Economic Theory manuscript No.
(will be inserted by the editor)

Enumeration of Nash Equilibria for Two-Player Games

David Avis · **Gabriel D. Rosenberg** ·
Rahul Savani · **Bernhard von Stengel**

Received: date / Accepted: date

Abstract This paper describes algorithms for finding all Nash equilibria of a two-player game in strategic form. We present two algorithms that extend earlier work. Our presentation is self-contained, and explains the two methods in a unified framework using faces of best-response polyhedra. The first method is based on the known vertex enumeration program *lrs*, for "lexicographic reverse search". It enumerates the vertices of only one best-response polytope, which determine a complementary face in the other polytope. The second method is a modification of the known *EEE* algorithm, for "enumeration of extreme equilibria". We also describe a second, as yet not implemented, variant that is space efficient. We discuss details of implementations of the *lrs*-based and the *EEE* algorithm, and report on computational experiments that compare the two algorithms, which show that both have their strengths and weaknesses.

Keywords Bimatrix game · Nash equilibrium · Linear programming · Complementarity

JEL Classification C72

David Avis
McGill University, School of Computer Science and GERAD, Montreal, Quebec H3A 2A7, Canada
E-mail: avis@cs.mcgill.ca

Gabriel D. Rosenberg
Yale Law School, New Haven, CT 06511, USA
E-mail: grosenberg@gmail.com

Rahul Savani (supported in part by EPSRC project EP/D067170/1)
University of Warwick, Dept. of Computer Science and DIMAP, Coventry CV4 7AL, United Kingdom
E-mail: R.S.J.Savani@warwick.ac.uk

Bernhard von Stengel
London School of Economics, Department of Mathematics, London WC2A 2AE, United Kingdom
E-mail: stengel@nash.lse.ac.uk

1 Introduction

This paper describes algorithms for finding all Nash equilibria of a two-player game. These methods apply to games in strategic form, and have the potential to be extended to other game descriptions, for example games in extensive form (discussed briefly in Sect. 9). We present two main algorithms, and some variants, that extend earlier work. For both of the algorithms we give a variant that is space efficient, requiring memory polynomial in the input size only, in order to produce a duplicate free output list. As far as we know, these are the only known algorithms with this property. Our presentation is self-contained, and explains the two methods in a unified framework based on polyhedra.

The first method is based on the vertex enumeration program *lrs* (for "lexicographic reverse search") described in Avis and Fukuda (1992) and Avis (2000). The extensions of *lrs* to finding Nash equilibria are here described for the first time in a journal article. The second method is the *EEE* algorithm (for "enumeration of extreme equilibria") by Audet, Hansen, Jaumard, and Savard (2001), implemented in exact arithmetic by Rosenberg (2005), and presented here in modified form. We also give for the first time a geometric description of the *EEE* algorithm in terms of facets of polyhedra.

A Nash equilibrium is given by a mixed strategy for each player that is a best response to the fixed strategy of the other player. According to the well-known "best response condition" (Prop. 1, due to Nash (1951)), this means that the pure strategies in the support of the mixed strategy have maximal, and hence equal, expected payoff. This defines linear equations and inequalities for the mixed strategy probabilities of the other player. These are captured by a "best response polyhedron", an approach that has already been described by Vorob'ev (1958), Kuhn (1961), and Mangasarian (1964), explained in detail in Sect. 3. An equilibrium strategy of a player is a vertex of his best response polyhedron, or a convex combination of such vertices, as characterized in Prop. 4 (Winkels (1979); Jansen (1981)). Hence, the Nash equilibria of a two-player game can be found by enumerating all pairs of vertices of the two best reponse polyhedra, and checking the equilibrium property, which gives the *extreme* equilibria of the game.

A vertex enumeration program, such as *lrs*, enumerates all vertices of a polyhedron specified by inequalities (see Sect. 6). A straightforward enumeration of extreme equilibria generates the vertices of both best response polyhedra and outputs the vertex pairs that match as equilibria, as implemented by Canty (2003) and Savani (2005).

Here we describe a different approach, the basics of which have been outlined in von Stengel (1998). This approach considers the vertices of only one best response polyhedron, say for player 1. Each such vertex x is an equilibrium strategy of player 1 if and only if the "complementary" inequalities in the other polyhedron are tight, so these equations determine a face of that polyhedron, called the "complementary face" to x. (This complementary face is empty if x is not part of an equilibrium, and a single point if the equilibrium is isolated.) The approach thus considers the vertices x of one best response polyhedron, and enumerates the vertices y of the complementary face to x in the other polyhedron, which defines all extreme equilibria (x,y) of the game. This may involve only a small number of vertices y of the second polyhedron and

thereby save computation time. One may use a preliminary run of *lrs* to choose the first polyhedron judiciously.

The programs described in this paper use exact arithmetic with integers of arbitrary precision, given integer or fractional payoffs as input. This avoids rounding errors that can occur with floating-point arithmetic, and safely finds all equilibria even when the game is degenerate. Fractional numbers in the input can be scaled to become integers. The pivoting operations in *lrs* preserve integers in the linear programming tableaus via "integer pivoting" (see Sect. 5). This known technique is superior to using fractions of integers (rational arithmetic) because their cancellation requires greatest common divisor computations which tend to take the bulk of computation time.

The *EEE* approach due to Audet et al. (2001) enumerates all equilibria by alternately solving parameterized linear programs. It explores a binary search tree where in each step, a pure strategy is selected and converted to a tight inequality in one of the two best reponse polyhedra (which defines the binary choice). This ends when all strategies have been fixed, or the corresponding face of the polyhedron is empty, detected as an infeasible linear program. Rosenberg (2005) has implemented this approach with integer pivoting instead of floating-point arithmetic as done by Audet et al. (2001). We present variants of *EEE* that give some speedup for degenerate games.

In Sect. 2, we recall the best response condition. For nondegenerate games, this gives rise to an algorithm for finding all equilibria by enumerating all possible supports. Section 3 describes the best response polyhedra that are the basis of our algorithms. Degenerate games are discussed in Sect. 4. The possibly infinite set of all equilibria in a degenerate game can be described by "maximal Nash subsets". These are polytopes obtained from the finite set of extreme equilibria. The *Clique* Algorithm 2 shows how to determine these maximal Nash subsets, as well as their nondisjoint unions that define connected components of Nash equilibria. An extreme equilibrium is a pair of vertices of the best response polyhedra. Vertices are represented algebraically by linear programming tableaus or "dictionaries". We recall these standard techniques in Sect. 5 in order to explain the details of our algorithms, as well as the less known method of "integer pivoting" which is economical for keeping arbitrary precision. In Sect. 6, we explain our first algorithm that uses the *lrs* program for vertex enumeration. In Sect. 7, we explain the second algorithm *EEE*. We report experimental results in Sect. 8. A number of possible extensions, and open problems, are discussed in Sect. 9.

2 Bimatrix games and the best response condition

s-bimatrix

We use the following notation throughout. Let (A,B) be a bimatrix game, where A and B are $m \times n$ matrices of payoffs to the row player 1 and column player 2, respectively. Let M be the set of the m pure strategies of player 1 (the row player), and let N be the set of the n pure strategies of player 2 (the column player). It is useful to assume that these sets are disjoint, as in

$$M = \{1,\ldots,m\}, \qquad N = \{m+1,\ldots,m+n\}. \tag{1}$$

defMN

The payoff matrices A and B belong to $\mathbb{R}^{M\times N}$, so A has entries a_{ij} and B has entries b_{ij} for $i \in M$ and $j \in N$. When A and B define the input to an algorithm for finding all Nash equilibria, the payoffs are assumed to be rationals, or, by suitable scaling, integers.

A mixed strategy of player 1 is a vector x of probabilities x_i for playing rows $i \in M$, so $x \in \mathbb{R}^M$; similarly, a mixed strategy of player 2 is a probability vector $y \in \mathbb{R}^N$. All vectors are column vectors. The *support* of a mixed strategy is the set of pure strategies that have positive probability. A vector or matrix with all components zero is denoted by $\mathbf{0}$, and a vector of all ones by $\mathbf{1}$. Inequalities like $x \geq \mathbf{0}$ between two vectors hold for all components.

A *best response* to the mixed strategy y of player 2 is a mixed strategy x of player 1 that maximizes his expected payoff $x^\top Ay$. Similarly, a best response y of player 2 to x maximizes her expected payoff $x^\top By$. A *Nash equilibrium* is a pair (x,y) of mixed strategies that are best responses to each other.

The following well-known proposition states that a mixed strategy x is a best response to an opponent strategy y if and only if all pure strategies in its support are pure best responses to y. The same holds with the roles of the players exchanged.

p-bestresponse

Proposition 1 (Best response condition, Nash (1951)) *Let x and y be mixed strategies of player 1 and 2, respectively. Then x is a best response to y if and only if for all $i \in M$,*

$$x_i > 0 \implies (Ay)_i = u = \max\{(Ay)_k \mid k \in M\}, \tag{2}$$

bestresp

and y is a best response to x if and only if for all $j \in N$,

$$y_j > 0 \implies (B^\top x)_j = v = \max\{(B^\top x)_k \mid k \in N\}. \tag{3}$$

bestrespy

Proposition 1 is useful because it states a finite condition, which is easily checked, about all pure strategies of the player, rather than about the infinite set of all mixed strategies. It can also be used to find all Nash equilibria (see Algorithm 1 below), by trying out the different possible supports of mixed strategies. All pure strategies in the support must have maximum, and hence equal, expected payoff to that player. This leads to equations for the probabilities of the opponent's mixed strategy. These linear equations may not have full rank. To avoid this complication, we apply this algorithm only to *nondegenerate* games, defined as follows.

-nondegenerate

Definition 1 A two-player game is called *nondegenerate* if no mixed strategy with support of size k has more than k pure best responses.

The following observation is is immediate from Prop. 1.

Proposition 2 *In any Nash equilibrium (x,y) of a nondegenerate bimatrix game, x and y have supports of equal size.*

The following "support enumeration algorithm" has been described by Dickhaut and Kaplan (1991).

a-suppenum

Algorithm 1 (Equilibria by support enumeration) *Input:* A nondegenerate bimatrix game. *Output:* All Nash equilibria of the game. *Method:* For each $k = 1, \ldots,$ $\min\{m,n\}$ and each pair (I,J) of k-sized subsets I of M and J of N, respectively, solve the equations $\sum_{i\in I} x_i b_{ij} = v$ for $j \in J$, $\sum_{i\in I} x_i = 1$, $\sum_{j\in J} a_{ij} y_j = u$ for $i \in I$, $\sum_{j\in J} y_j = 1$, and check that $x \geq \mathbf{0}$, $y \geq \mathbf{0}$ and that (2) holds for x and (3) for y.

The linear equations considered in this algorithm may not have solutions, which means that there is no equilibrium for that support pair. Nonunique solutions occur only for degenerate games, because a linear dependency allows to reduce the support of a mixed strategy. Degenerate games are discussed in Sect. 4 below.

3 Equilibria via labeled polytopes

s-poly

In order to identify the possible supports of equilibrium strategies, one can use "best response polytopes". These express directly that best-response payoffs are not only equal to each other, but also at least as large as the expected payoffs for pure strategies that are not in the support.

We first recall some notions from the theory of (convex) polyhedra. An *affine combination* of points $z_1, \ldots, z_k$ in some Euclidean space is of the form $\sum_{i=1}^k z_i \lambda_i$ where $\lambda_1, \ldots, \lambda_k$ are reals with $\sum_{i=1}^k \lambda_i = 1$. It is called a *convex combination* if $\lambda_i \geq 0$ for all i. A set of points is *convex* if it is closed under forming convex combinations. Given points are *affinely independent* if none of these points is an affine combination of the others. A convex set has *dimension* d if and only if it has $d+1$, but no more, affinely independent points.

A *polyhedron* P in $\mathbb{R}^d$ is a set $\{z \in \mathbb{R}^d \mid Cz \leq q\}$ for some matrix C and vector q. It is called *full-dimensional* if it has dimension d. It is called a *polytope* if it is bounded. A *face* of P is a (possibly empty) set $\{z \in P \mid c^\top z = q_0\}$ for some $c \in \mathbb{R}^d$, $q_0 \in \mathbb{R}$ so that the inequality $c^\top z \leq q_0$ holds for all z in P. A *vertex* of P is the unique element of a 0-dimensional face of P. An *edge* of P is a one-dimensional face of P. A *facet* of a d-dimensional polyhedron P is a face of dimension $d-1$. It can be shown that any nonempty face F of P can be obtained by turning some of the inequalities that define P into equalities, which are then called *binding* inequalities. That is, $F = \{z \in P \mid c_i z = q_i,\ i \in I\}$, where $c_i z \leq q_i$ for $i \in I$ are some of the rows in $Cz \leq q$. A facet is characterized by a single binding inequality that is *irredundant*, that is, the inequality cannot be omitted without changing the polyhedron. A d-dimensional polyhedron P is called *simple* if no point belongs to more than d facets of P, which is true if there are no special dependencies between the facet-defining inequalities.

The *best response polyhedron* $\overline{P}$ for player 1 is the set of player 1's mixed strategies x together with the "upper envelope" of expected payoffs (and any larger payoffs) v to player 2. The best response polyhedron $\overline{Q}$ for player 2 is defined analogously:

$$\begin{aligned} \overline{P} &= \{(x,v) \in \mathbb{R}^M \times \mathbb{R} \mid x \geq \mathbf{0},\ \mathbf{1}^\top x = 1,\ B^\top x \leq \mathbf{1}v\}, \\ \overline{Q} &= \{(y,u) \in \mathbb{R}^N \times \mathbb{R} \mid Ay \leq \mathbf{1}u,\ y \geq \mathbf{0},\ \mathbf{1}^\top y = 1\}. \end{aligned} \tag{4}$$

hedra

As an example, consider the 3×2 bimatrix game (A,B) with

$$A = \begin{bmatrix} 3 & 3 \\ 2 & 5 \\ 0 & 6 \end{bmatrix}, \qquad B = \begin{bmatrix} 3 & 2 \\ 2 & 6 \\ 3 & 1 \end{bmatrix}. \tag{5}$$

example

In this example, $\overline{Q}$ is the set of triples (y_4, y_5, u) that fulfill $3y_4 + 3y_5 \le u$, $2y_4 + 5y_5 \le u$, $0y_4 + 6y_5 \le u$, $y_4 \ge 0$, $y_5 \ge 0$, and $y_4 + y_5 = 1$. The left picture in Fig. 1 shows $\overline{Q}$ for $0 \le y_4 \le 1$ which uniquely determines y_5 as $1 - y_4$. The circled numbers indicate the facets of $\overline{Q}$, which are either strategies $i \in M$ of the other player or own strategies $j \in N$. Facets 1, 2, 3 of player 1 indicate his best responses together with his expected payoff u. For example, strategy 1 is a best response when $y_4 \ge 2/3$. Facets 4 and 5 of player 2 tell when the respective own strategy has probability zero, namely $y_4 = 0$ or $y_5 = 0$.

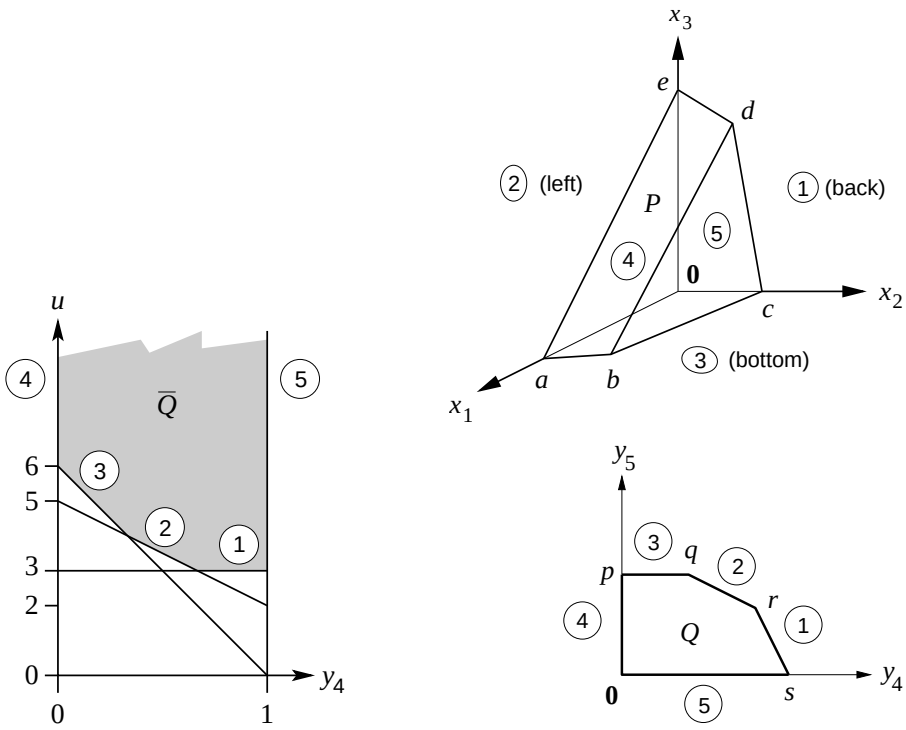

Fig. 1 Left: Best reponse polyhedron $\overline{Q}$ for (5). Bottom right: Corresponding polytope Q, which has vertices $\mathbf{0}, p, q, r, s$. Top right: Best response polytope P with vertices $\mathbf{0}, a, b, c, d, e$.

f-upper

We say a point (y, u) of $\overline{Q}$ has *label* $k \in M \cup N$ if the kth inequality in the definition of $\overline{Q}$ is binding, which for $k = i \in M$ is the ith binding inequality $\sum_{j \in N} a_{ij} y_j = u$ (meaning i is a best response to y), and for $k = j \in N$ is the binding inequality $y_j = 0$. In the example, $(y_4, y_5, u) = (2/3, 1/3, 3)$ has labels 1 and 2. The labels of a point

(x,v) of $\overline{P}$ are defined correspondingly: It has label i in M if $x_i = 0$, and label j in N if $\sum_{i\in M} b_{ij}x_i = v$.

With these labels, an equilibrium is a pair (x,y) of mixed strategies so that with the corresponding expected payoffs v and u, the pair $((x,v),(y,u))$ in $\overline{P}\times\overline{Q}$ is *completely labeled*, which means that every label $k \in M\cup N$ appears as a label of (x,v) or of (y,u). This is equivalent to the best response conditions (2) and (3), which say that in equilibrium, every pure strategy is a best response or has probability zero.

The constraints (4) defining $\overline{P}$ and $\overline{Q}$ can be simplified by eliminating the payoff variables u and v, which works if these are always positive. For that purpose, assume that

$$A \text{ and } B^\top \text{ are nonnegative and have no zero column.} \tag{6}$$ ABpos

We could simply assume $A > \mathbf{0}$ and $B > \mathbf{0}$, but it is useful to admit zero matrix entries (e.g. as in the identity matrix). Even negative entries are possible as long as the upper envelope remains positive; for example, a_{34} (currently zero) in (5) could be negative, as Fig. 1 shows.

We change $\overline{P}$ by dividing each inequality $\sum_{i\in M} b_{ij}x_i \le v$ by v, where v is positive by (6). This gives the new inequality $\sum_{i\in M} b_{ij}(x_i/v) \le 1$, where we treat x_i/v as a new variable that we call again x_i. The resulting polyhedron is P. Similarly, $\overline{Q}$ is replaced by Q by dividing each inequality in $Ay \le \mathbf{1}u$ by u. Then

$$\begin{aligned} P &= \{x \in \mathbb{R}^M \mid \quad x \ge \mathbf{0},\ B^\top x \le \mathbf{1}\},\\ Q &= \{y \in \mathbb{R}^N \mid Ay \le \mathbf{1}, \quad y \ge \mathbf{0}\}. \end{aligned} \tag{7}$$ defPQ

It is easy to see that (6) implies that P and Q are full-dimensional polytopes, unlike $\overline{P}$ and $\overline{Q}$. In effect, we have normalized the expected payoffs to be 1, and dropped the conditions $\mathbf{1}^\top x = 1$ and $\mathbf{1}^\top y = 1$. Nonzero vectors $x \in P$ and $y \in Q$ are multiplied by $v = 1/\mathbf{1}^\top x$ and $u = 1/\mathbf{1}^\top y$ to turn them into probability vectors. The scaling factors v and u are the expected payoffs to the other player.

The set $\overline{P}$ is in one-to-one correspondence with $P - \{\mathbf{0}\}$ with the map $(x,v) \mapsto x\cdot(1/v)$. Similarly, $(y,u) \mapsto y\cdot(1/u)$ defines a bijection $\overline{Q} \to Q - \{\mathbf{0}\}$. These bijections are not linear, but are known as "projective transformations" (for a visualization see von Stengel (2002, Fig. 2.5)). They preserve the face incidences since a binding inequality in $\overline{P}$ (respectively, $\overline{Q}$) corresponds to a binding inequality in P (respectively, Q) and vice versa. In particular, points have the same *labels* defined by the binding inequalities, which are some of the $m+n$ inequalities that define P and Q in (7). An equilibrium is then defined by a completely labeled pair $(x,y) \in P\times Q - \{(\mathbf{0},\mathbf{0})\}$; for brevity, we say (x,y) "is" a Nash equilibrium, with the understanding that x and y have to be rescaled to become probability vectors $x\cdot 1/\mathbf{1}^\top x$ and $y\cdot 1/\mathbf{1}^\top y$, respectively.

For the example (5), the polytopes P and Q are shown on the right in Fig. 1. Any point x in P has at most three labels, and any y in Q has at most two labels, and only vertices have that many labels. The following three completely labeled vertex pairs define the Nash equilibria of the game: The pure strategy equilibrium (a,s), and the mixed equilibria (b,r) and (d,q). For example, vertex $b = (2/7, 1/14, 0)^\top$ of P has labels 3, 4, 5, and vertex $r = (1/6, 1/9)^\top$ of Q has labels 1 and 2, so (b,r) is completely labeled. This corresponds to the mixed strategy pair $((2/3,1/3,0)^\top, (2/3,1/3)^\top)$. The vertices c and e of P, and p of Q, are not part of an equilibrium.

r-label *Remark 1* A bimatrix game (A,B) is nondegenerate if the polytopes P and Q in (7) have the property that no point in P has more than m labels, and no point in Q has more than n labels.

Proof. If $x \in P$ and x has support of size k and L is the set of labels of x, then $|L \cap M| = m-k$, so $|L| > m$ implies x has more than k best responses in $L \cap N$. □

If (A,B) is nondegenerate, P and Q are simple polytopes, because a point of P, say, that is on more than m facets would have more than m labels. Even if P and Q are simple polytopes, the game can still be degenerate if the *description* of P or Q is redundant in the sense that some inequality can be omitted, but nevertheless is sometimes binding. This occurs if a player has a pure strategy that is weakly dominated by or payoff equivalent to some other mixed strategy. Redundant inequalities of this kind, or non-simple polytopes, do not occur for generic payoffs. A strictly dominated strategy may occur generically, but it defines a redundant inequality that is never binding, so this does not lead to a degenerate game.

If the game is nondegenerate, only vertices of P can have m labels, and only vertices of Q can have n labels. Otherwise, a point of P with m labels that is not a vertex would be on a higher-dimensional face, and a vertex of that face, which is a vertex of P, would have additional labels. Consequently, only vertices of P and Q have to be inspected as possible equilibrium strategies.

4 Degenerate games

s-degen In a degenerate game, a vertex of P, for example, may have more than m labels. As an example, consider the 3×2 game

$$A = \begin{bmatrix} 3 & 3 \\ 2 & 5 \\ 0 & 6 \end{bmatrix}, \qquad B = \begin{bmatrix} 3 & 3 \\ 2 & 6 \\ 3 & 1 \end{bmatrix}, \tag{8}$$ degen

which agrees with (5) except that $b_{15} = 3$. The polytope Q for this game is the same as before, shown on the right in Fig. 2. The polytope P, shown in the left in Fig. 2, differs from P in Fig. 1 only in that vertex b has merged with a.

Degenerate games may have infinite sets of equilibria. In the example (8), vertex a of P, which represents the pure strategy $(1,0,0)^\top$ of player 1, together with the entire edge of Q that joins vertices r and s, defines a component of Nash equilibria, where player 2 plays some mixed strategy $(y_4, 1-y_4)$ for $2/3 \le y_4 \le 1$.

The following central observation characterizes all Nash equilibria of a general bimatrix game (A,B) with P and Q as defined in (7).

l-face **Proposition 3** *For $K, L \subseteq M \cup N$, let*

$$\begin{aligned} P(K) &= \{x \in P \mid \forall i \in K \cap M: \quad x_i = 0, \ \ \forall j \in K \cap N: (B^\top x)_j = 1\} \\ Q(L) &= \{y \in Q \mid \forall i \in L \cap M: (Ay)_i = 1, \ \ \forall j \in L \cap N: \quad y_j = 0\} \end{aligned} \tag{9}$$ PKQL

Then $(x,y) \in P \times Q - \{\mathbf{0},\mathbf{0}\}$ is a Nash equilibrium if and only if there are sets K and L so that $K \cup L = M \cup N$ and $(x,y) \in P(K) \times Q(L)$.

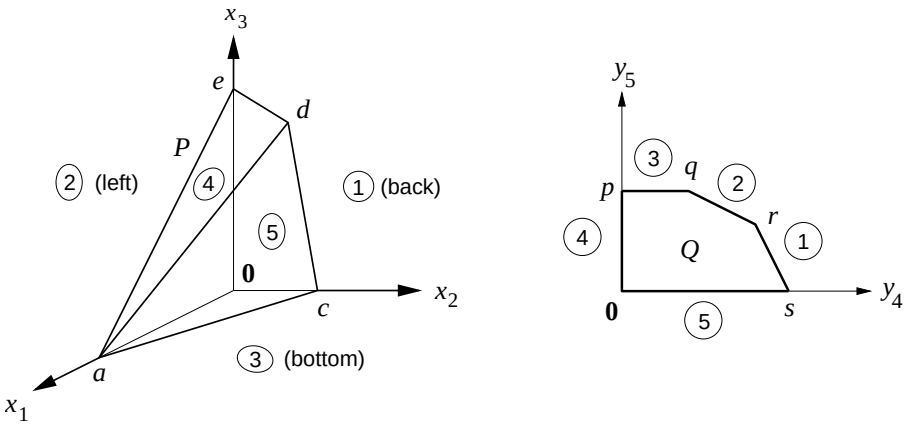

Fig. 2 Best reponse polytopes for the degenerate game (8). f-degen

Proof. Given K and L so that $K \cup L = M \cup N$, any $(x,y) \in P(K) \times Q(L)$ is by (9) completely labeled. If $x = \mathbf{0}$, then $B^\top x < \mathbf{1}$, so x has no label in N (i.e., $K \subseteq M$), which implies $N \subseteq L$ and therefore $y = \mathbf{0}$ (and thus $Ay < \mathbf{1}$ and $L = N$, $K = M$); similarly, $y = \mathbf{0}$ implies $x = \mathbf{0}$. However, the case $(x,y) = (\mathbf{0},\mathbf{0})$ is excluded, so (x,y) is a Nash equilibrium.

Conversely, given a Nash equilibrium (x,y) in $P \times Q - \{\mathbf{0},\mathbf{0}\}$, it belongs to $P(K) \times Q(L)$ where K and L are the sets of labels of x and y, respectively. □

Clearly, the set $P(K)$ in (9) is a face of P, and $Q(L)$ is a face of Q. By Prop. 3, the set of Nash equilibria is the union of products $P(K) \times Q(L)$ of faces of the polytopes P and Q. The following proposition, due to Winkels (1979) and Jansen (1981), characterizes these products in terms of pairs of vertices of P and Q. We write $\mathrm{conv}U$ for the convex hull of a set U.

p-winkels **Proposition 4** *Let (A,B) be a bimatrix game, and $(x,y) \in P \times Q$. Then (x,y) is a Nash equilibrium of (A,B) if and only if there is a set U of vertices of $P - \{\mathbf{0}\}$ and a set V of vertices of $Q - \{\mathbf{0}\}$ so that $x \in \mathrm{conv}U$ and $y \in \mathrm{conv}V$, and every $(u,v) \in U \times V$ is completely labeled.*

Proof. By Prop. 3, any Nash equilibrium (x,y) belongs to $P(K) \times Q(L)$ for suitable K,L with $K \cup L = M \cup N$. Let U and V be the sets of vertices of $P(K)$ and $Q(L)$, which are also vertices of P and Q, respectively. Then $P(K) = \mathrm{conv}U$ and $Q(L) = \mathrm{conv}V$, which shows the "only if" part.

Conversely, given vertex sets U and V so that every $(u,v) \in U \times V$ is completely labeled, let K be the set of labels common to all $u \in U$, and let L be the set of labels common to all $v \in V$. Then $K \cup L = M \cup N$, because otherwise there would be some label that was missing from some $u \in U$ and from some $v \in V$, so that (u,v) is not completely labeled, contrary to the assumption. Then $\mathrm{conv}U \subseteq P(K)$ and $\mathrm{conv}V \subseteq Q(L)$, which implies the "if" part by Prop. 3. □

Proposition 4 shows that the set of all Nash equilibria can be completely described by the (finitely many) Nash equilibria that are vertex pairs of $P \times Q$. These

are also called *extreme* equilibria in the sense that they are not convex combinations of other equilibria. For example, the two extreme equilibria (a,s) and (a,r) of the game (8) represent the component $\{a\} \times \operatorname{conv}\{r,s\}$ of equilibria mentioned above.

Consider the bipartite graph R on the vertices of $P - \{\mathbf{0}\}$ and $Q - \{\mathbf{0}\}$ whose edges are the completely labeled vertex pairs (x,y), which are the extreme equilibria of (A,B). The maximal "cliques" (maximal complete bipartite subgraphs) of R of the form $U \times V$ then define sets of Nash equilibria $\operatorname{conv} U \times \operatorname{conv} V$, as in Prop. 4, whose union is the set of all Nash equilibria. These sets are called "maximal Nash subsets" (Millham 1974). They are also the maximal sets of the form $X \times Y$ so that any two Nash equilibria (x,y) and (x',y') in $X \times Y$ are *interchangeable* in the sense that then (x',y) and (x,y') are equilibria as well, which is a property of equilibria in zero-sum games.

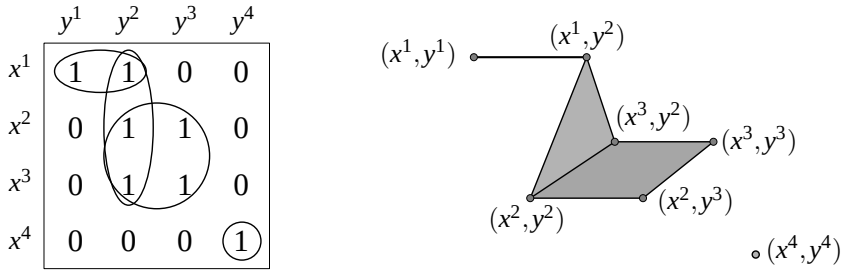

Fig. 3 Left: Incidence matrix of a bipartite graph R of extreme equilibria, with its maximal cliques. Right: Geometry of the two equilibrium components. One of them is the union of the three maximal Nash subsets $\{x^1\} \times \operatorname{conv}\{y^1,y^2\}$, $\operatorname{conv}\{x^1,x^2,x^3\} \times \{y^2\}$, and $\operatorname{conv}\{x^2,x^3\} \times \operatorname{conv}\{y^2,y^3\}$, and the other consists of a single vertex pair (x^4,y^4).

f-bipart

Maximal Nash subsets may be nondisjoint, as in the abstract example in Fig. 3, or the game in (17) below. The inclusion-maximal connected sets of Nash equilibria are usually called the (topological) equilibrium *components*. The concept of "stable" equilibria applies to such components; see Kohlberg and Mertens (1986).

The set of extreme equilibria suffices to describe all equilibrium components as well as their maximal Nash subsets, because if two Nash subsets are not disjoint, they have a common vertex pair (because by Prop. 3, both Nash subsets are products of faces of P and Q, and so is their intersection). Hence, equilibrium components are obtained as connected components of the bipartite graph R above, which are found by a straightforward graph search algorithm (e.g., Cormen et al. 2001).

a-comp

Algorithm 2 (*Clique* **– Equilibrium components)** *Input:* All pairs (x,y) of extreme equilibria. *Output:* All components of Nash equilibria, given as unions of maximal Nash subsets. *Method:* Consider the set of extreme equilibria as a bipartite graph R. Each connected component C of R defines an equilibrium component; enumerate the maximal cliques of C, which define the maximal Nash subsets.

All maximal complete bipartite subgraphs of R can be found by a variant of the elegant clique enumeration algorithm by Bron and Kerbosch (1973). An implementation of the *Clique* Algorithm 2 by von Stengel (1998) is used in the computer systems of McKelvey, McLennan and Turocy (2007), Canty (2003), and Savani (2005).

In the rest of paper, we are concerned with algorithms for finding all extreme equilibria, which define the input for the *Clique* Algorithm 2.

5 Vertices and pivoting

s-intpiv

We consider algorithms for enumerating the extreme equilibria of a bimatrix game. These are vertex pairs of polyhedra derived from the payoff matrices. The algorithms use standard techniques for representing polyhedra as they are known from linear programming. For easy reference and in order to explain the details of our algorithms, we summarize these methods in this section.

The inequalities defining a polyhedron are converted to equations with the help of nonnegative *slack variables*, and vertices are represented as *basic feasible solutions* to these equations. Moving from one vertex to another along an edge of the polyhedron is done by the algebraic operation of *pivoting*. Pivoting is used by the simplex algorithm for solving a linear program, and by the algorithm of Lemke and Howson (1964) for finding one equilibrium of a bimatrix game.

Consider a polyhedron such as $Q = \{y \in \mathbb{R}^n \mid Ay \le q,\ y \ge \mathbf{0}\}$ for an $m \times n$ matrix A and m-vector q. Then $y \in Q$ if and only if there exists a vector of slack variables $r \in \mathbb{R}^m$ so that

$$Ay + r = q, \qquad y \ge \mathbf{0},\ \ r \ge \mathbf{0}. \tag{10}$$

slacks

The system (10) is of the form

$$Cz = q \tag{11}$$

tabl

for a matrix C, right-hand side q, and a vector z of nonnegative variables. The matrix C has full row rank, so that q always belongs to the space spanned by the columns C_j of C. A *basis* β is given by a basis $\{C_j \mid j \in \beta\}$ of this column space, so that the square matrix C_β formed by these columns is invertible. The corresponding *basic solution* is the unique vector $z_\beta = (z_j)_{j\in\beta}$ with $C_\beta z_\beta = q$, where the variables z_j for j in β are called *basic variables*, and $z_j = 0$ for all *nonbasic* variables z_j for $j \notin \beta$, which implies (11). If this solution also fulfills $z \ge \mathbf{0}$, then the basis β is called *feasible*. If β is a basis for (11), then the corresponding basic solution can be read directly from the equivalent system $C_\beta^{-1}Cz = C_\beta^{-1}q$, called a *tableau*, because the columns of $C_\beta^{-1}C$ for the basic variables form the identity matrix. The tableau and thus (11) is equivalent to the system, also called a *dictionary*,

$$z_\beta = C_\beta^{-1}q - \sum_{j\notin\beta} C_\beta^{-1}C_j z_j \tag{12}$$

dict

which shows how the basic variables depend on the nonbasic variables.

The basic feasible solutions to (11) represent the vertices of the polyhedron, for the following reason. Setting any variables z_j in (11) to zero defines a face of the polyhedron. If these variables are the nonbasic variables of a basic feasible solution,

that face contains only a single point of the polyhedron, which is therefore a vertex. Conversely, consider a vertex of the polyhedron, given by a vector z in (11). The vertex is a zero-dimensional face, defined by the binding inequalities that correspond to the zero components of z. The positive components of z define linearly independent columns of C, because otherwise it is easy to see that there would be additional positive solutions for the same binding inequalities, so that the face would not be zero-dimensional. The linearly independent columns can be extended with suitable additional columns C_j (for which $z_j = 0$) to form a basis. In a *degenerate* basic feasible solution, some basic variables are zero; the respective vertex can typically be represented by more than one degenerate basis. If all basic variables are positive, the basis is called *nondegenerate*.

We use algorithms that move from one vertex of a polyhedron to another vertex along an edge. This corresponds to a change of the basis β in (12) known as *pivoting*. Thereby, a nonbasic variable z_j for some j not in β *enters* and a basic variable z_i for some i in β *leaves* the set of basic variables. The pivot step is possible if and only if the coefficient of z_j in the ith row of the current tableau is nonzero, and is performed by solving the ith equation for z_j and then replacing z_j by the resulting expression in each of the remaining equations.

For a given entering variable z_j, the leaving variable is chosen to preserve feasibility of the basis. Let the components of $C_\beta^{-1}q$ be $\overline{q}_i$ and of $C_\beta^{-1}C_j$ be $\overline{c}_{ij}$, for $i \in \beta$. Then the largest value of z_j such that in (12) $z_\beta = C_\beta^{-1}q - C_\beta^{-1}C_j z_j$ is nonnegative is given by

$$\min\{\,\overline{q}_i/\overline{c}_{ij} \mid i \in \beta,\ \overline{c}_{ij} > 0\,\}. \tag{13}$$

ratio

This is called a *minimum ratio test*. If i in β achieves the minimum in (13), then z_i can be chosen as a leaving variable. After pivoting, the new basis is $\beta \cup \{j\} - \{i\}$.

The minimum in (13) may be zero, if the current basis β is degenerate and $\overline{q}_i = 0$ for some $i \in \beta$ with $\overline{c}_{ij} > 0$. Then the pivoting step changes the basis but not the basic feasible solution z, so the corresponding vertex stays the same.

If the minimum in (13) is not unique, two (or more) variables can leave the basis, but only one variable does. The other variable stays basic and becomes zero after the pivoting step, so that the new basis is degenerate.

The *lexicographic method* extends the minimum ratio test (13) in such a way that the leaving variable is always unique, even in degenerate cases. The method simulates an infinitesimal perturbation of the right-hand side q of the given linear system (11) and works as follows. For any $\varepsilon \geq 0$, consider the system

$$Cz = q + (\varepsilon^1, \ldots, \varepsilon^m)^\top \tag{14}$$

perturb

which is equal to (11) for $\varepsilon = 0$ and which is a *perturbed* system for $\varepsilon > 0$. Let β be a basis for this system with basic solution

$$z_\beta = C_\beta^{-1}q + C_\beta^{-1} \cdot (\varepsilon^1, \ldots, \varepsilon^m)^\top = \overline{q} + C_\beta^{-1} \cdot (\varepsilon^1, \ldots, \varepsilon^m)^\top \tag{15}$$

lexico

and $z_j = 0$ for $j \notin \beta$. It is easy to see that z_β is positive for all sufficiently small ε if and only if all rows of the matrix $[\overline{q}, C_\beta^{-1}]$ are *lexico-positive*, that is, the first nonzero component of each row is positive. Then β is called a *lexico-positive* basis. This holds

in particular for $\overline{q} > \mathbf{0}$ when β is a nondegenerate basis for the unperturbed system. Because C_β^{-1} has no zero row, any feasible basis for the perturbed system is nondegenerate. In consequence, the leaving variable for the perturbed system is always unique. It is determined by the *lexico-minimum ratio test* which is a straightforward extension of (13) (see Chvátal (1983) or von Stengel (2002, p. 1741)). Pivoting with the lexico-minimum ratio test moves from one lexico-positive basis to another. It uses only the entries of C_β^{-1} and does not need an actual perturbation with positive ε.

Our algorithms use exact arithmetic with *integers* of arbitrary precision, which avoids rounding errors of floating-point arithmetic. We use *integer pivoting*, which is superior to using fractions of integers (rational arithmetic) because their cancellation requires greatest common divisor computations which tend to take the bulk of computation time. In integer pivoting, the dictionary (12) is stored with all numbers multiplied by the determinant of C_β, so that (by Cramer's rule) these numbers are integers if the entries of C are integers; the determinant is stored separately. Pivoting is done by row operations on the system followed by a division by the old determinant, which always produces integers (see Avis 2000, Sect. 7 or Azulay and Pique 2001). In that way, the dictionary entries are kept from growing indefinitely.

6 Finding all extreme equilibria using vertex enumeration

s-lrs

We first describe a straightforward method to generate all extreme equilibria, which define the input to the "Clique" Algorithm 2.

a-vertenum

Algorithm 3 (Enumerating and matching vertices of both polytopes) *Input:* Bimatrix game (A,B). *Output:* All extreme equilibria (x,y). *Method:* Enumerate all vertices x of $P-\{\mathbf{0}\}$ and y of $Q-\{\mathbf{0}\}$ in (7), and output every completely labeled pair (x,y).

Enumerating all vertices of polytope is an important, well-studied and difficult problem in polyhedral computation. It is still unknown if it is possible to do this efficiently in general, i.e., in time polynomial in both the input and output size. Two basic ways to solve this problem are by the double description method (see Motzkin et al. (1953), Fukuda and Prodon (1995)) and by pivoting (see Chvátal (1983), Avis and Fukuda (1992)). Both methods have their strengths and weaknesses, as is discussed in detail in Avis et al. (1997). As either method may be used for Algorithm 3, we give a few remarks here.

Double description methods tend to work well for highly degenerate polyhedra, especially those with relatively few vertices. These polyhedra cause pivoting methods to behave very badly. A drawback is that a large amount of memory may be required for intermediate steps, even when the output size is small. When the output size is large, this can cause the program to run out of memory. We remark that the polytopes P and Q may have as many as $(m+n)^{\lfloor m/2 \rfloor}$ and $(m+n)^{\lfloor n/2 \rfloor}$ vertices, respectively.

General pivoting pivoting methods may use large amounts of memory also, but this problem has been eliminated in Avis and Fukuda's (1992) *reverse search* algorithm. This uses space proportional only to the input size, i.e. $O(mn)$, and produces

the output as a stream that need not be saved, or can be saved off-line. These properties will be exploited in the *lrsNash* Algorithm 4 to be presented later. The problems associated with degenerate polyhedra have been overcome to some extent in the *lexicographic reverse search* algorithm *lrs* (Avis 2000; 2006).

Consider the polytope P as defined in (7). It has a known vertex $\mathbf{0}$, which is the unique point of P at which the linear objective function $x \mapsto -\mathbf{1}^\top x$ is maximized. The simplex algorithm for maximizing this linear function computes from any vertex of P a path of pivoting steps to $\mathbf{0}$. With a deterministic pivoting rule, that path is unique. In *lrs*, that pivoting rule chooses the variable with the least index (i.e., smallest subscript) that improves the objective function as entering variable, and the leaving variable via the lexicographic rule described after (15). (In contrast, the earlier *reverse search* by Avis and Fukuda (1992) used Bland's least-index rule for the leaving variable.)

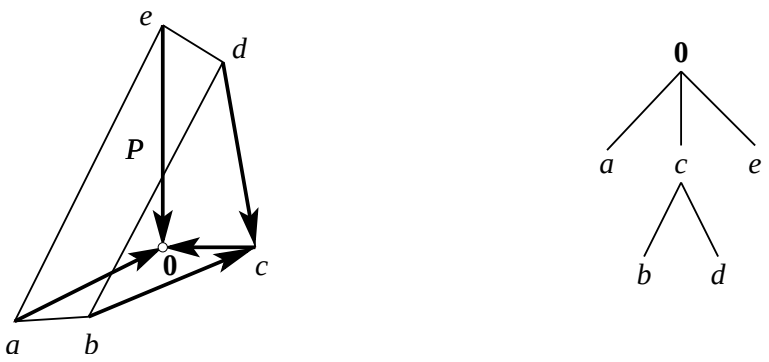

Fig. 4 Left: Tree of simplex steps for maximizing $-\mathbf{1}^\top x$ on the polytope P for the example (5). Right: The corresponding reverse search tree. f-lrsP

The unique paths of simplex steps from the vertices of P to $\mathbf{0}$ define a tree with root $\mathbf{0}$. For the polytope P resulting from the example (5), as in the top right picture of Fig. 1, that tree is shown on the left in Fig. 4. The *lrs* algorithm explores this tree by traversing the edges in the reverse direction using a depth-first search, which in the example outputs the vertices in the order $\mathbf{0}, a, c, b, d, e$. For a given vertex u of P, the children v of u in the tree are found by considering possible reverse pivots from u to v and then checking if the simplex rule would actually move from v to u.

The simplex method moves from basis to basis, but several degenerate bases may represent the same vertex, which should be output only once. For a given vertex, it is straightforward to determine if a given basis β that represents it is lexicographically smallest, that is, there is no basis β' representing the vertex with $j \in \beta' - \beta$ and $j < i$ for all $i \in \beta - \beta'$ (Avis 2000, Prop. 5.1). Moreover, that lexicographically smallest basis is also lexico-positive (Avis 2000, Prop. 5.2). The vertex is only output when this lexicographically smallest basis is encountered, so there are no duplicate vertices in the output list.

When using *lrs* for enumerating the vertices x of P (and similarly of Q) in Algorithm 3, the missing labels k of x can be identified from the positive slack vari-

ables (which are only visible as $x_k > 0$ when $k \in M$, but not when $k \in N$) via the `printslack` option of *lrs*. This allows a straightforward implementation of Algorithm 3 with *lrs*, which is used in the website of Savani (2005). For each vertex, its set of labels is stored as a pattern of $m+n$ bits. For each vertex x of P, the matching vertices y of Q are searched linearly to find the equilibria (x,y). If P has p and Q has q vertices, this matching process takes time $O(pq)$, but it tends to be negligible in comparison to enumerating the vertices in the first place, unless p and q are very large. The space required is proportional to $p+q$, the output size of the two vertex enumeration problems, both of which must be completely solved.

The following algorithm has several advantages over Algorithm 3. Firstly it requires that only one of the two vertex enumeration problems needs to be completely solved. Since P and Q need not be related in any way, one of them may well be easier than the other in terms of vertex enumeration. Secondly, the new algorithm requires only memory proportional to the input size $O(mn)$ rather than the output size $O(p+q)$, which as we saw may be super-exponential in m and n. Thirdly the equilibria are produced as as a stream while the vertex enumeration is being performed, so useful output may be obtained even when a complete enumeration of all equilibria is not tractable. Fourthly, it does not require a separate matching process.

The *lrsNash* Algorithm 4 enumerates the vertices x of only one polytope, say P. The set K of labels of x defines a set $L = (M \cup N) - K$ of labels missing from x that a vertex y of Q must have in order to obtain an equilibrium (x,y). The labels in L define the face $Q(L)$ of Q according to (9). If the game is nondegenerate, then, by Remark 1, $|K| = m$ and $|L| = n$ and $Q(L)$ is either empty or a single vertex of Q. If the game is degenerate, then it is possible that $|L| < n$ so that $Q(L)$ may be a polytope of higher dimension, although typically still of much smaller dimension than Q. If $Q(L)$ is not empty, it has a vertex that can be used as a starting point for enumerating its vertices with *lrs*.

`a-singlepol`

Algorithm 4 (*lrsNash***)** *Input:* Bimatrix game (A,B). *Output:* All extreme equilibria (x,y). *Method:* For each vertex x of $P - \{\mathbf{0}\}$ and set L of labels missing from x,
(a) determine whether $Q(L)$ is empty or else find a vertex of $Q(L)$, and then
(b) enumerate the vertices y of $Q(L)$ and output (x,y).

The *lrsNash* Algorithm 4 is implemented as the method `nash` of the *lrs* program (Avis 2006). We explain the implementation of part (a), where we assume some familiarity with linear programming terminology (see, for example, Chvátal (1983)). A standard way to solve (a) is the phase-1 simplex method. However, we use a specialized approach which is adapted to the use of *lrs* for enumerating the vertices of P and therefore very fast.

We distinguish two types of dictionaries. A *full dictionary* as in (12) is an equivalent way of representing all linear constraints that define Q. The corresponding basis does not have to be feasible. In a *reduced dictionary*, some nonbasic variables are fixed at zero. The corresponding columns are omitted, so the reduced dictionary no longer represents the full information about Q. It may happen that a row of the reduced dictionary has all coefficients and right-hand side zero (so the basic variable is

zero); that row is then omitted. (A zero row cannot appear in a full dictionary because the system has full row rank.)

A vertex x of P defines a set L of missing labels. We first create a full dictionary where as many elements of L as possible become nonbasic variables (we identify the variables of the dictionary with the elements of $M \cup N$). This is done by starting with some full dictionary and pivoting iteratively a basic variable in L out of the basis using any entering column that does not belong to L. The pivot element can be any nonzero coefficient because the dictionary does not have to be feasible. Let $D(L)$ be any full dictionary so that the nonbasic variables contain a maximal subset of L (often L itself).

When using *lrs* to enumerate the vertices x of P (with missing label set L), the next vertex x' (with missing label set L') is often adjacent to x. In that case, the full dictionary $D(L')$ is usually quickly obtained from $D(L)$. Moreover, *lrs* keeps a cache for storing previous vertices x to speed up backtracking, and *lrsNash* also caches the corresponding full dictionaries $D(L)$. This creates the main speedup compared to using a standard phase-1 simplex method for part (a) of the *lrsNash* Algorithm 4.

After obtaining a full dictionary $D(L)$ (which is saved for finding the next dictionary $D(L')$ as described), it is converted to a reduced dictionary by eliminating all columns that belong to L, and afterwards omitting any zero rows, which may include further elements of L. Suppose some element of L is basic in the reduced dictionary. Then in that row, all coefficients of nonbasic variables (none of which belongs to L) are zero, because otherwise the basic variable could have been pivoted out. Hence the basic variable is always equal to the constant in that row, which is nonzero (otherwise the entire row would have been omitted), and the set $Q(L)$ is empty, which completes (a). Hence, we can assume that the reduced dictionary has no variable in L, so that the set of its feasible solutions is $Q(L)$.

The nondegenerate case is that $|L| = n$ and the reduced dictionary has no nonbasic columns at all. Then $Q(L)$ is nonempty if and only if the basic solution is feasible, which is then the sole vertex in $Q(L)$. In general, the reduced dictionary may have some nonbasic variables. If the basic solution is feasible, it defines a vertex of $Q(L)$. Otherwise, *lrs* finds such a vertex, or determines that the system is infeasible, with the dual simplex method. This completes part (a). Because the reduced dictionary has typically low dimension, this part is negligible compared to the enumeration of the vertices of P. Finally, given a vertex of $Q(L)$, a standard run of *lrs* solves part (b) of Algorithm 4.

In the *lrsNash* Algorithm 4, the roles of P and Q can be exchanged, which one could call *lrsNash*$^\top$. The running time of *lrs* is determined by the number of feasible bases, so enumerating the vertices of P is faster if P is the polytope with fewer bases. This is usually, but not always, the polytope of smaller dimension, that is, for the player with fewer strategies. A feature of *lrs* is that it can rapidly provide an unbiased estimate of the number of feasible bases of a given polyhedron, and this can be used as a preprocessing step to choose the polytope that plays the role of P in Algorithm 4.

In terms of running time, part (a) can typically be done quickly, as mentioned after Algorithm 4. Part (b) is necessary because it produces the equilibria, so their number is always relevant for the running time. If Q has much more feasible bases than P, only a fraction of them are visited by *lrsNash*, which is then much better than

Algorithm 3 because the overall running time only depends on the enumeration of the vertices of P.

A bimatrix game may have an exponential number of equilibria. For example, the coordination game where each payoff matrix is the $n \times n$ identity matrix has $2^n - 1$ equilibria; $n \times n$ games with more than 2.4^n equilibria are described by von Stengel (1999). Hence, enumerating all equilibria cannot be done in a running time that is polynomial in the input size. A running time that is polynomial in the *output* size cannot be expected either, because this would give a polynomial-time algorithm that decides if a game has a unique Nash equilibrium, which is an NP-hard problem (Gilboa and Zemel 1989). Algorithm 4 is however *space efficient* if *lrs* is used to do the vertex enumeration. It requires only $O(mn)$ space to produce the possibly exponential number of equilibria, output as a duplicate-free stream.

Algorithms 3 and 4 can be extended so that only equilibria with a given minimum payoff, say $\bar{u}$ for player 1 and $\bar{v}$ for player 2, are enumerated. When enumerating all vertices of polytope P, say, with *lrs*, this is done by terminating the reverse search at vertices x where the objective function $-\mathbf{1}^\top x$ is less than $-1/\bar{v}$. One way to do this would be to add the additional constraint $\mathbf{1}^\top x \leq 1/\bar{v}$ to the definition of $P(L)$. However this would mean that additional vertices would now be produced which are not vertices of $P(L)$. They would need to be skipped, and the added constraint would create many unnecessary pivots. Fortunately the structure of the reverse search tree can be exploited. From the description given earlier in this section, we see that the value of the objective function is maximized at the root, and decreases monotonically along any path in the tree. We simply truncate the tree whenever a pivot would lead to a vertex that violates the constraint. A similar method is used in the vertex enumeration of $Q(L)$. Clearly the game may not have any equilibria with payoffs satisfying given bounds, and deciding whether such equilibria exist is NP-complete (Gilboa and Zemel 1989).

7 The modified EEE algorithm

s-EEE

Audet et al. (2001) describe an algorithm they call *EEE* for "Enumeration of Extreme Equilibria". The algorithm initially traverses a binary search tree. Each node of the search tree represents a pair of parameterized linear programs where certain pure strategies are constrained either to have probability zero or to be a best response. The two children of a node are obtained by forcing either constraint for an additional pure strategy. If the added constraint results in an infeasible system, the search terminates, which hopefully happens as early as possible. If all pure strategies are either best responses or have zero probability, the resulting solution is an extreme equilibrium. In a degenerate game, an additional search is needed at this stage to find all extreme equilibria.

We present two modifications of the original *EEE* algorithm by Audet et al. (2001). The first was implemented as an extension of work by Rosenberg (2005), the second is new and has not yet been implemented. Both algorithms are relatively similar and differ from *EEE* in how they handle degenerate games. We will explain the algorithms in geometric terms, rather than as finding alternate solutions to pairs

of parameterized linear programs, which clarifies their connection to Algorithms 3 and 4. We also specify each algorithm concisely as a recursive depth-first search. Audet et al. (2001) allow for other traversals of the search tree, even though their implementation is also a depth-first search. Further implementation issues are discussed in Sect. 8.

The *EEE* algorithm uses the polyhedra $\overline{P}$ and $\overline{Q}$ in (4). (It could also be implemented using the polytopes P and Q in (7).) In the course of the computation, certain inequalities that define $\overline{P}$ and $\overline{Q}$ in (4) are forced to be binding, represented by sets of labels K and L, which are subsets of $M \cup N$. In analogy to (9), let

$$\begin{aligned} \overline{P}(K) &= \{\, (x,v) \in \overline{P} \mid \forall i \in K \cap M : \quad x_i = 0, \;\; \forall j \in K \cap N : (B^\top x)_j = v \,\}, \\ \overline{Q}(L) &= \{\, (y,u) \in \overline{Q} \mid \forall i \in L \cap M : (Ay)_i = u, \;\; \forall j \in L \cap N : \quad y_j = 0 \,\}. \end{aligned} \tag{16}$$

PbarK

A *node* of the search tree of the algorithm is defined by disjoint sets of labels K, L so that the faces $\overline{P}(K)$ and $\overline{Q}(L)$ are not empty. In addition to K and L, a node stores *witnesses* x and y so that (x,v) is a vertex of $\overline{P}(K)$ and (y,u) is a vertex of $\overline{Q}(L)$ for suitable scalars v,u. These scalars are uniquely determined by x and y as the best-response payoffs against x and y, respectively.

Suppose (K,L,x,y) represents a node of the search tree so that $|K \cup L| < m+n$. Then a new label h not in $K \cup L$ is selected and added to either K or L, which defines the two children of that node. However, if the resulting face $\overline{P}(K \cup \{h\})$ or $\overline{Q}(L \cup \{h\})$ is empty, the respective child is omitted and the search tree pruned at that point.

The root of the search tree is given by $K = L = \emptyset$ and vertices (x,v) of $\overline{P}$ and (y,u) of $\overline{Q}$, respectively. The root has *level* zero, and the level of any other node is one more than the level of its parent. (The level of a node is one less than the search depth in Audet et al. (2001) who start with the root at depth one.) At level $m+n$, the label sets K, L fulfill $K \cup L = M \cup N$, so that the witness pair (x,y) defines an equilibrium. In a nondegenerate game, all equilibria are obtained in this way. In general, not all extreme equilibria are in obtained this way, because $\overline{P}(K)$ and $\overline{Q}(L)$ may not be singletons, and an additional enumeration of vertices is required.

The first variant of the *EEE* algorithm is as follows. Its details, in particular the selection of the added label h, are explained afterwards.

a-ieee

Algorithm 5 (*EEE-m* – Modified EEE) *Input:* Bimatrix game (A,B). *Output:* All extreme equilibria (x,y). *Method:* Implicit depth-first search on a binary tree by choosing any vertices (x,v) of $\overline{P}$ and (y,u) of $\overline{Q}$, and calling `visit`$(\emptyset,\emptyset,x,y)$ with the recursive `visit` method in Fig. 5.

Algorithm 5 is based on the `visit` method, which is a standard recursive depth-first exploration of a search tree. A node (K,L,x,y) of the search tree corresponds to a call to the `visit` method, and its children correspond to the two recursive calls (if they take place) in lines 4 and 6, respectively. The level of the node is given by $|K \cup L|$. At level $m+n$, no further recursion takes place, and the method performs the "else" part in lines 8–9.

Line 2 of the `visit` method asks for the selection of a label h, which is added to K or L in lines 4 and 6, respectively. Following Audet et al. (2001), h is chosen as follows. Consider the slack vectors $s = \mathbf{1}v - B^\top x$ and $r = \mathbf{1}u - Ay$. First, suppose that

```
visit (K,L,x,y):
[  assumption:  (x,v) vertex of P̄(K),  (y,u) vertex of Q̄(L)  ]
1     if  |K∪L| < m+n  then
2           select h ∈ (M∪N) − (K∪L)
3           if  |K| < m and ∃ vertex (x',v') of P̄(K∪{h})  then
4                 visit (K∪{h},L,x',y)
5           if  |L| < n  and ∃ vertex (y',u') of Q̄(L∪{h})  then
6                 visit (K,L∪{h},x,y')
7     else
8           for all vertices (x,v) of P̄(K) and (y,u) of Q̄(L):
9                 output (x,y) if not already output earlier.
```

Fig. 5 The recursive `visit` method used in the *EEE-m* Algorithm 5. `f-visit`

(x,y) is not an equilibrium of the game. Then there is a label h so that $x_h r_h > 0$ (that is, $h \in M$) or $y_h s_h > 0$ (that is, $h \in N$), and h is chosen so that $x_h r_h$ or $y_h s_h$ is maximal among these products, with smallest such h in case of ties. Suppose that product is $x_h r_h$, so that adding h to K means forcing the equation $x_h = 0$ when changing from the face $\overline{P}(K)$ to its subface $\overline{P}(K \cup \{h\})$, and adding h to L means forcing the equation $r_h = 0$ when changing from the face $\overline{Q}(L)$ to its subface $\overline{Q}(L \cup \{h\})$. With this heuristic choice of h, it is hoped to prune the search tree early when the smaller face $\overline{P}(K \cup \{h\})$ or $\overline{Q}(L \cup \{h\})$ is found to be empty.

If (x,y) is already an equilibrium, then $x_h r_h = 0$ and $y_h s_h = 0$ for all h in $M \cup N$. Then any h not in $K \cup L$ is selected, and one can use the same witness for one of the children in the search tree. For example, if $x_h = 0$, then adding h to K means x is already a witness for the face $\overline{P}(K \cup \{h\})$. However, then typically $r_h > 0$ holds and forcing $r_h = 0$ requires a new witness y' for the face $\overline{Q}(L \cup \{h\})$ corresponding to the other child, if it exists, and (x,y') may no longer be an equilibrium. In short, during the search it is irrelevant whether the witness pair (x,y) is an equilibrium.

Lines 3–4 and 5–6 of `visit` describe the branchings to the two children during the search. In line 3 of `visit`, a vertex (x',v') of $\overline{P}(K \cup \{h\})$ is found using the previous vertex (x,v) of $\overline{P}(K)$. This vertex (x,v) corresponds to a basic feasible solution with a dictionary that represents $\overline{P}$ where all variables in K are nonbasic. By omitting these nonbasic columns altogether, we obtain a *reduced dictionary* that represents $\overline{P}(K)$, as explained after Algorithm 4. Adding the constraint $x_h = 0$ (if $h \in M$) or $s_h = 0$ (if $h \in N$, with $s = \mathbf{1}v - B^\top x$) means driving that variable x_h or s_h out of the basis, so that the variable becomes nonbasic and its column can be omitted from the reduced dictionary so that it represents $\overline{P}(K \cup \{h\})$. If this is not possible, the system is usually infeasible. The only exception is if the variable corresponds to an all-zero row of the reduced dictionary, in which case that row is omitted. This is analogous to the discussion following Algorithm 4. In line 5 of `visit`, a vertex (y',u') of $\overline{Q}(L \cup \{h\})$ is found analogously.

As described so far, the search for a witness x' stops at the first vertex (x',v') found on the face $\overline{P}(K \cup \{h\})$, which suffices for the algorithm to work. Audet et al. (2001)

maximize an *objective function* subject to the constraints that define $\overline{P}(K \cup \{h\})$. Their objective function is ${x'}^{\top}(Ay) - v'$ (using the other witness y), and similarly $(x^{\top}B)y' - u'$ to find a vertex (y', u') of $\overline{Q}(L \cup \{h\})$ in line 5 of `visit`, in order to "guide" the computation towards equilibria (x,y) where the sum $x^{\top}(Ay) + (x^{\top}B)y - u - v$ of these two objective functions is zero and therefore maximal. Section 8 reports on computational experiments that compare this pair of objective functions by Audet et al. (2001) with other possibilities.

In a nondegenerate game, no vertex of $\overline{P}$ has more than m labels, and no vertex of $\overline{Q}$ has more than n labels. Hence, the condition $|K \cup L| = m + n$ that reaches the final "else" part in lines 7–9 of `visit`(K,L,x,y) occurs only when $|K| = m$ and $|L| = n$. Then it suffices to output the unique equilibrium (x,y) at this terminal node of the search tree. Indeed, then the enumerations in line 8 are trivial because then $\overline{P}(K) = \{(x,v)\}$ and $\overline{Q}(L) = \{(y,u)\}$.

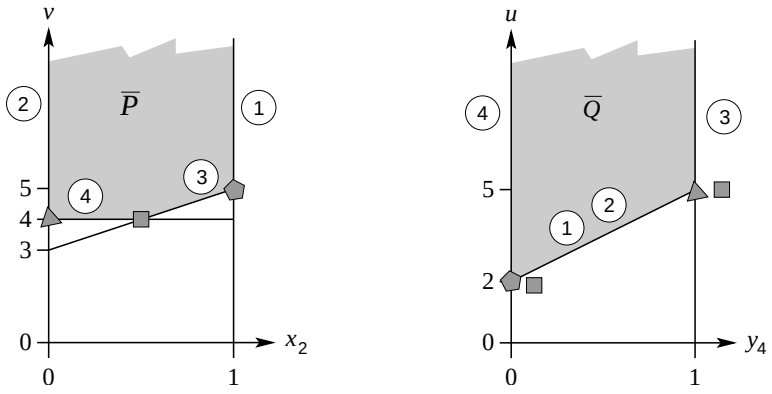

Fig. 6 The polyhedra $\overline{P}$ and $\overline{Q}$ for the degenerate game (17), and its extreme equilibria.

f-else

For degenerate games, we give an example that shows that we need the enumeration in line 8 of `visit`. Consider the degenerate game (A,B) defined by

$$A = \begin{bmatrix} 2 & 5 \\ 2 & 5 \end{bmatrix}, \qquad B = \begin{bmatrix} 3 & 4 \\ 5 & 4 \end{bmatrix}. \tag{17}$$

else

The polyhedra $\overline{P}$ and $\overline{Q}$ are shown in Fig. 6. The polyhedron $\overline{P}$ has three vertices (x_1, x_2, v), namely $(1,0,4)$ with label set $\{2,4\}$, and $(1/2,1/2,4)$ with label set $\{3,4\}$, and $(0,1,5)$ with label set $\{1,3\}$. The polyhedron $\overline{Q}$ has only two vertices (y_3, y_4, v), namely $(1,0,2)$ with label set $\{1,2,4\}$, and $(0,1,5)$ with label set $\{1,2,3\}$. The game has four extreme equilibria: The two pure equilibria $((1,0),(0,1))$ and $((0,1),(1,0))$ shown as pairs of triangles and pentagons, respectively, in Fig. 6, and two equilibria $((1/2,1/2),(1,0))$ and $((1/2,1/2),(0,1))$ that use the mixed strategy of player 1, indicated by the square in $\overline{P}$ paired with either square in $\overline{Q}$. The maximal Nash subsets (see Prop. 4) of the game are $\mathrm{conv}\{(1,0),(1/2,1/2)\} \times \{(0,1)\}$, $\{(1/2,1/2)\} \times \mathrm{conv}\{(1,0),(0,1)\}$, and $\mathrm{conv}\{(1/2,1/2),(0,1)\} \times \{(1,0)\}$.

The only two-element sets K so that $\overline{P}(K)$ is not empty are $\{2,4\}$, $\{3,4\}$, and $\{1,3\}$, which define the three vertices of $\overline{P}(K)$. Then the last level 4 of the search tree is reached for $L=\{1,3\}$, $\{1,2\}$, and $\{2,4\}$, respectively. For $L=\{1,3\}$ and $L=\{2,4\}$, the corresponding face $\overline{Q}(L)$ is a vertex; these two pairs (K,L) give the pure-strategy equilibria. For $K=\{3,4\}$, $x=(1/2,1/2)$, and $L=\{1,2\}$, however, $\overline{Q}(L)$ is a higher-dimensional face, an edge of $\overline{Q}$. Only one of its vertices gives a witness y, for example $y=(1,0)$. If one would now only output the equilibrium (x,y), one would miss the equilibrium (x,y') for the other vertex y' of $\overline{Q}(L)$, in the example $y'=(0,1)$, so the enumeration in line 8 of `visit` is needed.

The necessity of doing additional enumeration after reaching level $m+n$ of the search tree was already observed by Audet et al. (2001); we discuss their implementation of this stage in Sect. 8. The following proposition asserts that Algorithm 5 is correct, which is slightly more involved than the original correctness proof of Audet et al. (2001) for their *EEE* algorithm which does not test for the conditions $|K|<m$ and $|L|<n$ in lines 3 and 5 of the `visit` method.

p-ee **Proposition 5** *Algorithm 5 enumerates all extreme Nash equilibria of the game.*

Proof. Let (x,y) be an extreme equilibrium, with vertices (x,v) of $\overline{P}$ and (y,u) of $\overline{Q}$. Let K_x and L_y be the sets of labels of x and y, respectively. We claim that there are disjoint sets K and L so that $|K|=m$, $|L|=n$, and $K\subseteq K_x$ and $L\subseteq L_y$ (note that K and L need not define bases that represent the vertices). Namely, the set $M\cup N$ is the disjoint union of K_x-L_y, $K_x\cap L_y$ and L_y-K_x. With $k=|K_x-L_y|$ and $l=|L_y-K_x|$, we have $k\le m$ because $|L_y|\ge n$ and $l\le n$ because $|K_x|\ge m$. Partition $K_x\cap L_y$ arbitrarily into sets K' and L' of sizes $m-k$ and $n-l$, respectively. Then the claim holds with $K=(K_x-L_y)\cup K'$ and $L=(L_y-K_x)\cup L'$.

Then (x,v) and (y,u) are vertices of the faces $\overline{P}(K)$ and $\overline{Q}(L)$, respectively, so these two faces are nonempty, and the two vertices are found in line 8 of the `visit` method in Fig. 5. □

Because of the size constraints $|K|<m$ and $|L|<n$ in lines 3 and 5 of the `visit` method in Fig. 5, the enumeration in line 8 is reached only for sets K,L with $|K|=m$ and $|L|=n$. This has the advantage that even when only one of the polyhedra $\overline{P}$ or $\overline{Q}$ is simple and has no redundant inequalities due to weakly dominated or payoff equivalent pure strategies, as in Fig. 6, the list of extreme equilibria is free of duplicates:

p-nodupl **Proposition 6** *If $\overline{P}$ or $\overline{Q}$ are nondegenerate in the sense that no vertex of $\overline{P}$ has more than m labels or no vertex of $\overline{Q}$ has more than n labels, then the extreme equilibria are enumerated without the need to check for duplicates in lines 8–9 of the* `visit` *method.*

Proof. Suppose that $\overline{P}$ is nondegenerate as described; the case for $\overline{Q}$ is analogous. Then $\overline{P}(K)$ in line 8 of the `visit` method consists of a single vertex (x,v) because $|K|=m$, so distinct sets K will produce distinct equilibria (x,y). □

As the proof of Prop. 6 shows, an even weaker condition is that $\overline{P}(K)$ (or correspondingly $\overline{Q}(L)$) is a singleton for any m-element set K encountered during the search. A sufficient condition for this is that all m basic variables are positive in the reduced dictionary that represents $\overline{P}(K)$. As long as this holds for all bases found for either polyhedron, one can omit the search for duplicates in line 9 of `visit`.

The tests for $|K| < m$ and $|L| < n$ in lines 3 and 5 of `visit` are new. In the original *EEE* algorithm of Audet et al. (2001), larger sets K or L are considered until $K \cup L = M \cup N$. This may lead to unnecessary duplicates: In the example (17), all equilibria are found again with the sets K, L of labels $K = \{4\}$, $L = \{1,2,3\}$, and $K = \{3\}$, $L = \{1,2,4\}$, as Fig. 6 shows.

We now give another version of *EEE* that handles the problem of degenerate games in a different manner. In this version we allow the set K to exceed m elements.

a-ieee2

Algorithm 6 (*EEE-2* – Modified EEE, version 2) Identical to Algorithm 5, except that the `visit` method is replaced by the `visit′` method in Fig. 7.

```
visit′(K,L,x,y):
1    if |K∪L| < m+n then
2        select h ∈ (M∪N) − (K∪L)
3        if ∃ vertex (x′,v′) of P̄(K∪{h}) then
4            visit′(K∪{h},L,x′,y)
5        if |L| < n and ∃ vertex (y′,u′) of Q̄(L∪{h}) then
6            visit′(K,L∪{h},x,y′)
7    else
8        if K is the set of all labels of x then
9            enumerate all vertices (y,u) of Q̄(L) and output (x,y).
```

Fig. 7 The recursive `visit′` method used in Algorithm 6.

f-visit2

Note that if $\overline{P}$ is nondegenerate as in Prop. 6 (in particular if the game is nondegenerate), then the label set K cannot contain more than m labels and $\overline{P}(K)$ is a singleton, so that the *EEE-2* Algorithm 6 behaves identically to the *EEE-m* Algorithm 5. In the game (17), the two algorithms behave differently when A and B are interchanged, that is, when the polyhedra in Fig. 6 switch roles; for simplicity, assume we switch the names $\overline{P}$ and $\overline{Q}$ in that figure but keep the labels. Then the only cases where K in line 8 of `visit′` is the set of *all* labels of a vertex (of the right polyhedron in Fig. 6) is for $K = \{1,2,3\}$, $L = \{4\}$ and $K = \{1,2,4\}$, $L = \{3\}$. Then the enumeration in line 9 of `visit′` produces each extreme equilibrium exactly once. The following proposition asserts that this is the case in general.

p-ee2

Proposition 7 *Algorithm 6 enumerates all extreme Nash equilibria of the game without duplicates.*

Proof. Let (x,y) be an extreme equilibrium, so that (x,v) and (y,u) are vertices of $\overline{P}$ and $\overline{Q}$, respectively, with equilibrium payoffs v and u. Let K be the complete set of labels of x, so that $\overline{P}(K) = \{(x,v)\}$. Now $L = (M \cup N) - K$, so L is the set of labels missing from x. All labels in L are labels of y because (x,y) is an equilibrium. Therefore the face $\overline{Q}(L)$ of $\overline{Q}$ contains (y,u) and is not empty, and lines 7–9 of `visit′` are reached with parameters K and L. Then the vertex (y,u) of $\overline{Q}(L)$ is found in line 9,

and (x,y) is output. Because K is the set of all labels of x, the enumeration in lines 9 is performed at most once for each vertex x of $\overline{P}$. Therefore, if the vertex enumeration of $\overline{Q}(L)$ produces vertices without repetition, e.g. by using *lrs*, then the equilibria (x,y) are also output without repetition. □

Like the *lrsNash* Algorithm 4, the *EEE-2* Algorithm 6 is space efficient if *lrs* is used to enumerate vertices. The search tree has depth $m+n$, so an efficient implementation may need to cache up to this many dictionaries, each of size $O(mn)$. The vertex enumeration in line 9 requires an additional $O(mn)$ space.

We conclude with an improvement of Algorithm 6 omitted initially for simplicity. If $\{(x',y')\} = \overline{P}(K \cup \{h\})$ in lines 3–4 of `visit'`, then there is no need for subsequent branching in lines 1–6. We may add all labels of x' to K, set L to be the remaining labels, set $x = x'$ and go directly to line 8. Note that $Q(L)$ may in this case be empty, in which case no output is produced. If x' is a highly degenerate vertex, this eliminates uneccessarily creating a large subtree at the current node. However, this shortcut to the search may create duplicate outputs, because the complete label set for x' may now be produced in different ways. It is therefore necessary to modify line 8 so that vertex enumeration is only done once for each vertex x, which requires maintaining a list of such vertices. This in turn means that the algorithm is no longer space efficient, as this list may have exponential size.

Detecting the condition $\{(x',y')\} = \overline{P}(K \cup \{h\})$ depends on the implementation. If we have an explicit reduced dictionary that represents $\overline{P}(K \cup \{h\})$, this condition happens when the dictionary has no cobasic columns.

Finally, the *EEE-2* Algorithm 6 is not symmetric between the two players. Implementations and tests, which have yet to be done, should show which order of players is best, and how the algorithm competes with the *EEE-m* Algorithm 5.

8 Implementation and computational experiments

s-exper

In this section, we discuss aspects of the implementations of the *EEE* algorithm by Audet et al. (2001) and Rosenberg (2005), and its modification in Algorithm 5. We report on the empirical performance of these variants for some instances of games.

We also describe results of computational experiments that compare the *EEE* algorithm (its original version *EEE*-o as well as *EEE-m*) with the *lrsNash* Algorithm 4. Both algorithms have their strengths and weaknesses: *EEE* is not very suitable for degenerate games, already acknowledged as a possibility by Audet et al. (2001). However, for larger nondegenerate games, in particular square games, it scales better than an algorithm based on vertex enumeration such as *lrsNash*.

The implementation of the *EEE* algorithm by Audet et al. (2001) uses the commercial CPLEX solver for linear programs. It uses floating-point arithmetic, which may produce rounding errors. Equalities are assumed to hold whenever the compared numbers differ by less than 10^{-5}, so the computation is not exact. In contrast, Rosenberg (2005) has implemented the *EEE* algorithm as a stand-alone program in Java with exact arithmetic and integer pivoting (see the end of Sect. 5).

Both Audet et al. (2001) and Rosenberg (2005) use multiway branching as an implementation of the vertex enumeration in line 8 of the `visit` method in Fig. 5.

```
multiway(K,L,x,y):
[ assumptions: K∪L = M∪N, equilibrium (x,y),
               (x,v) vertex of P̄(K), (y,u) vertex of Q̄(L) ]
1   output (x,y) if not already output earlier
2   s = 1v − Ay, r = 1u − B^⊤ x
3   for all h ∈ M so that x_h > 0 and for all h ∈ N so that s_h > 0
4       if ∃ vertex (x′,v′) of P̄(K∪{h}) then
5           multiway(K∪{h},L,x′,y)
6   for all h ∈ M so that r_h > 0 and for all h ∈ N so that y_h > 0
7       if ∃ vertex (y′,u′) of Q̄(L∪{h}) then
8           multiway(K,L∪{h},x,y′)
```

Fig. 8 Recursive `multiway` method that implements the vertex enumeration in line 8 of `visit` in Fig. 5 by Audet et al. (2001) and Rosenberg (2005). `f-multiway`

This is shown as the recursive method `multiway` in Fig. 8. The vectors s and r of slack variables in line 2 are already stored in the reduced dictionaries that represent $\overline{P}(K)$ and $\overline{Q}(L)$, and are also available when using CPLEX. The possible indices h in lines 3 and 6 are positive basic variables of these dictionaries, and therefore not elements of $K \cup L$. When all cobasic variables have been eliminated from the reduced dictionary, the recursion is terminated immediately in the implementation by Rosenberg (2005) which has explicit access to the dictionary. This is slightly faster than the method of Audet et al. (2001) which terminates after unsuccessfully trying to set all positive basic variables to zero at that point.

size	5×5	5×10	5×15	5×20	5×25	5×30	10×10	10×15
sparse payoff matrices, nonzero with density 0.5								
#NE	6.3	5.6	12.8	9.6	7.1	18.1	6.8	9.5
EEE-m	0.30	0.02	0.45	0.31	0.30	1.14	0.17	0.69
EEE-o	0.54	0.03	1.23	0.44	0.32	2.89	0.14	0.77
lrsNash	0.01	0.03	0.03	0.04	0.07	0.11	0.30	1.43
lrsNash$^\top$	0.01	0.04	0.24	1.08	3.43	8.47	0.31	5.01
sparse payoff matrices, nonzero with density 0.2								
#NE	13.3	38.3	27.7	36.1	51.9	35.0	334.6	1967.3
EEE-m	0.05	0.24	3.14	0.26	9.84	25.07	26.94	388.98
EEE-o	0.09	13.63	10.83	1.23	44.33	148.41	3183.81	1658.49
lrsNash	0.02	0.02	0.03	0.03	0.05	0.05	0.21	0.75
lrsNash$^\top$	0.01	0.04	0.11	0.32	1.43	3.44	0.25	1.75

Table 1 Average running times in seconds for degenerate games obtained by sparse payoff matrices. Depending on the game size, around ten random games were tested for each size and density. #NE is the average number of extreme Nash equilibria, *EEE*-o is the original *EEE* algorithm without the tests $|K| < m$ and $|L| < n$ in lines 3 and 5 of the `visit` method in Fig. 5, and *lrsNash*$^\top$ is the *lrsNash* Algorithm 4 with P and Q interchanged. `t-testdeg`

The `multiway` branching in Fig. 8 is an extension of the binary branching in `visit`. In effect, it is a brute-force and inefficient vertex enumeration as required in line 8 of `visit`. This is probably one of the reasons that *EEE* performs poorly on degenerate games, as shown in the comparison of running times[1] in Fig. 1. For payoff matrices with a density 0.2 of nonzero entries, *EEE*-o may perform very badly on some instances. For *EEE*-m, the tests $|K| < m$ and $|L| < n$ in lines 3 and 5 of `visit` improve this behavior, which is still much worse than that of *lrsNash* for these game sizes. In a future implementation, line 8 of `visit` should be performed by a call to *lrs*, which so far has not happened because of the different programming languages (Java and C) used for the two programs. These also affect the running times to some extent.

size	5×5	5×10	5×15	5×20	5×25	5×30
#NE	3.0	3.6	7.4	9.0	7.6	14.4
EEE-m	0.06	0.05	0.51	0.94	1.66	3.24
lrsNash	0.01	0.03	0.04	0.04	0.08	0.11
size		10×10	10×15	10×20	10×25	10×30
#NE		11.2	20.0	30.0	31.6	34.5
EEE-m		1.18	1.68	5.00	10.19	21.79
lrsNash		0.35	1.38	3.20	5.24	9.75
lrsNash$^\top$		0.32	6.42	90.81	466.01	3079.38
size			15×15	15×20	15×25	15×30
#NE			36.6	55.8	117.8	174.8
EEE-m			10.21	31.88	137.18	398.09
lrsNash			50.28	199.26	651.26	772.83
lrsNash$^\top$			42.20			
size				20×20	20×25	20×30
#NE				140.7	320.2	651.0
EEE-m				299.40	1728.48	8341.56
lrsNash				5628.06	23154.05	∞
lrsNash$^\top$				4836.70		
size					25×25	25×30
#NE					354.0	327
EEE-m					6309.72	19000.81
lrsNash					∞	∞

Table 2 Average running times in seconds for nondegenerate games with random full matrices. Typically ten games were tested per size, fewer for large sizes. Tests showing ∞ were cancelled because they took longer than a day.

`t-testfull`

For random games with full payoff matrices, which are in most cases nondegenerate, the *EEE* algorithm behaves better and scales well, as shown in Table 2. The original *EEE* algorithm *EEE*-o is then identical to *EEE-m*. The *lrsNash* Algorithm 4 becomes inferior to *EEE* when each player has 15 or more strategies. For these games, the algorithm works better on the polytope P with fewer vertices, which is here the

[1] Tests were run on a standard 32-bit processor with 1.2 GHz clockspeed, provided by the "Amazon Elastic Compute Cloud" webservice, `http://aws.amazon.com/`

polytope of lower dimension. This is demonstrated by the entries for *lrsNash*$^\top$, which uses the higher-dimensional polytope, for games of size $10 \times n$. For square games, either polytope can be chosen; we also tested sizes 15×15 and 20×20, and observed that the running time often differs by a factor of two or more. Current computers are often dual- or quad-core with several processors that can be used simultaneously. These can work on the same game with different algorithms in parallel, cancelling the other computations when the first one finishes. This has been implemented for *lrsNash* and *lrsNash*$^\top$, and is available from the *lrs* website (Avis 2006). The space efficiency of *EEE-m*(Algorithm 4) means that there is little competition for memory or other resources, and the parallel version indeed runs in the shorter of the two running times.

As mentioned in Sect. 7, Audet et al. (2001) consider linear programs parameterized by x and y, with an objective function to find new vertices (x', v') of $\overline{P}(K \cup \{h\})$ and (y', u') of $\overline{Q}(L \cup \{h\})$ in lines 3 and 5, respectively, of the `visit` method. Table 3 compares various objective functions; the "guessing game" and "dollar game" are defined in Rosenberg (2005, pp. 41–43). The first line in Table 3 gives the original objective function. A motivation for that choice is that the sum of the objective functions for the two polyhedra is indeed maximized at an equilibrium, when the two duality gaps of the parameterized linear programs are both zero. The `visit` method would also run with an objective function that finds the lowest point on the upper envelope (minimizing v' and u', respectively) or when only looking for any feasible vertex with a constant objective function. Indeed, this provides a slight speedup for games that have many equilibria, as in the last column in Table 3. However, for games with fewer equilibria, or random games, the original objective functions are better. Compared to that, the last line in the figure shows even better objective functions ${x'}^\top (A+B)y - v'$ and $x^\top (A+B)y' - u'$ which seem to guide the search towards vertex pairs with good payoffs for both players; maximizing the sum of these two functions also closes the duality gaps. This last objective function is used in the computational experiments in Tables 1 and 2.

objective function for finding $(x', v') \in \overline{P}(K \cup \{h\})$	objective function for finding $(y', u') \in \overline{Q}(L \cup \{h\})$	**Random (average)** 17×17 45.8 NE	**Guessing Game** 22×22 3 NE	**Dollar Game** 15×15 211 NE
max ${x'}^\top Ay - v'$	max $x^\top By' - u'$	36.25	9.03	93.38
max $-v'$	max $-u'$	70.00	37.48	52.53
0	0	84.80	40.23	72.30
max ${x'}^\top (A+B)y - v'$	max $x^\top (A+B)y' - u'$	28.45	6.55	98.82

Table 3 Running times in seconds for various objective functions to determine the vertices (x', v') in line 3 and (y', u') in line 5 of the `visit` method in Fig. 5.

`t-objective`

9 Further work and open questions

s-open

The algorithms that we have described in Sect. 6 and Sect. 7 work for two-player games in strategic form. In general, one may consider "constrained" bimatrix games where the set of strategies of a player is not a simplex but a polytope define by more complex linear constraints. One such description is the "sequence form" that allows an efficient representation of behavior strategies of a game in extensive form (von Stengel 1996). An enumeration of equilibria based on this description should be possible by extending the presented algorithms. Crucially, there is typically more than one equality constraint that defines a player's strategy space. In consequence (by using linear programming duality), there will be more than one unbounded payoff variable. This is one of the reasons why we have studied the *EEE* algorithm using unbounded polyhedra and not polytopes. Equilibrium enumeration for constrained bimatrix games has been studied by Audet, Belhaïza, and Hansen (2006; 2009) and is also a topic for future work.

As the computational experiments show, the running times for equilibrium enumeration are exponential, so that sizes of games that can be solved soon hit a limit. An alternative to enumeration is the simpler problem of finding one Nash equilibrium. This can addressed by the algorithm by Lemke and Howson (1964), which seems to be efficient in practice. Apart from finding one equilibrium, the modeller is typically also interested in its uniqueness. This can only be decided in the negative with an algorithm that finds a second equilibrium if there is one. However, if many starting points always lead to the same equilibrium, this may be considered sufficient reason to accept it as "the" equilibrium that players are likely to play in practice. For this purpose, an algorithm with a large choice of starting points is desirable, such as that by van den Elzen and Talman (1991).

Even if many starting points are attempted, the path-following methods by Lemke and Howson (1964) and van den Elzen and Talman (1991) only find equilibria of positive index (for a definition of the index see Shapley (1974) or, for example, von Schemde and von Stengel 2008). Negatively indexed equilibria can be found as soon as one has two different positively index equilibria, by following the path backwards from one such equilibrium with the starting point that led to the other equilibrium. This may still fail to find all equilibria. At any rate, this approach has to our knowledge not yet been implemented.

Another useful feature would be to output the index of an equilibrium component, as an additional information provided by the *Clique* Algorithm 2. This is nontrivial in degenerate games where the index is computed for a component. It would be nice to find this lexicographically, that is, by means of symbolic, and not actual, perturbation techniques.

The *Clique* Algorithm 2 represents equilibrium components via their maximal Nash subsets, as the convex hull of their extreme equilibria. For a highly degenerate game, the definition by inequalities of the facets of P and Q that define the maximal Nash subset may be of use. This representation is trivial, but it is suitable to answer, for example, the query as to which component a given equilibrium belongs.

A shortcoming of many published algorithms for equilibrium computation is that they have been implemented ad hoc, to demonstrate some computational experi-

ments, but not robustly for public use. The *lrs* program by Avis (2000; 2006) is under constant development to be as useful as possible. We hope to make the algorithms presented in this paper easily available and convenient to use for the community.

Acknowledgements We thank Charles Audet for helpful comments.

10 References

Audet, C., Belhaïza, S., Hansen, P., Enumeration of all extreme equilibria in game theory: Bimatrix and polymatrix games. J. Optim. Theory Appl. 129, 349–372 (2006)

Audet, C., Belhaïza, S., Hansen, P., A new sequence form approach for the enumeration of all extreme nash equilibria for extensive form games. International Game Theory Review, to appear (2009)

Audet, C., Hansen, P., Jaumard, B., Savard G., Enumeration of all extreme equilibria of bimatrix games. SIAM J. Sci. Comput. 23, 323–338 (2001)

Avis, D. lrs: A revised implementation of the reverse search vertex enumeration algorithm. In: Kalai, G., Ziegler, G. (eds.): Polytopes – Combinatorics and Computation, DMV Seminar Band 29, pp. 177–198. Birkhäuser, Basel (2000)

Avis, D.: User's Guide for lrs. `http://cgm.cs.mcgill.ca/~avis` (2006)

Avis, D., Fukuda K., A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. Discrete and Computational Geometry 8, 295–313 (1992)

Azulay, D.-O., Pique, J.-F.: A revised simplex method with integer Q-matrices. ACM Trans. Math. Software 27, 350–360 (2001)

Bron, C., Kerbosch, J.: Finding all cliques of an undirected graph. Comm. ACM 16, 575–577 (1973)

Canty, M. J.: Resolving Conflicts with Mathematica: Algorithms for Two-Person Games. Academic Press, Amsterdam (2003)

Chvátal, V.: Linear Programming. Freeman, New York (1983)

Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C.: Introduction to Algorithms, 2nd ed. MIT Press, Cambridge, Mass. (2001)

Dickhaut, J., Kaplan, T.: A program for finding Nash equilibria. The Mathematica Journal 1:4, 87–93 (1991) Typesetter: 1:4 STET – this journal numbers pages separately per issue; delete this box

Fukuda, K., Prodon A.: Double description method revisited. In: Deza, M. et al. (eds.) Combinatorics and Computer Science, Lecture Notes in Computer Science, Vol. 1120, pp. 91–121. Springer, Heidelberg (1996)

Gilboa, I., Zemel, E.: Nash and correlated equilibria: some complexity considerations. Games Econom. Behav. 1, 80–93 (1989)

Jansen, M. J. M.: Maximal Nash subsets for bimatrix games. Naval Res. Logist. Quart. 28, 147–152 (1981)

Kohlberg, E., Mertens, J.-F.: On the strategic stability of equilibria. Econometrica 54, 1003–1037 (1986)

Kuhn, H. W.: An algorithm for equilibrium points in bimatrix games. Proc. Nat. Acad. Sci. U.S.A. 47, 1657–1662 (1961)

Lemke, C. E., Howson, J. T., Jr.: Equilibrium points of bimatrix games. J. Soc. Indust. Appl. Math. 12, 413–423 (1964)

Mangasarian, O. L.: Equilibrium points in bimatrix games. J. Soc. Indust. Appl. Math. 12, 778–780. (1964)

McKelvey, R. D., McLennan, A. M., Turocy, T. L.: Gambit: Software Tools for Game Theory, Version 0.2007.01.30 `http://econweb.tamu.edu/gambit` (2007)

Millham, C. B.: On Nash subsets of bimatrix games. Naval Res. Logist. Quart. 21, 307–317 (1974)

Motzkin, T. S., Raiffa, H., Thompson, G. L., Thrall, R. M.: The double description method. In: Kuhn, H. W., Tucker, A. W. (eds.) Contributions to the Theory of Games II, pp. 51–73. Annals of Mathematics Studies 28, Princeton Univ. Press, Princeton (1953)

Nash, J. F.: Non-cooperative games. Ann. Math. 54, 286–295 (1951)

Rosenberg, G. D.: Enumeration of All Extreme Equilibria of Bimatrix Games with Integer Pivoting and Improved Degeneracy Check. CDAM Research Report LSE-CDAM-2005-18, London School of Economics (2005)

Savani, R.: Solve a bimatrix game. Interactive website. `http://banach.lse.ac.uk/form.html` (2005)

Shapley, L.S.: A note on the Lemke–Howson algorithm. Mathematical Programming Study 1: Pivoting and Extensions, pp. 175–189 (1974)

van den Elzen, A. H. and Talman, A. J. J.: A procedure for finding Nash equilibria in bi-matrix games. Mathematical Methods of Operations Research 35, 27–43 (1991)

von Schemde, A., von Stengel, B.: Strategic characterization of the index of an equilibrium. In: Monien, B., Schroeder, U.-P. (eds.) Symposium on Algorithmic Game Theory (SAGT) 2008, Lecture Notes in Computer Science, Vol. 4997, pp. 242–254. Springer-Verlag, Berlin (2008)

von Stengel, B.: Efficient computation of behavior strategies. Games Econom. Behav. 14, 220–246 (1996)

von Stengel, B.: Improved equilibrium enumeration for bimatrix games. Extended Abstract, International Conference on Operations Research, ETH Zurich, Aug 31–Sept 3. `http://www.maths.lse.ac.uk/Personal/stengel/TEXTE/complement-enum.pdf` (1998)

von Stengel, B.: New maximal numbers of equilibria in bimatrix games. Discrete and Computational Geometry 21, 557–568 (1999)

von Stengel, B.: Computing equilibria for two-person games. In: Aumann, R. J., Hart, S. (eds.) Handbook of Game Theory, Vol. 3, pp. 1723–1759. North-Holland, Amsterdam (2002)

Vorob'ev, N. N.: Equilibrium points in bimatrix games. Theory of Prob. Appl. 3, 297–309 (1958)

Winkels, H.-M.: An algorithm to determine all equilibrium points of a bimatrix game. In: Moeschlin, O., Pallaschke, D. (eds.) Game Theory and Related Topics, pp. 137–148. North-Holland, Amsterdam (1979)