# 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 (RACTI), Greece
Email: tsaknak@cti.gr, spirakis@cti.gr
[2] Dept. of Computer Eng. and Informatics, Patras University, Patras, Greece
Email: 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 $\epsilon$-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 $\epsilon$-approximate equilibria for $\epsilon = \frac{3}{4}$ and $\epsilon = \frac{1}{2}$ ([2], [6]). Furthermore, some more complicated polynomial time algorithms have recently been presented, achieving approximations $\epsilon = 0.38$, $\epsilon = 0.36$ and $\epsilon = 0.3393$ ([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. This work is concerned with the implementation and evaluation of the performance of this approach as well as the presentation of a new complexity result on the convergence rate.

Let $R, C$ denote the $m \times n$ 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 $\Delta_k = \{u : u \in R^k, u \geq 0, e_k^\tau u = 1\}$ denote the $k$-dimensional standard simplex (superscript $\tau$ denotes transpose). Also, let $(1, k)$ denote the set of indices from 1 to $k$. We will need the following additional definitions for any vector $u \in R^k$:

- $supp(u) = \{i \in (1,k) : u_i \neq 0\}$ (the set of indices in $(1,k)$ for which $u$ is non-zero).
- $suppmax(u) = \{i \in (1,k) : u_i \geq u_j \ \forall j \in (1,k)\}$ (the set of indices in $(1,k)$ for which the maximum value of $u$ is attained).
- $max(u) = \{u_i : i \in (1,k), u_i \geq u_j \ \forall j \in (1,k)\}$ (the maximum value of $u$).
- $max_S(u) = \{u_i, i \in S : u_i \geq u_j \ \forall j \in S\}$ (the maximum value of $u$ in the index set $S \subset (1,k)$).

The problem of finding an $\epsilon$-approximate Nash equilibrium of the game $(R, C)$, for some $\epsilon \geq 0$, is to compute a pair of strategies $\hat{x} \in \Delta_m$ and $\hat{y} \in \Delta_n$ such that the following relationships hold: $x^\tau R\hat{y} \leq \hat{x}^\tau R\hat{y} + \epsilon \ \forall x \in \Delta_m$ and $\hat{x}^\tau C y \leq \hat{x}^\tau C\hat{y} + \epsilon \ \forall y \in \Delta_n$.

Following the analysis in [1], we define the following function, mapping $\Delta_m \times \Delta_n$ into $[0, 1]$: $f(x, y) = max\{f_R(x, y), f_C(x, y)\}$, where, $f_R(x, y) = max(Ry) - x^\tau Ry$ and $f_C(x, y) = max(C^\tau x) - x^\tau Cy$. For any pair of strategies $(x, y) \in \Delta_m \times \Delta_n$, 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 obtain an $\epsilon$-approximate Nash equilibrium, where $\epsilon$ is $\leq 0.3393$ (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 $(x, y) \in \Delta_m \times \Delta_n$, we consider motions away from it along feasible directions of the form $(1-\epsilon)[x^\tau, y^\tau] + \epsilon[(x')^\tau, (y')^\tau]$, where, $(x', y') \in \Delta_m \times \Delta_n$ is another pair of strategies and $\epsilon : 0 \leq \epsilon \leq 1$ (vectors in brackets denote $m + n$-dimensional vectors). The computation of a feasible direction $(x', y') \in \Delta_m \times \Delta_n$ 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 with respect to $x'$:

$$min_{x'}\{max(Ry) - (x')^\tau Ry\}$$

subject to: $max(C^\tau x') - (x')^\tau Cy \leq max(Ry) - (x')^\tau Ry$ and $x' \in \Delta_m$.
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 with respect to $y'$:

$$min_{y'}\{max(C^\tau x) - x^\tau Cy'\}$$

subject to: $max(Ry') - x^\tau Ry' \leq max(C^\tau x) - x^\tau Cy'$ and $y' \in \Delta_n$.
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:

$$min_{(x',y')}max_{(w,z,\rho)}[\rho w^\tau, (1-\rho)z^\tau]G(x,y)\begin{bmatrix} y' \\ x' \end{bmatrix}$$

where:

(i) The maximum is taken with respect to dual variables $w, z, \rho$ such that:
$w \in \Delta_m$ and $supp(w) \subset S_R(y) \equiv suppmax(Ry)$
$z \in \Delta_n$ and $supp(z) \subset S_C(x) \equiv suppmax(C^\tau x)$
$\rho \in [0,1]$

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

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

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

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:*

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

*(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.* For fixed $y$, 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 $x'$ such that $supp(x') \subset suppmax(Ry)$.

Now, choose any $x_1 \in \Delta_m$ such that: $supp(x_1) \subset suppmax(Ry)$. Consider the following function:

$$g(\epsilon) = f_C((1-\epsilon)x + \epsilon x_1, y) - f_R((1-\epsilon)x + \epsilon x_1, y) \text{ for } 0 \leq \epsilon \leq 1$$

It is easy to verify that $g(\epsilon)$ is continuous and convex in $\epsilon$ and that it satisfies: $g(0) = -(f_R(x,y) - f_C(x,y)) < 0$ and $0 \leq g(1) = f_C(x_1, y) \leq 1$. So, there is an $\epsilon'$ such that $g(\epsilon') = 0$. By convexity, we should have $g(\epsilon') \leq (1-\epsilon')g(0) + \epsilon' g(1)$ which, in view of the above relationships, implies: $(1-\epsilon') \leq \frac{1}{1+f_R(x,y)-f_C(x,y)}$. Furthermore, choosing $x' = (1-\epsilon')x + \epsilon' x_1$, we have $f_C(x', y) = f_R(x', y) = (1-\epsilon')f_R(x,y)$. Finally, the assertion of the Lemma follows from the last two relationships. □

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 $\leq \frac{1}{2}$. 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 $\leq \frac{1}{2}$ 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 $f(x+\epsilon(x'-x), y+\epsilon(y'-y))$ with respect to the scalar parameter $\epsilon$, 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 $\epsilon$ (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 $\epsilon^\star$ 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 $(x', y') \in \Delta_m \times \Delta_n$ 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 $\epsilon^\star > 0$ such that $f(x+\epsilon(x'-x), y+\epsilon(y'-y)) - f(x,y) < 0$ for all $\epsilon \in (0, \epsilon^\star]$.*

*Proof.* The proof is based on the construction of the above difference for any $\epsilon : 0 \leq \epsilon \leq 1$ 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 the index sets $(1,m)$ and $(1,n)$ respectively (we have defined $S_R(y) \equiv suppmax(Ry)$ and $S_C(x) \equiv suppmax(C^\tau x)$). Let $\epsilon^\star = min\{\epsilon_1^\star, \epsilon_2^\star, 1\}$, where:

$$\epsilon_1^\star = min_i \left[ \frac{\max(Ry) - (Ry)_i}{\max(Ry) - (Ry)_i + (Ry')_i - \max_{S_R(y)}(Ry')} \right]$$

(where the minimum is taken over $i$ such that: $i \in \overline{S_R}(y)$ and $(Ry')_i - \max_{S_R(y)}(Ry') \geq 0$)

$$\epsilon_2^\star = min_j \left[ \frac{\max(C^\tau x) - (C^\tau x)_j}{\max(C^\tau x) - (C^\tau x)_j + (C^\tau x')_j - \max_{S_C(x)}(C^\tau x')} \right]$$

(where the minimum is taken over $j$ such that: $j \in \overline{S_C}(x)$ and $(C^\tau x')_j - \max_{S_C(x)}(C^\tau x') \geq 0$)
Also, let

$$\Delta = min\big[min_{i \in \overline{S_R}(y)}[max(Ry) - (Ry)_i], min_{j \in \overline{S_C}(x)}[max(C^\tau x) - (C^\tau x)_j]\big]$$

It is clear that we always have $\Delta > 0$ and that $\epsilon^\star \geq \frac{\Delta}{1+\Delta}$.

Following the derivations in [1] and after several manipulations that were performed taking into account that the algorithm minimizes the difference with respect to $\epsilon \in (0, \epsilon^\star]$ 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:

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

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. Notice that a stationary point always exists as a limit point of the sequence of values of $f$ produced by the descent algorithm. Such a sequence has always a limit since it is monotonically decreasing and bounded from below by 0. □

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 $V(x,y) - f(x,y) = 0$ is satisfied. 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
1. Apply **Descent Direction**
2. Apply **Line Search**
3. Stop if termination condition is satisfied. If not, return to step 1.

## 3 Convergence rate

We provide an improved complexity result of the descent algorithm compared to the result 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:

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

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

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

**Lemma 3.** *The descent algorithm converges to a $\delta$-stationary point in $O\left(\frac{1}{\delta}\log(\frac{1}{\delta})\right)$ loops.*

*Proof.* Let us denote by $b$ the value of $f$ at a limit point of the descent algorithm. We should have $V \leq b \leq f$ at every loop of the algorithm ($b$ cannot be more than 0.3393 and the initial value of $f$ is $\leq \frac{1}{2}$). 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 $f_{new} - b \leq (f - b) - \frac{(f-b)^2}{4(1-b)}$, or a descent of the form $f_{new} - b \leq \left(1 - \frac{\Delta}{1+\Delta}\right)(f - b) - \frac{\Delta^2}{(1+\Delta)^2}b$. So we have two types of positive sequences $s_k, k = 1, 2, ...$ starting from values $< \frac{1}{2}$ and converging to 0 according to the relationships $s_{k+1} \leq s_k - \frac{s_k^2}{4(1-b)}$ and $s_{k+1} \leq \left(1 - \frac{\Delta}{1+\Delta}\right)s_k - \frac{\Delta^2}{(1+\Delta)^2}b$. It can be verified that the first sequence requires at most $O\left(\frac{1}{\delta}\right)$ steps to reach a point $s_k$ such that $s_k \leq \delta$ and the second sequence requires at most $O\left(\frac{1}{\Delta}\log(\frac{1}{\delta})\right)$ steps to reach such a point. The result follows from these facts and from $\delta < \Delta$. □

## 4 Evaluation scenaria

For the evaluation of performance we have applied the algorithm to a large number of bi-matrix game problem instances with sizes ranging from $10 \times 10$ to $100 \times 100$. 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, $2\times 2$ 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 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 $2 \times 2$) 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 ([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 $R, C$. 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 $\delta$ was fixed to $\delta = 0.001$ for all instances.

## 5 Evaluation of the results

The results obtained from all the runs that were performed generally indicate that the descent algorithm is a highly efficient and practical algorithm that converges fast to stationary points providing high quality approximate Nash equilibria. In fact, for almost all cases, the approximations achieved are less than the precision parameter $\delta$, far better than the theoretical worst case approximation (0.3393). This experimental conclusion is typical for all categories of instances (as described above) upon which the algorithm was applied.

The worst approximation obtained across all experimental instances that we considered was 0.015 (notably, this happened for a small game). 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 the above-mentioned 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.

In regard to convergence rate, the experiments indicate that the number of loops required for convergence differ across starting points and categories but typically it was between 4 and 20 with a median value of 10. Also, the supports of the resulting (approximate equilibria) distributions were relatively large, typically between $n/3$ and $n/2$ ($n$ being the size of the game).

Overall, the number of loops required for convergence to a $\delta$-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 $O\left(\frac{1}{\delta}\log(\frac{1}{\delta})\right)$ (formulated and proved here in Lemma 3), a significant improvement over the previous $O\left(\frac{1}{\delta^2}\right)$.

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, (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, and (d) Some further investigation of the convergence rate of the algorithm for an improved bound.

## References


1. H. Tsaknakis, P. G. Spirakis. 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 (EC07), 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.