Simple Search Methods for Finding a Nash Equilibrium

Ryan Porter and **Eugene Nudelman** and **Yoav Shoham**
Computer Science Department
Stanford University
Stanford, CA 94305
{rwporter,eugnud,shoham}@cs.stanford.edu

Abstract

We present two simple search methods for computing a sample Nash equilibrium in a normal-form game: one for 2-player games and one for n-player games. We test these algorithms on many classes of games, and show that they perform well against the state of the art– the Lemke-Howson algorithm for 2-player games, and Simplicial Subdivision and Govindan-Wilson for n-player games.

Introduction

Game theory has had a profound impact on multi-agent systems research, and indeed on computer science in general. Nash equilibrium (NE) is arguably the most important concept in game theory, and yet remarkably little is known about the problem of computing a sample NE in a normal-form game. All evidence points to this being a hard problem, but its precise complexity is unknown (Papadimitriou 2001).

At the same time, several algorithms have been proposed over the years for the problem. In this paper, three previous algorithms will be of particular interest. For 2-player games, the Lemke-Howson algorithm (Lemke & Howson 1964) is still the state of the art, despite being 40 years old. For n-player games, until recently the algorithm based on Simplicial Subdivision (van der Laan, Talman, & van der Heyden 1987) was the state of the art. Indeed, these two algorithms are the default ones implemented in Gambit (McKelvey, McLennan, & Turocy 2003), the best-known game theory software. Recently, a new algorithm, which we will refer to as Govindan-Wilson, was introduced by (Govindan & Wilson 2003) and extended and efficiently implemented by (Blum, Shelton, & Koller 2003).

In a long version of this paper we provide more intuition behind each these methods. Here we simply note that they have surfaced as the most competitive algorithms for the respective class of games, and refer the reader to two thorough surveys on the topic (von Stengel 2002; McKelvey & McLennan 1996). Our goal in this paper is to demonstrate that for both of these classes of games (2-player, and n-player for $n > 2$) there exists a relatively simple, search-based method that performs very well in practice. For 2-player games, our algorithm performs substantially better than Lemke-Howson. For n-player games, our algorithm outperforms both Simplicial Subdivision and Govindan-Wilson.

The basic idea behind our search algorithms is simple. Recall that, while the general problem of computing a NE is a complementarity problem, computing whether there exists a NE with a *particular support*[2] for each player is a relatively easy feasibility program. Our algorithms explore the space of support profiles using a backtracking procedure to instantiate the support for each player separately. After each instantiation, they prune the search space by checking for actions in a support that are strictly dominated, given that the other agents will only play actions in their own supports.

Both algorithms order the search by giving precedence to supports of small size. Since it turns out that games drawn from classes that researchers have focused on in the past tend to have (at least one) NE with a very small support, our algorithms are often able to find one quickly. Thus, this paper is as much about the properties of NE in games of interest as it is about novel algorithmic insights.

We emphasize, however, that we are not cheating in the selection of games on which we test. Past algorithms were tested almost exclusively on "random" games. We tested on these too (indeed, we will have more to say about how "random" games vary along at least one important dimension), but also on many other distributions (24 in total). To this end we use GAMUT, a recently introduced computational testbed for game theory (Nudelman et al. 2004). Our results are quite robust across all games tested.

The rest of the paper is organized as follows. After formulating the problem and the basis for searching over supports, we describe our two algorithms. The n-player algorithm is essentially a generalization of the 2-player algorithm, but we describe them separately, both because they differ slightly in the ordering of the search, and because the 2-player case admits a simpler description of the algorithm. Then, we describe our experimental setup, and separately present our results for 2-player and n-player games. In the final section, we conclude and describe opportunities for future work.

[1]This work was supported in part by DARPA grant F30602-00-2-0598 and in part by the National Science Foundation under ITR IIS-0205633.

[2]The support specifies the pure strategies played with nonzero probability.

Notation

We consider finite, n-player, normal-form games $G = \langle N, (A_i), (u_i) \rangle$:

- $N = \{1, \ldots, n\}$ is the set of players.
- $A_i = \{a_{i1}, \ldots, a_{im_i}\}$ is the set of actions available to player i, where m_i is the number of available actions for that player. We will use a_i as a variable that takes on the value of a particular action a_{ij} of player i, and $a = (a_1, \ldots, a_n)$ to denote a profile of actions, one for each player. Also, let $a_{-i} = (a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n)$ denote this same profile excluding the action of player i, so that (a_i, a_{-i}) forms a complete profile of actions. We will use similar notation for any profile that contains an element for each player.
- $u_i : A_1 \times \ldots \times A_n \rightarrow \Re$ is the utility function for each player i. It maps a profile of actions to a value.

Each player i selects a mixed strategy from the set $\mathcal{P}_i = \{p_i : A_i \rightarrow [0,1] | \sum_{a_i \in A_i} p_i(a_i) = 1\}$. A mixed strategy for a player specifies the probability distribution used to select the action that the player will play in the game. We will sometimes use a_i to denote the pure strategy in which $p_i(a_i) = 1$. The support of a mixed strategy p_i is the set of all actions $a_i \in A_i$ such that $p_i(a_i) > 0$. We will use $x = (x_1, \ldots, x_n)$ to denote a profile of values that specifies the size of the support of each player.

Because agents use mixed strategies, u_i is extended to also denote the expected utility for player i for a strategy profile $p = (p_1, \ldots, p_n)$: $u_i(p) = \sum_{a \in A} p(a) u_i(a)$, where $p(a) = \Pi_{i \in N} p_i(a_i)$.

The primary solution concept for a normal form game is that of Nash equilibrium. A mixed strategy profile is a Nash equilibrium if no agent has incentive to unilaterally deviate.

Definition 1 *A strategy profile $p^* \in \mathcal{P}$ is a Nash equilibrium if:* $\forall i \in N, a_i \in A_i : \quad u_i(a_i, p^*_{-i}) \leq u_i(p^*_i, p^*_{-i})$

Every finite, normal form game is guaranteed to have at least one Nash equilibrium (Nash 1950).

Searching Over Supports

The basis of our two algorithms is to search over the space of possible instantiations of the support $S_i \subseteq A_i$ for each player i. Given a support profile as input, Feasibility Program 1, below, gives the formal description of a program for finding a Nash equilibrium p consistent with S (if such an strategy profile exists).[3] In this program, v_i corresponds to the expected utility of player i in an equilibrium. The first two classes of constraints require that each player must be indifferent between all actions within his support, and must not strictly prefer an action outside of his support. These imply that no player can deviate to a pure strategy that improves his expected utility, which is exactly the condition for the strategy profile to be a Nash equilibrium.

[3]We note that the use of Feasibility Program 1 is not novel–it was used by (Dickhaut & Kaplan 1991) in an algorithm which enumerated all support profiles in order to find all Nash equilibria.

Because $p(a_{-i}) = \prod_{j \neq i} p_j(a_j)$, this program is linear for $n = 2$ and nonlinear for all $n > 2$. Note that, strictly speaking, we do not require that each action $a_i \in S_i$ be in the support, because it is allowed to be played with zero probability. However, player i must still be indifferent between action a_i and each other action $a'_i \in S_i$.

Feasibility Program 1

Input: $S = (S_1, \ldots, S_n)$, a support profile
Output: NE p, if there exists both a strategy profile $p = (p_1, \ldots, p_n)$ and a value profile $v = (v_1, \ldots, v_n)$ s.t.:

$$\begin{aligned}
\forall i \in N, a_i \in S_i : &\quad \sum_{a_{-i} \in S_{-i}} p(a_{-i}) u_i(a_i, a_{-i}) = v_i \\
\forall i \in N, a_i \notin S_i : &\quad \sum_{a_{-i} \in S_{-i}} p(a_{-i}) u_i(a_i, a_{-i}) \leq v_i \\
\forall i \in N : &\quad \sum_{a_i \in S_i} p_i(a_i) = 1 \\
\forall i \in N, a_i \in S_i : &\quad p_i(a_i) \geq 0 \\
\forall i \in N, a_i \notin S_i : &\quad p_i(a_i) = 0
\end{aligned}$$

Algorithm for Two-Player Games

In this section we describe Algorithm 1, our 2-player algorithm for searching the space of supports. There are three keys to the efficiency of this algorithm. The first two are the factors used to order the search space. Specifically, Algorithm 1 considers every possible support size profile separately, favoring support sizes that are balanced and small. The motivation behind these choices comes from work such as (McLennan & Berg 2002), which analyzes the theoretical properties of the NE of games drawn from a particular distribution. Specifically, for n-player games, the payoffs for an action profile are determined by drawing a point uniformly at random in a unit sphere. Under this distribution, for $n = 2$, the probability that there exists a NE consistent with a particular support profile varies inversely with the size of the supports, and is zero for unbalanced support profiles.

The third key to Algorithm 1 is that it separately instantiates each players' support, making use of what we will call "conditional (strict) dominance" to prune the search space.

Definition 2 *An action $a_i \in A_i$ is conditionally dominated, given a profile of sets of available actions $R_{-i} \subseteq A_{-i}$ for the remaining agents, if the following condition holds:* $\exists a'_i \in A_i \; \forall a_{-i} \in R_{-i} : \quad u_i(a_i, a_{-i}) < u_i(a'_i, a_{-i})$.

The preference for small support sizes amplifies the advantages of checking for conditional dominance. For example, after instantiating a support of size two for the first player, it will often be the case that many of the second player's actions are pruned, because only two inequalities must hold for one action to conditionally dominate another.

Pseudo-code for Algorithm 1 is given below. Note that this algorithm is complete, because it considers all support size profiles, and because it only prunes actions that are *strictly* dominated.

Algorithm 1

for all support size profiles $x = (x_1, x_2)$, sorted in increasing order of, first, $|x_1 - x_2|$ and, second, $(x_1 + x_2)$ **do**
  for all $S_1 \subseteq A_1$ s.t. $|S_1| = x_1$ **do**
    $A'_2 \leftarrow \{a_2 \in A_2$ not cond. dominated, given S_1 $\}$
    if $\nexists a_1 \in S_1$ cond. dominated, given A'_2 **then**
      for all $S_2 \subseteq A'_2$ s.t. $|S_2| = x_2$ **do**
        if $\nexists a_1 \in S_1$ cond. dominated, given S_2 **then**
          if Feasibility Program 1 is satisfiable for $S = (S_1, S_2)$ **then**
            Return the found NE p

Algorithm for N-Player Games

Algorithm 1 can be interpreted as using the general backtracking algorithm (see, e.g., (Dechter 2003)) to solve a constraint satisfaction problem (CSP) for each support size profile. The variables in each CSP are the supports S_i, and the domain of each S_i is the set of supports of size x_i. While the single constraint is that there must exist a solution to Feasibility Program 1, an extraneous, but easier to check, set of constraints is that no agent plays a conditionally dominated action. The removal of conditionally dominated strategies by Algorithm 1 is similar to using the AC-1 to enforce arc-consistency with respect to these constraints. We use this interpretation to generalize Algorithm 1 for the n-player case. Pseudo-code for Algorithm 2 and its two procedures, Recursive-Backtracking and Iterated Removal of Strictly Dominated Strategies (IRSDS) are given below.[4]

IRSDS takes as input a domain for each player's support. For each agent whose support has been instantiated, the domain contains only that instantiated support, while for each other agent i it contains all supports of size x_i that were not eliminated in a previous call to this procedure. On each pass of the *repeat-until* loop, every action found in at least one support of a player's domain is checked for conditional domination. If a domain becomes empty after the removal of a conditionally dominated action, then the current instantiations of the Recursive-Backtracking are inconsistent, and IRSDS returns *failure*. Because the removal of an action can lead to further domain reductions for other agents, IRSDS repeats until it either returns *failure* or iterates through all actions of all players without finding a dominated action.

Finally, we note that Algorithm 2 is not a strict generalization of Algorithm 1, because it orders the support size profiles first by size, and then by a measure of balance. The reason for the change is that balance (while still significant) is less important for $n > 2$ than it is for $n = 2$. For example, under the model of (McLennan & Berg 2002), for $n > 2$, the probability of the existence of a NE consistent with a particular support profile is no longer zero when the support profile is unbalanced.

[4] Even though our implementation of the backtracking procedure is iterative, for simplicity we present it here in its equivalent, recursive form. Also, the reader familiar with CSPs will recognize that we have employed very basic algorithms for backtracking and for enforcing arc consistency, and we return to this point in the conclusion.

Algorithm 2

for all $x = (x_1, \ldots, x_n)$, sorted in increasing order of, first, $\sum_i x_i$ and, second, $max_{i,j}(x_i - x_j)$ **do**
  $\forall i : S_i \leftarrow NULL$ *//uninstantiated supports*
  $\forall i : D_i \leftarrow \{S_i \subseteq A_i : |S_i| = x_i\}$ *//domain of supports*
  if Recursive-Backtracking$(S, D, 1)$ returns a NE p **then**
    Return p

Procedure 1 Recursive-Backtracking

Input: $S = (S_1, \ldots, S_n)$: a profile of supports
$D = (D_1, \ldots, D_n)$: a profile of domains
i: index of next support to instantiate
Output: A Nash equilibrium p, or $failure$
if $i = n + 1$ **then**
  if Feasibility Program 1 is satisfiable for S **then**
    Return the found NE p
  else
    Return $failure$
else
  for all $d_i \in D_i$ **do**
    $S_i \leftarrow d_i$
    $D_i \leftarrow D_i - \{d_i\}$
    if IRSDS$((\{S_1\}, \ldots, \{S_i\}, D_{i+1}, \ldots, D_n))$ succeeds **then**
      if Recursive-Backtracking$(S, D, i+1)$ returns NE p **then**
        Return p
  Return $failure$

Procedure 2 Iterated Removal of Strictly Dominated Strategies (IRSDS)

Input: $D = (D_1, \ldots, D_n)$: profile of domains
Output: Updated domains, or $failure$
repeat
  $changed \leftarrow false$
  for all $i \in N$ **do**
    for all $a_i \in d_i \in D_i$ **do**
      for all $a'_i \in A_i$ **do**
        if $\forall a_{-i} \in d_{-i} \in D_{-i}, u_i(a_i, a_{-i}) < u_i(a'_i, a_{-i})$ **then**
          $D_i \leftarrow D_i - \{d_i \in D_i : a_i \in d_i\}$
          $changed \leftarrow true$
          if $D_i = \emptyset$ **then**
            return $failure$
until $changed = false$
return D

D1	Bertrand Oligopoly	D2	Bidirectional LEG, Complete Graph
D3	Bidirectional LEG, Random Graph	D4	Bidirectional LEG, Star Graph
D5	Covariance Game: $\rho = 0.9$	D6	Cov. Game: $\rho \in [-1/(N-1), 1]$
D7	Covariance Game: $\rho = 0$	D8	Dispersion Game
D9	Graphical Game, Random Graph	D10	Graphical Game, Road Graph
D11	Graphical Game, Star Graph	D12	Graphical Game, Small-World
D13	Minimum Effort Game	D14	Polymatrix Game, Complete Graph
D15	Polymatrix Game, Random Graph	D16	Polymatrix Game, Road Graph
D17	PolymatrixGame, Small-World	D18	Uniformly Random Game
D19	Travelers Dilemma	D20	Uniform LEG, Complete Graph
D21	Uniform LEG, Random Graph	D22	Uniform LEG, Star Graph
D23	Location Game	D24	War Of Attrition

Table 1: Descriptions of GAMUT distributions.

Experimental Results

To evaluate the performance of our algorithms we ran several sets of experiments. All games were generated by GAMUT (Nudelman et al. 2004), a test-suite that is capable of generating games from a wide variety of classes of games found in the literature. Table 1 provides a brief description of the subset of distributions on which we tested.

A distribution of particular importance is the one most commonly tested on in previous work: D18, the "Uniformly Random Game", in which every payoff in the game is drawn independently from an identical uniform distribution. Also important are distributions D5, D6, and D7, which fall under a "Covariance Game" model studied by (Rinott & Scarsini 2000), in which the payoffs for the n agents for each action profile are drawn from a multivariate normal distribution in which the covariance ρ between the payoffs of each pair of agents is identical. When $\rho = 1$, the game is common-payoff, while $\rho = \frac{-1}{N-1}$ yields minimal correlation, which occurs in zero-sum games. Thus, by altering ρ, we can smoothly transition between these two extreme classes of games.

Our experiments were executed on a cluster of 12 dual-processor, 2.4GHz Pentium machines, running Linux 2.4.20. We capped runs for all algorithms at 1800 seconds. When describing the statistics used to evaluate the algorithms, we will use "unconditional" to refer to the value of the statistic when timeouts are counted as 1800 seconds, and "conditional" to refer to its value excluding timeouts.

When $n = 2$, we solved Feasibility Program 1 using CPLEX 8.0's callable library. For $n > 2$, because the program is nonlinear, we instead solved each instance of the program by executing AMPL, using MINOS as the underlying optimization package. Obviously, we could substitute in any nonlinear solver; and, since a large fraction of our running time is spent on AMPL and MINOS, doing so would greatly affect the overall running time.

Before presenting the empirical results, we note that a comparison of the worst-case running times of our two algorithms and the three we tested against does not distinguish between them, since there exist inputs for each which lead to exponential time.

Results for Two-Player Games

In the first set of experiments, we compared the performance of Algorithm 1 to that of Lemke-Howson (implemented in Gambit, which added the preprocessing step of iterated removal of weakly dominated strategies) on 2-player, 300-action games drawn from 24 of GAMUT's 2-player distributions. Both algorithms were executed on 100 games drawn from each distribution. The time is measured in seconds and plotted on a logarithmic scale.

Figure 1(a) compares the unconditional median runtimes of the two algorithms, and shows that Algorithm 1 performs better on all distributions.[5] However, this does not tell the whole story. For many distributions, it simply reflects the fact that there is a greater than 50% chance that the distribution will generate a game with a pure strategy NE, which our algorithm will then find quickly. Two other important statistics are the percentage of instances solved (Figure 1(b)), and the average runtime conditional on solving the instance (Figure 1(c)). Here, we see that Algorithm 1 completes far more instances on several distributions, and solves fewer on just a single distribution (6 fewer, on D23). Additionally, even on distributions for which we solve far more games, our conditional average runtime is 1 to 2 orders of magnitude smaller.

Clearly, the hardest distribution for our algorithm is D6, which consists of "Covariance Games" in which the covariance ρ is drawn uniformly at random from the range $[-1, 1]$. In fact, neither Algorithm 1 nor Lemke-Howson solved any of the games in another "Covariance Game" distribution in which $\rho = -0.9$, and these results were omitted from the graphs, because the conditional average is undefined for these results. On the other hand, for the distribution "CovarianceGame-Pos" (D5), in which $\rho = 0.9$, both algorithms perform well.

To further investigate this continuum, we sampled 300 values for ρ in the range $[-1, 1]$, with heavier sampling in the transition region and at zero. For each such game, we plotted a point for the runtime of both Algorithm 1 and Lemke-Howson in Figure 1(d).[6] The theoretical results of (Rinott & Scarsini 2000) suggest that the games with lower covariance should be more difficult for Algorithm 1, because they are less likely to have a pure strategy Nash equilibrium. Nevertheless, it is interesting to note the sharpness of the transition that occurs in the $[-0.3, 0]$ interval. More surprisingly, a similarly sharp transition also occurs for Lemke-Howson, despite the fact that the two algorithms operate in unrelated ways. Finally, it is important to note that the transition region for Lemke-Howson is shifted to the right by approximately 0.3, and that, on instances in the easy region for both algorithms, Algorithm 1 is still an order of magnitude faster.

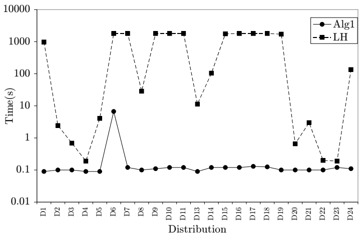
In the third set of experiments we explore the scaling behavior of both algorithms on the "Uniformly Random Game" distribution (D18), as the number of actions increases from 100 to 1000. For each multiple of 100, we generated 20 games. Because space constraints preclude an analysis similar to that of Figures 1(a) through 1(c), we instead plot in Figure 1(e) the *unconditional* average runtime over 20 instances for each data size, with a timeout counted as 1800s. While Lemke-Howson failed to solve any game with more than 600 actions and timed out on some 100-action games, Algorithm 1 solved all instances, and, without the help of cutoff times, still had an advantage of 2 orders of magnitude at 1000 actions.

Results for N-Player Games

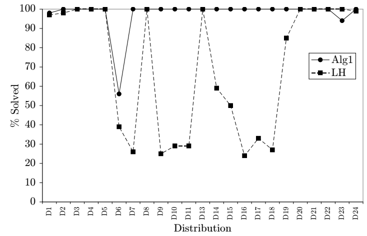
In the next set of experiments we compare Algorithm 2 to Govindan-Wilson and Simplicial Subdivision (which was implemented in Gambit, and thus combined with iterated removal of weakly dominated strategies). First, to compare performance on a fixed problem size we tested on 6-player,

[5] Obviously, the lines connecting data points across distributions for a particular algorithm are meaningless– they were only added to make the graph easier to read.

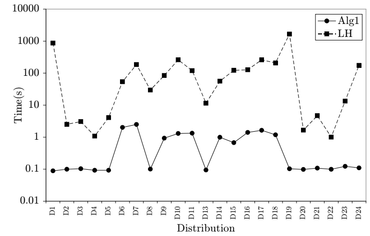
[6] The capped instances for Algorithm 1 were perturbed slightly upward on the graph for clarity.



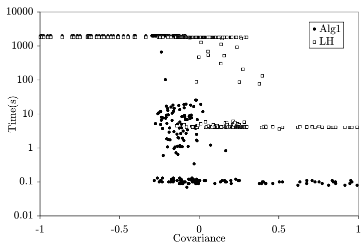
(a) Unconditional median runtime



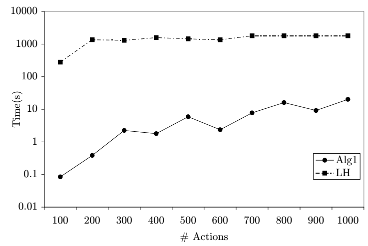
(b) Percentage solved



(c) Average time on solved instances

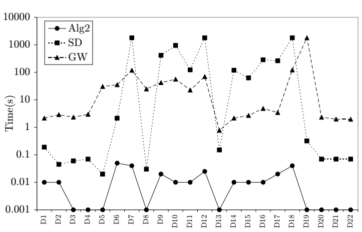


(d) Runtime vs. Covariance

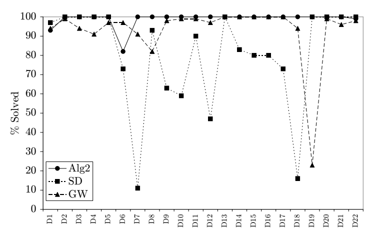


(e) Unconditional average vs. Actions

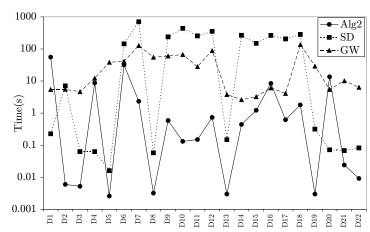
Figure 1: Comparison of Algorithm 1 and Lemke-Howson on 2-player games. Subfigures (a)-(d) are for 300-action games.



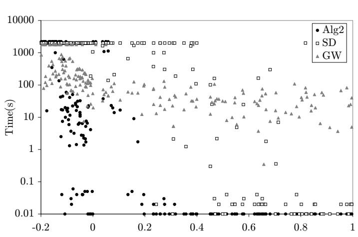
(a) Unconditional median runtimes



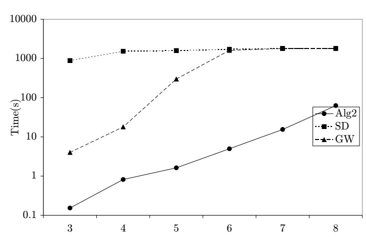
(b) Percentage solved



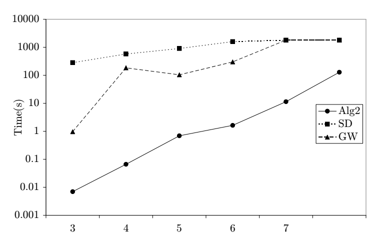
(c) Average time on solved instances



(d) Runtime vs. Covariance



(e) Unconditional average runtime vs. Actions, on 6-player games



(f) Unconditional average runtime vs. Players, on 5-action games

Figure 2: Comparison of Algorithm 2, Simplicial Subdivision, and Govindan-Wilson. Subfigures (a)-(d) are for 6-player, 5-action games.

5-action games drawn from 22 of GAMUT's n-player distributions.[7] While the numbers of players and actions appear small, note that these games have 15625 outcomes and 93750 payoffs. Once again, Figures 2(a), 2(b), and 2(c) show unconditional median runtime, percentage of instances solved, and conditional average runtime, respectively. Algorithm 2 has a very low unconditional median runtime, for the same reason that Algorithm 1 did for two-player games, and outperforms both other algorithms on all distributions. While this dominance does not extend to the other two metrics, the comparison still favors Algorithm 2.

We again investigate the relationship between ρ and the hardness of games under the "Covariance Game" model. For general n-player games, minimal correlation under this model occurs when $\rho = -\frac{1}{n-1}$. Thus, we can only study the range $[-0.2, 1]$ for 6-player games. Figure 2(d) shows the results for 6-player 5-action games. Algorithm 2, over the range $[-0.1, 0]$, experiences a transition in hardness that is even sharper than that of Algorithm 1. Simplicial Subdivision also undergoes a transition, which is not as sharp, that begins at a much larger value of ρ (around 0.4). However, the running time of Govindan-Wilson is only slightly affected by the covariance, as it neither suffers as much for small values of ρ nor benefits as much from large values.

Finally, Figures 2(e) and 2(f) compare the scaling behavior (in terms of unconditional average runtimes) of the three algorithms: the former holds the number of players constant at 6 and varies the number of actions from 3 to 8, while the latter holds the number of actions constant at 5, and varies the number of players from 3 to 8. In both experiments, both Simplicial Subdivision and Govindan-Wilson solve no instances for the largest two sizes, while Algorithm 2 still finds a solution for most games.

Conclusion and Future Work

In this paper, we presented two algorithms for finding a sample Nash equilibrium. Both use backtracking approaches (augmented with pruning) to search the space of support profiles, favoring supports that are small and balanced. Both also outperform the current state of the art.

The most difficult games we encountered came from the "Covariance Game" model, as the covariance approaches its minimal value, and this is a natural target for future algorithm development. We expect these games to be hard in general, because, empirically, we found that as the covariance decreases, the number of equilibria decreases, and the equilibria that do exist are more likely to have support sizes near one half of the number of actions, which is the support size with the largest number of supports.

One direction for future work is to employ more sophisticated CSP techniques. The main goal of this paper was to show that our general search method performs well in practice, and there are many other CSP search and inference strategies which may improve its efficiency. Another promising direction to explore is local search, in which the state space is the set of all possible supports, and the available moves are to add or delete an action from the support of a player. While the fact that no equilibrium exists for a particular support does not give any guidance as to which neighboring support to explore next, one could use a relaxation of Feasibility Program 1 that penalizes infeasibility through an objective function. More generally, our results show that AI techniques can be successfully applied to this problem, and we have only scratched the surface of possibilities along this direction.

[7] Two distributions from the tests of 2-player games are missing here, due to the fact that they do not naturally generalize to more than 2 players.

References

Blum, B.; Shelton, C. R.; and Koller, D. 2003. A continuation method for Nash equilibria in structured games. In *IJCAI-03*.

Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann.

Dickhaut, J., and Kaplan, T. 1991. A program for finding Nash equilibria. *The Mathematica Journal* 87–93.

Govindan, S., and Wilson, R. 2003. A global newton method to compute Nash equilibria. In *Journal of Economic Theory*.

Lemke, C., and Howson, J. 1964. Equilibrium points of bimatrix games. *Journal of the Society for Industrial and Applied Mathematics* 12:413–423.

McKelvey, R., and McLennan, A. 1996. Computation of equilibria in finite games. In H. Amman, D. Kendrick, J. R., ed., *Handbook of Computational Economics*, volume I. Elsevier. 87–142.

McKelvey, R.; McLennan, A.; and Turocy, T. 2003. Gambit: Software tools for game theory, version 0.97.0.5. Available at http://econweb.tamu.edu/gambit/.

McLennan, A., and Berg, J. 2002. The asymptotic expected number of Nash equilibria of two player normal form games. Mimeo, University of Minnesota.

Nash, J. 1950. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences of the United States of America* 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-04*.

Papadimitriou, C. 2001. Algorithms, games, and the internet. In *STOC-01*, 749–753.

Rinott, Y., and Scarsini, M. 2000. On the number of pure strategy Nash equilibria in random games. *Games and Economic Behavior* 33:274–293.

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. *Mathematics of Operations Research*.

von Stengel, B. 2002. Computing equilibria for two-person games. In Aumann, R., and Hart, S., eds., *Handbook of Game Theory*, volume 3. North-Holland. chapter 45, 1723–1759.