Performance evaluation of a descent algorithm for bi-matrix games

Haralampos Tsaknakis[1], Paul G. Spirakis[1,2], Dimitrios Kanoulas[2]

[1]Research Academic Computer Technology Institute,
N. Kazantzaki, University of Patras Campus, 26500, Rio, Patras, Greece. E-mail:
tsaknak@cti.gr, spirakis@cti.gr
[2]University of Patras, Department of Computer Engineering, 26500, Rio, Patras,
Greece. E-mail: kanoulas@ceid.upatras.gr

Abstract.
In this paper we present an implementation and performance evaluation of a descent algorithm that was proposed in [1] for the computation of approximate Nash equilibria of non-cooperative bi-matrix games. This algorithm, which achieves the best polynomially computable ε- approximate equilibria till now, is applied here to several problem instances designed so as to avoid the existence of easy solutions. Its performance is analyzed in terms of quality of approximation and speed of convergence. The results demonstrate significantly better performance than the theoretical worst case bounds, both for the quality of approximation and for the speed of convergence. This motivates further investigation into the intrinsic characteristics of descent algorithms applied to bi-matrix games. We discuss these issues and provide some insights about possible variations and extensions of the algorithmic concept that could lead to further understanding of the complexity of computing equilibria. We also prove here a new significantly better bound on the number of loops required for convergence of the descent algorithm.

1. Introduction and definitions

The problem of computing approximate Nash equilibria in polynomial time has received attention due to the intractability results for the problem of finding exact Nash equilibria ([4]), even for two-player games. The two-player game is intractable in the sense that it is PPAD-complete. Simple polynomial time algorithms have been presented in the past two years for finding ε- approximate equilibria for $\varepsilon = 3/4$ and $\varepsilon = 1/2$ ([2], [6]). Furthermore, some more complicated polynomial time algorithms have recently been presented, achieving approximations $\varepsilon = 0.38$, $\varepsilon = 0.36$ and $\varepsilon = 0.3393$ ([5], [7], [1]). The last result achieves the best constant approximation reported in the literature so far and is based on an optimization approach applied to the problem of finding approximate equilibria for bi-matrix games. In particular, it is based on a descent algorithm aiming at minimizing the value of a regret function representing the approximation of equilibria. The present work is concerned with the implementation, convergence analysis and evaluation of the performance of this approach for a variety of bi-matrix game examples.

Let R, C denote the $m\,X\,n$ row and column players' payoff matrices respectively, for m, n any positive integers. We assume, without loss of generality, that both payoff matrices are positively normalized, i.e. the maximum entry is 1 and the minimum entry is 0 for each matrix.
Let us denote by e_k the k- dimensional column vector having all its entries equal to 1 (for positive integer k) and let $\Delta_k = \left\{u : u \in R^k, u \geq 0, e_k{}^T u = 1\right\}$ denote the k- dimensional standard simplex (superscript T denotes transpose).

We will also need the following definitions:
For any vector $u \in R^k$:

$supp\,(u) = \{i \in (1,k) : u_i \neq 0\}$, the support index subset of $u \in R^k$

$supp\_max\,(u) = \{i \in (1,k) : u_i \geq u_j \;\forall j \in (1,k)\}$, the index subset where all entries are equal to the maximum entry of $u \in R^k$.

$max\,(u) = \{u_i :\; u_i \geq u_j \;\; \forall j \in (1,k)\}$, the value of the maximum entry of the vector.

$\max_S(u) = \{u_i,\ i \in S : u_i \geq u_j \ \forall\, j \in S\}$, the value of the maximum entry of the vector within an index subset $S \subset (1,k)$.

The problem of finding an ε-approximate Nash equilibrium of the game (R,C), for some $\varepsilon > 0$, is to compute a pair of strategies $\hat{x} \in \Delta_m, \hat{y} \in \Delta_n$ such that the following relationships hold:

$$x^T R \hat{y} \leq \hat{x}^T R \hat{y} + \varepsilon \ \forall\, x \in \Delta_m \quad \text{and} \quad \hat{x}^T C y \leq \hat{x}^T C \hat{y} + \varepsilon \ \forall\, y \in \Delta_n$$

Following the analysis in [1], we define the following function, mapping $\Delta_m\ X\ \Delta_n$ into $[0,1]$:

$$f(x,y) = \max\{f_R(x,y), f_C(x,y)\}$$

where: $f_R(x,y) = \max(Ry) - x^T R y$ and $f_C(x,y) = \max(C^T x) - x^T C y$.

For any pair of strategies $(x,y) \in \Delta_m\ X\ \Delta_n$, this function measures the maximum distance between the players' payoffs (achieved by that pair of strategies) and their respective best response payoffs. We call this a “regret” function.

The descent approach attempts to minimize the regret function through an iterative process moving along feasible descent directions in the space of strategies for both players simultaneously. The descent directions are computed by solving linear programs. A descent algorithm produces a monotonically decreasing regret function which always terminates with a ***stationary point*** in the space of strategies. It was proven in [1] that at any stationary point we have an ε - approximate Nash equilibrium, where ε cannot exceed the number 0.3393 (for positively normalized payoff matrices).

2. Algorithm description

In this section we give the main points underlying the descent algorithm as derived in [1].
From any given point $(x,y) \in \Delta_m\ X\ \Delta_n$, we consider motions along feasible directions, where, a feasible direction is specified by another pair of strategies $(x', y') \in \Delta_m\ X\ \Delta_n$ and has the following form:

$$(1-\varepsilon)\,[x,\ y] + \varepsilon\,[x',\ y'],\ where,\ 0 \leq \varepsilon \leq 1,\ (x', y') \in \Delta_m\ X\ \Delta_n$$

(The vectors in brackets denote $(m+n)$ - dimensional vectors).

The computation of a feasible direction $(x', y') \in \Delta_m\ X\ \Delta_n$ that is also a descent direction for the regret function at a point (x,y) involves the solution of an appropriate linear program which is formulated as follows, for two distinct cases (A) and (B):

(A) If $f_R(x,y) \neq f_C(x,y)$, then:

(a1) If $f_R(x,y) > f_C(x,y)$, keep y fixed and solve the following LP:

$$\min_{x' \in \Delta_m} \{\max(Ry) - x'^T R y\}$$
$$s.t.\ \max(C^T x') - x'^T C y \leq \max(Ry) - x'^T R y$$

A minimizer x' defines a descent direction (x', y)

(a2) If $f_R(x,y) < f_C(x,y)$, keep x fixed and solve the following LP:

$$\min_{y' \in \Delta_n} \{\max(C^T x) - x^T C y'\}$$
$$s.t.\ \max(Ry') - x^T R y' \leq \max(C^T x) - x^T C y'$$

A minimizer y', defines a descent direction (x, y')

(B) If $f_R(x,y) = f_C(x,y)$, then solve the following $(m+n)$ - dimensional LP in mini-max form:

$$\min_{x',y'} \max_{w,z,\rho} \left[\rho\, w^T, (1-\rho)\, z^T \right] G(x,y) \begin{bmatrix} y' \\ x' \end{bmatrix}$$

where:

(i) the maximum is taken with respect to dual variables w, z, ρ such that:
$w \in \Delta_m$, $supp(w) \subset S_R(y) \triangleq supp\_\max(Ry)$,
$z \in \Delta_n$, $supp(z) \subset S_C(x) \triangleq supp\_\max(C^T x)$,
$\rho \in [0,1]$

(ii) the minimum is taken with respect to $(x', y') \in \Delta_m \, X \, \Delta_n$, and

(iii) the matrix $G(x,y)$ is the following $(m+n)\, X\, (m+n)$ matrix:

$$G(x,y) = \begin{bmatrix} R - e_m x^T R & -e_m y^T R^T + e_m e_m^T x^T R y \\ -e_n x^T C + e_n e_n^T x^T C y & C^T - e_n y^T C^T \end{bmatrix}$$

The descent direction is specified by a minimizer (x', y') of the above problem.

It should be pointed out that in case (A) a solution of the corresponding LP always leads to a point where the values of the two components of the regret function are equal, i.e. $f_R(x',y) = f_C(x',y)$ (or $f_R(x,y') = f_C(x,y')$) and the regret function at this point is strictly smaller than the previous one. We make this statement precise in the following Lemma:

Lemma 1. At a given pair of strategies (x,y), if $f_R(x,y) > f_C(x,y)$ and x' is a minimizer of the LP in (a1) above, then:

$$f(x',y) = f_R(x',y) = f_C(x',y) \le \frac{f_R(x,y)}{1 + f_R(x,y) - f_C(x,y)}$$

(A similar statement holds if $f_R(x,y) < f_C(x,y)$ as in case (a2) by interchanging the roles of the row and column players)

Proof. Since y is fixed, the equality of the two regrets (for the row and column players) follows from the fact that at an optimal solution of the LP in (a1) above, not all constraints can be inactive since the upper bound of these constraints (which is the regret of the row player and the objective function to be minimized) can always be made equal to 0 by a choice of a best response strategy x', i.e. such that $supp(x') \subset supp\_\max(Ry)$.
Now, choose any $x_1 \in \Delta_m$ such that: $supp(x_1) \subset supp\_\max(Ry)$. Consider the following function:

$$g(\varepsilon) = f_C\left((1-\varepsilon)x + \varepsilon x_1, y\right) - f_R\left((1-\varepsilon)x + \varepsilon x_1, y)\right) \quad \text{for } \varepsilon : 0 \le \varepsilon \le 1$$

It is easy to verify that $g(\varepsilon)$ is continuous and convex in ε and that it satisfies:

$$g(0) = -\left(f_R(x,y) - f_C(x,y)\right) < 0$$
$$0 \le g(1) = f_C(x_1, y) \le 1$$

So, there is an $\varepsilon' > 0$ such that $g(\varepsilon') = 0$. By convexity, we should have $g(\varepsilon') \le (1-\varepsilon')\, g(0) + \varepsilon'\, g(1)$ which, in view of the above relationships, implies:

$$(1-\varepsilon') \le \frac{1}{1 + f_R(x,y) - f_C(x,y)}$$

Furthermore, choosing $x' = (1-\varepsilon')x + \varepsilon' x_1$, we have:

$$f_C(x',y) = f_R(x',y) = (1-\varepsilon')\, f_R(x,y)$$

Finally, the assertion of the Lemma follows from the last two relationships.
QED

From the above Lemma it can also be observed that if we take any arbitrary strategy y for the column player and a strategy x for the row player that is a best response to it, then, the resulting value of the regret function in the first step will always be $\le 1/2$. So, for the computations in the main step (B) of the algorithm we can always start at a point with a regret function value that is $\leq 1/2$ and for which the two components of the regret are equal, i.e. $f_R(x,y) = f_C(x,y)$.

We call the overall process of computing a descent direction a ***"Descent Direction"*** step.

After obtaining a descent direction, say (x',y'), from any given point (x,y), we move on to compute the minimum of the function $f\left(x+\varepsilon(x'-x), y+\varepsilon(y'-y)\right)$ with respect to the scalar parameter ε, producing thus a new pair of strategies with smaller regret. We call this process a ***"Line Search"*** step.

For the line search computations, we exploit the fact that the above function is piece-wise quadratic with respect to ε (for any x,y,x',y') and the total number of switches from one quadratic to another is less than $m+n$. For the purposes of the implementation described here, we have considered stepsizes equal to the switching point ε^* that is closest to 0. We provide explicit estimates of this stepsize in the proof of Lemma 2 below.

Lemma 2. Let (x,y) be a pair of strategies such that $f_R(x,y) = f_C(x,y)$ and let $(x',y') \in \Delta_m \, X \, \Delta_n$ be a solution of the LP defined in (B) above. Also, let $V(x,y)$ denote the optimal value of the LP. Then, if $V(x,y) - f(x,y) < 0$, there is an $\varepsilon^* > 0$ such that $f\left(x+\varepsilon(x'-x), y+\varepsilon(y'-y)\right) - f(x,y) < 0$ for all $\varepsilon \in (0,\varepsilon^*]$.

Proof. The proof is based on the construction of the above difference for any $\varepsilon : 0 \leq \varepsilon \leq 1$ and on using the properties of a solution of the mini-max LP in (B). We also provide a new explicit bound on the decrease of the function f at each step of the algorithm (compared to the one given in [1]), which is useful for a more refined analysis of its convergence properties.
At first, it can be easily verified (by setting $x' = x,\ y' = y$) that we always have:

$$V(x,y) - f(x,y) \leq 0$$

Let $\overline{S_R}(y)$ and $\overline{S_C}(x)$ be the complements of the index sets $S_R(y)$ and $S_C(x)$ with respect to index sets $(1,m)$ and $(1,n)$ respectively (we have defined $S_R(y) = supp\_\max(Ry)$ and $S_C(x) = supp\_\max(C^T x)$).
Let $\varepsilon^* = \min\{\varepsilon_1^*, \varepsilon_2^*, 1\}$, where:

$$\varepsilon_1^* = \min_i \left\{ \frac{\max(Ry) - (Ry)_i}{\max(Ry) - (Ry)_i + (Ry')_i - \max\limits_{S_R(y)}(Ry')} \right\}$$

(the minimum is taken over $\left\{ i :\ i \in \overline{S_R}(y) \ \wedge\ (Ry')_i - \max\limits_{S_R(y)}(Ry') \geq 0 \right\}$)

and
$$\varepsilon_2^* = \min_j \left\{ \frac{\max(C^T x) - (C^T x)_j}{\max(C^T x) - (C^T x)_j + (C^T x')_j - \max\limits_{S_C(x)}(C^T x')} \right\}$$

(the minimum is taken over $\left\{ j : j \in \overline{S_C}(x) \ \wedge\ (C^T x')_j - \max\limits_{S_C(x)}(C^T x') \geq 0 \right\}$)

Also, let

$$\Delta = \min\left\{ \min_{i \in \overline{S_R}(y)} \{\max(Ry) - (Ry)_i\},\ \min_{j \in \overline{S_C}(x)} \{\max(C^T x) - (C^T x)_j\} \right\}$$

It is clear that we always have $\Delta > 0$ and that $\varepsilon^* \geq \dfrac{\Delta}{1+\Delta}$.
Following the derivations in [1] and after several manipulations that were performed taking into account that the algorithm minimizes the difference with respect to $\varepsilon \in (0,\varepsilon^*]$ at every step, we finally get the following relationship for the new value of f (we drop the indices for notational simplicity) that is obtained at every step:

$$f_{new} - f \le -\min\left\{ \frac{|V-f|^2}{4\,(1-V)}, \left(\frac{\Delta}{1+\Delta}\right)^2 f + \frac{\Delta}{(1+\Delta)^2} |V-f| \right\}$$

From the above relationship it is clear that there is always a decrease of the value of f unless $V - f = 0$. In the latter case we have a stationary point. We notice that a stationary point always exists as a limit point of the sequence of values of the function f produced by the descent algorithm. Such a sequence has always a limit since it is monotonically decreasing and bounded from below by 0.
QED

The descent algorithm essentially consists of an iterative loop containing the two basic steps as defined above. The execution of these steps continue until the stationarity condition is satisfied. The stationarity condition is equivalent to the condition that no descent direction exists and is expressed by the relationship $V(x,y) - f(x,y) = 0$, where, $V(x,y)$ is the value of the solution of the mini-max problem as defined in the above Lemma 2. We call the common value of $V(x,y)$ and $f(x,y)$ at stationarity the ***value*** of the stationary point.

Schematically, the basic steps of the algorithm can be described as follows:

0. Start at an arbitrary pair of strategies (x_0, y_0)
1. Apply ***Descent Direction***
2. Apply ***Line Search***
3. Stop if termination condition is satisfied. If not, return to step 1.

A flow chart of the algorithm with more detail is presented in Appendix A.

3. Convergence rate

We provide an improved complexity result of the descent algorithm compared to the one presented in [1]. This is based on an estimate of a tighter bound on the number of loops required for convergence to a stationary point.

Fix $\delta > 0$ and let (x, y) be the current pair of strategies obtained during the descent procedure for which $f_R(x,y) = f_C(x,y)$. We define the following index sets:

$$S_R(y,\delta) = \{ i \in (1,m) : (R\,y)_i \ge \max(R\,y) - \delta \}$$

$$S_C(x,\delta) = \{ j \in (1,n) : (C^T x)_j \ge \max(C^T x) - \delta \}$$

Using the above index sets throughout the algorithm (for the computation of descent directions and all quantities associated with them), we define a δ - stationary point as a point that satisfies the relationship $V(x,y) - f(x,y) \ge -\delta$. Let $\overline{S_R}(y,\delta)$ and $\overline{S_C}(x,\delta)$ be the complements of these sets with respect to index sets $(1,m)$ and $(1,n)$ respectively. Then, we have $\max(Ry) - (Ry)_i > \delta$ for all $i \in \overline{S_R}(y,\delta)$ and $\max(C^T x) - (C^T x)_j > \delta$ for all $j \in \overline{S_C}(x,\delta)$. Therefore $\delta < \Delta$, where Δ is as defined in the proof of Lemma 2 above.
The convergence rate result is expressed in Lemma 3 below.

<u>Lemma 3.</u> The descent algorithm converges to a δ - stationary point in $O\big((1/\delta)\log(1/\delta)\big)$ loops.

<u>Proof.</u> Let us denote by b the value of f at a limit point of the descent algorithm. We should have $V \le b \le f$ at every loop of the algorithm (b cannot be more than 0.3393 and the initial value of f is $\le 1/2$). Using the last expression in the proof of Lemma 2 for the new value of f as a function of the previous value, it can be verified by direct calculation, that at every loop we obtain either a descent of the form:

$$f_{new} - b \le (f - b) - \frac{(f-b)^2}{4(1-b)}$$

or, a descent of the form:

$$f_{new} - b \le \left(1 - \frac{\Delta}{1+\Delta}\right)(f-b) - \frac{\Delta^2}{(1+\Delta)^2} b$$

So we have two types of positive sequences $\{s_k, k=1,2,...\}$ starting from values $<1/2$ and converging to 0 according to the relationships $s_{k+1} \le s_k - \frac{s_k^{\ 2}}{4(1-b)}$ and $s_{k+1} \le \left(1 - \frac{\Delta}{1+\Delta}\right) s_k - \frac{\Delta^2}{(1+\Delta)^2} b$

It can be verified that the first sequence requires at most $O(1/\delta)$ steps to reach a point s_k such that $s_k \le \delta$ and the second sequence requires at most $O\left((1/\Delta)\log(1/\delta)\right)$ steps to reach such a point. The result follows from these facts and from $\delta < \Delta$.

QED

4. Evaluation scenaria

For the evaluation of performance we have applied the algorithm to a large number of bimatrix game problem instances with sizes ranging from 10X10 to 100X100. The games that were generated were basically of two kinds: Those consisting of randomly generated real-valued payoff matrices R, C and those consisting of 0-1 (win-loose) matrices. The example matrices were generated so as to avoid the existence of easy solutions such as pure strategy equilibria, 2x2 equilibria and uniform distribution equilibria. Furthermore, the instances were selected so as to avoid being close to constant sum games (for which every stationary point is a Nash equilibrium). All example matrices were positively normalized before running the algorithm.

The 0-1 payoff matrices R, C that were generated were of two kinds:

(a) Randomly generated matrices, under the constraint that no small support equilibria exist (pure and 2X2) and the games are not close to constant sum games.

(b) Specifically designed matrices satisfying the above constraints and also containing arbitrary cycles of consecutively interchanging assignments of ordered pairs (0,1) and (1,0) to the entries of $(R_{i,j}, C_{i,j})$ horizontally and perpendicularly (ref. [3]). In particular, according to this pattern, we start from an arbitrary entry (i, j) and assign either (0,1) or (1,0) to it. Then, keeping i (or j) fixed, pick an arbitrary column (or row) and assign the opposite pair to it, i.e. (1,0) or (0,1). The next step is to move in the perpendicular direction, i.e. to keep j (or i) fixed and continue this process an arbitrary number of times to finally close the cycle. We can have several such cycles in the specification of the matrices R, C (a specific numerical example of how this is done is presented in the sequel). It is expected that the presence of such cycles makes the problem of finding equilibria more difficult. Indeed, uniform distributions of either small support or large support (close to the dimension of the matrices) are not likely to be close to equilibria under such patterns of payoff assignments.

For each example, we used several different starting points in order to check the response of the algorithm to the variation of the starting distributions. In particular, we used the uniform distributions, as one starting point, but also several arbitrarily chosen pure strategies for both players.

The algorithm was implemented in C and the CPLEX Program was used as an LP solver. The parameter δ was fixed to $\delta = 0.001$ for all instances.

In the figures below we present some representative results of the algorithm applied to 100X100 examples drawn from three different categories of instances as described above: Random real payoff matrices, 0-1 matrices with constraints, and 0-1 matrices with cycles. In particular, we show graphs of how fast the value of the regret function f decreases with the number of loops until convergence to a stationary point. For each figure, several such graphs depicting the evolution of the regret function are shown, each representing the algorithm's response to different starting points. In order to show as much as possible detail of the graphs near convergence, the starting values of the regret function before the algorithm are not shown explicitly in the graphs (they are only mentioned in the legend) due to the fact that they are usually large and create scaling problems.

It is pointed out that the results presented below are typical of the results we got after extensive experimentation with a wide range of instances of all categories and of sizes ranging from 10X10 to 100X100.

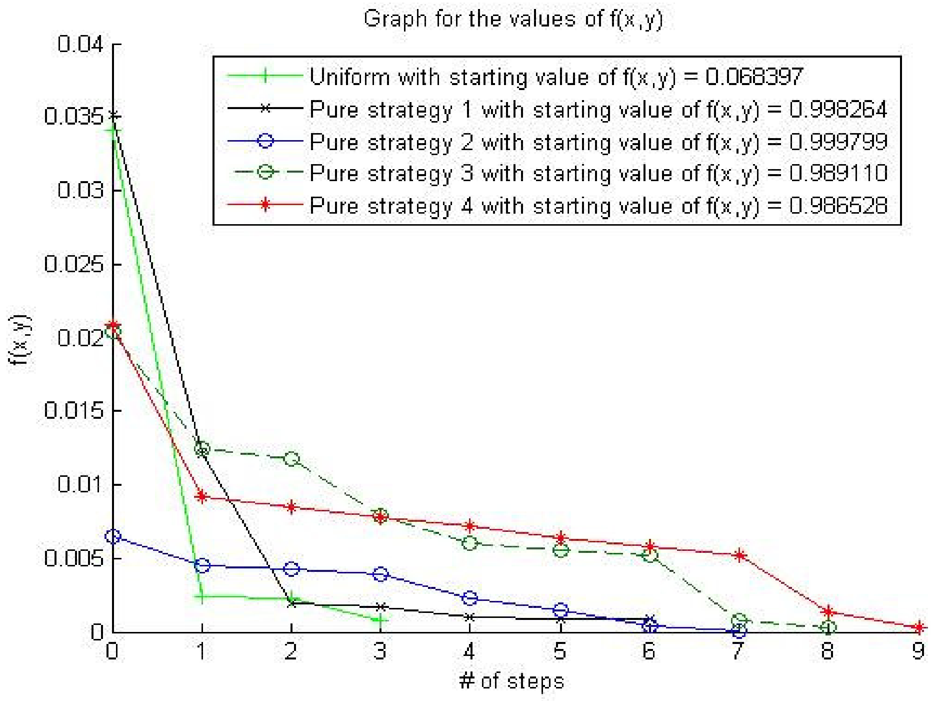


Figure 1. Regret function f vs. number of loops for random real matrices

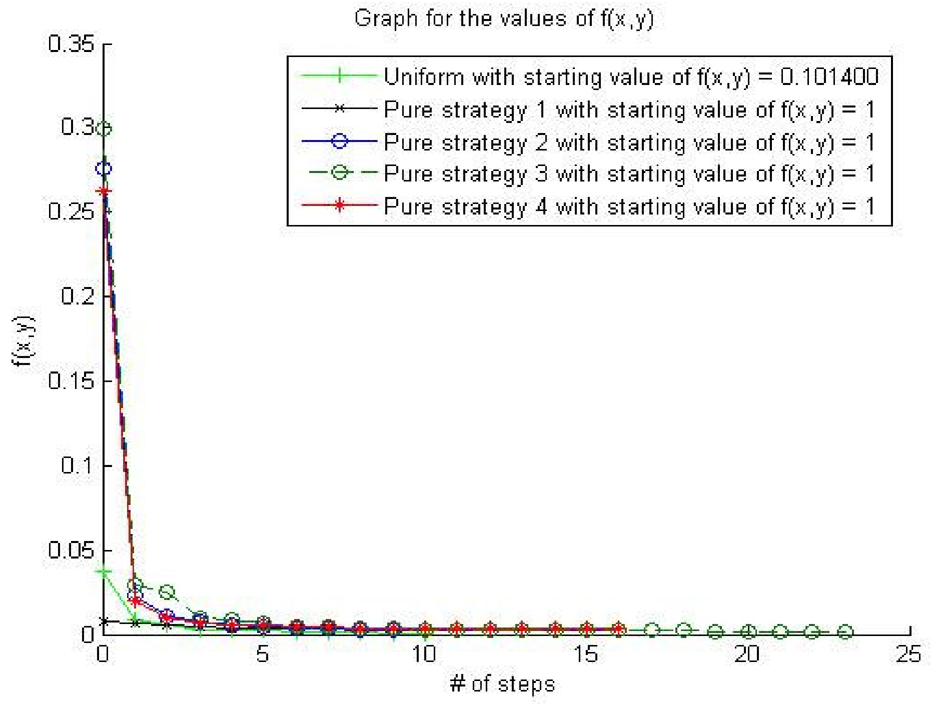


Figure 2. Regret function f vs. number of loops for 0-1 random matrices with constraints

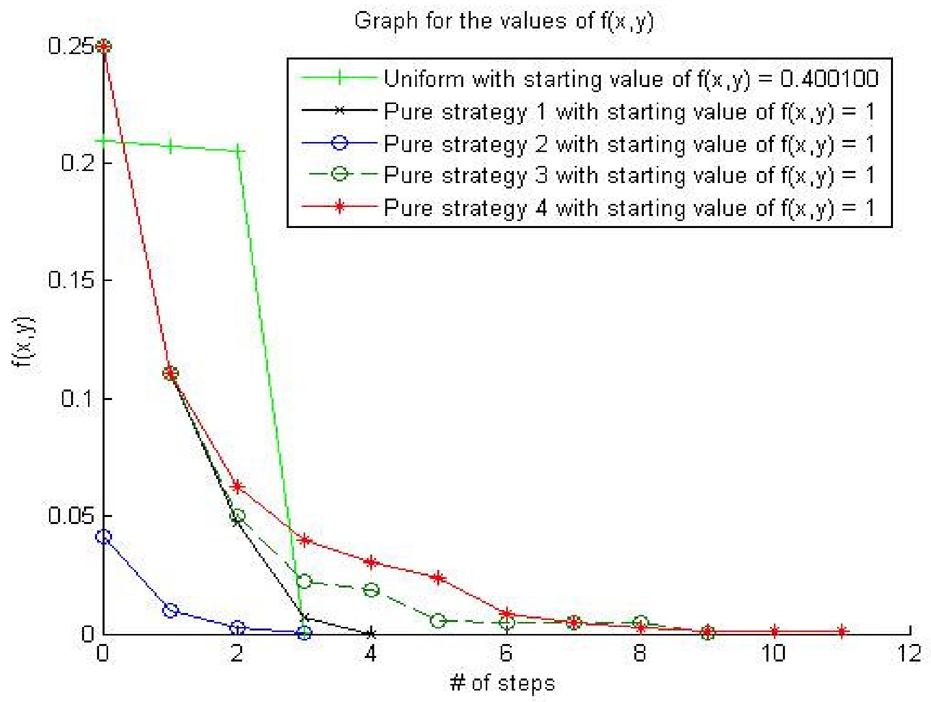


Figure 3. Regret function f vs. number of loops for 0-1 matrices with cycles

For the first category (random real payoff matrices) the resulting stationary points provided exact Nash equilibria (approximation less than the parameter $\delta = 0.001$) from all starting points.

For the second category (0-1 matrices with constraints) the worst approximation to a Nash equilibrium that was achieved was 0.003, and for one starting point (the uniform distributions) an exact equilibrium was obtained.

Finally, for the third category (0-1 matrices with cycles) we obtained exact Nash equilibria from all starting points.

The number of loops required to converge differ across starting points and categories but typically it was between 4 and 20 with a median value of 10.

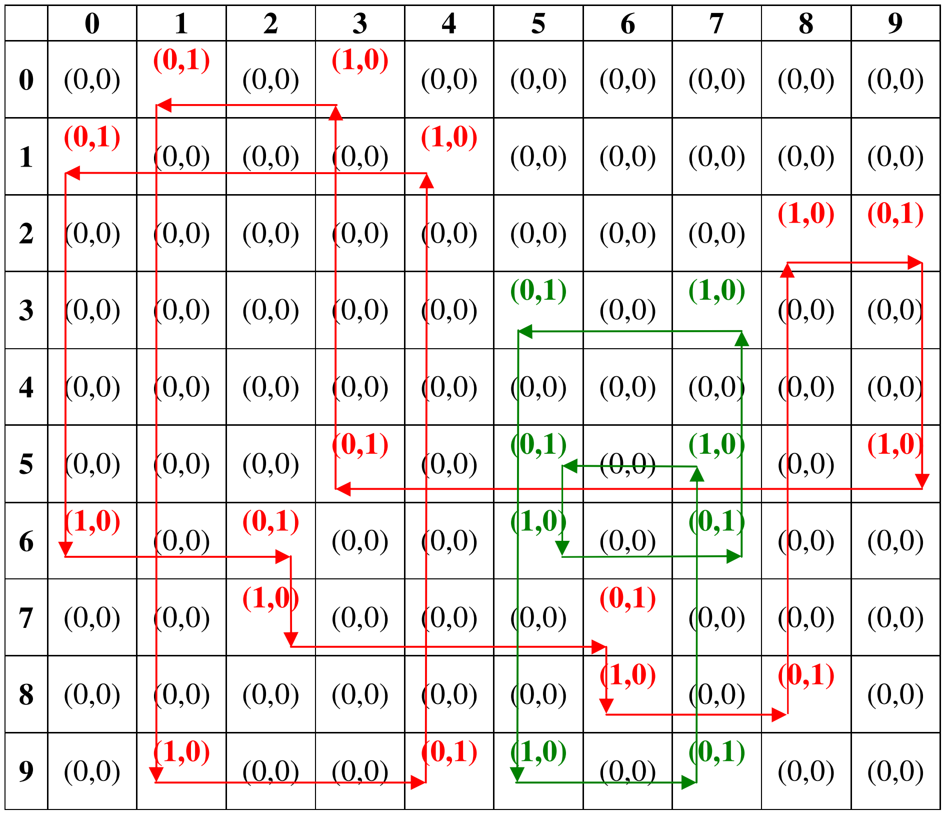
The supports of the resulting approximate equilibria distributions were relatively large, typically between $n/3$ an $n/2$ (where n is the size of the game).

A small example

We present in detail a 10X10 example from the third category of instances (0-1 matrices with cycles), where the entries of the two payoff matrices R, C are shown as ordered pairs in the same box. We have run the algorithm using different starting points including the uniform pair of strategies, as well as four different arbitrary pairs of pure strategies. The results are shown below, including the approximate Nash equilibria obtained at convergence and the respective approximations achieved for each starting point.

It is pointed out that the worst approximation to a Nash equilibrium that was obtained is 0.015, which occurred for one case with an arbitrary pair of pure strategies as starting point. Also, a 0.01 approximation was obtained with the uniform distributions as starting point.

The 0.015 approximation is the worst approximation that we obtained from all the experiments that we performed so far.



	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	(0,0)	**(0,1)**	(0,0)	**(1,0)**	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)
1	**(0,1)**	(0,0)	(0,0)	(0,0)	**(1,0)**	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)
2	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	**(1,0)**	**(0,1)**
3	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	**(0,1)**	(0,0)	**(1,0)**	(0,0)	(0,0)
4	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)
5	(0,0)	(0,0)	(0,0)	**(0,1)**	(0,0)	**(0,1)**	(0,0)	**(1,0)**	(0,0)	**(1,0)**
6	**(1,0)**	(0,0)	**(0,1)**	(0,0)	(0,0)	**(1,0)**	(0,0)	**(0,1)**	(0,0)	(0,0)
7	(0,0)	(0,0)	**(1,0)**	(0,0)	(0,0)	(0,0)	**(0,1)**	(0,0)	(0,0)	(0,0)
8	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	**(1,0)**	(0,0)	**(0,1)**	(0,0)
9	(0,0)	**(1,0)**	(0,0)	(0,0)	**(0,1)**	**(1,0)**	(0,0)	**(0,1)**	(0,0)	(0,0)

Figure 4. Payoff matrices R, C for 10X10 example with cycles

1. Starting point: Uniform strategies
   Solution at convergence:
   x = [2/13, 2/13, 2/13, 0, 0, 2/13, 0, 1/13, 2/13, 2/13]
   y = [1/7, 1/7, 0, 1/7, 1/7, 0, 1/7, 1/7, 1/7, 0]
   Approximation: 0.01 – approximate Nash equilibrium

2. Starting point: Pure strategies, Row index = 0, Column index = 0
   Solution at convergence:
   x = [1/4, 1/4, 0, 0, 0, 1/4, 0, 0, 0, 1/4]
   y = [0, 1/4, 0, 1/4, 1/4, 0, 0, 1/4, 0, 0]
   Approximation: Exact Nash equilibrium

3. Starting point: Pure strategies, Row index = 3, Column index = 0
   Solution at convergence:
   x = [0, 0, 1/5, 1/5, 0, 0, 1/5, 1/5, 1/5, 0]
   y = [0, 0, 1/5, 0, 0, 1/5, 1/5, 1/5, 1/5, 0]
   Approximation: Exact Nash equilibrium

4. Starting point: Pure strategies, Row index = 5, Column index = 0
   Solution at convergence:
   x = [1/7, 1/7, 1/7, 0, 0, 1/7, 1/7, 1/7, 1/7, 0]
   y = [2/15, 2/15, 2/15, 2/15, 1/15, 0, 2/15, 2/15, 2/15, 0]
   Approximation: 0.009 – approximate Nash equilibrium

5. Starting point: Pure strategies, Row index = 7, Column index = 0
   Solution at convergence:
   x = [0, 1/6, 1/6, 0, 0, 1/6, 1/6, 1/6, 1/6, 0]
   y = [2/11, 0, 2/11, 0, 1/11, 0, 2/11, 2/11, 2/11, 0]
   Approximation: 0.015 – approximate Nash equilibrium

For the above example, the graphs in the figure below show the decrease of the regret function with the number of loops of the descent algorithm for each starting point.

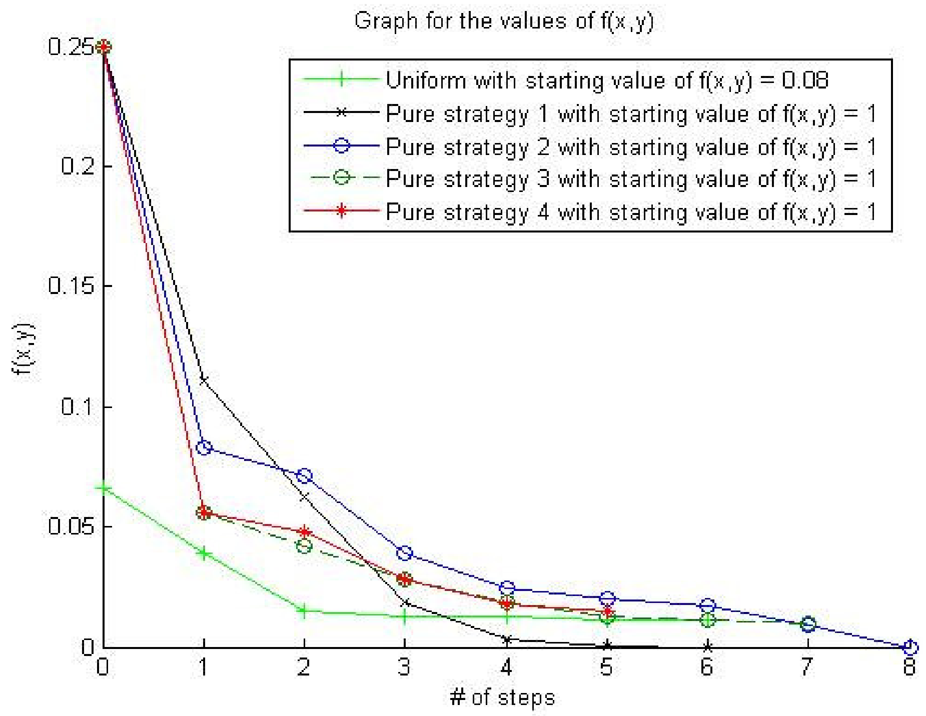


Figure 5. Regret function f vs. number of loops for 10X10 example

5. Evaluation of the results

Two basic conclusions for the performance of the descent algorithm can be derived based on the experimental results, in terms of approximation quality and convergence rate:

(a) Quality of approximation
The approximations to Nash equilibria appear in all experimental cases to be much better than the theoretical worst case approximation (0.3393). The worst approximation was 0.015 across all experimental instances. In fact, for most instances the approximation was of the order of the precision parameter δ. The reduction factor of the regret function f from the start to the end appears to be much larger than the one dictated by the theoretical worst case approximation. The curves showing the reduction of the regret function f with respect to the number of loops appear to exhibit a more or less similar pattern across all instances of all categories of experiments. Furthermore, for instances with relatively large approximation (i.e. large values of f at stationarity but still less than 0.015), it was sufficient to choose a different starting point (for example, an arbitrary pure strategy pair) for the algorithm to converge to an exact equilibrium (for the same instance).

Overall, it appears that if one moves along paths determined by the descent algorithm, it is very likely to hit a Nash equilibrium (or a stationary point close to a Nash equilibrium) along the way.

These experimental results indicate that the stationary points of bi-matrix games are rather "unstable" to the operation of the descent process and that this "instability" tends to increase rapidly with their value f. It is also possible that the stationary points with larger values are less probable (in fact significantly less probable) than the ones with smaller values. It is conjectured that with an appropriate definitions of "stability" of a stationary point, it may be possible to formulate a rigorous approach that could potentially lead to better approximation guarantees than the one currently available, including the possibility of obtaining a PTAS for the equilibrium problem. More specifically, it is conjectured that a modification of the descent algorithm to include restarts from new strategies obtained by small perturbations around a stationary point, could provide an effective way to bypass stationary points with high values.

Significant further insight into the problem will certainly be achieved if we can find harder instances. In this respect, it is worth investigating the existence and construction of instances for which the descent process could be more or less easily "trapped" into relatively high stationary points.

(b) Convergence rate
The number of loops required for convergence to a δ- stationary point was found in all experiments to be much smaller than the worst case bound theoretically predicted in [1]. This motivated a closer look into the convergence properties of the descent algorithm which resulted in a new bound of the order of $O\left((1/\delta)\log(1/\delta)\right)$, a significant improvement over the previous $O\left(1/\delta^2\right)$.

However, it appears that a better complexity bound of the descent algorithm is possible, in terms of speed of convergence to a stationary point. A possible way to obtain improved convergence results is to investigate the maximum number of "small" steps that the algorithm can go through as a function of the size of the game. It appears from the experiments that for each "small" step there is an increase of the size of the support (which often occurs in large chunks rather than one at a time), so, the total number of such "small" steps in a row cannot be more than a fraction of n. Also, it was observed that the "large" steps were too often large enough to enforce very fast convergence to a stationary point. Actually, only a small number of large steps were typically needed for convergence.

6. Discussion

The experimental results were surprising, particularly in regard to the quality of approximation to Nash equilibria. It seems that it is quite hard to create hard instances for the descent algorithm. We believe that the results motivate further investigation into the complexity of finding Nash equilibria along the following lines:

(a) Study of the issue of stability of stationary points as a function of their value $f(x, y)$
(b) Investigation of ways to bypass stationary points via small perturbations around them
(c) Creation of hard instances for the descent algorithm, i.e. instances for which it is possible to get stuck to stationary points with large values
(d) Some further investigation of the convergence rate of the algorithm for an improved bound

References

[1] Tsaknakis, H., Spirakis, P. G.: An Optimization Approach for Approximate Nash Equilibria. In: WINE 2007, LNCS 4858, pp. 42-56, 2007, Springer-Verlag, Berlin, Heilderberg (2007).

[2] S. Kontogiannis, P. Panagopoulou and P. Spirakis. Polynomial algorithms for approximating nash equilibria in bimatrix games. In *Proc. of the 2nd W. on Internet and Net Econ. (WINE '06), vol. 4286 of LNCS*, pp. 286-296, Springer, 2006.

[3] S. Kontogiannis, P. G. Spirakis. Efficient algorithms for constant well supported approximate equilibria of bimatrix games. ICALP 2007.

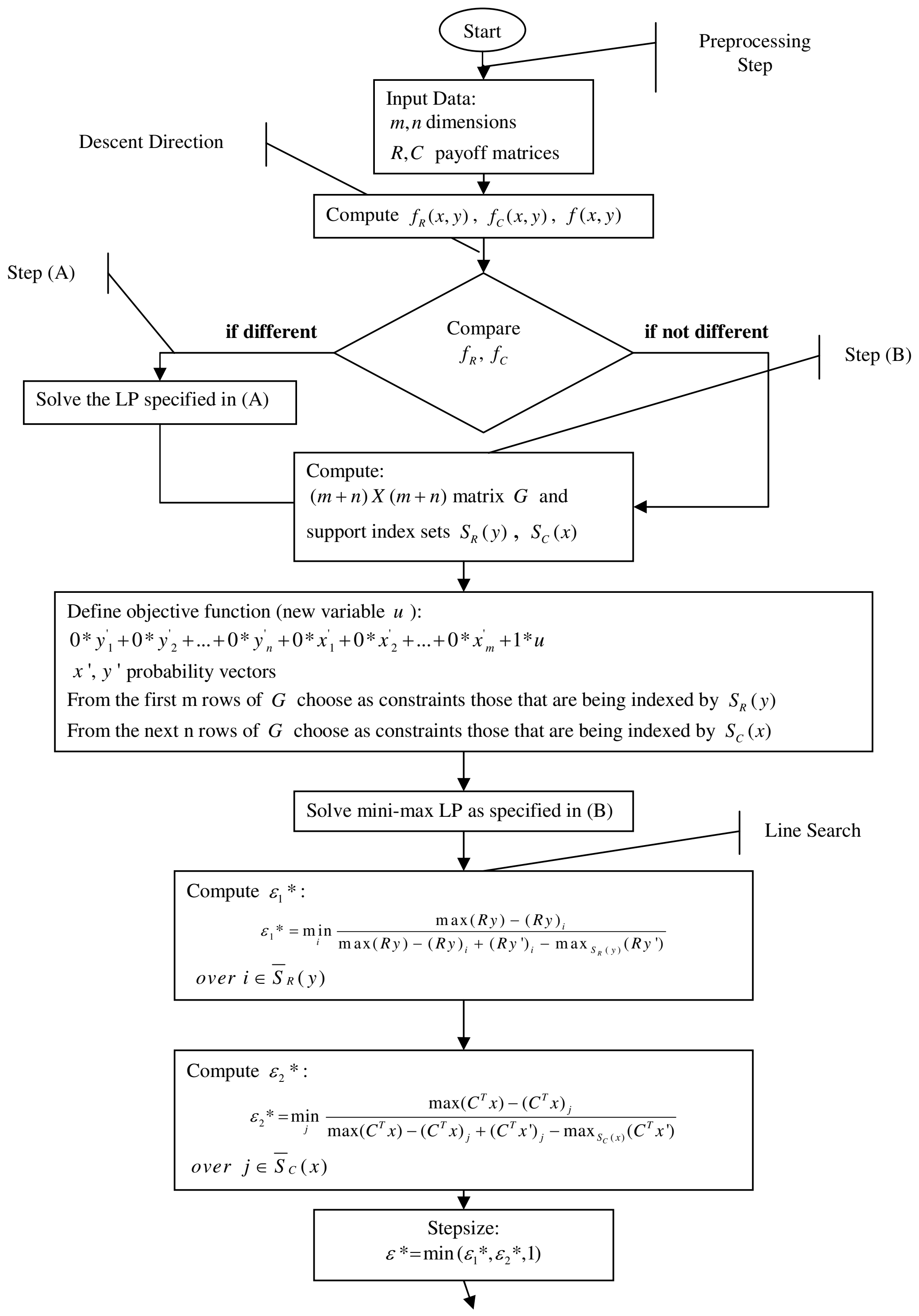
[4] X. Chen and X. Deng. Settling the complexity of 2-player nash equilibrium. In *Proc. of the 47th IEEE Symp. on Found. of Comp. Sci. (FOCS '06)*, pp. 261-272, IEEE Comp. Soc. Press, 2006.

[5] C. Daskalakis, A. Mehta, and C. Papadimitriou. Progress in approximate nash equilibrium. In *Proc. Of the 8th ACM Conf. on Electr. Commerce (EC'07)*, 2007.

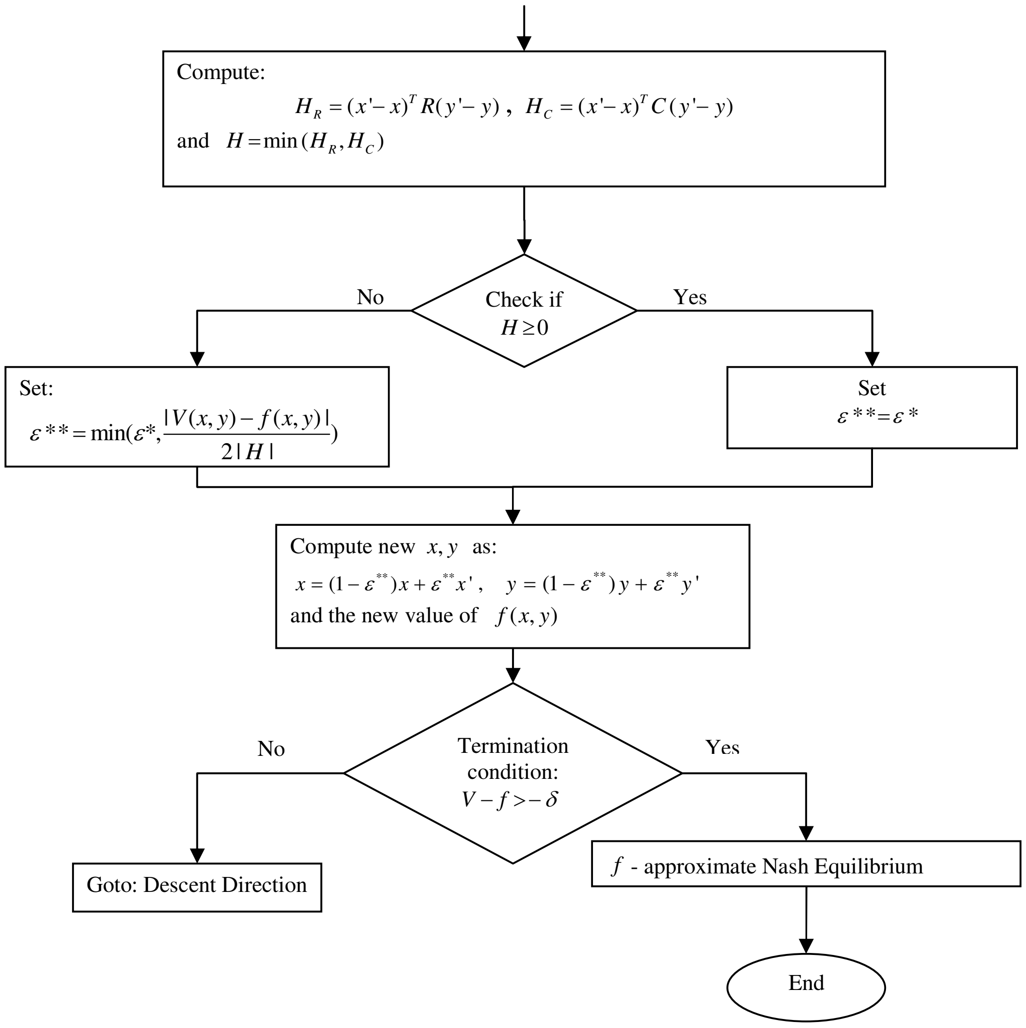
[6] C. Daskalakis, A. Mehta, and C. Papadimitriou. A note on approximate nash equilibria. In *Proc. of the 2nd W. on Internet and Net Econ. (WINE '06), vol. 4286 of LNCS*, pp. 297-306, Springer, 2006.

[7] H. Boss, J. Byrka and E. Markakis. New Algorithms for Approximate Nash Equilibria in Bimatrix Games. In: WINE 2007.

Appendix A: Flow Chart of the descent algorithm



Start
Preprocessing Step
Input Data:
m, n dimensions
R, C payoff matrices
Descent Direction
Compute $f_R(x,y)$, $f_C(x,y)$, $f(x,y)$
Step (A)
if different
Compare f_R, f_C
if not different
Step (B)
Solve the LP specified in (A)
Compute:
$(m+n)\,X\,(m+n)$ matrix G and
support index sets $S_R(y)$, $S_C(x)$
Define objective function (new variable u):
$0*y'_1+0*y'_2+\ldots+0*y'_n+0*x'_1+0*x'_2+\ldots+0*x'_m+1*u$
x', y' probability vectors
From the first m rows of G choose as constraints those that are being indexed by $S_R(y)$
From the next n rows of G choose as constraints those that are being indexed by $S_C(x)$
Solve mini-max LP as specified in (B)
Line Search
Compute ε_1*:
$\varepsilon_1* = \min_i \frac{\max(Ry)-(Ry)_i}{\max(Ry)-(Ry)_i+(Ry')_i-\max_{S_R(y)}(Ry')}$
$over\ i \in \overline{S}_R(y)$
Compute ε_2*:
$\varepsilon_2* = \min_j \frac{\max(C^Tx)-(C^Tx)_j}{\max(C^Tx)-(C^Tx)_j+(C^Tx')_j-\max_{S_C(x)}(C^Tx')}$
$over\ j \in \overline{S}_C(x)$
Stepsize:
$\varepsilon* = \min(\varepsilon_1*, \varepsilon_2*, 1)$



Compute:
$H_R = (x'-x)^T R(y'-y)$, $H_C = (x'-x)^T C(y'-y)$
and $H = \min(H_R, H_C)$
Check if $H \geq 0$
No
Yes
Set: $\varepsilon^{**} = \min(\varepsilon^*, \frac{|V(x,y) - f(x,y)|}{2|H|})$
Set $\varepsilon^{**} = \varepsilon^*$
Compute new x, y as:
$x = (1-\varepsilon^{**})x + \varepsilon^{**}x'$, $y = (1-\varepsilon^{**})y + \varepsilon^{**}y'$
and the new value of $f(x,y)$
Termination condition: $V - f > -\delta$
No
Yes
Goto: Descent Direction
f - approximate Nash Equilibrium
End