HAL
open science

Computing the Equilibria of Bimatrix Games using Dominance Heuristics

Raghav Aras, Alain Dutech, François Charpillet

To cite this version:

Raghav Aras, Alain Dutech, François Charpillet. Computing the Equilibria of Bimatrix Games using Dominance Heuristics. 2006. hal-00118840

HAL Id: hal-00118840

https://hal.science/hal-00118840v1

Preprint submitted on 6 Dec 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Computing the Equilibria of Bimatrix Games using Dominance Heuristics

Raghav Aras
LORIA
Nancy, France
aras@loria.fr

Alain Dutech
LORIA
Nancy, France
dutech@loria.fr

François Charpillet
LORIA
Nancy, France
charpillet@loria.fr

hal-00118840, version 1 - 6 Dec 2006

Abstract

We propose a formulation of a general-sum bimatrix game as a bipartite directed graph with the objective of establishing a correspondence between the set of the relevant structures of the graph (in particular elementary cycles) and the set of the Nash equilibria of the game. We show that finding the set of elementary cycles of the graph permits the computation of the set of equilibria. For games whose graphs have a sparse adjacency matrix, this serves as a good heuristic for computing the set of equilibria. The heuristic also allows the discarding of sections of the support space that do not yield any equilibrium, thus serving as a useful preprocessing step for algorithms that compute the equilibria through support enumeration.

1 Introduction

Game theory is the study of strategic decision-making. The decision-makers are called players. In bimatrix games, only 2 players are involved. Each player makes one decision in the game. This decision pertains to the probability distribution the player conceives over the set of strategies available to him. As a function of the two decisions, each player receives a real-valued number, called his payoff. When each player tries to maximize his own payoff, their decisions form an *equilibrium* in which neither player can increase his own payoff by changing his own decision given that the other player sticks to his part of the pair. Using Brouwer's fixed-point theorem, John Nash [8] proved the existence of such an equilibrium, since named after him, for every finite game. It is equally true that a game may have more than one such equilibrium.

The problem of determining the Nash equilibria of a game has occupied much of research in computational game theory (see [6], [13] for excellent surveys). The principle algorithm for finding a Nash equilibrium of a general-sum game is the Lemke-Howson algorithm ([4], [5]). It solves a linear complementarity program (LCP) formulation of the game. While in practice this algorithm is quite efficient, its worst case complexity is exponential [11]. Despite its age, the algorithm remains the state of the art.

Heuristics about the game structure, therefore, invite interest in finding a sample Nash equilibrium, particularly when the game is large-sized. It is well-understood that most randomly-generated games allow an equilibrium with *small and balanced* supports [1]. There is theoretical and empirical evidence that in randomly generated bimatrix games ([7]) as n, the size of the game increases, the probability of the game having an equilibrium where each player's support has size n becomes vanishingly small. This heuristic is also used by [2] and [10] who use a much simpler algorithm than the Lemke-Howson to find a sample Nash equilibrium. Their algorithm enumerate all support pairs (starting with the smallest-sized ones), and checks if a Nash equilibrium can be formed from a given support pair. In [10] it is reporter that in exhaustive computational experience on a variety of games, this algorithm outperforms the Lemke-Howson in finding one Nash equilibrium [10].

In this paper, we investigate the larger problem of defining a good heuristic for computing the *set* equilibria. Our approach in general shall be of support enumeration. Our contribution to this line of research is that we formulate the bimatrix game as a bipartite directed graph that captures the inter-dependencies of the strategies of the game. We term this graph as the *dominance graph* of the game. We then establish a correspondence between the *set* of elementary structures of the graph and the set of the equilibria of the game. In particular, we show that the set of the elementary cycles of the graph is sufficient to compute the set of equilibria. Roughly speaking, we equate a cycle with a support pair. This heuristic also allows us to *discard* certain portions of support space that will never yield a Nash equilibrium.

The motivation for this approach is that graph theory has a predictably large body of work on finding the set of the relevant structures of the graph. For example, efficient

[1] A support is a subset of strategies that the player uses with positive probability; balanced here implies that the support size of both players is equal

linear-time algorithms that find the set of elementary cycles ([3])or strongly connected components ([12], for large-sized, sparsely connected graphs have been known for quite some time.

While support enumeration is a simpler technique to implement than the Lemke-Howson (a given pair of supports can be checked in polynomial time via a linear program if it yields a Nash equilibrium or not), we do state the following caveats. First, enumerative methods of LCPs are in general faster than support enumeration ([13], [6], [1]) and they require less memory storage. Second, a potential drawback of our approach is that if the dominance graph is not sparse, the number of elementary cycles increases faster (with the game size) than support enumeration. Thirdly, no efficient algorithm is known that computes only the pair-wise distinct elementary cycles of a graph. Since two or more cycles composed of the same vertices but in different order could be elementary, there would clearly be a waste if the set of elementary cycles is computed.

The rest of the paper is organized as follows. In Section 2 we define bimatrix games, their solutions as well as the support enumeration approach. Then in Section 3 we define the formulation the game as a bipartite digraph. In Section 4 we establish some results about the correspondence between the structures of the graph and the set of equilibria. Then, in Section 5, we discuss methods of finding the set of elementary cycles. We also show how the dominance graph can be feasibly constructed using a more generalized formulation. Finally in Section 6, we summarize this work and discuss its future direction.

2 Bimatrix Game

In this section we recall standard definitions from game theory. A bimatrix game g (henceforth, game) is played by two players called player 1 and player 2 respectively, and is defined by four elements $g = (M, N, A, B)$. M and N are the strategy sets of players 1 and 2 respectively. Strategies in M and N are also called *pure* strategies. Player 1 has m pure strategies and player 2 has n pure strategies. A and B are $m \times n$ matrices and are called respectively player 1's and player 2's payoff matrix. If player 1 chooses strategy $x \in M$ and player 2 chooses $y \in N$, player 1 receives the entry A_{xy} as payoff and player 2 receives the entry B_{xy} as payoff.

Notice. Henceforth, for convenience, we shall be give definitions and notations only for player 1 that are also, by obvious analogy, applicable to player 2, unless we state to the contrary.

A *mixed* strategy p for player 1 is a m-column vector where p_x represents the probability with which player 1 chooses the strategy $x \in M$. In a mixed strategy, pure strategies that receive nonzero probability are said to be in its *support*. The support of a mixed strategy p shall be denoted by s_p. $|s_p|$ denotes the size of the set s_p. The set of mixed strategies of player 1 shall be denoted by $\Delta(M)$.

Notice. Henceforth, unless specified otherwise: x shall denote a pure strategy in M and y a pure strategy in N, A_x shall denote the xth row vector of A and B_y shall denote the yth column vector of B. p shall denote a mixed strategy from the set $\Delta(M)$ and q a mixed strategy from the set $\Delta(N)$.

2.1 Dominated Strategies

Strategy x is said to be a *best response* to the strategy y if, $\forall\ x' \in M\backslash\{x\}$, $A_{xy} \geq A_{x'y}$. x is said to be a best response to a mixed strategy $q \in \Delta(N)$, if $\forall\ x' \in M\backslash\{x\}$, $A_x \cdot q \geq A_{x'} \cdot q$. Given a mixed strategy q, the set of pure strategies that are a best response to q is denoted by $BR(q)$. Finally, the mixed strategy p is said to a best response to the mixed strategy q, if $\forall\ r \in \Delta(M)\backslash\{p\}, p^T \cdot A \cdot q \geq r^T \cdot A \cdot q$. Given two strategies, a and b, pure or mixed, we shall use $a \rightarrow b$, to mean "a is a best response to b".

A pure strategy that is not a best response to any pure or mixed strategy is called *strictly dominated*. Formally, x is strictly dominated, if $\forall\ q \in \Delta(N)$, $\exists\ x' \in M\backslash\{x\}$, such that $A_{x'} \cdot q > A_x \cdot q$. The set of strategies of a player can be made smaller by removing from it all strictly dominated strategies. The following linear program **LP1** checks if the pure strategy x is strictly dominated or not.

LP1:
variables: $\forall y \in N$, q_y; ϵ
maximize: ϵ
subject to:
1. $\sum_{y \in N} A_{x'y} q_y + \epsilon \leq \sum_{y \in N} A_{xy} q_y, \forall x' \in M\backslash\{x\}$
2. $\sum_{y \in N} q_y = 1$
3. $q_y \geq 0, \forall y \in N$

If **LP1** has a feasible solution and $\epsilon < 0$, then x is strictly dominated. In other words, if x is not strictly dominated, there exists a mixed strategy q (obtained from the q values in **LP1**) such that $x \rightarrow q$. For strictly dominated strategies no such q exists. Henceforth, we assume that from the sets M and N strictly dominated strategies have been removed by the path-independent process of the iterated elimination of strictly dominated strategies.

2.2 Nash Equilibrium

We now recall two equivalent definitions of Nash equilibrium [8], the central solution concept in game theory.

Definition 1 *The mixed strategy pair (p, q) is a Nash equilibrium, if for every mixed strategy $p' \neq p$ of player 1 and*

every mixed strategy $q' \neq q$ of player 2,

$$p^T \cdot A \cdot q \geq p'^T \cdot A \cdot q$$
$$p^T \cdot B \cdot p \geq p^T \cdot B \cdot q'$$

Thus, if player 2 is playing q, player 1 cannot improve his payoff by playing a mixed strategy different than p. This is analogously true for player 2 as well. The following theorem ([8]) leads to an equivalent definition of Nash equilibrium.

Theorem 1 *The mixed strategy pair (p, q) is a Nash equilibrium, iff $\forall\, x \in s_p$, $x \rightarrow q$ and $\forall\, y \in s_q$, $y \rightarrow p$.*

Thus, in the Nash equilibrium (p, q), the expected payoff to player 1 on playing any pure strategy $x \in s_p$ when player 2 chooses his pure strategies according to q is the same, and the expected payoff to player 1 may be lesser if he uses a pure strategy that lies outside s_p. This is analogously true for player 2 as well. This implies that strictly dominated strategies cannot be used in the support of any Nash equilibrium. A Nash equilibrium can be defined in these terms.

Definition 2 *The mixed strategy pair (p, q) with supports s_p and s_q respectively is a Nash equilibrium if:*

$$A_x^T \cdot q = u_1, \;\; \forall x \in s_p$$
$$A_{x'}^T \cdot q \leq u_1, \;\; \forall x' \in M \backslash s_p$$
$$p^T \cdot B_y = u_2, \;\; \forall y \in s_q$$
$$p^T \cdot B_{y'} \leq u_2, \;\; \forall y' \in N \backslash s_q$$

u_1 and u_2 are the expected payoffs for player 1 and player 2 respectively for playing the mixed strategies (p, q).

2.3 Non-Degenerate Games

A game is said to be *non-degenerate* if for every mixed strategy, the number of pure strategies that are a best response to it is less than or equal to the size of its support. Formally,

Definition 3 *Let $g = (M, N, A, B)$ be a bimatrix game. If for every mixed strategy $p \in \Delta(M)$ and for every mixed strategy $q \in \Delta(N)$, $|BR(p)| \leq |s_p|$ and $|BR(q)| \leq |s_q|$, then g is said to be a non-degenerate game.*

A straightforward corollary due to Definition 3 is as follows.

Corollary 1 *If (p, q) is a Nash equilibrium of a non-degenerate game g, then $|s_p| = |s_q|$ (the supports are said to be balanced).*

2.4 Computing Equilibria using Support Enumeration

Since the set of mixed strategies of each player is an infinite one, the set of Nash equilibria may be an infinite one. For example, in a two-strategy per player game, where the payoff matrices are both the identity matrix, every mixed strategy of one player forms a Nash equilibrium with every mixed strategy of the other player (although the payoffs will be same for every pair). We are therefore interested in determining a subset Ω of Nash equilibria that we define as follows. Let $P(M)$ and $P(N)$ denote the power sets of M and N respectively.

Definition 4 *Let $g = (M, N, A, B)$ be a game. Then, $\Omega = \{s : s \in P(M) \times P(N), \exists\, (p, q)$ such that $s_p \cup s_q = s$ and (p, q) is a Nash equilibrium $\}$.*

We are thus interested in determining all the support set pairs that form a Nash equilibrium. Definition 2 can be directly converted into what [10] call a *feasibility program*, which is a linear program that accepts as arguments two support sets, $s_p \subseteq M$ and $s_q \subseteq N$, and checks if they constitute a Nash equilibrium or not. Since the game involves only two players, the constraints in Definition 2 are all linear. We denote the linear program corresponding to Definition 2 by **FP1**. A simple algorithm to compute Ω is to run **FP1** for every pair (s_p, s_q), where $s_p \in P(M)$ and $s_q \in P(N)$. There are $(2^m - 1)(2^n - 1)$ elements in $P(M) \times P(N)$ and hence this algorithm becomes intractable as m or n grows.

3 Dominance Graph of the Game

We now develop our idea of deriving a graph from the game g that in such a way that the graph's relevant structural properties (in particular, the number of its elementary cycles) serve as a good heuristic to compute Ω. We call this graph the game's *dominance graph*. Each pure strategy is a vertex in this graph. The graph's adjacency matrix is based on two kinds of sets that we call the *domain* and the *relevancy set* respectively of each pure strategy. We describe these concepts and the construction of the graph in this section. The central idea of the construction is that the elements of these two sets represent the vertices of the graph.

3.1 Domain $D(x)$

Theorem 1 says that in a Nash equilibrium mixed strategy pair, each pure strategy of a player is a best response to the other player's mixed strategy. Therefore a starting point to compute Ω, is to compute for each pure strategy of a player, the set of the mixed strategies to which the pure strategy is a

best response. We define the *domain* of x, denoted by $D(x)$, to be the set of subsets of N, such that from every element in $D(x)$, a mixed strategy can be formulated to which x is a best response.

Definition 5 *The domain of strategy x is the set $D(x) = \{s : s \subseteq N, \exists\, q$ such that $s_q = s$ and $x \rightarrow q\}$*

A strategy x that is not strictly dominated may have upto $2^n - 1$ elements (subsets of N) in its domain. The domain of a strategy x can be computed by enumerating the elements of the power set of N and checking them individually to see if they belong to $D(x)$. We can check via linear programming if, given $s_1, s_0 \subseteq N$, there exists a mixed strategy q such that $x \rightarrow q$ and $\forall\, j \in s_1$, $q_j > 0$ and $\forall\, k \in s_0$, $q_k = 0$. The linear program **LP2** corresponds to this check. It takes as arguments the sets s_1, s_0.

LP2(s_1, s_0):
variables: $\forall y \in N$, q_y; ϵ
maximize: ϵ
subject to:
1. $\sum_{y \in N} A_{x'y} q_y \leq \sum_{y \in N} A_{xy} q_y, \forall x' \in M \backslash \{x\}$
2. $\sum_{y \in N} q_y = 1$
3. $q_j \geq \epsilon, \forall j \in s_1$
4. $q_k = 0, \forall k \in s_0$

Only if **LP2** has a feasible solution and $\epsilon > 0$, is it true that there exists a mixed strategy q such that $x \rightarrow q$, and $\forall\, j \in s_1$, $q_j > 0$ and $\forall\, k \in s_0$, $q_k = 0$. This mixed strategy is obtained from the values of the variables q_y (which are the probabilities of pure strategies y). Strictly dominated strategies have empty domains.
$D(x)$ can be computed by enumerating the elements of $P(N)$, the power set of N, and executing **LP2(s_1, s_0)** for each element $s \in P(N)$ setting $s_1 = s$ and $s_0 = (N - s)$. The computation of $D(x)$ of each $x \in M$ requires $(2^n - 1)$ runs of **LP2**. In non-degenerate games, the number **LP2** runs can be reduced somewhat by keeping track of (x, q) pairs where $s_q \in D(x)$. For example, if $|s_q|$ = 2, and we have determined that $s_q \in D(x)$, then we do not need to run **LP2**$(s_q, N \backslash s_q)$ after having found another x' with $s_q \in D(x')$.

3.2 Relevancy Set $R(x)$

The computation of $D(x)$ becomes intractable for large values of n. Besides, the *size* of $D(x)$ might be too large. Therefore to use $D(x)$ of each x to obtain the adjacency matrix of the graph is impractical. In Section 5.1 we discuss how a subset of $D(x)$ containing supports of small sizes only can be used to construct the graph. The measure that we shall therefore use to construct the graph is what we term as the *relevancy set* of x, denoted by $R(x)$. It is the set of the pure strategies of player 2 such that every strategy in $R(x)$ is in the support of *some* mixed strategy to which x is a best response.

Definition 6 *The relevancy set of strategy x is the set $R(x) = \{y : y \in N, \exists\, q$ such that $q_y > 0$ and $x \rightarrow q\}$*

Thus, the relevancy set of x is just the union of the elements of $D(x)$. The worst-case (as well as best-case) complexity of computing $D(x)$ is exponential. While computing $R(x)$ has this same worst-case complexity, its best-case complexity is much lower. We describe one method of computing $R(x)$ that works quite well in practice.
By the definition of the relevancy set, $R(x)$ is non-empty iff x is non-strictly dominated. We now show how to determine if a strategy $y^* \in N$ is an element of $R(x)$. In **LP1**, we make two modifications: we change constraint **3.**, to $q_y \geq 0, \forall y \in N \backslash \{y^*\}$, and we add the constraint, q_{y^*} = 0. We call this LP, **Mod LP1**.
If x is non-strictly dominated then $\epsilon \geq 0$ in **LP1**. Therefore, if in **Mod LP1** (which is the same as **LP1** but with $q_{y^*} = 0$), $\epsilon < 0$, then it implies that x is now strictly dominated only because player 2 plays y^* with zero probability. In other words, only by re-setting $q_{y^*} \geq 0$, we establish that there exists a mixed strategy q of player 2 in which he plays q_{y^*} with non-zero probability, to which x is a best response. Thus, we conclude that if $\epsilon < 0$ in **Mod LP1**, then $y^* \in R(x)$.
However, if $\epsilon \geq 0$, it does not rule out that y^* could still be in $R(x)$. By executing **Mod LP1** n times, each time taking a different strategy from N, we might obtain only a subset of $R(x)$. Assume that this set is $R'(x) \subset N$. To determine $R(x)$ given $R'(x)$, we need to execute **LP2** for either all the pairs (s_1, s_0) (where s_1 takes on values from the power set of $N - R'(x)$ and s_0, from the power set of $R'(x)$) or until $R(x) = N$. [2]
In view of the worst-case complexity of computing $R(x)$, for the purpose of the present discussion we make the following simplifying assumption.

Assumption 1 *$\forall\, x \in M$, $R(x) = N$, and $\forall\, y \in N$, $R(y) = M$.*

This assumption does not affect any of the theoretical results we give, but only affects our ability to construct the graph, an issue we address in Section 5.1. As a matter of fact, in practice, we have found that executing **Mod LP1** on randomly generated bimatrix games, n times for each x, almost always gives us $R(x) = N$. Our use $of R(x)$ to describe a graph in the following is purely for expository reasons. Theorem 1 and Definition 6 give us the following simple corollary.

[2] Other methods too exist for this purpose. For example, if x is not strictly dominated, **LP1** will return a set of pure strategies of player 2 that are in $R(x)$. Therefore, only those not found by **LP1** need to be iteratively checked by **LP2**.

Corollary 2 *If (p, q) is a Nash equilibrium of a game g, then $s_q \subseteq R(x)$ for every $x \in s_p$ and $s_p \subseteq R(y)$ for every $y \in s_q$.*

3.3 Dominance Graph based on $R(x)$

The relevancy sets $R(x)$ of each $x \in M$, and $R(y)$ of each $y \in N$ as well as the sets M and N lend a certain structure to the game that can be formulated as a *bipartite directed graph* or digraph. A bipartite digraph is a tuple $G = (U, W, E)$, where U, W are finite disjoint sets, and E is a $|U| \times |W|$ matrix called the *adjacency matrix* or the *arc set* such that $E_{uw} = 1$ if an arc exists from $u \in U$ to $w \in W$, and 0 otherwise. We say that the arc $(u, w) \in E$, if $E_{uw} = 1$. The *vertex set* of G is the union of U and W.
By setting $U = M$ and $W = N$, and defining E such that $E_{xy} = 1$ iff $y \in R(x)$ and 0 otherwise, we obtain the dominance graph of the game g, the bipartite digraph $G_r = (M, N, E)$. The vertex set of G_r is denoted by $V = M \cup N$. Thus for every arc in E, one endpoint lies in M and the other in N or vice-versa. G_r is a 1-graph, hence between every ordered pair of vertices, not more than one arc exists.
A bipartite digraph $G = (U, W, E)$ is said to be *completely connected* if for every vertex $u \in U$ and every vertex $w \in W$, the arcs (u, w) and (w, u) exist in E. Note that if Assumption 1 is made, G_r is completely connected. In a digraph, the *out-degree* of a vertex v, denoted by $O(v)$ is the number of arcs emanating from the vertex, while the *in-degree* of v, denoted by $I(v)$ is the number of arcs entering it. Note that in G_r, for each $v \in V$, $O(v) \geq 1$, since the relevancy set of each v is non-empty.

3.4 Support Cycle Basis

Some basic structural definitions from graph theory about digraphs that we require to represent the equilibria of the game g in terms of the digraph G_r are as follows:
We are given the digraph G_r as defined above. A *path* is a sequence of vertices $(v_1 \ldots, v_k)$ such that $\forall\ 1 \leq i < k$, the arc (v_i, v_{i+1}) exists in E. The first vertex v_1 in the sequence is called the initial endpoint and the last vertex v_k is called the terminal endpoint. A *cycle* [3] is a path whose initial endpoint is the same as its terminal endpoint. An *elementary* (or simple) cycle is a cycle in which no vertex (barring the initial endpoint) occurs twice. Note that a cycle is a sequence of pure strategies where each strategy is alternatively picked from the two strategy sets. The *length* of a cycle is the number of vertices in it (not counting the repeating vertex). A cycle of length k is called a k-cycle. The longest cycle in G_r has $2K + 1$ vertices where K is the size of the smaller of the two strategy sets, M and N. We denote the set of elementary cycles of G_r by C_{G_r}.
Henceforth, for convenience, we shall refer to elementary cycles as cycles unless we state to the contrary. The set of vertices that appear in a cycle μ is denoted by $V(\mu)$ and is called its vertex set. For $i \in \{1, 2\}$, the set of vertices (pure strategies) of player i in cycle μ is denoted by $V_i(\mu)$. Two cycles μ, μ' are called *equivalent* if $V(\mu) = V(\mu')$, else they are said to be *distinct*. The set C_{G_r} of the elementary cycles of G_r can be partitioned into *equivalence* classes, $V^1, \ldots, V^J$ such that any two cycles from the same class are equivalent and any two cycles from different classes are distinct. A class is represented by the vertex set of the cycles that belong to that class. The cycles of a given class are permutations of the vertices of that class.
To eventually be able to compute Ω, it is enough to know how many equivalence classes there are in G and the definition (i.e., vertex set) of each class. It is not necessary to compute the members of each class. By drawing one member (any member) from each equivalence class, we obtain a set of pairwise distinct cycles. We define the *support cycle basis* of G_r denoted by δ_{G_r} as follows.

Definition 7 *Let $P(V)$ denote the power set of V. The support cycle basis of G_r is the set $\delta_{G_r} = \{s : s \in P(V), \exists\ \mu \in C_{G_r}$, such that $V(\mu) = s\}$.*

Thus each element of a support cycle basis (henceforth called the cycle basis) [4] is a *subset* of V. There exists atleast one cycle whose vertex set equals this subset. Naturally, it may be possible that other cycles also exist whose vertex set equals this subset. In the forthcoming discussion we describe a property of cycles that is such that if it is applicable to one cycle, then it is also applicable to every cycle in that cycle's class. Thus we can refer without ambiguity to an element of δ_{G_r} as a cycle as well a subset. As we shall see in the next section, the cycle basis is important in formulating Nash equilibria in graph-theoretic terms (note the certain similitude between Definitions 4 and 7).

4 Expressing Equilibria as Cycles

We now discuss the motivation behind the preceding constructions, that of the dominance graph and the cycle basis. Our objective in using these two constructs is that they might provide heuristics that enable the computation of the equilibria set Ω more efficiently than comprehensive support pair enumeration. We first show how Nash equilibria are related to elementary cycles.
We say that a mixed strategy (p, q) *generates* a given cycle μ if $V(\mu) \subseteq (s_p \cup s_q)$. We say that a cycle μ generates a given mixed strategy (p, q) if $s_p = V_1(\mu)$ and $s_q = V_2(\mu)$.

[3] sometimes also called a circuit, esp. in undirected graphs

[4] Not to be confused with the *cycle basis* of a graph which is a set of fundamental cycles of the graph

Theorem 2 *Let g be a game and G_r its dominance graph. Then, for every Nash equilibrium (p, q) of g, there exists atleast one cycle $\mu \in \delta_{G_r}$ of length $2K + 1$ where $K = \min(|s_p|, |s_q|)$, such that $V(\mu) \subseteq (s_p \cup s_q)$. Moreover, if $|s_p| = |s_q|$, then $V(\mu) = (s_p \cup s_q)$.*

Proof: Assume (p, q) is a Nash equilibrium of g. Denote s_p by s_1 and s_q by s_2. $\forall\, i \in \{1, 2\}$, let L_i be a stack in which the elements of s_i have been pushed in any order. Let K denote the size of the smaller of the two stacks and k the subscript of that stack. Let μ be a list. The jth element of μ is denoted by μ_j. Now, remove K elements from L_k and L_{-k} [5] each, and place them *alternatively* in μ with an element of L_k being μ_1. By Corollary 2, $\forall\, i \in \{1, 2\}$, $\forall\, u \in L_i$, $s_{-i} \subseteq R(u)$ and thus, $\forall\, v \in s_{-i}$, $(u, v) \in E$. Therefore, for every $1 \leq j < 2K$, the arc (μ_j, μ_{j+1}) is an element of E. Since, $\mu_1 \in L_k$ and there are $2K$ elements in μ, $\mu_{2K} \in L_{-k}$. But the arc (μ_{2K}, μ_1) exists in E (by Corollary 2). Therefore, adding the element μ_1 at position $2K+1$, gives us an elementary cycle, as claimed in the first statement.
The vertex set of μ is $L_k \cup L_{-k}(K)$, where $L_{-k}(K) \subseteq L_{-k}$ such that $|L_{-k}(K)| = K$. If (as in a non-degenerate game) $|L_k| = |L_{-k}| = K$, the elementary cycle μ, as constructed above, has a vertex set that equals $L_k \cup L_{-k}(K) = L_k \cup L_{-k}$ as claimed in the second statement. QED.
Since the order in which the vertices are put in the stacks does not matter in the proof of Theorem 2, it follows that (p, q) generates every cycle of the class to which μ belongs. This allows us, as stated before, to refer to an element of δ_{G_r} as a cycle as well as a subset. We can also refer to the cycle generated by (p, q).

4.1 Support Trees

Theorem 2 implies that every Nash equilibrium of a game generates a cycle. If the game is non-degenerate or if the supports of the equilibrium are balanced, then the cycle also generates the Nash equilibrium. In particular, *every 3-cycle of G_r generates a Nash equilibrium of g*. We cannot generalize this statement, however. That is, not every Nash equilibrium of g can be generated by a cycle of size ≥ 5.

Corollary 3 *Given a game g and its dominance graph G_r, and a Nash equilibrium (p, q) of g that generates the cycle $\mu \in \delta_{G_r}$ of length ≥ 5. Then it is possible that μ does not generate (p, q), that is, it is possible that $s_p \neq V_1(\mu)$ or $s_q \neq V_2(\mu)$.*

Due to this corollary, it would appear that computing the cycle basis may not be sufficient to compute the set of Nash equilibria Ω. However, as we describe in the following, while a cycle itself may not generate a particular equilibrium, a cycle and an auxiliary set of 3-cycles would generate that equilibrium. Note that every 3-cycle is necessarily elementary.
Consider a mixed strategy (p, q) that is a Nash equilibrium. By Theorem 2 it generates a cycle. Let this cycle be $\mu \in \delta_{G_r}$. If $|s_p| = |s_q| = K$, then the cycle generates the equilibrium as well. So, the case that requires generalization is if $|s_p| \neq |s_q|$. Let $|s_p| < |s_q|$, and let $s = s_q - V_2(\mu)$ (s contains the pure strategies in s_q that have not been "used up" in μ). By Corollary 2, for each $v \in s$, and for each $u \in V_1(\mu)$, the arcs (u, v) and (v, u) exist in E. Thus, each element $v \in s$ forms the cycle (v, u, v) with atleast one vertex of $u \in V_1(\mu)$.
Therefore, every Nash equilibrium (p, q) is such that the union of its support sets equals the union of the vertex sets of a set of cycles $\tau(p, q)$ where each cycle is from δ_{G_r}. In this set, there is a cycle μ of length $2K + 1$ and some other 3-cycles, whose vertex sets have one element in common with the vertex set of μ. We call this set of cycles a *support tree* (henceforth, tree) [6]. We say that the mixed strategy (p, q) generates the tree $\tau(p, q)$ if the latter is obtained is the manner just described. Therefore if (p, q) generates $\tau(p, q)$, $(s_a \cup s_q) = \bigcup_{c \in \tau(p,q)} V(c)$. Moreover, as in the case of cycles and balanced supports, here $\tau(p, q)$ generates (p, q) as well. We can thus find a tree of G_r that generates a given Nash equilibrium (In the case of an equilibrium with balanced supports, there are no 3-cycles in τ).

Theorem 3 *Let g be a game and G_r its dominance graph. Every Nash equilibrium (p, q) of g generates atleast one support tree $\tau(p, q)$ of G_r.*

An important consequence of the two preceding theorems is that, we can use them for deciding if a certain strategy is eliminable i.e., it does not occur in the support of any Nash equilibrium. More generally, we can use the theorems to discard a subset of strategies, if we find that they do not yield any cycle.

5 Computing the Support Cycle Basis

The two theorems of the last section establish that the set of Nash equilibria Ω can be computed from the cycle basis only. So a general scheme to compute Ω that we call *support tree enumeration* as follows. We first determine the cycle basis δ_{G_r} from C_{G_r} which also gives us all 3-cycles. Denote the set of 3-cycles by $\delta^3_{G_r}$, and by P^3 its power set. The set of support trees is obtained by keeping those elements of $(\delta_{G_r} - \delta^3_{G_r}) \times P^3$ that satisfy the definition of a support tree. Finally, for each cycle or support tree μ found,

[5] $-k$ denotes "not k"

[6] A structure such as $\tau(p, q)$ is a *tree* of the underlying undirected graph of G_r

we run **FP1** with arguments $V_1(\mu)$ and $V_2(\mu)$.
The cycle basis is just a set of cycles of G_r. The problem of determining the set of elementary cycles of a directed graph is a well studied one in graph theory. To our knowledge, the algorithm due to Johnson [3] is the most efficient in this regard. Its run-time is bounded by $O((v+e)(c+1))$, where v is the number of vertices, e the number of arcs and c the number of elementary cycles of the graph. It computes the set C_{G_r}. We do not know of any algorithm that computes efficiently the subset δ_{G_r} of C_{G_r}.
Johnson's algorithm detects the strongly connected components (SCCs) of a digraph G and then finds all the elementary cycle of each SCC. An SCC is a subset V' of the vertices of G such that for every pair of vertices $u, v, \in V'$ there exists an elementary path of vertices of V' such that its initial endpoint is u and terminal endpoint is v. There exist efficient, linear-time algorithms that find all the SCCs of digraph. The efficiency of Johnson's algorithm depends on the density of the matrix E (and on the number of SCCs; the more SCCs, the better it is).
A completely connected bipartite graph has just one SCC and has the maximum number of elementary cycles that a graph of its size (in the number of vertices, say k) can have. This number ([3]), grows, faster than the number of total supports of the game as k grows. Therefore, constructing G_r by making Assumption 1 and then enumerating its cycles using Johnson's algorithm (or any other) is guaranteed to be worse than enumerating the elements of $P(M) \times P(N)$. On the other hand, as noted before, $R(x)$ is difficult to compute as well.
So, we would like to construct a graph without the (forced) complete connectedness of G_r but without actually computing R_x. Moreover, we would like Theorems 2 and 3 to be true for this graph as well. We now describe how a graph that is based on $D(x)$ can satisfy these criteria.

5.1 Dominance Graph based on $D(x)$

We define a dominance graph based on the domain, denoted by G_d as the bipartite digraph $G_d = (P(M), P(N), E)$. In this graph a vertex is an element of the power set of the set of pure strategies and thus corresponds to either a pure strategy or to a set of pure strategies. In G_d an arc is made from a vertex s to the vertex t, if every pure strategy in s is a best response to some mixed strategy with support t. Thus, given $s \in P(M)$ and $t \in P(N)$, $E_{st} = 1$, iff, $\forall$ $x \in s$, $t \in D(x)$. Using Theorem 1, it can be verified that all 3-cycles of G_d are Nash equilibria, just as the 3-cycles of G_r are. Additionally, in G_d only 3-cycles generate Nash equilibria. Cycles of longer lengths need not be considered. It is easy to see the motivation behind the construction of G_d. Even if the sets $R(x)$ are considered as given, G_r contains a lot of superfluous information in terms of the arcs it contains. For example, let x be a vertex in G_r, and let the arcs (x, y) and (x, w) exist in E of G_r. Suppose that x is not a best response to y, x is not a best response to w but $(y, w) \in D(x)$. This implies, that x is a best response to a mixed strategy q that has in its support y, if and only if w (or some other pure strategies) also occur in the support. A graph such as G_d contains more precise information.
Thus on the one hand, we have the small but quasi-completely connected graph G_r, and on the other, the very large, but possibly sparsely connected graph G_d. We can therefore seek to construct a dominance graph that is an *intermediate* between G_r and G_d. The intermediacy is in the size of the two vertex sets of G_i. In G_d, they are $P(M)$ and $P(N)$. In general, they can be any subsets of $P(M)$ and $P(N)$. We define an intermediate graph denoted by G_i as $G_i = (P_k(M), P_l(N), E)$ where $P_k(M) \in P(M)$ consists only of elements of $P(M)$ of size k or less and $P_l(N) \in P(N)$ consists only of elements of $P(N)$ of size l or less. The definition of E requires some care. Given an $s \in P_k(M)$ we define all the outgoing arcs from s as follows. (by analogy, the following discussion is also applicable for every $t \in P_l(N)$). Let $D_l(x) \subseteq D(x)$ denote the subset of the domain of x such that its elements are of size l or less. For example, the $D_2(x)$ can contain only pure strategies and pairs of pure strategies from N. Let $S(v)$ denote the set of pure strategies in the vertex v and $L(v)$ the size of $S(v)$. Then,

1. Let $T(s) = \{t \in P_l(N) : \forall\, x \in s, t \in D_l(x)\}$. Then, $\forall\, t \in T(s)$, $E_{st} = 1$
2. If $T(s)$ is empty, let $D(s) = \{t \in P_l(N) : S(t) = \bigcap_{x \in s} D_l(x)\}$. Then, $\forall\, t$ such that $t = \arg\max_{L(w)}\{w \in D(s)\}$, $E_{st} = 1$.
3. If $T(s)$ and $D(s)$ are both empty, then $\forall\, t \in (\bigcup_{x \in s} D_l(x))$, $E_{st} = 1$.
4. If $s = x$ (i.e., it is a pure strategy) and $D_l(x)$ is empty, then $\forall\, t \in P_l(N)$, $E_{st} = 1$.

In G_d only Case 1 is needed. In Cases 2, 3 and 4, we are creating *artificial arcs*. These are needed, since in G_i, we disallow vertices $v \in P_l(N)$ such that $L(v) > l$. Thus, it is possible that for a given s, there is no outgoing arc (the vertex is isolated) using just Case 1. This would happen either because $D_l(x)$ is empty for some $x \in s$ or that none of x in s have a common element in their domains D_l. Hence we need Cases 2, 3 and 4. If the values of l is small, the computation of $D_l(x)$ is tractable. The intuition behind the definition of E above, is that in most games even for small values of l, Cases 2, 3 and 4 are not needed, and hence artificial arcs (that introduce artificial cycles into G_i) need not be made. We define a support tree τ of G_i to be a set of cycles

Table 1. Average size of δ_{G_r} in random bimatrix games of sizes 7 to 11. $S = P(M) \times P(N)$

$m = n =$	7	8	9	10	11
$\lvert\delta_{G_r}\rvert$	757	3775	11772	48768	252567
$\lvert\delta_{G_r}\rvert/\lvert S\rvert$	0.04	0.058	0.045	0.046	0.06
$T(secs)$	< 1	< 1	5.3	9.8	67

such that each pair of cycles in it has one vertex in common. The definition of G_i leads to the following theorem.

Theorem 4 *Given an intermediate dominance graph $G_i = (P_k(M), P_l(N), E)$ of a game g, every Nash equilibrium (p, q) of g generates atleast one support tree τ of G_i and τ generates (p, q) as well.*

We summarize a general scheme to compute the set Ω as follows. For players 1 and 2, we set the values k and l respectively, to fix the sizes of $P_k(M)$ and $P_l(N)$. Then, using **LP2**, we compute $\forall\, x \in M, \forall\, 1 \leq i \leq k$, $D_i(x)$, and $\forall\, y \in N, \forall\, 1 \leq i \leq l$, $D_i(y)$. We then fill the entries of the matrix E as described in Section 5.1 to obtain the graph G_i. We then use an elementary cycle-finding algorithm such as [3] to get the cycle basis of G_i. We then find sets of cycles from δ_{G_i} that contain pairwise intersecting cycles. Then the program **FP1** is run for each such set to obtain Ω.

5.2 Results

Table 1 shows some preliminary results (the games were generated by the GAMUT software [9]) about the sizes of the cycle basis in G_r. The relevancy sets were obtained through **Mod LP2**, and in fact in all cases, the relevancy sets equaled the other player's strategy set. We did not use Johnson's algorithm since it does not directly compute the cycle basis. For our purpose, we have conceived a simple enumerating algorithm that builds elementary cycles of length $k + 1$ from those of length k. The different cycle lengths to be considered are $k = 3, 4, \ldots (2K + 1)$. Each elementary k-cycle of length k is stored in a vector. Before storing, the vertices are sorted, and converted into a number using a coding scheme. In the first step, all 3-cycles are computed by a simple search. When searching for a $(k + 1)$-cycle from a k-cycle, an expansion is done (using fixed look-ahead) only if the resulting cycle is not already present in the set of $(k + 1)$-cycles. We also conducted several experiments to generate statistics about the domain set for a variety of 10×10 games using GAMUT. We do not report our findings here for want of space, but we do mention that (predictably) for random games, the adjacency matrix of G_d is very dense (about 65% of entries are 1). Games with sparse matrices were "WarOfAttrition"(15%), "LocationGame"(10%) and "GuessTwoThirdsAve"(15%) among others.

6 Conclusion and Future Work

We have presented a heuristic for the computation of the set of equilibria of bimatrix games as well as for identifying eliminable strategies (those that are not in any Nash equilibrium). We have formulated the heuristic in graph-theoretic terms with the idea that certain games can be converted to sparsely connected digraphs, which can then be mined for interesting structures. In this paper, we showed that we can re-design a game to be a digraph whose elementary cycles can be checked directly to see if they yield Nash equilibria. The bulk of the paper concerned graphs conceived with the relevancy set. As we stated, G_r was used mainly for expository purposes. Our immediate work concerns more focused computational experience with intermediate graphs G_i. At the present time, there are not many approaches in the literature for computing the set of Nash equilibria, and we hope that our approach is a useful contribution.

Acknowledgements. We are thankful to Martin Allen and Vishesh Vikas for helpful discussions.

References

[1] C. Audet, P. Hansen, B. Jaumard, and G. Savard. Enumeration of all extreme equilibria of bimatrix games. *SIAM Journal on Scientific Computing*, 23:323–338, 2001.

[2] J. Dickhaut and T. Kaplan. A program for finding nash equilibria. *The Mathematica Journal*, pages 87–93, 1991.

[3] D. B. Johnson. Finding all the elementary circuits of a directed graph. *SIAM Journal of Computing*, 4(1):77–84, 1975.

[4] C. E. Lemke. Bimatrix equilibrium points and mathematical programming. *Management Science*, pages 681–689, 1965.

[5] C. E. Lemke and J. Howson. Equilibrium points of bimatrix games. *Journal of the Society for Industrial and Applied Mathematics*, pages 413 – 423, 1964.

[6] R. McKelvey and A. McLennan. Computation of equilibria in finite games. *Handbook of Computational Economics*, 1996.

[7] A. McLennan and J. Berg. The asymptotic expected number of nash equilibria of two player normal form games. *Mimeo, University of Minnesota*, 2002.

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

[9] R. Porter, E. Nudelman, and Y. Shoham. Run the gamut: A comprehensive approach to evaluating game-theoretic algorithms. *Third International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS-04)*, 2004.

[10] R. Porter, E. Nudelman, and Y. Shoham. Simple search methods for finding a nash equilibrium. *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI)*, pages 664–669, 2004.

[11] R. Savani and B. von Stengel. Exponentially many steps for finding a nash equilibrium in a bimatrix game. *45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 258–267, 2004.

[12] R. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal of Computing*, 1(2):146 – 160, 1972.

[13] B. von Stengel. Computing equilibria of two-person games. *Handbook of Game Theory (R.J. Aumann and S. Hart, eds)*, 3, 2002.