the search space, i.e., when the lower bounds of the probabilities for the hyperrectangle sum to more than, or equal to, one: $\sum_{x \in p'_i} l_x \geq 1$ for any $i \in N$. Moreover, we automatically subdivide the region if the middle point p^0 is outside the search space.

We use a bisection method and split the hyperrectangle along the longest edge, that is, we find the dimension with $\max_{x \in p'} u_x - l_x$. Thus, the method forms a binary search tree. The algorithm is presented below.

Algorithm 1

```
Initialize the regions and compute the rankings with $g$.
    Compute the constants $M_i^*$, $i \in N$.
Repeat until any stopping condition is met
    1. Select the region with minimal value of $g$ in Eq. (2).
    2. Select $p^0$. If it is outside the search space, goto 3.
       Else compute $r(p^0)$. If Eq. (4) is satisfied,
       exclude the region and return to 1. Else goto 3.
    3. Bisect the region and compute $g$ for the new
       regions with $r(p^0)$ and $M_i(p^0)$ in Eq. (3).
```

We can stop when $r(p)$ is small enough. If we stop the algorithm when $r(p) \leq \epsilon$, we can bound the number of iterations needed. The result is based on enumerating all the regions where $r(p) \geq \epsilon$. The result holds for the bisection method with any region-selection heuristic.

Theorem 3. *Any bisection algorithm excludes all the points with $r(p) \geq \epsilon$ within $2^{(m-1)n\lceil \log\left(\frac{2M^*}{\epsilon}\right)/\log(2)\rceil} - 1$ iterations, where $M^* = \max_{i \in N} M_i^*$.*

Proof. Theorem 2 gives the diameter that is required to exclude a point with $r(p) \geq \epsilon$, which is $\frac{\epsilon}{2M^*}$. In each iteration, we bisect the longest edge which means that it takes $d' = (m-1)n$ iterations to bisect all the edges. The initial diameter is 1 as the probabilities are between 0 and 1. The number of bisections needed to reach the required diameter is $\lceil \log\left(\frac{2M^*}{\epsilon}\right) / \log(2) \rceil$. In a binary search tree, the total number of nodes up to depth x is $1+2+4+\ldots = 2^x - 1$. $\square$

To our knowledge, the best prior complete algorithms for the problem are by Lipton and Markakis (2004), which is polynomial in $\log 1/\epsilon$, and by Babichenko et al. (2014), which is $O(m^{\log m})$ and $O((\log n)^n)$. Specifically, the complexity of Lipton and Markakis (2004) is $poly(\log 1/\epsilon, n^{nm}, L)$, where L is the maximum bit size of the payoff data. Babichenko et al. (2014) proved two bounds for their method: m^{nk}, where $k > 8(\log m + \log n - \log \epsilon + \log 8)/\epsilon^2$ and $m = poly(n)$, and $(k+1)^{nm}$, when m is a constant. The former is better with respect to m, giving $O(m^{\log m})$, and the latter is better with respect to n and ϵ, giving $O((\log n)^n)$ and $O((\epsilon^{-2} \log 1/\epsilon)^c)$, where c is a constant. Our method is better than Babichenko et al. (2014) with respect to n and ϵ, since by Theorem 3 ours is $O(c^n)$ and $O(1/\epsilon^c)$. Ours is better than Lipton and Markakis (2004) with respect to n, as good (exponential) in m, and worse in ϵ. In summary, the best-known bounds are $O(c^n)$ by us, $O(m^{\log m})$ by Babichenko et al. (2014), and $O(\log 1/\epsilon)$ by Lipton and Markakis (2004).

Search algorithm over uniform strategies

As an improvement to the prior exhaustive complete method (Babichenko, Barman, and Peretz 2014), to get a method that we can compare ours to, we implemented an algorithm that goes through all the k-uniform strategies in the following way. The idea is to start with small k which means that the pure-strategy equlibria are found fast, and increase k if a good-enough solution has not been found.

Algorithm 2

```
Initialize the value $k$ and select constants $c > 1$ and $k^*$.
Repeat until $r(p) \leq \epsilon$ or $k = k^*$
    1. Compute $r(p)$ for all k-uniform strategies.
    2. Update $k = ck$.
```

We start the algorithm with $k = m$ and choose c and k^* so that we do only two repetitions. For example, we go through all the k-uniform strategies for $k = 3$ and $k = 18$ for 3-player 3-action games. This means $\binom{m+k-1}{k}^n = 6859000$ points in the probability space for $k = 18$ and $\epsilon = 10^{-3}$. Note that it is impossible to compute all the strategies for k in the range that the theoretical results of Babichenko et al. require; k would need to be $1.9 \cdot 10^7$ which means $5.7 \cdot 10^{42}$ points. We chose the k^* values so that it takes roughly 15 minutes to go through all the points, so that we can do enough repetitions. Our choice of k^* is not critical, since the results are similar for all $k^* \leq 30$ which is the range of values that can be computed in reasonable time.

Experiments

We tested the algorithms on randomly-generated games and on the games produced by the GAMUT generator (Nudelman et al. 2004). For random games, the payoffs were randomly drawn from a uniform distribution between zero and one. We ran $rep = 1000$ repetitions and set the stopping conditions as $r(p) \leq \epsilon = 10^{-3}$ and $maxtime = 900$ (seconds); for the other experiments, we did not use any time limits. The results are presented in Table 1. The Dim column shows the dimension of the search space p', and Time gives the average run time. OverTime% shows the percentage of instances that hit the time limit, and OverTime ϵ gives the average errors for the repetitions that hit the time limit. The runs were conducted on Intel Core i7-6500U at 2.50 GHz with 16 GB of RAM under 64-bit Windows 7. We implemented the algorithms in Matlab R2015b.

The results for GAMUT games are in Table 2; the results for all classes are available in the extended version. For these games, we used $\epsilon = 10^{-3}$, and no time limit. NotSolved% is the percentage of runs where $\epsilon > 10^{-3}$ for Algorithm 2. We can see big differences between the game classes. Polymatrix games are solved clearly slower than the other games. Note that our method solved all the instances to the given accuracy, while Algorithm 2 did not in some classes.

The results show that the run times grow rapidly as the search space dimension increases. Our method has higher average run time, but it finds an ϵ-Nash (for given small ϵ) on all instances. In contrast, Algorithm 2 is faster on average but has problems on some instances and ϵ can be high. Our method seems to work better on hard instances, i.e., ϵ is smaller for those instances that take a long time to solve. In many games, especially GAMUT games, equilibria are in pure strategies or can be found with small k; thus it is reasonable that Algorithm 2 works well on those easy instances. The run time of our method can be made equally fast as Algorithm 2's on those instances by first searching pure strategies and strategies in small support (but we did not want to design an algorithm that would take advantage of those properties that only hold for certain classes of games).

We also compared against the algorithms available in the well-known game-solving software package GAMBIT. We tested the GAMBIT algorithms on the GAMUT games; see Table 3. The algorithms are the homotopy method of (Govindan and Wilson 2003) (gnm), its modification using iterated polymatrix approximation (Govindan and Wilson 2004) (ipa), the polynomial equation solver (enumpoly), the simplicial subdivision method (van der Laan, Talman, and van der Heyden 1987) (simpdiv), a function minimization approach (liap), and the quantal response method (McKelvey and Palfrey 1995; Turocy 2005) (logit).[1] The numbers

[1]We used GAMBIT version 15.0 (except 16.0 for simpdiv as it had a bug in 15.0; only simpdiv changed from 15.0 to 16.0, so we reran only that algorithm when version 16.0 was recently released).

			Algorithm 1				Algorithm 2			
n	m	Dim	Time	Median	OverTime%	OverTime ϵ	k	Time	OverTime%	OverTime ϵ
3	2	3	0.04	0.02	0	-	2/180	49	1	0.0013
3	3	6	26	1.2	1	0.003	3/18	191	29	0.003
3	5	12	900	900	100	0.07	5	94	33	0.0059
4	2	4	99	0.48	8	0.0054	2/40	23	15	0.002
4	3	8	352	87	30	0.003	3/8	85	33	0.004
5	2	5	125	2.7	10	0.0057	2/8	1.0	30	0.0049
5	3	10	520	589	46	0.0022	3	7.9	36	0.0082

Table 1: Results for the randomly-generated general-sum games.

	Algorithm 1		Algorithm 2			
Game class	Time	95% bound	Time	NotSolved%	NotSolved Time	NotSolved ϵ
Bertrand oligopoly	13.7	19.3	0.01	0	-	-
Bidirectional LEG	159	337	0.013	0	-	-
Collaboration	2.8	3.7	0.0009	0	-	-
Congestion	29	71	0.027	0	-	-
Covariant r=-0.5	95	202	80	16	434	0.003
Polymatrix	172	358	27.2	7	373	0.003
Random LEG	880	1970	0.02	0	-	-
Random graphical	35000	35500	0.05	0	-	-
Uniform LEG	793	1850	0.02	0	-	-

Table 2: Results for three-player three-action GAMUT games.

Game class	gnm	ipa	enumpoly	simpdiv	liap	logit
Bertrand oligopoly	0.05 (30)	0.05 (75)	0.04 (50)	0.05	0.24 (99)	0.06
Bidirectional LEG	0.09 (0.3)	0.05 (58)	0.84 (1)	0.06 (0.1)	0.24 (99)	0.06 (0.1)
Collaboration	0.24 (0.1)	0.04	3.3 (50)	0.05	0.34 (99)	0.06 (0.3)
Congestion	0.05 (0.2)	0.05 (85)	0.05 (0.6)	0.05 (0.1)	0.21 (100)	0.05
Covariant r=-0.5	0.13 (3)	0.05 (94)	36	0.67 (2.8)	0.31 (100)	0.05 (1)
Polymatrix	0.06 (1)	0.04 (79)	0.04 (50)	0.07 (0.3)	0.3 (92)	0.05 (0.4)
Random LEG	0.05 (1)	0.04 (59)	8.1 (2)	0.05 (0.6)	0.24 (99)	0.06
Random graphical	0.08 (3)	0.04 (96)	6.3 (6)	0.17 (3)	0.31 (99)	0.06 (0.3)
Uniform LEG	0.07 (0.4)	0.05 (55)	0.04 (17)	0.05	0.23 (99)	0.06

Table 3: Computation times in seconds for the GAMBIT algorithms and percentage of instances not solved.

are the average computation times for the algorithms and the parentheses show the percentage of instances that were not solved (code got stuck, empty output, or accuracy not within the given ϵ). One can see that the algorithms are much faster than the complete methods, but all of them fail to solve some instances. Moreover, there are game classes (bidirectional LEG and polymatrix) where some instances are not solved by any of those algorithms. The best of those methods are logit, gnm and simpdiv.

Conclusions and future research

We introduced a complete method for finding an ϵ-Nash equilibrium, for any given ϵ, in normal-form multi-player games. It has the best run-time bound in n, improving prior results of Babichenko et al. (2014) with respect to n and ϵ. Moreover, the method is the first implemented algorithm, to our knowledge, that manages to find an approximate solution on all instances. The main components of our tree-search-based method are the node-selection strategy, the exclusion oracle, and the subdivision scheme. The node-selection strategy determines the next region to be explored. The exclusion oracle provides a provably correct sufficient condition for there not to be an equilibrium in the region. The subdivision scheme determines how the region is split if it cannot be excluded. The method has the benefit that it need not proceed locally, which avoids it getting stuck in a local minimum that may be far from any actual Nash equilibrium. The method produces a set—the non-excluded regions—that contains all 0-Nash equilibria, and this set keeps decreasing.

Experiments confirmed that our method finds an ϵ-equilibrium for a given target ϵ on all instances. This was not the case with prior methods from GAMBIT which are incomplete, or even with the prior theoretically-complete methods. On the other hand, the incomplete methods were often faster on instances they managed to solve. This suggests a hybrid scheme where one runs a homotopy method (or some other relatively fast incomplete method) first, and if it fails to find an equilibrium, then one runs our algorithm. Or, they can be run in parallel. That essentially achieves the best of both worlds: the speed of incomplete methods and the completeness of ours.

Acknowledgments

This material is based on work supported by the NSF under grants IIS-1617590, IIS-1320620, and IIS-1546752, the ARO under award W911NF-16-1-0061.

References

Babichenko, Y.; Barman, S.; and Peretz, R. 2014. Simple approximate equilibria in large games. EC '14, 753–770. New York, NY, USA: ACM.

Babichenko, Y. 2014. Query complexity of approximate Nash equilibria. STOC '14, 535–544.

Berg, K., and Sandholm, T. 2016. Exclusion Method for Finding Nash Equilibrium in Multi-Player Games. AAMAS, 1417–1418.

Boryczka, U., and Juszczuk, P. 2013. Differential evolution as a new method of computing Nash equilibria. In Nguyen, N., ed., *Trans on Computational Collective Int IX*. 192–216.

Buttler, J., and Akchurina, N. 2013. Nash equilibria in normal form games via optimization methods. *2013 European Control Conference* 724–729.

Chatterjee, B. 2009. An optimization formulation to compute Nash equilibrium in finite games. In *Int. Conf. on Methods and Models in Comp. Science*.

Chen, X.; Deng, X.; and Teng, S.-H. 2009. Settling the complexity of computing two-player Nash equilibria. *Journal of the ACM* 56(3).

Daskalakis, C.; Goldberg, P. W.; and Papadimitriou, C. H. 2009. The complexity of computing a Nash equilibrium. *Communications of the ACM* 52(2):89–97.

Daskalakis, C. 2013. On the complexity of approximating a Nash equilibrium. *ACM Transactions on Algorithms* 9(3).

Datta, R. S. 2010. Finding all Nash equilibria of a finite game using polynomial algebra. *Econ Theory* 42:55–96.

Etessami, K., and Yannakakis, M. 2010. On the complexity of Nash equilibria and other fixed points. *SIAM Journal of Computing* 39(6):2531–2597.

Ganzfried, S., and Sandholm, T. 2010. Computing equilibria by incorporating qualitative models. *AAMAS*.

Goldberg, P. W.; Papadimitriou, C. H.; and Savani, R. 2013. The complexity of the homotopy method, equilibrium selection, and Lemke-Howson solutions. *ACM TEAC* 1(2).

Govindan, S., and Wilson, R. 2003. A global Newton method to compute Nash equilibria. *JET* 110:65–86.

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

Hémon, S.; de Rougemont, M.; and Santha, M. 2008. *Approximate Nash Equilibria for Multi-player Games*. Springer Berlin Heidelberg. 267–278.

Herings, P. J.-J., and Peeters, R. 2005. A globally convergent algorithm to compute all Nash equilibria for n-person games. *Annals of Operations Research* 137:349–368.

Herings, P. J.-J., and Peeters, R. 2010. Homotopy methods to compute equilibria in game theory. *Economic Theory* 42(1):119–156.

Herings, P. J.-J., and van den Elzen, A. 2002. Computation of the Nash equilibrium selected by the tracing procedure in n-person games. *GEB* 38:89–117.

Jiang, A. X., and Leyton-Brown, K. 2011. Polynomial-time computation of exact correlated equilibrium in compact games. *EC*.

Jiang, A. X.; Leyton-Brown, K.; and Bhat, N. A. 2011. Action-graph games. *GEB* 71:141–173.

Lipton, R. J., and Markakis, E. 2004. *Nash Equilibria via Polynomial Equations*. Springer. 413–422.

Lipton, R. J.; Markakis, E.; and Mehta, A. 2003. Playing large games using simple strategies. EC '03, 36–41. ACM.

McKelvey, R. D., and McLennan, A. 1996. Computation of equilibria in finite games. In Amman, H.; Kendrick, D.; and Rust, J., eds., *Handbook of Comp Econ*, Vol 1. 87–142.

McKelvey, R. D., and Palfrey, T. R. 1995. Quantal response equilibria for normal form games. *GEB* 10:6–38.

McLennan, A. 2005. The expected number of Nash equilibria of a normal form game. *Econometrica* 73(1):141–174.

Nash, J. 1950. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences* 36:48–49.

Nudelman, E.; Wortman, J.; Shoham, Y.; and Leyton-brown, K. 2004. Run the GAMUT: A comprehensive approach to evaluating game-theoretic algorithms. In *AAMAS*, 880–887.

Papadimitriou, C. H., and Roughgarden, T. 2005. Computing correlated equilibria in multi-player games. *16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*.

Porter, R.; Nudelman, E.; and Shoham, Y. 2008. Simple search methods for finding a Nash equilibrium. *Games and Economic Behavior* 63:642–662. Early version in AAAI-04.

Rubinstein, A. 2015. Inapproximability of Nash equilibrium. STOC '15, 409–418.

Rubinstein, A. 2016. Settling the complexity of computing approximate two-player Nash equilibria. In *arXiv:1606.04550*.

Sandholm, T.; Gilpin, A.; and Conitzer, V. 2005. Mixed-integer programming methods for finding Nash equilibria. *AAAI* 495–501.

Turocy, T. L. 2005. A dynamic homotopy interpretation of the logistic quantal response equilibrium correspondence. *Games and Economic Behavior* 51:243–263.

Turocy, T. L. 2008. Towards a black-box solver for finite games: computing all equilibria with gambit and phcpack. In Stillman, M.; Verschelde, J.; and Takayama, N., eds., *Software for algebraic geometry*. 133–147.

Turocy, T. L. 2010. Software for solving noncooperative strategic form games. In Cochran, J. J.; Cox, L. A.; Keskinocak, P.; Kharoufeh, J. P.; and Smith, J. C., eds., *Wiley encycl. of operations research and management science*.

van der Laan, G.; Talman, A.; and van der Heyden, L. 1987. Simplicial variable dimension algorithms for solving the nonlinear complementarity problem on a product of unit simplices using a general labelling. *MOR* 12(3):377–397.

von Stengel, B. 2012. Rank-1 games with exponentially many Nash equilibria. *arXiv:1221.2405*.