PRALINE: A Tool for Computing Nash Equilibria in Concurrent Games

Romain Brenguier

Département d'informatique, Université Libre de Bruxelles (U.L.B), Belgium
LSV, CNRS & ENS Cachan, France
brenguier@lsv.ens-cachan.fr

Abstract. We present PRALINE, which is the first tool to compute Nash equilibria in games played over graphs. We consider concurrent games: at each step, players choose their actions independently. There can be an arbitrary number of players. The preferences of the players are given by payoff functions that map states to integers, the goal for a player is then to maximize the limit superior of her payoff; this can be seen as a generalization of Büchi objectives. PRALINE looks for pure Nash equilibria in these games. It can construct the strategies of the equilibrium and users can play against it to test the equilibrium. We give the idea behind its implementation and present examples of its practical use.

1 Introduction

In computer science, two-player games have been successfully used for solving the problem of controller synthesis. Multiplayer games appear when we want to model interaction between several agents, where each agent has its own preference concerning the evolution of the global system. Think for instance of several users behind their computers on a shared network. When designing a protocol, maximizing the overall performance of the system is desirable, but if a deviation can be profitable to the users, it should be expected that one of them takes advantage of this weakness. This happened for example to the bit-torrent protocol where selfish clients became more popular. Such deviations can harm the global performance of the protocol. In these situations, equilibrium concepts are particularly relevant. These notions aim at describing rational behaviors. In a Nash equilibrium, each agent plays in such a way that none of them can get a better payoff by switching to another strategy.

In the context of controller synthesis, games are generally played on graphs. The nodes of the graph represent the possible configurations of the system. The agents take actions in order to move a token from one node to another. Among those games, the simplest model is that of turn-based games, where in each state, one player decides alone on which outgoing edge to take. The model we consider is concurrent games, which is more expressive. For these games, in each state, the players choose their actions independently and the joint move formed by these choices determines the next state.

There has recently been a lot of focus on the algorithmic aspect of Nash equilibria for games played on graphs [4,11,12]. Thanks to these efforts, the theoretical bases are understood well enough, so that we have developed effective algorithms [1,2]. We implemented them in PRALINE. Other tools exist which can compute Nash equilibria for classical models, however this is the first tool to compute Nash equilibria in games played on graphs. The tool PRISM-games [5] can also analyse multiplayer games on graphs. In particular, it can generate an optimal strategy for one player and can be used to check that a given strategy is a Nash equilibrium. It can deal with randomized games, which PRALINE cannot, however it is unable to generate Nash equilibria, as PRALINE does.

We give an overview of the features of PRALINE. First, we present the model of games that is used, illustrated with examples. We also present the suspect game transformation [3], which gives the idea of the underlying algorithm and makes it possible to test the equilibrium by playing against it. We also ran some experiments on the given examples to evaluate the performances of the tool. The tool is available from `http://www.lsv.ens-cachan.fr/Software/praline/`.

2 Concurrent Games

The model of game we consider is concurrent games. These are played on a graph which we call the *arena* of the game. A *state* of the game is a vertex of the arena. At the beginning of a turn, each of the players chooses an action, and the tuple of these actions defines a *move*. The next state of the game is given by following the edge that goes from the current state and is labeled by this move, and a new turn begins from that state. This is then repeated *ad infinitum*, to define an infinite path on the graph, called a *play*. Players are assumed to see the sequence of states, but not the actions played by other players.

An example of an arena is given in Fig. 2. If, for example, in state `1,0,1,0`, player p_1 chooses action 1 and player p_2 chooses action 0, the play follows the edge labeled by the move $1, 0$ and the next state is `0,1,1,0`. Then a new turn begins. If both players keep on playing $0, 0$ forever, the system will stay in configuration `0,1,1,0`; this defines the play $\texttt{1,0,1,0} \cdot (\texttt{0,1,1,0})^{\omega}$.

To describe games with a big state space it is convenient to write small programs which generate the arena, like the one in Fig. 1. Each state of the arena corresponds to a possible valuation of the variables, the move function describes the actions available in each state and the update function computes the new state according to the actions of each player.

Example 1 (Medium Access Control). This example was first formalized from the point of view of game theory in [8]. Several users share access to a wireless channel. During each slot, they can choose to either transmit or wait for the next slot. If too many users are emitting in the same slot, then they fail to send data. Furthermore each attempt at transmitting costs energy to the players. They have to maximize the number of successful attempts using the energy available to them. We give in Fig. 1 a possible way to model this game in PRALINE. In

this example the players are `p1` and `p2`. Their energy levels are represented by variables `energy1` and `energy2`, and variables `trans1` and `trans2` keep track of the number of successful attempts. The players can always wait (represented by the action 0), and if there energy is not zero they can transmit (represented by action 1). The generated arena for an initial energy allowing only one attempt for each player is represented in Fig. 2. The labels of the nodes correspond to the valuation of the variables `energy1`, `trans1`, `energy2` and `trans2`.

```
/* ... */
move {
  legal p1 0;
  legal p2 0;
  if (energy1 > 0) legal p1 1;
  if (energy2 > 0) legal p2 1;
}

update {
  if (action p1 == 1)
    energy1 = energy1 - 1;
  if (action p2 == 1)
    energy2 = energy2 - 1;

  if (action p1 == 1 && action p2 == 0)
    trans1 = trans1 + 1;
  if (action p1 == 0 && action p2 == 1)
    trans2 = trans2 + 1;
}
```

Fig. 1: Part of the game file "medium_access.game"

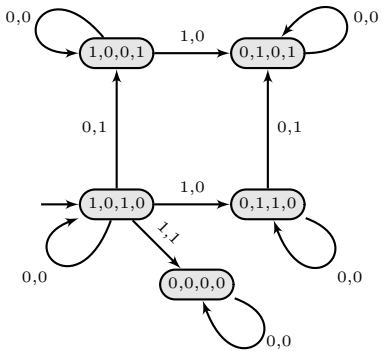

Fig. 2: Arena generated from this file.

3 Computing Nash Equilibria

The preference of a player p_i is specified by a function payoff_{p_i} which assigns an integer to each state of the game. The payoff of a run is the limit superior of this function, and the goal is to maximize it. This is a generalization of Büchi objectives which can be specified with payoff either 0 or 1 for each state.

Example 2 (Power Control). This example is inspired by the problem of power control in cellular networks. Game theoretical concepts are relevant for this problem and Nash equilibria are actually used to describe rational behaviors of agents [6,7]. We consider the situation where several phones are emitting over a cellular network. Each agent p_i can choose the emitting power pow_i of his phone. From the point of view of agent p_i, using a stronger power results in a better transmission, but it is costly since it uses energy, and it lowers the quality of the transmission for the others, because of interferences. We model this game by the arena presented in Fig. 3, for a simple situation with two players which at each step can choose to increase or not their emitting power until they reach the maximum level of 2. The payoff for player p_i can be modeled by this expression from [10]: $\mathsf{payoff}_{p_i} = \frac{R}{\mathsf{pow}_i}\left(1 - e^{-0.5\gamma_i}\right)^L$ where γ_i is the signal-to-interference-and-noise ratio for player p_i, R is the rate at which the wireless system transmits the information in bits per seconds and L is the size of the packets in bits.

To describe the rational behavior of the agents in a non-zero-sum game, the concept that is most commonly used is Nash equilibria [9]: a Nash equilibrium is a choice of strategies (one for each player), such that there is no player which can increase her payoff, by changing her own strategy, while the other players keep theirs unchanged.

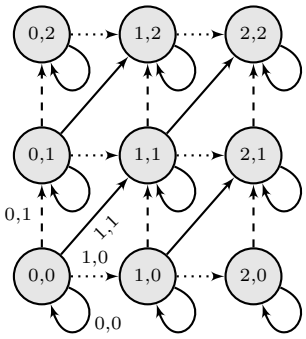

Fig. 3: Arena of the power control game

The tool PRALINE looks for pure (i.e. non randomized) Nash equilibria in the kind of games we described. Note that a pure Nash equilibrium is resistant to randomized strategies. On the other hand the existence of a randomized Nash equilibrium with a particular payoff is undecidable [12]. If the game contains a (pure) Nash equilibrium with some payoff payoff_i for each player p_i, then PRALINE returns at least one Nash equilibrium with payoff payoff'_i such that for every player p_i, $\mathsf{payoff}'_i \geq \mathsf{payoff}_i$. For an overview of the Nash equilibrium, the tool can output the *shape* of the solution. That is, the moves that are effectively taken by the players if none of them deviates from the equilibrium. For example, in the power control game, our tool gives two solutions, one Nash equilibrium with payoff 110 for each player and another one with payoff 94 for each, their shapes are represented in Fig. 4 and Fig. 5 respectively. For each solution, the tool can also output a file containing the full strategies, represented as automata. These automata are usually big: the first solution of the power control example is implemented by an automaton containing 125 edges. They can be difficult to analyze. A convenient way to look at the generated equilibrium, is to play the suspect game against it. We now explain the suspect game construction, which is the core of the implemented algorithm.

4 The Suspect Game

The idea behind the algorithm implemented is that of the suspect game, which allows to think about Nash equilibria as winning strategies in a two-player turn-based game [3]. This transformation makes it possible to use algorithmic techniques from zero-sum games to synthesize Nash equilibria.

The suspect game is played between `Eve` and `Adam`. `Eve` wants the players to play a Nash equilibrium, and `Adam` tries to disprove that it is a Nash equilibrium, by finding a possible deviation that improves the payoff of one of the players.

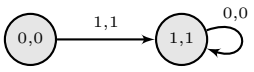

Fig. 4: Shape of solution 1

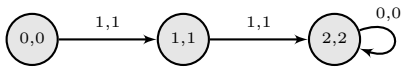

Fig. 5: Shape of solution 2

In the beginning of the game all the players are considered suspect, since they can potentially deviate from the equilibrium. Then `Eve` chooses a legal move and `Adam` chooses some successor state. When the state chosen by `Adam` is the state resulting from `Eve`'s move, we say that `Adam` obeys `Eve`, and the suspects are the same than before. Otherwise we keep among the suspects those that can unilaterally change their action from the one suggested by `Eve` to activate the transition suggested by `Adam`. The game then continues from the state played by `Adam`. Given the desired payoff, the outcome of the game is winning for `Eve`, if all players that are ultimately suspect have a payoff inferior or equal to the given one. There is a Nash equilibrium in the original game with payoff_i for each player p_i if, and only if, there is a winning strategy for `Eve` such that when `Adam` obeys the payoff is exactly payoff_i for each player p_i [3].

Using this transformation for the preferences under consideration, `Eve`'s objective can be expressed as a co-Büchi condition. This game can be solved in polynomial time and we have indeed a polynomial time algorithm for Nash equilibria [2]. In order to understand how a Nash equilibrium is enforced, the tool PRALINE allows the user to play against the winning strategy of `Eve`.

Example 2 (cont'd). We come back to the example of the power control game, and the first solution, where players use a power of 1 and get a payoff of 110. When we play against `Eve` in this game, she first suggests the move $1, 1$. If we obey this move, the power of each players is raised to 1. `Eve` then plays $0, 0$ as long as we stay in this same state. If we continue to obey, the payoff will be 110 for each player, which would make `Eve` win. If we want to find a deviation profitable to one player we might want to raise the power of one of them. For instance, if we change the power of the first one, then she is suspect in the next state and her current payoff is 131. But then, `Eve` will suggest the move $0, 1$ whose natural outcome is $2, 2$ which has a payoff of 94 for the first player. If we obey this move we failed to improve our payoff, and we cannot change the next state by changing only the action of the first player, so the game is lost for `Adam`.

5 Experiments and Conclusions

In order to show the influence of the size of the graph on the time taken to compute Nash equilibria, we ran the tool on examples with different parameters. The experimental results are given in Table 1, they were obtained on a PC with an Intel Core2 Duo processor at 2.8GHz with 4GB of RAM. We observe from these experiments that our prototype works well for games up to one hundred states. The execution time then quickly increases. This is because the algorithm as described in [2], requires computation of the winning regions in a number of subgames that might be quadratic in the number of states of the game. Computing winning regions takes quadratic time in itself.

PRALINE is the first tool to compute pure Nash equilibria in games played on graphs. Experimental results are encouraging since we managed to synthesize Nash equilibria for several examples. For future implementations, we hope to improve the tool's scalability by using symbolic methods.

Table 1: Experiments

Power Control					
Players	Emission Levels	States	Edges	Solutions	Time (sec.)
2	2	9	25	2	0.01
4	2	81	625	6	4.28
3	5	216	1331	83	64.50
6	2	729	15625	23	2700.97
Medium Access Control					
Players	Initial Energy	States	Edges	Solutions	Time (sec.)
2	2	14	35	1	0.02
3	2	100	347	1	0.69
3	4	1360	6303	1	160.22

References

1. P. Bouyer, R. Brenguier, and N. Markey. Nash equilibria for reachability objectives in multi-player timed games. In CONCUR'10, LNCS 6269, p. 192–206. Springer, 2010.
2. P. Bouyer, R. Brenguier, N. Markey, and M. Ummels. Nash equilibria in concurrent games with Büchi objectives. In FSTTCS'11, p. 375–386. Leibniz-Zentrum für Informatik, 2011.
3. P. Bouyer, R. Brenguier, N. Markey, and M. Ummels. Concurrent games with ordered objectives. In FoSSaCS'12, LNCS. Springer, 2012.
4. K. Chatterjee. Two-player nonzero-sum ω-regular games. In CONCUR'05, LNCS 3653, p. 413–427. Springer, 2005.
5. T. Chen, V. Forejt, M. Kwiatkowska, D. Parker, and A. Simaitis. PRISM-games: A model checker for stochastic multi-player games. In TACAS'13, LNCS 7795, p. 185–191. Springer, 2013.
6. A. MacKenzie, S. Wicker. Game theory and the design of self-configuring, adaptive wireless networks. *Communications Magazine, IEEE*, 39(11):126–131, 2001.
7. A. MacKenzie, S. Wicker. Game theory in communications: Motivation, explanation, and application to power control. In GLOBECOM'01, p. 821–826. IEEE, 2001.
8. A. MacKenzie, S. Wicker. Stability of multipacket slotted aloha with selfish users and perfect information. *IEEE INFOCOM*, 3:1583–1590, 2003.
9. J. F. Nash, Jr. Equilibrium points in n-person games. *Proc. National Academy of Sciences of the USA*, 36(1):48–49, 1950.
10. C. Saraydar, N. Mandayam, and D. Goodman. Pareto efficiency of pricing-based power control in wireless data networks. In *Wireless Communications and Networking Conference, 1999. WCNC. 1999 IEEE*, p. 231–235. IEEE, 1999.
11. M. Ummels. The complexity of Nash equilibria in infinite multiplayer games. In FoSSaCS'08, LNCS 4962, p. 20–34. Springer, 2008.
12. M. Ummels and D. Wojtczak. The complexity of Nash equilibria in limit-average games. In CONCUR'11, LNCS 6901, p. 482–496. Springer, 2011.