Towards a Black-Box Solver for Finite Games

Finding All Nash Equilibria with Gambit and PHCpack

by Theodore L. Turocy

Department of Economics
Texas A&M University
College Station TX 77843

Email: `turocy@econmail.tamu.edu`

6 March 2007

Abstract

This paper describes a new implementation of an algorithm to find all isolated Nash equilibria in a finite strategic game. The implementation uses the game theory software package Gambit to generate systems of polynomial equations which are necessary conditions for a Nash equilibrium, and polyhedral homotopy continuation via the package PHCpack to compute solutions to the systems. Numerical experiments to characterize the performance of the implementation are reported. In addition, the current and future roles of support enumeration methods in the context of methods for computing Nash equilibria are discussed.

1 Introduction

It is no understatement that game theory has revolutionized many social sciences, most notably economics, in the last few decades. In recognition of this, the 1994 Nobel Memorial Prize in Economics was awarded to John Nash, John Harsanyi, and Reinhard Selten, three pioneers of the field. The scope of the influence of game theory is not limited to social science; for example, applications to biology date at least to Maynard Smith [7]. In more recent applications, game theory is now being used in computer science to inform the design of automated agents.

A central concept of game theory is the Nash equilibrium. Nash equilibrium can be thought of as a condition of mutual consistency among the beliefs and actions of a collection of agents, each of whom seeks to maximize his own objective function. (A more formal definition is given in Section 2.) Nash equilibria thus have a normative attraction, in that they represent points which are stable with respect to unilateral changes in action by any one player. At a Nash equilibrium, no individual player can change his action in such a way as to improve his outcome in the game.

For games in which there are a finite number of players, each of whom can choose one action from a finite set of possible actions, the existence of an equilibrium was established by Nash [10]. Nash's existence argument allows players to choose from the set of probability distributions over their actions. For many games, this is essential; the game "rock, paper, scissors" is a familiar example in which randomization is necessary for the existence of equilibrium. While the mutual consistency property of Nash equilibrium suggests a fixed-point formulation, the Nash equilibrium problem can be stated in several equivalent ways; see the survey of McKelvey and McLennan [8] for more details. Each formulation suggests a different computational approach for locating equilibria.

The mathematical versatility and beauty of the Nash equilibrium formulation aside, practitioners in the various fields in which the concept is applied are primarily interested in getting the set of equilibria of their games of interest, and in estimating what games are and are not feasible to solve with a given computational budget. The software package Gambit (McKelvey, McLennan, and Turocy [9]) aims, among other goals, to provide such users a set of routines for computing the equilibria of an arbitrary finite game. Currently, Gambit provides a number of methods which can compute one or more equilibria in games with more than two players. However, clear guidance from experience as to which method or methods to use remains absent. This paper documents one step in the direction of "black-boxing" the process of computing equilibria in those games.

One class of approaches which has received recent attention is a method based on the solution to systems of polynomial equations. The calculation is organized by enumerating supports, where a support is a set of strategies which are assigned positive probability. The Nash equilibrium conditions imply a set of polynomial equations, the solutions of which contain the set of Nash equilibria on that support.

This approach is not new. In fact, this is the way most humans go about trying to manually compute equilibria. The first published computer program to compute equilibria using support enumeration was Dickhaut and Kaplan [3]. A method which attempts to compute all isolated Nash equilibria, using essentially the support enumeration described in this paper, has been present in Gambit since the mid-1990s.

This paper reports computational results on a new implementation of this same support enumeration method. The program uses Gambit as the frontend for defining games and generating the systems of polynomial equations to be solved. Polyhedral homotopy continuation (Huber and Sturmfels [5]) is used as the backend to generate the solutions of these systems, using PHCpack (Verschelde [14]).

The program ties together two previous papers on the topic. Herings and Peeters [4] develop essentially the same algorithm, decomposing the game by means of "admissible" supports and showing how to solve the resulting systems of polynomial equations using a homotopy method. This paper augments their work by giving and proving an explicit algorithm for finding all admissible supports, and characterizing the scaling of the algorithm as the size of the game increases. The choice of PHCpack as the backend is inspired in part by the results of Datta [2], who gives a performance report on solving for all totally mixed Nash equilibria of a game using PHCpack. He shows that the numerical approach outperforms existing approaches to solve games via computer algebra using Grobner bases.

The paper proceeds as follows. Section 2 outlines the Nash equilibrium problem. It describes the support enumeration process to compute all supports which might harbor Nash equilibria, and argues the correctness of the process. Section 3 evaluates the performance and scalability of the program. Finally, Section 4 concludes with a discussion of future directions and extensions, and the role of support enumeration methods within the toolkit of computation in game theory.

2 Description of the Nash equilibrium problem

The algorithm operates on games in strategic form. There are N players, indexed by $i = 1, 2, \ldots, N$. Each player i chooses from a set of strategies $S_i = \{1, 2, \ldots, J_i\}$. The possible outcomes of the game is the set $S = \times_{i=1,\ldots,N} S_i$; in a standard abuse of notation, $S_{-i} = \times_{j\neq i} S_i$ is used to denote the set of choices made by all players other than player i. For each contingency $s \in S$, each player i has a utility function $u_i\colon S \to \mathbb{R}$.

Players are permitted to choose randomly from their strategy sets. Let Σ_i denote the set of distributions over S_i, for all players i. A typical randomized strategy for player i will be denoted π_i, with π_{ij} denoting the probability player i will choose strategy $j \in S_i$.

Players perform their random selections independently. The function u_i is extended to randomized strategies using the expected value,

$$u_i(\pi_1, \pi_2, \ldots, \pi_N) = \sum_{j_1=1}^{J_1} \sum_{j_2=1}^{J_2} \cdots \sum_{j_N=1}^{J_N} u_i(j_1, j_2, \ldots, j_N)\pi_{1j_1}\pi_{2j_2}\cdots\pi_{Nj_N}. \tag{1}$$

A randomized strategy profile $\pi = (\pi_i)_{i=1}^N$ is a Nash equilibrium if and only if it satisfies

$$u_i(\pi) \geq u_i(\pi_1, \pi_2, \ldots, \rho_i, \ldots, \pi_N)$$

for all $\rho_i \in \Sigma_i$ for all $i = 1, 2, \ldots, N$. That is to say, for each player i there is no other mixed strategy ρ_i which gives strictly higher expected value than π_i. An immediate consequence of this is that in a Nash equilibrium, all strategies which are played with positive probability under π_i must give the same value: $u_i(s, \pi_{-i}) = u_i(t, \pi_{-i})$ for any two strategies $s, t \in S_i$ such that $\pi_{is} > 0$ and $\pi_{it} > 0$. Therefore, the Nash equilibrium conditions can be written

$$\begin{aligned}
u_i(s, \pi_{-i}) &= u_i(t, \pi_{-i})\, \forall s, t \in S_i \,\text{such that}\, \pi_{is} > 0 \,\text{and}\, \pi_{it} > 0;\\
u_i(s, \pi_{-i}) &\geq u_i(t, \pi_{-i})\, \forall s \in S_i \,\text{such that}\, \pi_{is} > 0, \forall t \in S_i;\\
\pi_{is} &\geq 0\, \forall s \in S_i;\\
\sum_{j=1}^{J_i} &= 1.
\end{aligned}$$

The conditions given by the equality constraints are polynomial equations. Given any support, or selection of strategies assumed to have positive probability, these equations form necessary conditions that must be satisfied if π is a Nash equilibrium. Given a solution to the equations, the inequality conditions can then be checked.

To use this approach to compute all Nash equilibria, all supports must be checked. The number of possible supports in a game grows very rapidly in the size of the game. For any given game, the number of possible supports is

$$\prod_{i=1}^{N} 2^{J_i} - 1$$

which derives from the observation that each strategy $s_i \in S_i$ can independently be in or out of the support (thus generating the 2^{J_i}), except the support where all strategies are out is not valid (thus the minus one). However, some supports can be eliminated from consideration, because it can be seen by inspection that no solution of the equality constraints can also satisfy the inequality constraints. In the terminology of HERINGS AND PEETERS [4], such supports are "inadmissible."

Turning to the question of finding all admissible supports, in what follows, a capital letter, for example T, refers to a subset of the strategies in a game. A subscripted capital refers to the strategies in that set belonging to a player; thus, T_i denotes the strategies in T belonging to player i. Calligraphic capitals refer to collections of sets of strategies, or collections of supports. A *support* is defined a subset of the strategies in a game that satisfies the additional requirement that each player has at least one strategy in the set. That is, a support T is a subset of S such that, for each player i, $T_i \subseteq S_i$ and $T_i \neq \emptyset$.

A strategy $s_i \in S_i$ is said to dominate another strategy $r_i \in S_i$ if $u_i(s_i, t) > u_i(r_i, t)$ for all $t \in S_{-i}$. One can iteratively eliminate strictly dominated strategies from a game without eliminating any strategy profiles that form a Nash equilibrium. In doing this process, one constructs a sequence of supports $\{T^1, T^2, ..., T^k\}$, with $S \supset T^1 \supset T^2 \supset \cdots \supset T^k$, which has the property that no strategy in T^k is dominated by another strategy in T^k.[1] When constructing, say, T^2, it is enough to only ask whether a strategy in T^1 is dominated by some other strategy in T^1; it is not necessary to consider strategies eliminated in the process of constructing T^1 from the full support S. This follows because of a monotonicity property of the dominance operator. If a strategy s_i for player i dominates a strategy r_i for player i when considered against contingencies generated from the strategies of other players in T, then s_i will still dominate r_i when considered against contingencies generated from strategies in a strictly smaller support $V \subset T$.

In this algorithm, at some points there is an arbitrary assignment as to whether a given strategy is in a support. Therefore, it may occur that a strategy $s_i \in T_i$ that is dominated by another strategy $r_i \notin T_i$. A mixed strategy profile on such a support cannot form a Nash equilibrium. Therefore, the dominance elimination process is defined to consider all strategies for a player, and not just those in the support currently being considered.

To make this more precise, a mixed strategy π_i is said to dominate a pure strategy s_i against a support T if

$$\pi_i \succ_T s_i \equiv u_i(\pi_i, t_{-i}) > u_i(s_i, t_{-i}) \forall t_{-i} \in \times_{j \neq i} T_{-i}.$$

The undominated strategies s_i in a support T are then defined as

$$\mathcal{U}(T) = \{s_i : \nexists \pi_i \,\text{such that}\, \pi_i \succ_T s_i\}.$$

Let $\mathcal{U}^\infty(T)$ denote the transitive closure of the operator.

Define $\mathcal{N}$ to be the set of supports such that no strategy in the support is dominated in this sense; that is,

$$\mathcal{N} = \{T : \mathcal{U}(T) = T\}.$$

1. Since we are considering finite games, this process is guaranteed to terminate in a finite number k of steps.

These supports are the ones which are admissible, and thus may be the support of one or more Nash equilibria. This set can be enumerated recursively by considering sets of the form

$$\mathcal{P}(X,Y) = \{T \in \mathcal{N}: x \in T \text{ for all } x \in X, \text{ and } y \notin T \text{ for all } y \in Y\}. \tag{2}$$

That is, $\mathcal{P}(X, Y)$ is the set of supports which are admissible, and in which all the strategies in X are present in the support, and all the strategies in Y are not present in the support. Observe that $\mathcal{P}(\emptyset, \emptyset) = \mathcal{N}$, and that $\mathcal{P}(X, Y) = \emptyset$ by definition if X and Y are not disjoint; in what follows, X and Y will be assumed disjoint.

If $X \cup Y = S$, then the value of $\mathcal{P}(X,Y)$ is clearly

$$\mathcal{P}(X,Y) = \begin{cases} \{X\} & \text{if } \mathcal{U}(X) = X \text{ and } X_i \neq \emptyset \text{ for all } i, \\ \emptyset & \text{if } \mathcal{U}(X) \neq X \text{ or } X_i = \emptyset \text{ for some } i. \end{cases}$$

If $X \cup Y \subset S$, then the value of $\mathcal{P}(X,Y)$ can be written recursively as

$$\mathcal{P}(X,Y) = \mathcal{P}(X \cup \{s\}, Y) \cup \mathcal{P}(X, Y \cup \{s\}) \tag{3}$$

for any given $s \in S/(X \cup Y)$. Observe that the two sets on the right side of (3) are disjoint, and also that the recursion defined by (3) is monotonic, in the sense that the sets X and Y at deeper levels of the recursion are always supersets of those at the “parent” levels of the recursion.

Availing of the monotonicity of the dominance operator $\mathcal{U}(\,\cdot\,)$, the recursive process described by (3) can be optimized. Note that for any two sets of strategies T and U such that $T \subset U$, if $s \notin \mathcal{U}(S/T)$, then $s \notin \mathcal{U}(S/U)$. Because of the monotonicity of the process in (3), if at any point in time, it is true that there exists some $x \in X$ such that $x \notin \mathcal{U}(S/Y)$, then it must be that $\mathcal{P}(X,Y) = \emptyset$, because of the monotonicity of $\mathcal{U}$.

Therefore, the following procedure based on (3) efficiently uses dominance information to recursively compute $\mathcal{N}$. For any given $X, Y, Z = S/(X \cup Y)$,

- Compute $\mathcal{U}^\infty(X \cup Z)$.
  - If at any point in computing this support, no strategies remain for some player i, $\mathcal{P}(X,Y) = \emptyset$.
  - If there exists $s \in X$ but $s \notin \mathcal{U}^\infty(X \cup Z)$, then $\mathcal{P}(X,Y) = \emptyset$.
- Let $Y' = Y \cup \{s \in Z: s \notin \mathcal{U}^\infty(X \cup Z)\}$, $Z' = Z/\{s \in Z: s \notin \mathcal{U}^\infty(X \cup Z)\}$.
- If $Z' = \emptyset$, then $\mathcal{P}(X,Y) = \{X\}$.
- If $Z' \neq \emptyset$, select any strategy $z \in Z'$. Then define $\mathcal{P}(X,Y,Z)$ recursively by

$$\mathcal{P}(X,Y) = \mathcal{P}(X \cup \{z\}, Y') \cup \mathcal{P}(X, Y' \cup \{z\}).$$

Then, in view of the preceding argument, the following proposition holds.

Proposition 1. *Every support containing a Nash equilibrium is in the set $\mathcal{P}(\emptyset, \emptyset)$. Furthermore, the support enumeration process visits each of these supports exactly one time.*

Proof. The fact that $\mathcal{P}(\emptyset, \emptyset) = \mathcal{N}$ follows from the definition in (2), observing that the universal quantifiers hold trivially because X and Y are empty. The fact that the process visits any support at most once follows from the disjoint nature of the sets in the union in (3). □

3 Numerical results

This section reports on some numerical experiments which exercise the program combining Gambit and PHCpack to compute all Nash equilibria of a game in strategic form.

The first set of results consider the computational time for games of a given dimension with randomly-selected payoffs. Each payoff is drawn independently from the uniform distribtion on $[0, 1]$.

				Runtime (sec)		
Dimension	Supports	NE	non-NE	Enum	PHCpack	Games sampled
2x2x2	2.98	1.92	1.63	0.01	0.02	100
2x2x2x2	12.75	3.86	22.15	0.27	0.28	100
2x2x2x2x2	54.98	7.34	195.51	0.48	4.14	100
2x2x2x2x2x2	224.48	15.52	1559.54	3.19	82.48	100
2x2x2x2x2x2x2	866.11	34.09	12538.22	21.57	1881.51	100
2x2x2x2x2x2x2x2	3123.45	78.45	106185.27	164.70	51362.03	11
3x3x3	55.45	4.68	111.93	0.27	1.49	100
4x4x4	909.80	11.61	3469.58	6.40	108.71	100
5x5x5	12616.64	28.24	94374.07	136.99	6652.47	87
3x3x3x3	805.34	18.45	6101.96	8.90	318.53	100

Table 1. Some summary statistics on the average performance of the program on games with randomly-drawn payoffs.

Table 1 presents summary statistics on the average performance of the program. The dimension column gives the number of strategies for each player. The second column gives the average number of admissible supports considered in the games sampled. The third and fourth column together report the average number of solutions found by PHCpack, divided into those which correspond to Nash equilibria and those which do not. The fifth and sixth columns give the average runtime, in seconds, on a Pentium IV workstation with a single 2.8GHz processor. The column "Enum" gives the time spent performing the enumeration, and "PHCpack" the time spent solving the systems of equations.

The striking feature of Table 1 is that the time per invocation of PHCpack, and the number of solutions found that do not correspond to Nash equilibria, increase very rapidly in the size of the game. The average number of supports visited ranges from roughly one-third to one-half of the maximum possible for each size game.

It is possible, however, that these results are unduly pessimistic. Games which are of interest to practitioners are not drawn randomly from the set of possible games; rather, they often have a particular structure. This point is made, for example, by Nudelman et al [12], whose GAMUT database of games collects examples from several such classes. For some classes of games, the support enumeration process may be able to exploit this structure by eliminating many supports from consideration. Based on the timing results in the previous section, reducing the number of polynomial systems to be considered should permit significantly larger games to be solved feasibly.

As an example, consider the family of "minimum effort" games. In this family of games, each player selects a level of effort from an interval of integers $\{1, 2, ..., J\}$. A player selecting a level of effort e incurs a cost of effort $c(e)$, where the function $c(\,\cdot\,)$ is increasing in effort. Each player receives a benefit $b(\min\,(e_1, e_2, ..., e_N))$, which depends only on the lowest level of effort chosen. Each player's payoff, then, is given by

$$u_i(e_1, e_2, ..., e_N) = \omega_i - c(e_i) + b(\min\,(e_1, e_2, ..., e_N)),$$

where ω_i is player i's initial endowment. In these games, the pure strategy equilibria are those in which each player chooses the same effort level e, for each possible effort level e. In addition, there are generally many mixed-strategy equilibria.

For example, consider the case of $c(e) = 2e$ and $b(e) = 3e$, with $\omega_i = 0$. For $N = 3$ and $J = 4$, there are 336 admissible supports, and total runtime is 33.3 seconds. Similarly, for $N = 3$ and $J = 5$, there are 2677 admissible supports, and the program runs in 1590 seconds, and when $N = 4$ and $J = 4$, there are 3203 admissible supports, and runtime is 9138 seconds. These compare favorably with the "average" games from Table 1. In this example, the symmetry of the game is not exploited, which would further significantly decrease the computational load.

One care that must be addressed in working with the classes of games which might be of interest to practitioners is that the set of Nash equilibria might not consist of isolated points. As an example, consider a game due to Nau et al [11]. There are three players, each with two strategies; Player I can choose T or B; Player II can choose L or R; and Player III can choose 1 or 2. Table 2 gives the payoffs to the three players in each of the eight contingencies.

I	II	III	Payoffs
T	L	1	0, 0, 2
T	R	1	0, 3, 0
B	L	1	3, 0, 0
B	R	1	0, 0, 0
T	L	2	1, 1, 0
T	R	2	0, 0, 0
B	L	2	0, 0, 0
B	R	2	0, 0, 3

Table 2. A three-player game with a positive-dimensional component of Nash equilibria.

In this game, there are three pure-strategy Nash equilibria: $(T, R, 1)$; $(B, L, 1)$, and $(B, R, 2)$. The first two of these are actually parts of one connected component of equilibria $U_1 \cup U_2 \cup U_3$, where

$$\begin{aligned} U_1 &= (T; R; \alpha[1] + (1-\alpha)[2]) \,\text{for}\, \alpha \in \left[0, \frac{3}{4}\right], \\ U_2 &= (B; L; \alpha[1] + (1-\alpha)[2]) \,\text{for}\, \alpha \in \left[0, \frac{3}{4}\right], \\ U_3 &= \left(\frac{1-\alpha}{1-(1/3)\alpha}[T] + \frac{(2/3)\alpha}{1-(1/3)\alpha}[B]; \alpha[L] + (1-\alpha)[R]; (1/4)[1] + (3/4)[2]\right) \text{for}\, \alpha \in [0, 1]. \end{aligned}$$

Graphically, this component resembles the crossbar and uprights of an American football goal, with U_1 and U_3 being the upright posts, and U_2 the analogue of the crossbar.[2] The equilibria in this component thus arise from a total of five admissible supports. The current implementation of the program will detect the possible presence of a component of solutions through the return value from PHCpack, but does not attempt to characterize the nature of the component. Such "degenerate" games are of interest in practice, however, which means future development in this area is needed to create a true "black-box" solver for general games.

4 Discussion

It is interesting, and perhaps ironic, that this method for computing equilibria is precisely the one that humans often use. This has an advantage in that the operation of the algorithm can be easily understood by human users. Practitioners are often interested, not necessarily in finding all equilibria, but in finding out whether or not there exist equilibria with a particular structure; that is, whether there are equilibria in which certain strategies, or sets of strategies, are or are not used.

CHEN ET AL [1] illustrates one such application, which used Gambit and PHCpack to establish the nonexistence of equilibria in which any player randomizes. This paper studied the performance of two mechanisms for sharing the costs of using a common resource that suffers from congestion, such as, for example, a computer network. The mechanisms both have unique Nash equilibria when all players may choose the amount of service they demand from a continuous set. However, in passing to a discrete strategy set, uniqueness is lost, and a multiplicity of pure-strategy Nash equilibria exist.

The experimental design used by Chen et al involved four players, each choosing from a set of 21 demand levels. For games of this size, investigating each support is clearly infeasible. However, in this game, if one holds fixed the total demands of other players, each player's payoff function is quasiconcave in his own choice of demand. Therefore, in order for the equilibrium conditions to hold, a player can be randomizing only over at most two choices, q and $q+1$. Thus, one needs to consider only 20 possible supports per player, instead of $2^{21} - 1$. Through numerical exploration, it was established that there were no equilibria involving randomization; based on the explorations, the authors conjectured that this was a general fact in discretizing this class of games, which was proven formally in the final version of the paper.

The support enumeration process presented here covers all possible supports, which is necessary to compute all equilibria. In some cases, it is not necessary to compute all equilibria. For some applications, computing one equilibrium is enough. In other cases, one might be interested in finding out whether the game has a unique equilibrium, in which case the objective would be to minimize the time it takes to find two equilibria, if two are present. Some heuristics in this direction are investigated by PORTER ET AL [13], who find that support enumeration again stacks up well against other methods for finding a sample equilibrium.

2. The crossbar in the football goal is straight, while component U_3 is curved.

Despair at the scalability of the support enumeration process should also be tempered by the observation that many games are naturally represented in extensive, or tree, form. Using the sequence form representation of Koller, Megiddo, and von Stengel [6], the strategy space for each player may be represented efficiently. Converting an extensive game to strategic form for use with the support enumeration method described here may result in a strategic form which is exponential in the size of the original tree. The sequence form representation, on the other hand, is linear in the size of the tree. A support enumeration method similar to the one described here can be used on the extensive form, and the Nash equilibrium conditions for each support result in a system of polynomial equations and inequalities, just as in the strategic form. An implementation of the support enumeration process has been available in Gambit since the mid-1990s; interfacing this with PHCpack awaits future work.

At this writing, it appears support enumeration methods will have a significant future in the practice of computing Nash equilibria. This method is the only one known at this time that is guaranteed to find all Nash equilibria in finite time. Furthermore, investigation of interesting questions about the set of Nash equilibria often involves the enumeration of only a subset of supports. This suggests that future implementations of such a "black-box" solver for Nash equilibria should offer users flexibility in choosing how to search all or part of the set of supports which might harbor Nash equilibria.

A second avenue for future development is in solving the systems of equations generated by the Nash equilibrium conditions on a support. These systems have some regular structures. The equations generated by the indifference conditions for player i involve only the probabilities of strategies of players other than player i. In addition, the current implementation ignores the nonnegativity constraints on the variables in solving the systems. The equations generated by one support are similar to those generated by other supports which differ in the addition or removal of one strategy. In addition, Herings and Peeters [4] list several other possible optimizations for special-case supports. None of these possible optimizations are exploited by the current baseline implementation.

Acknowledgements

This paper is based on a talk given at the Institute for Mathematics and its Applications at the University of Minnesota, during the Software for Algebraic Geometry workshop in October 2006. Comments and suggestions from the participants there were greatly appreciated.

The programs described here, including the scripts used to build the randomly-drawn and minimum-effort games, are all available under the GNU General Public License from the Gambit website at http://econweb.tamu.edu/gambit.

Bibliography

[1] Yan Chen, Laura Razzolini, and Theodore L. Turocy. Congestion Allocation for Distributed Networks: An Experimental Study. *Economic Theory*, 2007. Forthcoming.

[2] R.S. Datta. Using Computer Algebra To Compute Nash Equilibria. In *Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation (ISSAC 2003)*, pages 74–79, 2003.

[3] John Dickhaut and Todd Kaplan. A Program for Finding Nash Equilibria. *The Mathematica Journal*, 1(4):87–93, 1992.

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

[5] Birk Huber and Bernd Sturmfels. A polyhedral method for solving sparse polynomial systems. *Mathematics of Computation*, 64:1541–1555, 1995.

[6] Daphne Koller, Nimrod Megiddo, and Bernhard von Stengel. Efficient computation of equilibria for extensive two-person games. *Games and Economic Behavior*, 14:247–259, 1996.

[7] John Maynard Smith. *Evolution and the Theory of Games*. Cambridge UP, Cambridge, 1982.

[8] Richard D. McKelvey and Andrew M. McLennan. Computation of Equilibria in Finite Games. In Hans Amman, David A. Kendrick, and John Rust, editors, *The Handbook of Computational Economics*, volume I, pages 87–142. Elsevier, 1996.

[9] Richard D. McKelvey, Andrew M. McLennan, and Theodore L. Turocy. Gambit: Software Tools for Game Theory. Version 0.2007.01.30.

[10] John F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36:48–49, 1950.

[11] Robert Nau, Sabrina Gomez Canovas, and Pierre Hansen. On the Geometry of Nash Equilibria and Correlated Equilibria. *International Journal of Game Theory*, 2005. Forthcoming.

[12] E. Nudelman, J. Wortman, Y. Shoham, and K. Leyton-Brown. Run the GAMUT: A comprehensive approach to evaluating game-theoretic algorithms. In *Proceedings of the Third International Joint Conference on Autonomous Agents & Multi Agent Systems*, 2004.

[13] Ryan W. Porter, Eugene Nudelman, and Yoav Shoham. Simple search methods for finding a Nash equilibrium. *Games and Economic Behavior*, 2006.

[14] Jan Verschelde. Algorithm 795: PHCpack: a general-purpose solver for polynomial systems by homotopy continuation. *ACM Transactions on Mathematical Software*, 25(2), 1999.