Using Answer Set Programming for Solving Boolean Games*

Sofie De Clercq and **Martine De Cock**
Dept. of Applied Mathematics, CS and Statistics
Ghent University, Ghent, Belgium
{SofieR.DeClercq, Martine.DeCock}@ugent.be

Kim Bauters
School of Electronics, Electrical Engineering and CS
Queen's University, Belfast, UK
K.Bauters@qub.ac.uk

Steven Schockaert
School of Computer Science and Informatics
Cardiff University, Cardiff, UK
S.Schockaert@cs.cardiff.ac.uk

Ann Nowé
Computational Modeling Lab
Vrije Universiteit Brussel, Brussels, Belgium
Ann.Nowe@vub.ac.be

Abstract

Boolean games are a framework for reasoning about the rational behaviour of agents, whose goals are formalized using propositional formulas. They offer an attractive alternative to normal-form games, because they allow for a more intuitive and more compact encoding. Unfortunately, however, there is currently no general, tailor-made method available to compute the equilibria of Boolean games. In this paper, we introduce a method for finding the pure Nash equilibria based on disjunctive answer set programming. Our method is furthermore capable of finding the core elements and the Pareto optimal equilibria, and can easily be modified to support other forms of optimality, thanks to the declarative nature of disjunctive answer set programming. Experimental results clearly demonstrate the effectiveness of the proposed method.

Introduction

Boolean games or BGs are a framework for analyzing the behaviour of competing rational agents, whose goals are formalized as propositional formulas. They offer an alternative to normal-form games (NFGs), in which the goals of agents are encoded implicitly in the form of a pay-off function. A BG is a tuple (N, V, π, Φ) with $N = \{1, \ldots, n\}$ a set of agents, V a set of propositional (action) variables, $\pi : N \rightarrow 2^V$ a control assignment function such that $\{\pi(1), \ldots, \pi(n)\}$ forms a partition of V, and Φ a set $\{\varphi_1, \ldots, \varphi_n\}$ of formulas in L_V, i.e. the logical language associated with V (Bonzon et al. 2006). The set $\pi(i)$ or π_i contains the action variables controlled by agent i. Note that each $p \in V$ is controlled by exactly one agent. Every agent i can choose to set $p \in \pi_i$ to true (undertake the action p) or to false (not undertake the action). The formula φ_i is agent i's goal. An interpretation of π_i is called a *strategy* s_i of agent i. A *strategy profile* of a BG is an n-tuple $S = (s_1, \ldots, s_n)$, with s_i a strategy of agent i for every $i \in N$. Since π partitions V and $s_i \subseteq \pi_i, \forall i \in N$, we unambiguously use the set notation $\cup_{i=1}^{n} s_i \subseteq V$ for a strategy profile $S = (s_1, \ldots, s_n)$. With s_{-i} we denote the projection of the strategy profile $(s_1, \ldots, s_n)$ on $N \setminus \{i\}$, i.e. $s_{-i} = (s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n)$. If s_i' is a strategy of agent i, then (s_{-i}, s_i') is a shorthand for $(s_1, \ldots, s_{i-1}, s_i', s_{i+1}, \ldots, s_n)$. In a natural way, agent i's goal induces a Boolean *utility function* u_i for agent i: for every strategy profile S, $u_i(S) = 1$ iff $S \models \varphi_i$, i.e. agent i reaches its goal in S, and $u_i(S) = 0$ otherwise. A straightforward solution concept in game theory is the notion of *pure Nash equilibrium* (PNE). A strategy profile $S = (s_1, \ldots, s_n)$ is a PNE iff, for every agent $i \in N$, s_i is a best response to s_{-i}, i.e. $u_i(S) \geq u_i(s_{-i}, s_i')$ for all strategies $s_i' \subseteq \pi_i$. An alternative solution concept of BGs is the *core*, i.e. the set of strategy profiles that are not blocked by any non-empty coalition $C \subseteq N$. A strategy profile S is blocked by a coalition $C \subseteq N$ ($C \neq \emptyset$) if there exists a strategy profile S' such that all agents outside C undertake the same actions in S and S' and if all agents in C have a strictly higher utility in S' than in S. In this paper we have restricted the discussion to PNEs, cores and Pareto optimality, since these are very common, intuitive solution concepts in the context of BGs, but our reduction to disjunctive answer set programming (ASP) can readily be generalized to alternative solution concepts (Bonzon, Lagasquie-Schiex, and Lang 2012; Dunne and Wooldridge 2012; Dunne et al. 2008).

BGs form a young and active research area (Harrenstein et al. 2001; Bonzon et al. 2006; Bonzon, Lagasquie-Schiex, and Lang 2007; Dunne et al. 2008; Bonzon, Lagasquie-Schiex, and Lang 2012; Wooldridge et al. 2013). The strength of BGs lies in their transparent and compact representation, since the propositional goals make it redundant to explicitly mention the utility for every strategy profile. Therefore, however, computing solutions such as PNEs or core elements is harder than for most other game representations. This is undoubtedly part of the reason why, to the best of our knowledge, no methods for computing the PNEs or core elements of general BGs have been proposed so far.

Currently available methods for solving BGs fall in two categories. First, given that translations from BGs to NFGs are available, we can readily use solvers for NFGs. However, since such translations are exponential, this circuitous method does not exploit the compactness of BGs. Moreover, its inefficiency increases rapidly with the number of action variables per agent, as shown in the Experiments sec-

*This research was funded by a Research Foundation-Flanders project.

tion. Second, several authors have proposed methods for sub-classes of BGs. For instance, in (Bonzon, Lagasquie-Schiex, and Lang 2007), an algorithm is provided to compute PNEs, but only for BGs with an acyclic irreflexive part of their dependency graph. In (Dunne et al. 2008), a bargaining protocol for BGs allows agents to negotiate in rounds. If the logical operators in the goals are restricted to $\wedge$ and $\vee$, specific negotiation strategies yield a Pareto optimal strategy profile during the first round. Otherwise, obtaining a solution is not guaranteed. More important, none of these techniques can compute the solutions of general BGs. In (De Clercq et al. 2014), the stochastic Win-Stay Lose-probabilistic-Shift algorithm (WSLpS) is used to find solutions of BGs. In case all agents are able to reach their goal simultaneously, WSLpS converges to a Pareto optimal PNE. In the Experiments section, we show that our disjunctive ASP technique outperforms a generalized version of WSLpS for sufficiently difficult problems.

In NFGs, deciding whether a PNE exists is NP-complete when the game is represented by: (i) a set of agents, (ii) a finite set of actions per agents, (iii) a function defining for each agent which other agents may influence their utility, and (iv) the utility of each agent, explicitly given for all joint strategies of the agent and the agents influencing it (Gottlob, Greco, and Scarcello 2003). Deciding whether a BG has a PNE, on the other hand, is Σ_2^{P}-complete[1], even for 2-player zero-sum BGs (Bonzon et al. 2006). Deciding whether the core of a BG is non-empty is also Σ_2^{P}-complete (Dunne et al. 2008). In particular, the problems we tackle in this paper are at the 2nd level of the Polynomial Hierarchy.

The aim of this paper is to introduce a solver that can compute PNEs and core elements of BGs, with the option of additionally enforcing Pareto optimality or taking into account constraints (Bonzon, Lagasquie-Schiex, and Lang 2012) and costs (Dunne et al. 2008). Our solver, available online[2], is based on disjunctive ASP, a form of declarative programming (Baral 2003). In recent years, a range of Σ_2^{P} problems have been identified on which ASP solvers outperform other state-of-the-art methods (Faber, Leone, and Ricca 2005). For our BG solver, we use the state-of-the-art ASP solver DLV to compute answer sets.

Computing PNEs and Core Elements of BGs with Disjunctive ASP Programs

We first show how to compute PNEs of a BG by associating it with a disjunctive ASP program. To this end, we use the saturation technique (Eiter, Gottlob, and Mannila 1997; Baral 2003) to compare answer sets and select the desired solutions. In our case, answer sets correspond to strategy profiles. Intuitively, the idea is to create a program with 3 parts: (i) a part describing strategy profiles and checking the satisfaction of the agents' goals, (ii) a part describing alternative strategies for all agents and checking the corresponding satisfaction of the agents' goals, (iii) a part checking whether or not the agents can improve their utility by deviating from the strategy profile in the first part and selecting the PNEs by saturation.

Let $G = (N, V, \pi, \Phi)$ be a BG. We start by adding the facts $\mathit{agent}(i) \leftarrow$ for every $i \in N$ and $\mathit{action}(p) \leftarrow$ for every $p \in V$ to the first program part $\mathcal{P}_1$. Next, we add a rule to express that every $p \in V$ is either undertaken or not:

$$\mathit{act}(P) \vee \neg\mathit{act}(P) \leftarrow \mathit{action}(P) \tag{1}$$

The capital letter P is a variable, and the rule means that if there exists a constant p such that the literal $\mathit{action}(p)$ is true, then either $\mathit{act}(p)$ or $\neg\mathit{act}(p)$ should be true. For every agent $i \in N$, we add a 'rule' that checks whether its goal is satisfied or not:

$$\mathit{goal}(i) \leftarrow \varphi_i(\mathit{act}(V)) \tag{2}$$

The notation $\varphi_i(\mathit{act}(V))$ represents the formula φ_i in which every $p \in V$ is replaced by $\mathit{act}(p)$. Note that the syntax of ASP does not allow an arbitrary formula in the body, but we translate these 'rules' into valid ASP rules. First our solver transforms the goals of the agents into negation-normal form (NNF). Then 'rules' of the form $\mathit{goal} \leftarrow \varphi$, with φ a propositional formula in NNF, are recursively translated to ASP rules by introducing new atoms. For instance, a 'rule' $\mathit{goal} \leftarrow ((a \wedge b) \vee \neg c) \wedge d$ is replaced with the set of rules $\{\mathit{goal} \leftarrow z, d; z \leftarrow a, b; z \leftarrow \neg c\}$, with z a newly introduced atom. We refer to the solver's implementation[2] for more details.

The second program part is denoted as $\mathcal{P}_2$. We add the following rules to $\mathcal{P}_2$, stating that every action is either undertaken or not in the alternative strategies:

$$\mathit{act'}(P) \vee \mathit{nact'}(P) \leftarrow \mathit{action}(P) \tag{3}$$

Using saturation requires simulating the strong negation $\neg$ with a prefix '*n*'. The intuitive idea is to make all the literals of $\mathcal{P}_2$ true in the third program part, therefore using $\neg\mathit{act'}(P)$ instead of $\mathit{nact'}(P)$ would lead to contradictions. By definition, the utility of every agent who individually deviates from a PNE cannot be strictly higher than its utility in the PNE. To know if the strategy s_i is a best response to s_{-i} for agent i, it suffices that either $u_i(s_{-i}, s_i) = 1$ or $u_i(s_{-i}, s_i') = 0$, $\forall s_i' \subseteq \pi_i$. Therefore it suffices to know whether φ_i is false for i's alternative strategy in $\mathcal{P}_2$. To this end, we add the following 'rules' to $\mathcal{P}_2$ for every $i \in N$:

$$\mathit{ngoal'}(i) \leftarrow n\varphi_i(\mathit{act}(\pi_{-i}), \mathit{nact'}(\neg\pi_i), \mathit{act'}(\pi_i)) \tag{4}$$

with $n\varphi_i$ the notation for a formula, equivalent to $\neg\varphi_i$, in NNF. The notation $n\varphi_i(\mathit{act}(\pi_{-i}), \mathit{nact'}(\neg\pi_i), \mathit{act'}(\pi_i))$ is the formula $n\varphi_i$ in which every occurrence of $p \in \pi_{-i}$ is replaced by $\mathit{act}(p)$, every occurrence of $\neg p$ with $p \in \pi_i$ is replaced by $\mathit{nact'}(p)$ and every other occurrence of $p \in \pi_i$ is replaced by $\mathit{act'}(p)$. As for (2), we translate (4) into valid ASP rules. With X' we denote the set of all newly introduced atoms during the translation of (4).

The third program part $\mathcal{P}_3$ checks whether the strategy of agent i chosen in $\mathcal{P}_1$ is a better response than its alternative strategy in $\mathcal{P}_2$, given the strategies of the other agents in $\mathcal{P}_1$. If so, we derive $\mathit{pleased}(i)$ with the following rules of $\mathcal{P}_3$:

$$\mathit{pleased}(I) \leftarrow \mathit{goal}(I); \quad \mathit{pleased}(I) \leftarrow \mathit{ngoal'}(I) \tag{5}$$

[1] This is also known as $\mathrm{NP}^{\mathrm{NP}}$-complete, a complexity class at the 2nd level of the Polynomial Hierarchy.

[2] http://www.cwi.ugent.be/BooleanGamesSolver.html.

If all agents have a better response in $\mathcal{P}_1$ than in $\mathcal{P}_2$, we derive *sat* using the following rule of $\mathcal{P}_3$, in which the body literals are compactly denoted in a set. Moreover, we exclude answer sets in which *sat* is not derived:

$$sat \leftarrow \{pleased(i) \mid i \in N\}; \qquad \leftarrow not\ sat \qquad (6)$$

As part of the saturation technique, we set the literals introduced in $\mathcal{P}_2$ to true if *sat* is derived, by adding the following rules to $\mathcal{P}_3$ for every $x \in X'$:

$$act'(P) \leftarrow sat, action(P); \quad ngoal'(I) \leftarrow sat, agent(I)$$
$$nact'(P) \leftarrow sat, action(P); \qquad x \leftarrow sat \qquad (7)$$

Together with (5), (7) implies that all literals of the form *pleased* will also be made true if *sat* is derived. The entire program $\mathcal{P}_1 \cup \mathcal{P}_2 \cup \mathcal{P}_3$ is called the *PNE-program induced by* G. Note that the Σ_2^{P}-complexity of deciding whether there is a PNE in a BG matches the Σ_2^{P}-complexity of our grounded disjunctive ASP programs (Baral 2003). We can prove that this ASP encoding is correct, i.e. that there is a one-to-one correspondence between the answer set of the induced PNE-program and the PNEs of the BG.

Adjusting the ASP program, we obtain an analogous result for core elements. Both encodings are generalized to an extended BG framework, allowing constraints on the possible strategies of the agents (Bonzon, Lagasquie-Schiex, and Lang 2012) and costs, imposed on agents depending on their chosen strategy (Dunne et al. 2008). Similarly, we have extended our encoding to additionally enforce Pareto optimality on the solutions. Due to space constraints, we refer to the implementation of our solver for more details.

Experiments

Experimental Set-up

We evaluate the performance of our ASP based method on instances of a structured problem, which we call *project BGs*. The problem consists of n persons, who can work on several projects. Depending on the project, each person has his preferences concerning the people he might collaborate with — partners — and the people he does not want to collaborate with — anti-partners. Someone who is not a partner, is a non-partner. Partnership and anti-partnership relations are anti-reflexive, not necessarily symmetric and project dependent. The project BGs are inspired by a BG about people being invited to a party (Bonzon, Lagasquie-Schiex, and Lang 2007; Dunne et al. 2008). We include 3 parameters for project BGs: (i) the number of agents, (ii) the number of projects and (iii) the probability of an agent being another agent's partner and the probability of a non-partner being an anti-partner. In a project BG, every agent i controls an action variable p_i^m per project m, interpreted as joining the project. For every project, every agent is randomly assigned one of 13 types, determined by three parameters: (i) personal preference: (a) join the project, (b) do not join, and (c) no preference; (ii) positive partnership (only relevant if level 1 is not (b)): (a) no condition, (b) only join a project if at least one partner joins, and (c) only join a project if all partners join; and (iii) negative partnership (only relevant if level 1 is not (b)): (a) no condition, and (b) only join the project if no anti-partners join. For each agent, we (uniformly) choose the personal preference, and based on this choice we (uniformly) choose among the allowed positive and negative partnerships. The type of an agent influences its goal. For example, agent i of type (c,b,b) has two partners j and k and one anti-partner l for a project m. Then the sub-goal of agent i corresponding to project m becomes $(p_i^m \wedge (p_j^m \vee p_k^m) \wedge \neg p_l^m) \vee \neg p_i^m$. So either agent i joins the project with j or k and without l, or agent i does not join. The overall goal of an agent is formed as the conjunction of the sub-goals corresponding to all the projects. All problem instances, problem generators and solvers have been made available at http://www.cwi.ugent.be/BooleanGamesSolver.html. The measurements have been performed on a dual CPU system with two 2.4GHz Intel Xeon six core E5-4610 processors and 8GB RDIMM.

Since there are currently no solvers for general BGs, there are only few options for a baseline. We consider two baselines: an NFG solver and a heuristic approach. In this paper we use on the one hand a standard approach to compute PNEs of NFGs in ASP (De Vos and Vermeir 1999), which we call the NFG baseline. To use it on BGs, we need to translate the BGs to NFGs, compute the PNEs and translate these PNEs back to BG format. Our second baseline uses the stochastic and iterative algorithm WSLpS on BGs as suggested in (De Clercq et al. 2014), but after each iteration, we verify whether the obtained strategy profile is a PNE or core element. For some classes of BGs, it can be shown that this algorithm is guaranteed to converge to a PNE without checking the best response condition (De Clercq et al. 2014) but in general it should be regarded as a heuristic method. Since WSLpS is stochastic, we run this baseline 25 times per BG.

Results

In a first experiment, we compute 1 PNE for project BGs with a timeout of 5 minutes. The number of agents varies from 5 to 50. For each fixed number of agents, we generate 100 BGs with 1, 2 or 3 projects and a 50% probability that an agent is another agent's partner and that a non-partner agent is an anti-partner. The median computation times, within a 95% confidence interval, are plotted in Figure 1. The results show that dASP strongly outperforms the NFG baseline. It also outperforms WSLpS if the number of agents is sufficiently high. The higher the number of projects, the smaller the number of agents at which dASP starts outperforming WSLpS. Note that for a wide range of parameters, dASP still succeeds in finding a solution for all problem instances, whereas both baselines consistently time out.

Secondly, we compute core elements instead of PNEs. Since the NFG baseline is not capable of computing core elements, we limit the experiments to WSLpS and dASP. For the same range of parameters as before, we compute 1 core element in 100 project BGs. The results are plotted in Figure 2. We observe that dASP consistently outperforms WSLpS for the more difficult problems instances.

Finally we investigate the scalability of our approach on project BGs with 2 projects, extended with costs and constraints. The constraints enforce that an agent can join at most 1 project. For both projects, an individual cost of

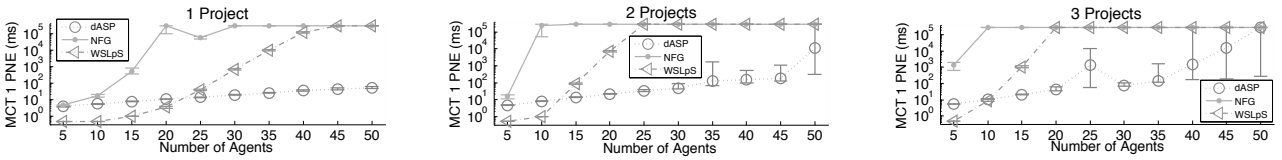

Figure 1: Median computation time (ms) of 1 PNE in project BGs.

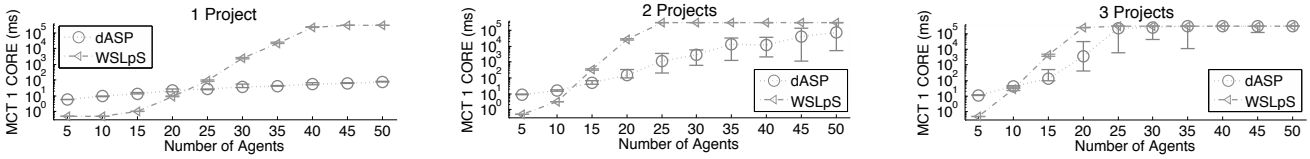

Figure 2: Median computation time (ms) of 1 core element in project BGs.

0, 1 or 2 is randomly assigned to joining the project. Not joining involves no costs. There are between 5 and 50 agents and we consider 100 BGs for each parameter setting. With a timeout of 10 minutes, respectively 1 PNE, 1 Pareto optimal PNE (POPNE), 1 core element or 1 Pareto optimal core element (POCORE) is computed. Note that no comparison with the baselines is made, since those are not equipped to deal with costs and constraints, nor to compute Pareto optimal solutions. The results are plotted in Figure 3. As expected, computing a PNE takes less time than computing a core element and enforcing Pareto optimality further increases the computation time of a PNE. For core elements, enforcing Pareto optimality does not demand more time. Although the computational cost is generally higher for stronger solution concepts or for BGs involving costs and constraints, our method remains suitable for medium-sized problems.

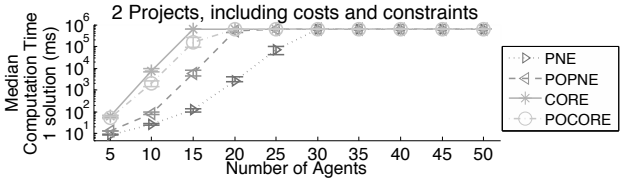

Figure 3: Computation of 1 solution in project BGs.

Conclusion

We proposed a method based on disjunctive ASP for finding the solutions of a BG. To the best of our knowledge, this is the first solver which is capable of handling general BGs (apart from methods based on an exponential translation to NFGs). Our solver can compute (Pareto optimal or arbitrary) PNEs and core elements, even in the presence of costs and constraints. Using a class of structured problems, called project BGs, we showed that our disjunctive ASP based method outperforms both a NFG based method and a heuristic method called WSLpS on sufficiently difficult problems, although for small instances WSLpS was faster on average. We also demonstrated the effectiveness of our method to find solutions under the presence of costs and constraints, and when Pareto optimality is required.

References

Baral, C. 2003. *Knowledge Representation, Reasoning and Declarative Problem Solving*. New York: Cambridge University Press.

Bonzon, E.; Lagasquie-Schiex, M.-C.; Lang, J.; and Zanuttini, B. 2006. Boolean games revisited. In *Proc. ECAI*, 265–269. ACM.

Bonzon, E.; Lagasquie-Schiex, M.-C.; and Lang, J. 2007. Dependencies between players in Boolean games. In *Proc. ECSQARU*, 743–754. Springer.

Bonzon, E.; Lagasquie-Schiex, M.-C.; and Lang, J. 2012. Effectivity functions and efficient coalitions in Boolean games. *Synthese* 187:73–103.

De Clercq, S.; Bauters, K.; Schockaert, S.; Mihaylov, M.; De Cock, M.; and Nowé, A. 2014. Decentralized computation of Pareto optimal pure Nash equilibria of Boolean games with privacy concerns. In *Proc ICAART '14*. Forthcoming.

De Vos, M., and Vermeir, D. 1999. Choice logic programs and Nash equilibria in strategic games. In *Proc. CSL*, 266–276. Springer.

Dunne, P. E., and Wooldridge, M. 2012. Towards tractable Boolean games. In *Proc. AAMAS*, 939–946. IFAAMAS.

Dunne, P.; van der Hoek, W.; Kraus, S.; and Wooldridge, M. 2008. Cooperative Boolean games. In *Proc. AAMAS*, volume 2, 1015–1022. IFAAMAS.

Eiter, T.; Gottlob, G.; and Mannila, H. 1997. Disjunctive datalog. *ACM T. Database Syst.* 22(3):364–418.

Faber, W.; Leone, N.; and Ricca, F. 2005. Solving hard problems for the 2nd level of the Polynomial Hierarchy: Heuristics and benchmarks. *Intelligenza Artificiale* 2(3):21–28.

Gottlob, G.; Greco, G.; and Scarcello, F. 2003. Pure Nash equilibria: hard and easy games. In *Proc. TARK*, 215–230. ACM.

Harrenstein, P.; van der Hoek, W.; Meyer, J.-J.; and Witteveen, C. 2001. Boolean games. In *Proc. TARK*, 287–298. Morgan Kaufmann Publishers Inc.

Wooldridge, M.; Endriss, U.; Kraus, S.; and Lang, J. 2013. Incentive engineering for Boolean games. *Artif. Intell.* 195:418–439.