Simple Search Methods For Finding A Nash Equilibrium

Ryan Porter, Eugene Nudelman, and Yoav Shoham [1]

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. Both algorithms bias the search towards supports that are small and balanced, and employ a backtracking procedure to efficiently explore these supports. Making use of a new comprehensive testbed, 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.

Key words: Nash equilibrium, Computer Science, Algorithms

[1] We would like to thank the members of Stanford multiagent research group, Bob Wilson, and Christian Shelton for many useful comments and discussions. We also thank anonymous reviewers for thoughtful comments. This work was supported in part by DARPA grant F30602-00-2-0598 and in part by the National Science

1 Introduction

This paper addresses the problem of finding a sample Nash equilibrium of a given normal form game. This notorious problem has been described as the "most fundamental computational problem" at the interface of computer science and game theory (Papadimitriou, 2001). Despite several decades of research into this problem it remains thorny; its precise computational complexity is unknown, and new algorithms have been relatively few and far between.

The present paper makes two related contributions to the problem, both bringing to bear fundamental lessons from computer science. In a nutshell, they are these:

- The use of heuristic search techniques in algorithms.
- The use of an extensive test suite to evaluate algorithms.

The surprising result of applying these insights is that, for relevant classes of games, even very simple heuristic methods are quite effective, and significantly outperform the relatively sophisticated algorithms developed in the past. We expand on these two points below.

In the developments of novel algorithms, one can identify two extremes. The first extreme is to gain deep insight into the structure of the problem, and craft highly specialized algorithms based on this insight. The other extreme is to identify relatively shallow heuristics, and hope that these, coupled with the ever increasing computing power, are sufficient. While the first extreme is certainly more appealing, the latter often holds the upper hand in practice.[2] For

Foundation under ITR IIS-0205633.

[2] Of course, there is no difference in kind between the two extremes. To be effective,

example, a deep understanding of linear programming yielded polynomial-time interior point methods (Karmarkar, 1984), though in practice one tends to use methods based on the exponential Simplex algorithm (Wood and Dantzig, 1949). In reality, neither extreme is common. It is more usual to find a baseline algorithm that contains many choice points based on some amount of insight into the problem, and then apply heuristics to making these choices. For example, for the problem of propositional satisfiability, the current state of the art solvers apply heuristics to search the space spanned by the underlying Davis-Putnam procedure (Cook and Mitchell, 1997).

The use of heuristics, while essential, raises the question of algorithm evaluation. The gold standard for algorithms is trifold: soundness, completeness, and low computational complexity. An algorithm is sound if any proposed solution that it returns is in fact a solution, and it is complete if, whenever at least one solution exists, the algorithm finds one. Low computational complexity is generally taken to mean that the worst-case running time is polynomial in the size of the input.[3]

heuristics too must embody some insight into the problem. However, this insight tends to be limited and local, yielding a rule of thumb that aids in guiding the search through the space of possible solutions or algorithms, but does not directly yield a solution.

[3] In rare cases this is not sufficient. For example, in the case of linear programming, Khachiyan's ellipsoid method (Khachiyan, 1979) was the first polynomial-time algorithm, but, in practice, it could not compete with either the existing, exponential simplex method (Wood and Dantzig, 1949), or polynomial interior point methods (Karmarkar, 1984) that would later be developed. However, this will not concern us here, since there is no known polynomial-time complexity algorithm for the problem in question – computing a sample Nash equilibrium – and there are strong

Of course, in many cases, it is not possible (or has turned out to be extremely difficult) to achieve all three simultaneously. This is particularly true when one uses heuristic methods. In this paper we focus on approaches that sacrifice the third goal, that of low worst-case complexity. Without a worst-case guarantee (and even worse, when one knows that the running time is exponential in the worst case), an algorithm must be evaluated using empirical tests, in which case the choice of problem distributions on which it is tested becomes critical. This is another important lesson from computer science – one should spend considerable effort devising an appropriate test suite, one that faithfully mirrors the domain in which the algorithms will be applied.

With these computer science lessons in mind, let us consider the extant game theoretic literature on computing a sample Nash equilibrium. Algorithm development has clearly been of the first kind, namely exploiting insights into the structure of the problem. For 2-player games, the problem can be formulated as a linear complementarity problem (LCP). The Lemke-Howson algorithm (Lemke and Howson, 1964) is based on a general method for solving LCPs. Despite its age, this algorithm has remained the state of the art for 2-player games. For n-player games, the best existing algorithms are Simplicial Subdivision (van der Laan et al., 1987) and Govindan-Wilson, which was introduced by Govindan and Wilson (2003) and extended and efficiently implemented by Blum et al. (2003). Each of these algorithms is based on non-trivial insights into the mathematical structure of the problem.

From the evaluation point of view, these algorithms are all sound and complete, but not of low complexity. Specifically, they all have a provably expo-

suspicions that one does not exist.

nential worst-case running time. The existing literature does not provide a useful measure of their empirical performance, because most tests are only on so-called "random" games, in which every payoff is drawn independently from a uniform distribution. This is despite the fact that this distribution is widely recognized to have rather specific properties that are not representative of problem domains of interest.

In this paper we diverge from the traditional game-theoretic approach in two fundamental ways: (1) we propose new algorithms that are based on (relatively) shallow heuristics, and (2) we test our algorithms, along with existing ones, extensively through the use of a new computational testbed. The result is a pair of algorithms (one for 2-player games, and another for n-player games, for $n > 2$) that we show to perform very well in practice. Specifically, they outperform previous algorithms, Lemke-Howson on 2-player games, and Simplicial Subdivision and Govindan-Wilson on n-player games, sometimes dramatically.

The basic idea behind our two search algorithms is simple. Recall that, while the general problem of computing a Nash equilibrium (NE) is a complementarity problem, computing whether there exists a NE with a *particular support* [4] 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 of the algorithms are biased towards simple solutions through their pref-

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

erence for small supports. Since it turns out that games drawn from classes that researchers have focused on in the past tend to have (at least one) "simple" NE, 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. While we too conduct tests on "random" games (indeed, we will have more to say about how "random" games vary along at least one important dimension), we also test 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 existing algorithms for the problem of finding a sample NE. We then define 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. We then briefly describe the nature of the equilibria that are found by different algorithms on our data. In the final section, we conclude and describe opportunities for future work.

2 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 of available strategies: $\mathcal{P}_i = \{p_i : A_i \rightarrow [0, 1] | \sum_{a_i \in A_i} p_i(a_i) = 1\}$. 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 (NE) if:* $\forall i \in N, a_i \in A_i : \quad u_i(a_i, p^*_{-i}) \leq u_i(p^*_i, p^*_{-i})$

3 Existing Algorithms

Despite the fact that Nash equilibrium is arguably the most important concept in game theory, remarkably little is known about the problem of computing a sample NE in a normal-form game. We know that every finite, normal form game is guaranteed to have at least one NE (Nash, 1950). Beyond that, all evidence points to this being a hard problem (Gilboa and Zemel, 1989; Conitzer and Sandholm, 2003), but it does not fall into a standard complexity class (Papadimitriou, 2001), because it cannot be cast as a decision problem. Instead, Megiddo and Papadimitriou (1991) developed a new (but rather limited) complexity class, TFNP, to encompass such problems.

In this section, we provide a brief overview of the relevant algorithms. In addition to specific references given for each algorithm, further explanation can be found in two thorough surveys on NE computation – von Stengel (2002); McKelvey and McLennan (1996).

The most commonly-used algorithm for finding a NE in a two-player game is the Lemke-Howson algorithm (Lemke and Howson, 1964), which is a special case of Lemke's method (Lemke, 1965) for solving linear complementarity problems. The Lemke-Howson algorithm is a complementary pivoting algorithm, where an arbitrary selection of an action for the first player determines the first pivot, after which every successive pivot is determined uniquely by the current state of the algorithm, until an equilibrium is found. Thus, each action for the first player can be thought of as defining a path from the starting point (the extraneous solution of all players assigning probability zero to all actions) to a NE. In the implementation of Lemke-Howson in Gambit (McKelvey et al., 2004), the first action of the first player is selected.

For n-player games, until recently, Simplicial Subdivision (van der Laan et al., 1987), and its variants, were the state of the art. This approach approximates a fixed point of a function (e.g., the best-response correspondence) which is defined on a simplotope (a product of simplices). The approximation is achieved by triangulating the simplotope with a mesh of a given granularity, and traversing the triangulation along a fixed path. The worst-case running time of this procedure is exponential in dimension and accuracy (McKelvey and McLennan, 1996).

More recently, Govindan and Wilson (2003) introduced a continuation method for a NE in an n-player game. Govindan-Wilson works by first perturbing a game to one that has a known equilibrium, and then by tracing the solution back to the original game as the magnitude of the perturbation approaches zero. The structure theorem of Kohlberg and Mertens (1986) guarantees that it is possible to trace both the game and a solution simultaneously. This method has been efficiently implemented by Blum et al. (2003), who also extended it to solve graphical games and Multi-Agent Influence Diagrams (Koller and Milch, 2001).

Our algorithm in spirit is closest to the procedure described by Dickhaut and Kaplan (1991) for finding all NE. Their program enumerates all possible pairs of supports for a two-player game. For each pair of supports, it solves a feasibility program (similar to the one we will describe below) to check whether there exists a NE consistent with this pair. A similar enumeration method was suggested earlier by Mangasarian (1964), based on enumerating vertices of a polytope.

Clearly, either of these two enumeration methods could be converted into an algorithm for finding a sample NE by simply stopping after finding the first NE. However, because the purpose of these algorithms is to instead find all NE, no heuristics are employed to speed the computation of the first NE.

4 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 $S = (S_1, \ldots, S_n)$ as input, Feasibility Program 1, below, gives the formal description of a program for finding a NE p consistent with S (if such an strategy profile exists). 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 NE.

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 $a'_i \in S_i$; thus, simply plugging in $S_i = A_i$ would not necessarily yield a Nash equilibrium as a solution.

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 $v = (v_1, \ldots, v_n)$ such that:

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

5 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 and 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:* $\exists a'_i \in A_i \; \forall a_{-i} \in R_{-i} : \quad u_i(a_i, a_{-i}) < u_i(a'_i, a_{-i})$

Observe, that this definition is strict, because, in a Nash Equilibrium, no action that is played with positive probability can be conditionally dominated given the actions in the support of the opponents' strategies.

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 conditionally dominated, given $S_1 \}$

    if $\nexists a_1 \in S_1$ conditionally dominated, given A'_2 **then**

      for all $S_2 \subseteq A'_2$ s.t. $|S_2| = x_2$ **do**

        if $\nexists a_1 \in S_1$ conditionally dominated, given S_2 **then**

          if Feasibility Program 1 is satisfiable for $S = (S_1, S_2)$ **then**

            Return the found NE p

6 Algorithm for N-Player Games

Algorithm 1 can be interpreted as using the general backtracking algorithm to solve a constraint satisfaction problem (CSP) for each support size profile (for an introduction to CSPs, see, for example, Dechter (2003)). 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 algorithm 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. [5]

[5] 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

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 in 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.

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

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 and 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

backtracking and for enforcing arc consistency, and we return to this point in the conclusion.

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 $failure$

support profile is unbalanced.[6]

[6] While this change of ordering does provide substantial improvements, the algorithm could certainly still be improved by adding more complex heuristics. For example, McKelvey and McLennan (1997) shows that, in a generic game, there cannot be a totally mixed NE if the size of one player's support exceeds the sum of the sizes of all other players' supports. However, we believe that handling cases such as this one would provide a relatively minor performance improvement, since our algorithm often finds a NE with small support.

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 $a_i \in \bigcup_{d_i \in D_i} d_i$ **do**

      for all $a'_i \in A_i$ **do**

        if a_i is conditionally dominated by a'_i, given $\bigcup_{d_{-i} \in D_{-i}} d_{-i}$ **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

7 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

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.

Game" model studied by Rinott and 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.

It is worth noting that, while the different distributions in GAMUT vary in their structure, some randomness must be injected into the generator in order to create a distribution over games of that structure. Our results then show that games drawn from these distributions are similar to that of the "Uniformly Random Game" in that they are likely to have a pure strategy NE, despite the imposed structure. Thus, the success of our algorithms is a reflection on the structure of games of interest to researchers as much as it is a demonstration of the techniques we employ.

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 (ILOG, 2004) [7]. For $n > 2$, because the program is nonlinear, we instead solved each instance of the program by executing AMPL, using MINOS (Murtagh and Saunders, 2004) 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 they all have exponential worst-case complexity.

7.1 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 preprocess-

[7] We note that Lemke-Howson uses a mathematical software that is less efficient than CPLEX, which undoubtedly has an effect on the running time comparisons between the two algorithms. However, the difference is not nearly great enough to explain the observed gap between the algorithms.

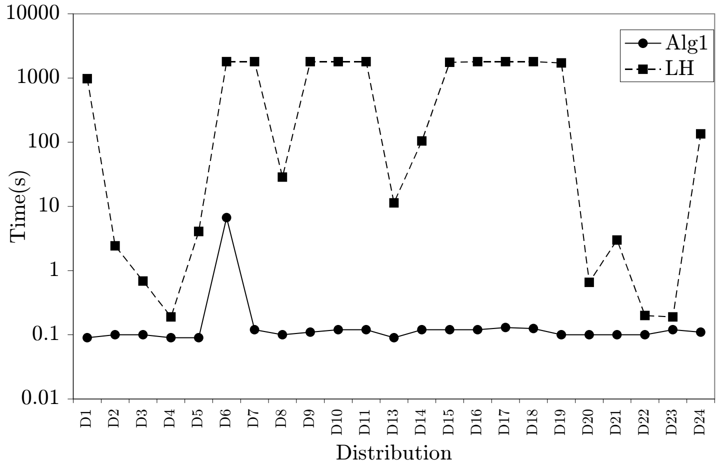

Fig. 1. Unconditional median running times for Algorithm 1 and Lemke-Howson on 2-player, 300-action games.

ing step of iterated removal of weakly dominated strategies) on 2-player, 300-action games drawn from 24 of GAMUT's 2-player distributions. All algorithms were executed on 100 games drawn from each distribution. The time is measured in seconds and plotted on a logarithmic scale.

Figure 1 compares the unconditional median running times of the algorithms, and shows that Algorithm 1 outperforms Lemke-Howson on all distributions. [8] 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 Algorithm 1 will then find quickly. Two other important statistics are the percentage of instances solved (Figure 2), and the average running time conditional on solving the instance (Figure 3). Here, we see that Algorithm 1 completes far more instances than Lemke-Howson on several distributions, and solves fewer on just a single distribution (6 fewer, on D23).

[8] 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.

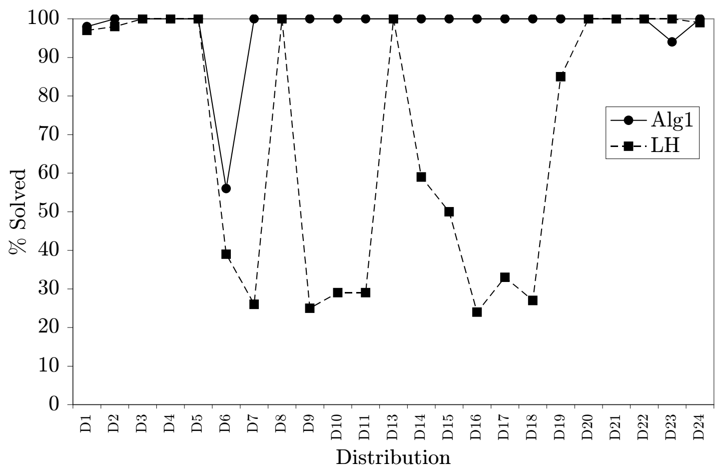

Fig. 2. Percentage solved by Algorithm 1 and Lemke-Howson on 2-player, 300-action games.

Fig. 3. Average running time on solved instances for Algorithm 1 and Lemke-Howson on 2-player, 300-action games.

Figure 3 further highlights the differences between the two algorithms. It demonstrates that even on distributions for which Algorithm 1 solves far more games, its conditional average running time is 1 to 2 orders of magnitude smaller than Lemke-Howson.

Clearly, the hardest distribution for both of the algorithms is D6, which con-

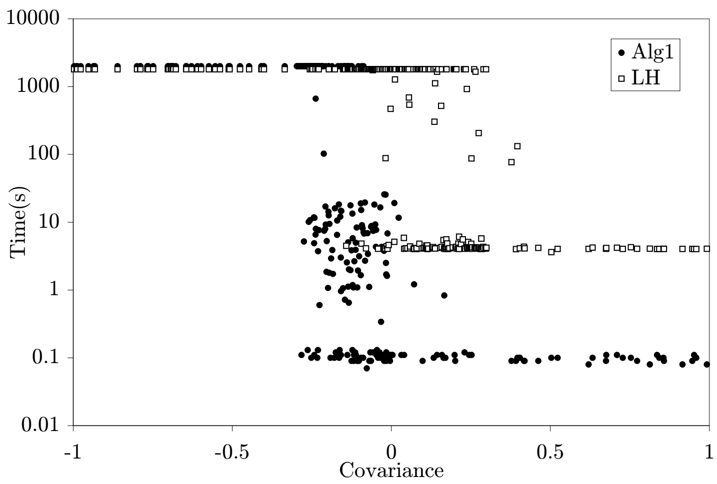

Fig. 4. Running time for Algorithm 1 and Lemke-Howson on 2-player, 300-action "Covariance Games".

sists of "Covariance Games" in which the covariance ρ is drawn uniformly at random from the range $[-1, 1]$. In fact, neither of the algorithms 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 $\rho = 0.9$, all 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 running time of Algorithm 1, Lemke-Howson in Figure 4. [9]

The theoretical results of Rinott and 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 NE. Nevertheless, it is interesting to

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

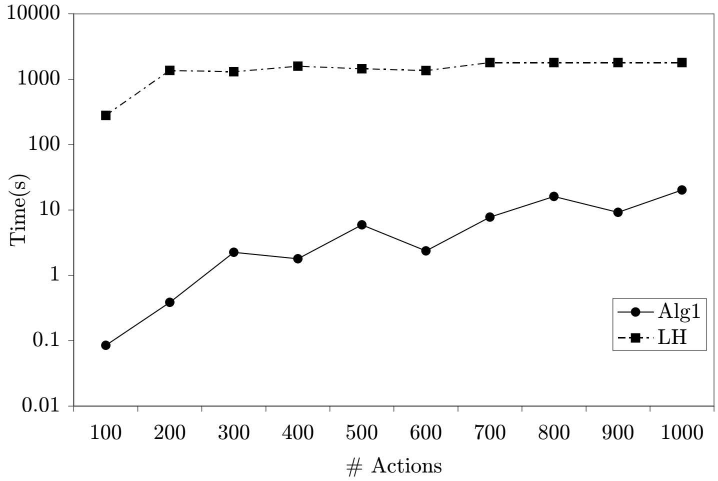

Fig. 5. Unconditional average running time for Algorithm 1 and Lemke-Howson on 2-player "Uniformly Random Games", as the number of actions is scaled.

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 it operates in an unrelated way to Algorithm 1. It is also important to note that the transition region for Lemke-Howson is shifted to the right by approximately 0.3 relative to that of Algorithm 1, and that on instances in the easy region for both algorithms, Algorithm 1 is still an order of magnitude faster.

Finally, we explored the scaling behavior of all algorithms by generating 20 games from the "Uniformly Random Game" distribution (D18) for each multiple of 100 from 100 to 1000 actions. Figure 5 presents the *unconditional* average running time, 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.

7.2 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-action games drawn from 22 of GAMUT's n-player distributions.[10] While the numbers of players and actions appear small, note that these games have 15625 outcomes and 93750 payoffs. Once again, Figures 6, 7, and 8 show unconditional median running time, percentage of instances solved, and conditional average running time, respectively. Algorithm 2 has a very low unconditional median running time, 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 9 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

[10] 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.

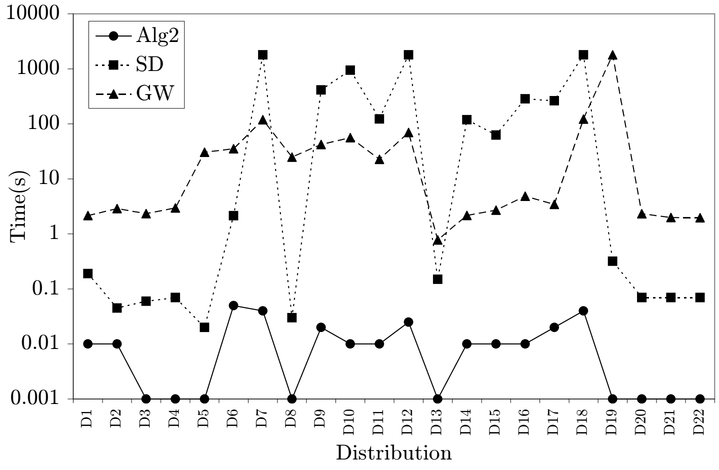

Fig. 6. Unconditional median running times for Algorithm 2, Simplicial Subdivision, and Govindan-Wilson on 6-player, 5-action games.

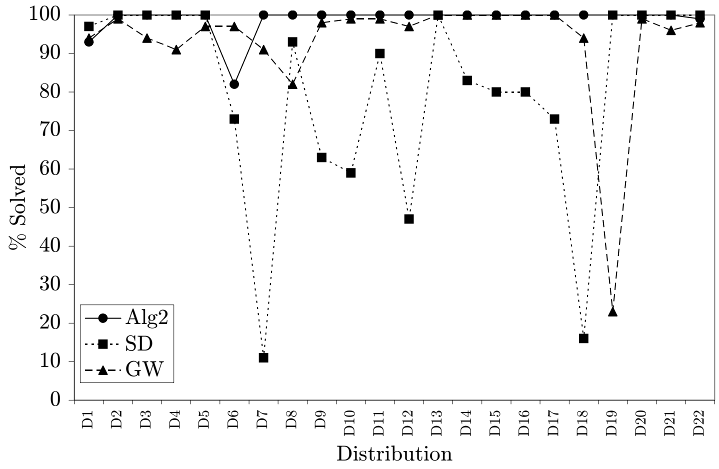

Fig. 7. Percentage solved by for Algorithm 2, Simplicial Subdivision, and Govindan-Wilson on 6-player, 5-action games.

of ρ nor benefits as much from large values.

Finally, Figures 10 and 11 compare the scaling behavior (in terms of unconditional average running times) 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

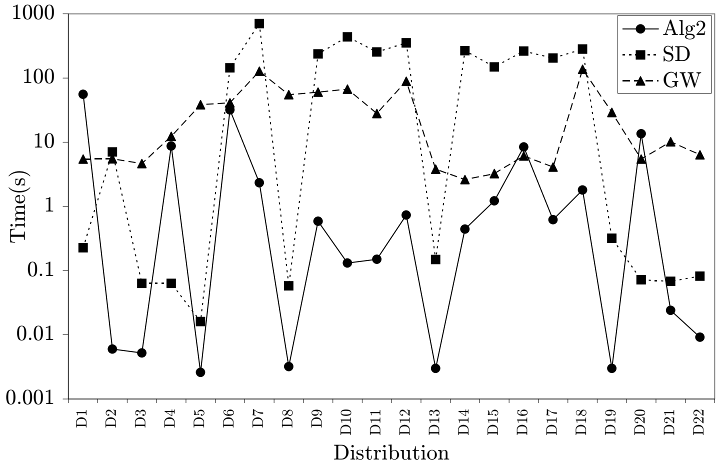

Fig. 8. Average running time on solved instances for Algorithm 2, Simplicial Subdivision, and Govindan-Wilson on 6-player, 5-action games.

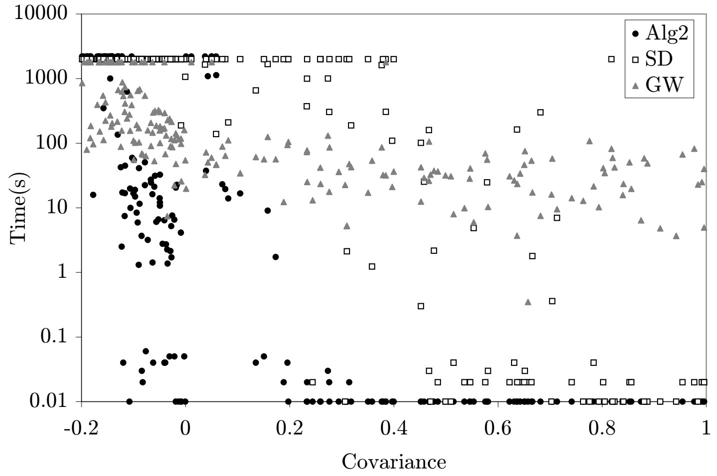

Fig. 9. Running time for Algorithm 2, Simplicial Subdivision, and Govindan-Wilson on 6-player, 5-action "Covariance Games".

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.

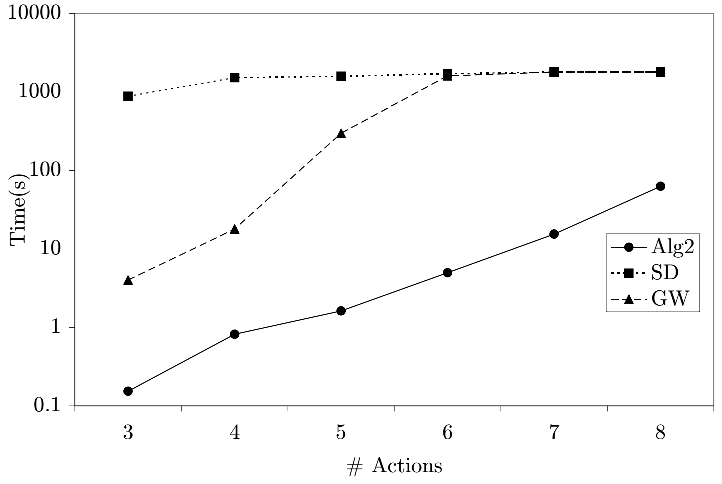

Fig. 10. Unconditional average time; for Algorithm 2, Simplicial Subdivision, and Govindan-Wilson on 6-player "Uniformly Random Games", as the number of actions is scaled.

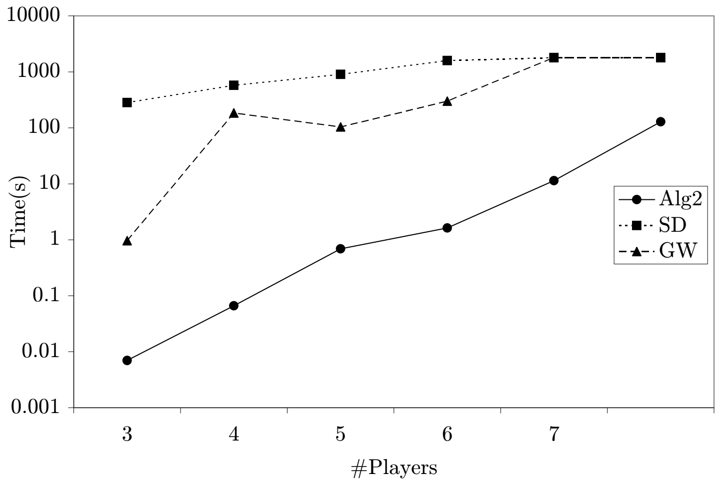

Fig. 11. Unconditional average time; for Algorithm 2, Simplicial Subdivision, and Govindan-Wilson on 5-action "Uniformly Random Games", as the number of players is scaled.

7.3 On the Distribution of Support Sizes

In this section we describe the nature of equilibria that are present in our dataset, and the kinds of equilibria discovered by different algorithms.

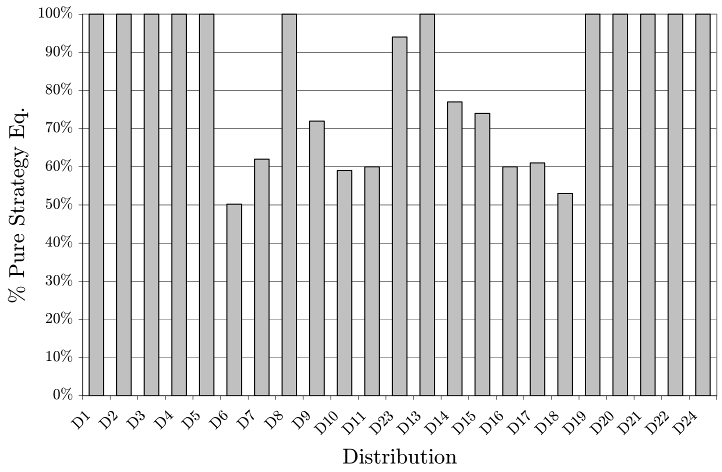

Fig. 12. Percentage of instances possessing a pure strategy NE, for 2-player, 300-action games.

7.3.1 Pure Strategy Equilibria

By definition, the first step of both Algorithm 1 and Algorithm 2 is to check whether the input game has a pure strategy equilibrium. This step can be performed very fast even on large games. It is interesting to see just how much of performance of these algorithms is due to the existence of pure strategy equilibria. Figures 12 and 13 show the fraction of games in each distribution that posses a PSNE, for 2-player, 300-action and 6-player, 5-action games, respectively. These figures demonstrate that a pure strategy equilibrium is present in many, though not all, games generated by GAMUT. We note, however, that our algorithms often perform well even on distributions that don't all have a PSNE, as they sometimes find equilibria of larger sizes.

These graphs thus demonstrate that looking for pure strategy Nash equilibria could be a useful preprocessing step for Lemke-Howson, Simplicial Subdivision , and Govindan-Wilson, but that at the same time, even with such a preprocessing step, these algorithms would not catch up with Algorithm 1

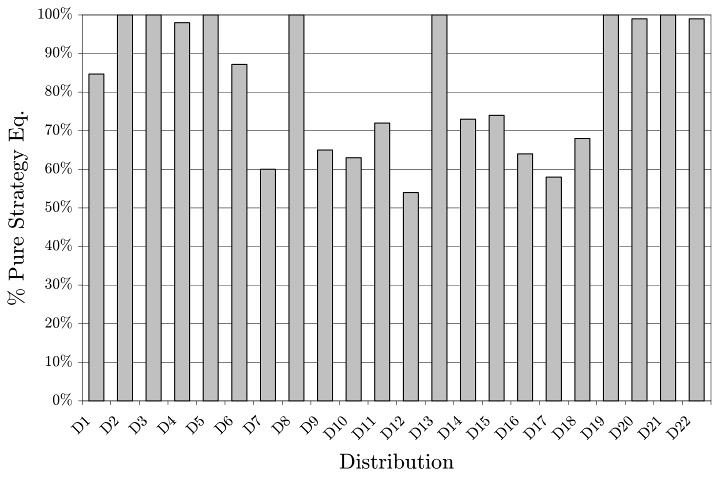

Fig. 13. Percentage of instances possessing a pure strategy NE, for 6-player, 5-action games.

and Algorithm 2. This step is essentially the first step of our algorithms, but in the cases that are not caught by this step, our algorithms degrade gracefully while the others do not.

7.3.2 Support Sizes Found

Many researchers feel that equilibria with small support sizes are easier to justify. It is, therefore, interesting to see what kinds of equilibria the different algorithms find in games of interest. Specifically, we would like to see whether other algorithms find equilibria that correspond to our own heuristics of small and balanced supports.

Figures 14 and 15 show the average total size of the support (i.e. $\sum_i x_i$) of the first equilibrium found by each algorithm on 2-player, 300-action and 6-player, 5-action games, respectively. Total size 2 in figure 14, and total size 6 in figure 15 correspond to a pure strategy equilibrium. As expected, our algorithms tend to find equilibria with much smaller supports than those of

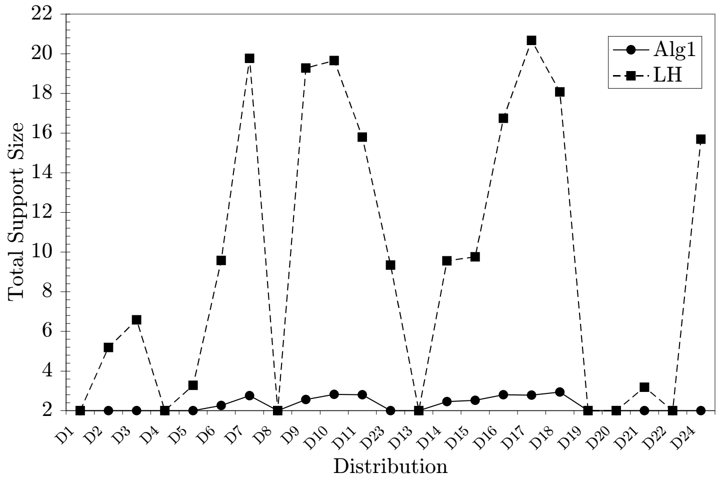

Fig. 14. Average of total support size for found equilibria, on 2-player, 300-action games.

other algorithms, even on distributions where pure strategy Nash equilibria often exist. Notice, however, that all of the algorithms have a bias towards somewhat small support sizes. On average, the equilibria found by Lemke-Howson in 2-player games have at most 10 actions per player (out of 300), while the equilibria found by Govindan-Wilson on 6-player games have on average 2.3 actions per player out of 5. For comparison, the absolute largest equilibrium found by Govindan-Wilson across all instances had between 3 and 5 actions for each player (4.17 on average). The games with these large equilibria all came from Covariance Game distributions. They all also possessed either a PSNE, or a NE with at most 2 actions per player, and were all quickly solved by Algorithm 2. The largest equilibrium among 2-player games that was found by Lemke-Howson had 122 actions for both players. That game came from distribution D23.

The second search bias that our algorithms employ is to look at balanced supports first. Figure 16 shows the average measure of support balance (i.e. $max_{i,j}(x_i - x_j)$) of the first equilibrium found by each algorithm on 6-player,

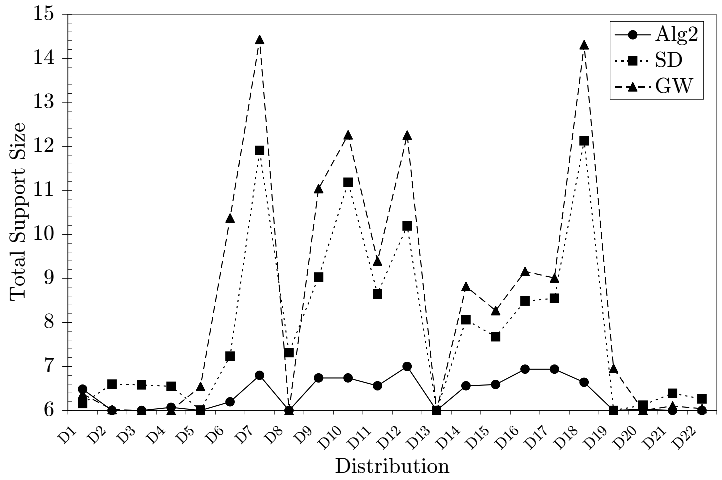

Fig. 15. Average of total support size for found equilibria, on 6-player, 5-action games.

5-action games. We omit a similar graph for 2-player games, since almost always perfectly balanced supports were found by both Algorithm 1 and Lemke-Howson. The only exception was distribution D24, on which Lemke-Howson found equilibria with average (dis)balance of 13.48. On 6-player games, as again expected, Algorithm 2 tends to find equilibria that are almost balanced, while the other two algorithms find much less balanced supports. Nevertheless, it is important to note that the balance of the supports found by Algorithm 2 is not uniformly zero, suggesting that it still finds many equilibria that are not pure strategy. Thus, once again, we see that using the preprocessing step of finding PSNEs first, while greatly improving previous algorithms, would not be enough to close the gap between them on the one hand and Algorithm 2 on the other.

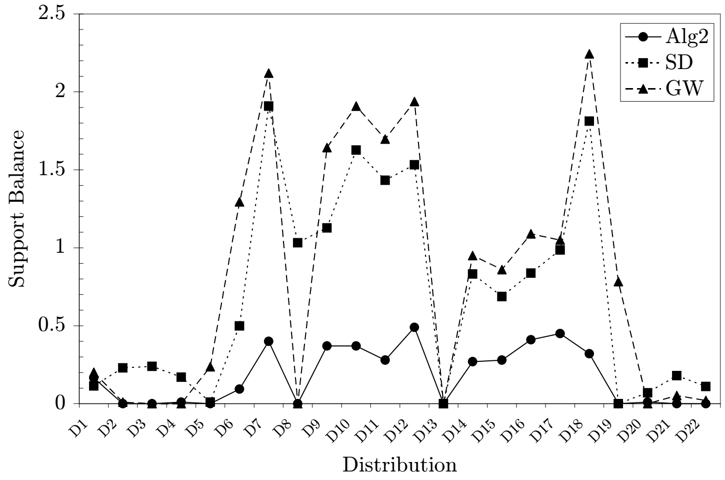

Fig. 16. Average of support size balance for found equilibria, on 6-player, 5-action games.

8 Conclusion and Future Work

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

Another approach that we have tried, and found to be successful, is to overlay a particular heuristic onto Lemke-Howson. Recall that, in the execution of Lemke-Howson, the first pivot is determined by an arbitrary choice of an action of the first player. This initial pivot then determines a path to a NE. Thus, we can construct an algorithm that, like our two algorithms, is biased towards "simple" solutions through the use of breadth-first search– initially, it branches on each possible starting pivot; then, on each iteration, it performs a single pivot for each possible Lemke-Howson execution, after which it checks whether a NE has been found. This modification significantly outperforms the standard Lemke-Howson algorithm on most classes of games. However, it still performs worse than our Algorithm 1.

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 as well, 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 methods perform well in practice, and there are many other CSP search and inference strategies which may improve its efficiency. Another promising direction is 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.

References

Blum, B., Shelton, C., Koller, D., 2003. A continuation method for Nash equilibria in structured games. In: Proceedings of the 18th International Joint

Conference on Artificial Intelligence.

Conitzer, V., Sandholm, T., 2003. Complexity results about Nash equilibria. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence.

Cook, S. A., Mitchell, D. G., 1997. Finding hard instances of the satisfiability problem: A survey. In: Du, Gu, Pardalos (Eds.), Satisfiability Problem: Theory and Applications. Vol. 35. American Mathematical Society, pp. 1–17. URL `citeseer.ist.psu.edu/cook97finding.html`

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

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

Gilboa, I., Zemel, E., 1989. Nash and correlated equilibria: Some complexity considerations. Games and Economic Behavior 1, 80–93.

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

ILOG, 2004. Cplex. Http://www.ilog.com/products/cplex.

Karmarkar, N., 1984. A new polynomial-time algorithm for linear programming. Combinatorica 4, 373–395.

Khachiyan, L., 1979. A polynomial time algorithm for linear programming. Dokl. Akad. Nauk SSSR 244, 1093–1096.

Kohlberg, E., Mertens, J., 1986. On the strategic stability of equilibria. Econometrica 54.

Koller, D., Milch, B., 2001. Multi-agent influence diagrams for representing and solving game. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence.

Lemke, C., 1965. Bimatrix equilibrium points and mathematical programming. Management Science 11, 681–689.

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

Mangasarian, O., 1964. Equilibrium points of bimatrix games. Journal of the Society for Industrial and Applied Mathematics 12, 780–780.

McKelvey, R., McLennan, A., 1996. Computation of equilibria in finite games. In: H. Amman, D. Kendrick, J. R. (Ed.), Handbook of Computational Economics. Vol. I. Elsevier, pp. 87–142.

McKelvey, R., McLennan, A., 1997. The maximal number of regular totally mixed nash equilibria. Journal of Economic Theory 72, 411–425.

McKelvey, R., McLennan, A., Turocy, T., 2004. Gambit: Software tools for game theory. Available at http://econweb.tamu.edu/gambit/.

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

Megiddo, N., Papadimitriou, C., 1991. A note on total functions, existence theorems and complexity. Theoretical Computer Science 81, 317–324.

Murtagh, B., Saunders, M., 2004. Minos. Http://www.sbsi-sol-optimize.com.

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., Leyton-Brown, K., 2004. 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.

Papadimitriou, C., 2001. Algorithms, games, and the internet. In: Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing. pp. 749–753.

Rinott, Y., 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., 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., Hart, S. (Eds.), Handbook of Game Theory. Vol. 3. North-Holland, Ch. 45, pp. 1723–1759.

Wood, M., Dantzig, G., 1949. Programming of interdependent activities. i. general discussion. Econometrica 17, 193–199.