Software for Solving Noncooperative Strategic Form Games

Theodore L. Turocy
School of Economics
University of East Anglia
Norwich NR4 7TJ, United Kingdom

June 8, 2010

Abstract

The problem of finding Nash equilibria in noncooperative strategic form games can be expressed using several equivalent formulations. These formulations admit a variety of algorithmic approaches. This article summarizes the current state of available software implementations for solving the Nash equilibrium problem, and outlines general advice for approaching the analysis of any given game.

1 Introduction and overview

Game theory is an elegant tool for modeling and analyzing strategic interaction among multiple agents. Over the last half-century, it has proven its worth and flexibility across disciplines from economics to computer science to operations research. This flexibility comes with some costs. It can be unwieldy to write down a formal specification of a noncooperative strategic form game, even for relatively small numbers of players and strategies available to those players. Furthermore, once the game is specified, computing predictions for how the game will or ought to be played, especially the Nash equilibrium (or equilibria) of the game, is tedious to do by hand for even small games, and completely impractical for games of moderate size. To move beyond toy examples and investigate models which are useful in operations and management contexts, software tools for specifying and solving noncooperative strategic form games are essential.

The good news for the practitioner interested in solving particular games is that the Nash equilibrium of a game can be formulated in terms of a variety of mathematical programming problems. These programming problems, in turn, can be solved using a combination of off-the-shelf general-purpose libraries, as well as implementations which may take advantage of the special structure of the programming problems which arise specifically from games.

This article first gives a high-level overview of Gambit, an integrated package for doing computation on finite games. The main body of the article provides a "field guide" to computing equilibria. This guide mixes an overview of the available algorithms and implementations with general advice on techniques and reasonable expectations for solving games. In a brief conclusion, some current and anticipated future directions for research in this area over the next few years are highlighted.

2 Gambit

The most comprehensive package available for computing Nash equilibria in finite games is Gambit, available at http://www.gambit-project.org. The Gambit projects offers a unified suite of tools for working with finite games, available for Windows, OS X, and Linux. Gambit is fully open source, and is licensed under the GNU GPL.

The Gambit project was initiated in the early 1990s by Richard McKelvey and developed extensively in the mid-1990s via a National Science Foundation grant to McKelvey and Andrew McLennan. The impetus behind the conception of the project was to provide a framework for developing and manipulating representations of finite games in extensive and strategic form, and providing implementations of the emerging array of algorithms for computing Nash equilibria and other solution concepts for games to researchers, students, and practitioners.

For most users, the front door to Gambit is the graphical user interface. This tool offers the ability to construct games of small to medium size interactively, including the specification of players, move structures, information structures, and payoffs. The interface also allows for the interactive elimination of dominated strategies and actions and conversion of an extensive game to its reduced strategic form equivalent. Finally, it serves as a frontend to the implementation of methods for computing one or more Nash equilibria of the game, and for inspecting the implications of the resulting equilibria.

As of this writing in early 2010, Gambit is set for a significant overhaul. The current architecture of Gambit dates to the mid-1990s, and is written primarily in C++. The architecture is modular, in that there is a self-contained library for representing games, entities within games such as players and outcomes, and concepts related to games such as mixed strategy profiles. Each algorithm for computing Nash equilibria in games is implemented separately, and wrapped as a standalone command-line tool for external scripting use. This current architecture makes Gambit modestly scriptable.

The state of the architecture also reflects the time at which it was written. In the mid-1990s, the term "open source" had yet to be coined officially, and libraries for scientific computing with licenses that encouraged reuse and redistribution were few. In addition, advances in computing power and algorithmic development have significantly expanded the scope of the type and size of game which can be analyzed using off-the-shelf hardware. The planned next iteration of Gambit will rebuild the architecture to be more friendly to modern scripting languages like Python, and to interface more cleanly with other tools relevant to the practice of game theory and scientific computing. Several of these changes are already available on an experimental basis in the current version. In addition, the current Gambit architecture represents all games by an explicit table of payoffs in memory. In the mid-1990s, any game which was large enough where custom code for generating payoffs on the fly would be useful was too large to be solved using available hardware. Today, this has changed, and in future Gambit plans to offer support for customizing off-the-shelf game representations for efficiency in particular applications.

In the discussion in this article, the focus will be on general-purpose tips for how to go about solving finite games. The organization will generally follow the structure of tools available in Gambit. Where other software resources exist which aren't well-integrated into the Gambit framework, these will also be indicated.

3 Field guide to solving games

A first piece of advice for the practitioner interested in computing Nash equilbria in finite games is to have reasonable expectations. Computing a Nash equilibrium is algorithmically a challenging problem; see, for instance, Daskalakis et al [5] and the references therein. In addition, there may be many Nash equilibria of a game (for example, McKelvey and McLennan [17]). Taken together, these imply that games which do not seem to be particularly large at first glance may be surprisingly difficult to solve. This suggests a good strategy in approaching a game-theoretic model in a field application is to begin with a simple game structure.

Basic results in game theory show that the set of Nash equilibria is a subset of the strategy profiles which survive iterated elimination of strictly dominated strategies. Because of the exponential nature of the complexity results cited, removing strategies from consideration a priori is very desirable from an efficiency standard. Checking for dominance is a very fast operation by comparison. Gambit can automate this elimination for you.

The focus of this article is on solving games in strategic form. Historically, methods for computing equilibria in the strategic form of a game were well-developed earlier. The last decade has seen significant algorithmic advancements in computing equilibria in extensive games (e.g., the sequence form of Koller, von Stengel, and Megiddo [14]). The representation of a game in extensive form is generally smaller than its strategic form counterpart, particularly when players have many information sets in the extensive version of the game. When possible, consider using an algorithm which exploits the extensive structure of the game; the discussion in this article will note where extensive form versions of equilibrium computation methods exist.

Complexity results such as the ones cited above are worst-case guarantees. Many games of interest in practice may have structure which can be exploited. The researcher may have prior beliefs that an equilbrium of a certain type might exist, or might be interested primarily in equilibria involving certain actions being played. Organizing the search for equilibria around this knowledge can help speed up the determination of answers to specific research questions, rather than waiting on an exhaustive analysis of the set of equilibria. There has been recent research in the direction of using heuristics to optimize the time to compute one Nash equilibrium.

For any game, an excellent place to being the analysis is a simple enumeration of pure strategy equilibria. While the number of possible pure strategy profiles increases exponentially in the size of the game, checking whether a given strategy profile is a Nash equilibrium is in practice fast, and since it is a brute-force search, it is straightforward to parallelize. The rest of this section gives an overview of the available tools for computing Nash equilibria in general, including those involving mixed strategies. These methods divide into those which are applicable for games with two players, and those which are applicable for games with arbitrary numbers of players. We now turn to each of those classes individually.

3.1 Two player games

In an strategic form game with two players, the expected payoff to each strategy is a linear function of the probabilities which specify the other player's mixed strategy. This special structure admits algorithms for computing the set of Nash equilibria which exploit linearity and convexity. In a general two-player game, the set of Nash equilibria can be expressed as the union of convex sets of mixed strategy profiles. When furthermore the game is constant-sum, the set of equilibria is itself one convex component. In both cases, the extreme points of the convex components are rational numbers when the payoffs specifying the game are rational. Gambit takes the perspective that all payoffs and probabilities in specifying a game are rational.

For two-player constant sum games, the Nash equilibrium problem can be formulated as a standard linear programming problem (Dantzig [4]). Therefore, any linear programming solver can be brought to bear on computing equilibria. Gambit provides a self-contained linear programming solver in its distribution, wrapped in the command-line tool gambit-lp. In addition, there is experimental support for formulating the linear programming problem based on any game and passing the problem to external solvers. Since there are many strong options for both free and commercial linear programming solvers, external solvers may be the appropriate choice for many applications.

For two-player games which are not constant sum, the equilibrium problem is a linear complementarity problem (Lemke and Howson [15]). There are two approaches to solving the linear complementarity problem. Lemke and Howson provided a linear path-following algorithm based on pivoting, reminiscent of the simplex method for linear programs. Experience has shown that this method, implemented as gambit-lcp in the Gambit package, is often efficient in finding an equilibrium of the game. However, Shapley [23] showed that not all equilibria of a game can be computed using the method of Lemke and Howson, and Savani and von Stengel [22] show how to construct pathological examples of games in which the method of Lemke and

Howson takes exponentially long to arrive at an equilibrium.

Instead, to guarantee computation of all equilibria, the reverse pivoting algorithm of Avis and Fukuda [1] can be used. This method, implemented in Gambit as gambit-enummixed, systematically enumerates all possible extreme points of the components of the set of equilibria, and determines whether or not they satisfy the Nash equilibrium conditions. An independent implementation of the algorithm, named lrslib, has been made available under the GPL by Avis at http://cgm.cs.mcgill.ca/~avis/C/lrs.html. Avis' implementation includes a driver program for using the library to compute the set of Nash equilibria. For access to lrslib without the need to download and compile it, Rahul Savani has made a Web-based frontend to lrslib for computing equilibria. [21] As of this writing, there is an experimental wrapping of lrslib available in the gambit-enummixed tool. The Gambit wrapping is often a version or two behind the latest lrslib, and therefore does not always include the latest features; for instance, the most recent lrslib has support for executing the algorithm on multiple processors, which is not yet available via Gambit. A good self-contained resource describing the reverse pivoting algorithm and other recent developments in computing all equilibria of two-player strategic form games is Avis et al [2], which includes a table summarizing how expected running times scale with the number of strategies in the game.

Because both the linear program and reverse pivoting algorithm are of independent interest outside of their applications to the equilibrium computation problem, external solvers generally offer better support than the native Gambit implementations. One strength of the Gambit native implementation not always found in external solvers is the ability to carry out the entire calculation in exact rational arithmetic. Rational arithmetic is significantly slower than floating-point arithmetic, but pivoting calculations can be done exactly without rounding error, and exact representations of the extreme points of the equilibrium set can be found. While generically the set of Nash equilibria is a collection of isolated points, games which are of interest in practice often have components of equilibria which have positive dimension. Rational arithmetic handles the implied degeneracies exactly. In addition, Gambit offers support in the gambit-enummixed tool for identifying the structure of the equilibrium components.

The existence of positive-dimension components of equilibria in strategic games often arises because they are the reduced strategic form representations of extensive games. The sequence form (Koller, von Stengel, and Megiddo [14]) allows for using linear programming and linear complementarity programming to compute equilibria while exploiting the extensive structure of the game.

3.2 General-player games

Games with more than two players lack the linear structure in payoffs present in games with only two players. Therefore, the problem of computing equilibria becomes more challenging. The development of effective methods for solving these games has occured more recently than for two-player games, and remains an active area for further development in algorithms and implementations.

There is one existing method for finding all equilibria in a generic game with more than two players. This method relies on enumerating all the possible supports (sets of strategies played with positive probability) in the game. On a given support, the Nash equilibrium conditions can be expressed as a collection of equalities which are polynomial in the probabilities with which strategies are played, and a collection of inequalities which are also polynomial in the same probabilities. Therefore, for a given support, one can use an external solver to compute all solutions of the system of polynomial equalities, and then check the candidate solutions and keep only those which satisfy the inequalities and form valid probability distributions. A version of this approach was implemented in Gambit by Andrew McLennan circa 1995, and the algorithm was formally defined and proven correct by Herings and Peeters [12].

Implementations of algorithms for finding the solutions to systems of polynomial equations are a recent development in the field of algebraic geometry. Two leading packages as of this writing are PHC-pack by Jan Verschelde (at http://www.math.uic.edu/~jan/download.html) and Bertini by Bates et al (at http://www.nd.edu/~sommese/bertini/). Turocy [26] describes preliminary results of integrating PHCpack with Gambit. The experimental interface in Gambit to this method, currently distributed separately from the main Gambit package, also includes support for using Bertini as the solver backend.

Carrying out this algorithm is a formidable task, because the number of potential supports is exponential in the number of strategies a player can choose. If player i has k_i strategies, then there are $2^{k_i} - 1$ potential supports for that player, and $\prod_{i=1}^{n}(2^{k_i} - 1)$ possible supports overall. As Herings and Peeters [12] and Turocy [26] show, many of these supports can be ruled out as candidates by checks which require little processing time, so in practice the number of supports which have to be passed to the polynomial solver is significantly less than the theoretical maximum. In addition, because each support is an independent problem, the solution process is embarrassingly parallelizable; it is straightforward to utilize multiple independent threads of computation if multiple processors are available.

For some applications, exhaustive enumeration of all equilibria may not be necessary. Given that multiplicity of Nash equilibria is the rule rather than the exception, one may simply be interested in obtaining a sample Nash equilibrium. There exist several approaches for computing a single equilibrium of a game, all based on homotopy methods. The leading approaches in this area are surveyed by Herings and Peeters [13]. Loosely speaking, the basic idea behind these methods is, given a game of interest, to construct a sequence of games converging to that game. The sequence is constructed in such a way that the first game in the sequence is easy to solve, perhaps because it has an equilibrium in strictly dominant strategies. Then, if the set of Nash equilibria changes in a continuous way along the sequence of games, it is possible to "trace" an equilibrium back to the original game of interest.

In fact, the path-following algorithm of Lemke and Howson ([15]) can be interpreted as implementing just such a homotopy, for the special case of two-player games. As noted, the linear structure of expected payoffs in two-player games means techniques for following the path of equilibria exactly are available. In games with more than two players, numerical procedures must be used to follow generally nonlinear paths approximately. The Lemke-Howson method is generalized to n-player games by the global Newton method of Govindan and Wilson [7, 8]. This has been implemented in the Gametracer package by Blum, Koller, and Shelton, and is available at http://dags.stanford.edu/Games/gametracer.html, distributed under the GNU GPL. Gambit incorporates the Gametracer implementation in the gambit-gnm and gambit-ipa command-line tools. The operation of the global Newton method is parameterized by specifying an initial perturbation vector, which specifies how the payoffs of the game are adjusted to find the initial starting game which is easy to solve. The resulting equilibria found by the procedure in general depend on the choice of this initial vector.

This last observation, that the equilibria found by an algorithm are a subset of the full set of equilibria, suggest the idea of selecting from the set of equilibria using an appropriately chosen algorithm. The multiplicity of Nash equilibria has always been a vexing problem in using the concept to predict behavior in a game, and the number of equilibria in a game tends to grow as the number of strategies and number of players are increased. This idea of using an algorithm to systematically select an equilibrium with desirable properties was made explicit in the form of the Tracing Procedure of Harsanyi [9] and Harsanyi and Selten [10]. The Tracing Procedure imagines players arriving at a Nash equilibrium of a game by beginning with initial beliefs about how other players will play the game. Based on this initial belief, through a process of introspection, the players adjust their intended actions based on their beliefs, while also adjusting their beliefs by realizing that other players are undergoing the same adjustment process. The Tracing Procedure results in paths which may not be differentiable at some points, complicating the numerical implementation of the tracing. Herings and Peeters [11] provide an implementation of an algorithm for following a differentiable path through the space of mixed strategy profiles to find a Nash equilibrium. While the path of strategy profiles followed is not exactly that of the Tracing Procedure, Herings and Peeters show that the equilibrium the method computes coincides with the equilibrium selected by the Tracing Procedure. A FORTRAN implementation of the algorithm is available from the authors at http://www.personeel.unimaas.nl/r.peeters.

Another equilibrium computation and selection approach based on a process of behavioral adjustment uses the quantal response equilibrium of McKelvey and Palfrey [18]. In a quantal response equilibrium, the payoffs to each strategy are perturbed by an additive, independent stochastic shock. Under specified conditions, as the variance of this random shock goes to zero, the set of quantal response equilibria converge to a subset of the set of Nash equilibria. Turocy [25] demonstrated how to use the results of McKelvey and Palfrey for the case when the shock follows the extreme value distribution to compute an arbitrarily good approximation of a Nash equilibrium by following a differentiable path of quantal response equilibria. This path can be interpreted as having the same conceptual structure of introspection and adjustment as the Tracing Procedure, but, since it uses a different dynamical system, does not necessarily result in the same equilibrium as selected by the Tracing Procedure. This is implemented in Gambit as the gambit-logit command-line tool. This approach has an independent interest in laboratory experiments in game theory, where the quantal response equilibrium is used to organize the behavior of subjects playing games in controlled experiments (see, e.g., Goeree and Holt [6]).

Individually, none of the preceding three methods (global Newton, Tracing Procedure, quantal response equilibria) are guaranteed to be able to find all equilibria, even in games where the equilibria are isolated points. However, taken together, they still are useful elements of a toolkit for analyzing strategic games. Since they are approximation-based methods, the intermediate output they generate has a meaningful game-theoretic interpretation either as the exact equilibrium of an approximation to the original game, or an approximate equilibrium to the exact original game. Also, they often find different equilibria when applied to a game. Therefore, some questions, such as whether a game has a unique equilibrium, may in practice be answered most easily by applying a combination of these methods, rather than setting up an exhaustive search involving the solution to many systems of polynomial equations.

A final word of caution is in order when working with games with more than two players. Unlike two player games, the set of Nash equilibria of a game with more than two players need not be expressable as the union of a finite number of convex sets. For a simple three-player example which illustrates this possibility, see Nau et al [19]. Since, as noted before, many games which arise in practical applications may have positive-dimensional sets of equilibria, this possibility should be kept in mind. As of this writing, there is no tool which reliably will find and report the existence of such positive-dimensional sets. There is some support in the Bertini package for exploring positive-dimensional solution sets which may be able to be brought to bear on the problem. In many cases, the existing PHCpack or Bertini implementations may fail to report any equilibria on a support, when in fact a positive-dimensional set of equilibria exists there. The global Newton, Tracing Procedure, and quantal response methods, if they find equilibria in such a component, generally will report only individual points from the component, and give no indication of the existence of equilibria nearby.

4 Summary and outlook

There exist a good set of tools for the practitioner to create, maniplate, and solve noncooperative strategic form games. Advances in algorithm development, particularly in the solution of systems of polynomial equations, and the steadily lowering price of computational power, have combined to expand significantly the size of game that can be analyzed on commondity hardware. For small applications, these tools may be able to provide essentially off-the-shelf, black-box solutions, particularly in the case of games with only two players.

In aiming to be a field guide for computing equilibria, this article omits many details of the algorithms mentioned, as well as many other approaches to computing Nash equilibria. The survey of McKelvey and McLennan [16] is a good, if becoming dated, deeper introduction to the topic. A good resource for updates is the January 2010 issue of the journal *Economic Theory*, which was devoted entirely to the computation of Nash equilibria in finite games, and included the surveys of Avis et al [2] and Herings and Peeters [13] mentioned earlier.

In view of the complexity results cited, no amount of algorithmic cleverness or hardware capacity will make the problem of computing the full set of equilibria of a strategic game fully tractable. Some current directions of research in this area involve heuristics which improve the average-time performance in, for instance, finding one equilibrium in particular classes of games; some progress in this direction has been reported by Porter et al [20]. Another direction in the analysis of games is being advanced by behavioral game theory. If a Nash equilibrium is difficult to find, as the results on computational complexity of finding equilibria indicate, then it seems unsatisfactory to expect players to arrive at a Nash equilibrium in practice. Instead, predictions based on simpler heuristics such as level-*k* reasoning (Stahl [24]) or cognitive hierarchy models (Camerer et al [3]) have been shown to organize how subjects in controlled laboratory experiments play games better than Nash equilibrium, and as such might be superior models for how players make strategic decisions in novel situations.

References

[1] Avis, D. and K. Fukuda (1992). "A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra." *Discrete and Computational Geometry* 8(3): 295-313.

[2] Avis, D., G. D. Rosenberg, R. Savani, and B. von Stengel (2010). "Enumeration of Nash equilibria for two-player games." *Economic Theory* 42(1): 9-38.

[3] Camerer, C. F., T.-H. Ho, and J.-K. Chong. (2004). "A cognitive hierarchy model of games." *Quarterly Journal of Economics* 119(3): 861-898.

[4] Dantzig, G. B. (1963). *Linear Programming and Extensions*. Princeton University Press.

[5] Daskalakis, C., P. W. Goldberg, and C. H. Papadimitriou. (2006). "The complexity of computing a Nash equilibrium." *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, 71-78.

[6] Goeree, J. K. and C. A. Holt. (2001). "Ten little treasures of game theory and ten intuitive contradictions." *American Economic Review* 91(5): 1402-1422.

[7] Govindan, S. and R. Wilson. (2003). "A global Newton method to compute Nash equilibria." *Journal of Economic Theory* 110(1): 65-86.

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

[9] Harsanyi, J. (1975). "The tracing procedure: A Bayesian approach to defining a solution for *n*-person noncooperative games." *International Journal of Game Theory* 4(2): 61-94.

[10] Harsanyi, J. and R. Selten. (1988). *A General Theory of Equilibrium Selection in Games*. MIT Press, Cambridge MA.

[11] Herings, P. J.-J. and R. Peeters. (2001). "A differentiable homotopy to compute Nash equilibria of *n*-person games." *Economic Theory* 18(1): 159-185.

[12] Herings, P. J.-J. and R. Peeters. (2005). "A globally convergent algorithm to compute all Nash equilibria of *n*-person games." *Annals of Operations Research* 137(1): 349-368.

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

[14] Koller, D., B. von Stengel and N. Megiddo. (1996). “Efficient computation of equilibria for extensive two-person games.” *Games and Economic Behavior* 14(2): 247-259.

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

[16] McKelvey, R. D. and A. McLennan. (1996). “Computation of equilibria in finite games.” In *Handbook of Computational Economics, Volume 1*, H. M. Amman, D. A. Kendrick, and J. Rust, eds. Elsevier.

[17] McKelvey, R. D. and A. McLennan. (1997). “The maximal number of regular totally mixed Nash equilibria.” *Journal of Economic Theory* 72(2): 411-425.

[18] McKelvey, R. D. and T. R. Palfrey. (1995). “Quantal response equilibria for normal form games.” *Games and Economic Behavior*, 10(1): 6-38.

[19] Nau, R. F., S. Gomez Canovas and P. Hansen. (2004). “On the geometry of Nash equilibria and correlated equilibria.” *International Journal of Game Theory* 32(4): 443-453.

[20] Porter, R. W., E. Nudelman, and Y. Shoham. (2008). “Simple search methods for finding a Nash equilibrium.” *Games and Economic Behavior* 63(2): 642-662.

[21] Savani, R. (2005). “Solve a bimatrix game.” http://banach.lse.ac.uk/form.html.

[22] Savani, R. and B. von Stengel. (2006). “Hard-to-solve bimatrix games.” *Econometrica* 74(2): 397-429.

[23] Shapley, L. S. (1974). “A note on the Lemke–Howson algorithm,” *Mathematical Programming Study, 1: Pivoting and Extensions*, ed. by M. Balinski. Amsterdam: North-Holland, 175–189.

[24] Stahl, D. O. (1993). “Evolution of smart-*n* players.” *Games and Economic Behavior* 5(4): 604-617.

[25] Turocy, T. L. (2005). “A dynamic homotopy interpretation of the logistic quantal response equilibrium correspondence.” *Games and Economic Behavior* 51(2): 243-263.

[26] Turocy, T. L. (2008). “Towards a black-box solver for finite games: Finding all Nash equilibria with Gambit and PHCpack.” In *Proceedings of the IMA Software for Algebraic Geometry Workshop*, M. E. Stillman, N. Takayama, and J. Verschelde, eds. New York: Springer.

[27] Turocy, T. L. (2010). “Using quantal response to compute Nash and sequential equilibria.” *Economic Theory* 42(1): 255-269