INT J COMPUT COMMUN, ISSN 1841-9836
8(4):571-577, August, 2013.

PyBNEq - A Tool for Computing Bayes-Nash Equilibria

I. Joldeş, B. Pârv, I. Parpucea, V. Lupşe

Iulian Joldeş, Bazil Pârv
Babeş-Bolyai University
Department of Computer Science
Romania, 400084 Cluj-Napoca, 1 M. Kogălniceanu Str.
E-mail: joldesiulian@yahoo.com, bparv@cs.ubbcluj.ro

Ilie Parpucea
Babeş-Bolyai University
Department of Mathematics and Statistics
Romania, 400591 Cluj-Napoca, 58-60 Teodor Mihali Str.
E-mail: ilie.parpucea@gmail.com

Vasile Lupşe
Technical University Cluj-Napoca
North Center Baia Mare
Romania, 430083 Baia Mare, 62/A Dr. Victor Babeş Str.
E-mail: vasilelupse@ubm.ro

Abstract:
This paper describes PyBNEq - a tool for computing Bayes-Nash equilibria for games of incomplete information. It is implemented in Python and has a graphical user interface, allowing the user to load/save/edit game data, and to find Bayes-Nash equilibria. Currently, PyBNEq implements Porter-Nudelman-Shoham algorithm for 2-player games and can be considered as a decision support system for solving games of incomplete information.
Keywords: Bayes-Nash equilibrium, decision support systems, game with incomplete information.

1 Introduction

Many real-world problems, including ones which contain germs of a crisis, are modeled as games of incomplete information. Examples include market competition, currency attacks, bank runs, liquidity crises, as well as military conflicts.

This paper describes a tool for computing Bayes-Nash equilibrium for games with incomplete information. It is structured in 5 sections, as follows. After this introductory section, the section 2 introduces the Bayesian games and a short discussion related to the computation of the Bayes-Nash equilibria. Section 3 describes the PyBNEq tool, especially its graphical user interface, while section 4 presents some case studies we used to test our tool. Finally, the last section compares our application with the existing ones and presents the future developments.

2 Theoretical background

In what follows, we consider the definition of a *Bayesian game* given in [1]. Such a game is defined as a tuple $(N, A, \Theta, \Omega, u)$, where:

- $N = \{1, ..., n\}$ is the set of agents or players;
- $A = (A_1, A_2, \cdots, A_n)$ is the set of agents' actions, A_i being the set of actions available to agent i;

- $\Theta = (\Theta_1, \Theta_2, \cdots, \Theta_n)$, with $\Theta_i = (\theta_{i,1}, \theta_{i,2}, \cdots, \theta_{i,m_i})$ being the set of types for agent i and $\theta_{i,j}$ the j-th type of agent i;
- $\Omega = (\Omega_1, \Omega_2, \cdots, \Omega_n)$, with $\Omega_i = (\omega_{i,1}, \omega_{i,2}, \cdots, \omega_{i,m_i})$ being the set of probabilities assigned to the types of agent i and $\omega_{i,j}$ the probability assigned to $\theta_{i,j}$;
- $u = (u_1, u_2, \cdots, u_n)$ is the set of utility (payoff) functions, with $u_i = (u_{i,1}, u_{i,2}, \cdots, u_{i,m_i})$ being the set of utility functions of agent i and $u_{i,j}$ the utility function of $\theta_{i,j}$ whose arguments are the joint action $\mathbf{a} = (a_1, a_2, \cdots, a_n)$ and the other agents' types $\theta_{-i} = (\theta_1, \cdots, \theta_{i-1}, \theta_{i+1}, \cdots, \theta_n)$.

Alternative definitions can be found for example in [3, 8, 9]. The solution of a Bayesian game is based on the concept of *Bayes-Nash equilibrium*. It is known that the computation of this equilibrium is NP-complete, and the usual approach uses two steps [12]:

1. reduce the Bayesian game to a complete-information game, *and*
2. compute the Nash equilibrium for the complete-information game obtained in Step 1.

Step 1 above can be addressed in several ways. The paper by Ceppi et al. [1] contains a detailed discussion regarding the ways of performing reduction, based especially on the material in [5]. Our approach is based on the use of sequence form, due to the smaller payoff matrixes than the ones in normal form.

Step 2 means computing of the Nash equilibria for two-player complete-information games. Ceppi et al. paper [1] analyze three algorithms for such games in strategic form: Lemke-Howson (LH), Porter-Nudelman-Shoham (PNS), and Sandholm-Gilpin-Conitzer (SGC), and propose an extension of PNS to Bayesian games (B-PNS), which is implemented in our tool.

The B-PNS algorithm has the following generic steps:

1. enumerate all the possible joint supports;
2. select a support for each possible type of agent 1;
3. prune the spaces of actions of the agent 2's types by strict conditional dominance;
4. check the strict conditional dominance on agent 1's support;
5. select a support for each possible type of agent 2;
6. check the strict conditional dominance on agent 2's suport;
7. check the feasibility problem.

3 The PyBNEq tool

In order to implement the B-PNS algorithm, there were two interrelated design problems to solve: data representation and the individual algorithms for each step. A future paper will describe the design solutions in greater detail.

The implementation language is Python 2.7 [11], while data representation and numerical computation capabilities of `numpy` package [6] were extensively used. Also, the application makes use of `pycplex` library [13], a Python interface to the ILOG CPLEX Callable Library [2], for solving the feasibility problem (step 7 of the above-described B-PNS algorithm). Finally, the graphical user interface was produced by using the `wxPython` toolkit [14].

The application is designed with a friendly user interface, having the following components:

- *main window* contains a menu bar for creating, loading or saving a game and also for running it. When a game is loaded, this window becomes the current game window, where the game parameters are displayed (see Figure 1), and the Bayes-Nash echilibria can then be computed;

- *new game window*, allowing the user to define actions, types and probabilities for both players (see Figure 2);

- *payoffs window*, where the payoffs could be added or edited (Figure 3).

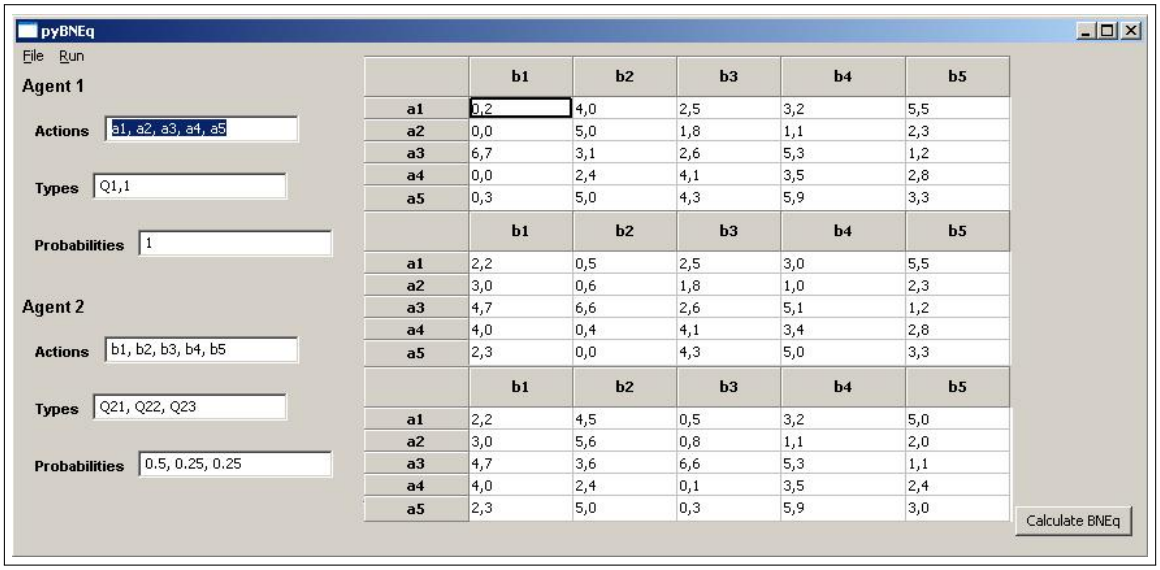

Figure 1: The main window of the application

The input data are stored in /data sub-directory in .dat files. The structure of input data file is:

Line1: first agent's index
Line2: first agent's actions
Line3: first agent's types
Line4: first agent's probabilities
Line5: second agent's index
Line6: second agent's actions
Line7: second agent's types
Line8: second agent's probabilities
Line9 and below: the payoffs of the two players

The result consists of the computed Bayes-Nash equilibria and it's saved in /output sub-directory, in an .out file. It represents the strategies profile.

4 Case studies

We tested our tool with three examples of Bayesian Games: market entry game (see [7]), gift game (see [4], and the example game from [1]. All examples consider two players, whose actions are denoted by a_i and b_j, respectively.

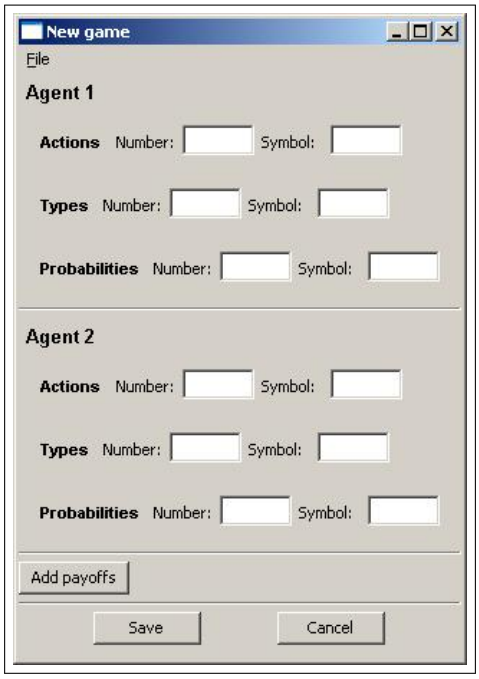

Figure 2: The Add new game window

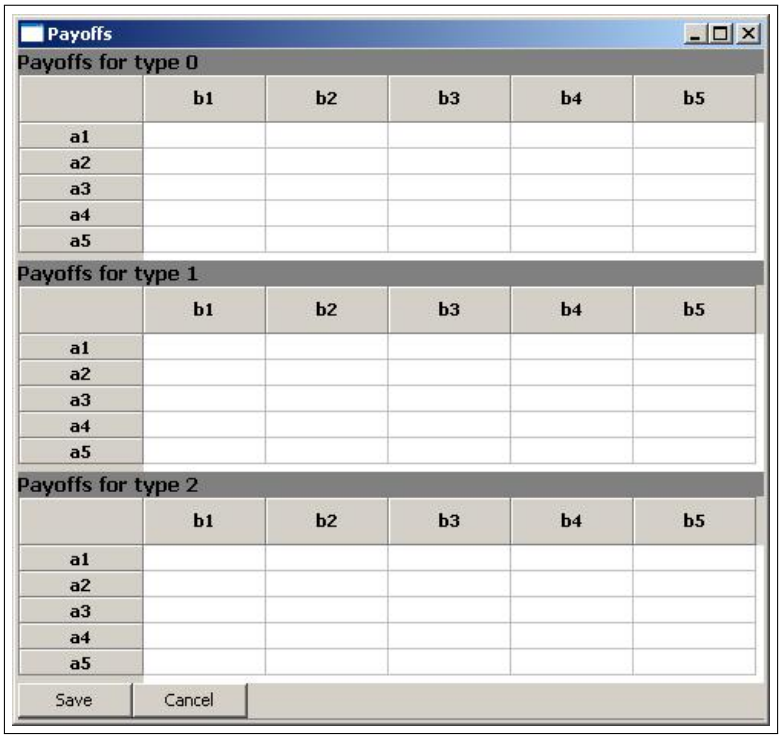

Figure 3: The payoffs window

4.1 Market entry game

Both players have a single type. The following payoff matrix represents the input data:

	b_1	b_2
a_1	$-\frac{1}{4}$, $-\frac{1}{4}$	$\frac{3}{4}$, 0
a_2	0, $\frac{1}{4}$	$\frac{1}{2}$, 0
a_3	$-\frac{1}{4}$, $\frac{1}{4}$	$\frac{1}{4}$, 0
a_4	0, $\frac{3}{4}$	0, 0

The application outputs three solutions: (a_1, b_2), (a_2, b_1), and (a_4, b_1).

4.2 Gift giving game

Player 1 has two types $(\theta_{1,1}, \theta_{1,2})$, and player 2 has one type $(\theta_{2,1})$. Type $\theta_{1,1}$ means $p > \frac{1}{2}$, while $\theta_{1,2}$ means $p < \frac{1}{2}$. The input data are represented by the following payoff matrix:

	b_1	b_2
a_1	0, 0	0, 0
a_2	1 - p, p - 1	p - 1, 0
a_3	p, p	-p, 0
a_4	1, $2 \cdot p$ - 1	-1, 0

a) The type combination $(\theta_{1,1}, \theta_{2,1})$. Player 1 is more likely to be a friend $(p > \frac{1}{2})$; the application outputs two solutions: (a_1, b_2) and (a_4, b_1).

b) The type combination $(\theta_{1,2}, \theta_{2,1})$. Player 1 is more likely to be an enemy $(p < \frac{1}{2})$; the application outputs one solution: (a_1, b_2).

4.3 Third example: the game from [1]

Player 1 has one type $(\theta_{1,1})$ and player 2 has three types $(\theta_{2,1}, \theta_{2,2}, \theta_{2,3})$. There are three payoff matrices as input, corresponding to the combinations of players' types:

a) $(\theta_{1,1}, \theta_{2,1})$: $\omega_{2,1} = \frac{1}{2}$

	b_1	b_2	b_3	b_4	b_5
a_1	0, 2	4, 0	2, 5	3, 2	5, 5
a_2	0, 0	5, 0	1, 8	1, 1	2, 3
a_3	6, 7	3, 1	2, 6	5, 3	1, 2
a_4	0, 0	2, 4	4, 1	3, 5	2, 8
a_5	0, 3	5, 0	4, 3	5, 9	3, 3

The solutions are: (a_1, b_5), (a_2, b_1), and (a_3, b_4).

b) $(\theta_{1,1}, \theta_{2,2})$: $\omega_{2,2} = \frac{1}{4}$

	b_1	b_2	b_3	b_4	b_5
a_1	2, 2	0, 5	2, 5	3, 0	5, 5
a_2	3, 0	0, 6	1, 8	1, 0	2, 3
a_3	4, 7	6, 6	2, 6	5, 1	1, 2
a_4	4, 0	0, 4	4, 1	3, 4	2, 8
a_5	2, 3	0, 0	4, 3	5, 0	3, 3

The solutions are: (a_1, b_3), (a_2, b_1), (a_4, b_5), and (a_5, b_1).

c) $(\theta_{1,1}, \theta_{2,3})$: $\omega_{2,3} = \frac{1}{4}$

	b_1	b_2	b_3	b_4	b_5
a_1	2, 2	4, 5	0, 5	3, 2	5, 0
a_2	3, 0	5, 6	0, 8	1, 1	2, 0
a_3	4, 7	3, 6	6, 6	5, 3	1, 1
a_4	4, 0	2, 4	0, 1	3, 5	2, 4
a_5	2, 3	5, 0	0, 3	5, 9	3, 0

The solutions are: (a_2, b_2), (a_3, b_2), (a_3, b_1), (a_4, b_5), and (a_5, b_4).

5 Conclusions and future work

In this paper we described a tool for computing Bayes-Nash equilibrium for two-player games of incomplete information.

The authors of [1] implemented a C version of the B-PNS algorithm, while we selected Python as the implementation language due to its benefits as rapid application development by using Component-based Software Development and the powerful libraries like numpy, pycplex, as well as the Graphical User Interface toolkit wxPython. The graphical user interface allows a very convenient way of playing with such games.

In the future we would like to extend the tool functionalities so that it can easely be integrated in more elaborate decision support systems, including custom-made GUIs for different classes of games. Also, the application programming interface will be improved and the tool will fully support games generated by the common generators like GAMUT.

Acknowledgements

This work was supported by the grant ID_2586, sponsored by NURC - Romanian National University Research Council (CNCSIS).

Bibliography

[1] Ceppi, S., N. Gatti, N. Basilico, Computing Bayes-Nash Equilibria through Support Enumeration Methods in Bayesian Two-Player Strategic-Form Games in WI-IAT, *Proc.of the 2009 IEEE/WIC/ACM Int. Joint Conference on Web Intelligence and Intelligent Agent Technology*, 2: 541-548, 2009, DOI: 10.1109/WI-IAT.2009.209.

[2] `cplex` reference, ftp://ftp.software.ibm.com/software/websphere/ilog/docs/optimization/cplex/refcallablelibrary.pdf

[3] Eichberger, I., *Game Theory for Economists*, Academic Press, 1993.

[4] Fehr, E. and K.M. Schmidt, Theories of Fairness and Reciprocity - Evidence and Economic Applications, in *M. Dewatripont, L.P. Hansen and S.J. Turnovsky (eds.) Advances in Economics and Econometrics, Econometric Society Monographs, 8th World Congress*, 1: 208-257, 2003.

[5] Koller, D., N. Megiddo, and B. von Stengel, Efficient computation of equilibria for extensive two-person games, Games and Economic Behavior, 14(2): 220-246, 1996.

[6] `numpy` home page, http://numpy.scipy.org/

[7] Ochs, J. Coordination problems. In J.H. Kagel and A.E. Roth (eds.) Handbook of Experimental Economics, Princeton University Press, 195-251, 1995.

[8] Parpucea, I., B. Pârv, and T. Socaciu, T. Modeling Uncertainty in a Decision Problem by Externalizing Information, *INT J COMPUT COMUN*, 6(2): 328-336, 2011.

[9] Pârv, B. and I. Parpucea, Bayes-Nash Equilibrium in the Presence of Information Sources: Computational Issues, Studia Univ. Babes-Bolyai, Informatica, LVI (2011), 3: 33-38, 2011.

[10] Porter, R., E. Nudelman, and Y. Shoham, Simple search methods for finding a Nash equilibrium, *Proc. of the AAAI Conference on Artificial Intelligence (AAAI)*, 664-669, 2004.

[11] Python home page, http://www.python.org/

[12] Shoham, Y. and K. Leyton-Brown, *Multiagent Systems: Algorithmic*, Game Theoretic and Logical Foundations. Cambridge, USA: Cambridge University Press, 2008.

[13] `pycplex` reference, http://www.cs.toronto.edu/ darius/software/pycplex/

[14] `wxpython` reference, http://wxpython.org/