Exact and heuristic methods for solving Boolean games

Sofie De Clercq , Kim Bauters, Steven Schockaert, Mihail Mihaylov, Ann Nowé, Martine De Cock

Autonomous Agents and Multi-Agent Systems, 2015, 10.1007/s10458-015-9313-5

Autonomous Agents and Multi-Agent Systems manuscript No.
(will be inserted by the editor)

Exact and Heuristic Methods for Solving Boolean Games

Sofie De Clercq · Kim Bauters · Steven Schockaert · Mihail Mihaylov · Ann Nowé · Martine De Cock

Received: date / Accepted: date

Abstract Boolean games are a framework for reasoning about the rational behavior of agents whose goals are formalized using propositional formulas. Compared to normal form games, a well-studied and related game framework, Boolean games allow for an intuitive and more compact representation of the agents' goals. So far, Boolean games have been mainly studied in the literature from the Knowledge Representation perspective, and less attention has been paid on the algorithmic issues underlying the computation of solution concepts. Although some suggestions for solving specific classes of Boolean games have been made in the literature, there is currently no work available on the practical performance. In this paper, we propose the first technique to solve general Boolean games that does not require an exponential translation to normal-form games. Our method is based on disjunctive answer set programming and computes solutions (equilibria) of arbitrary Boolean games. It can be applied to a wide variety of solution concepts, and can naturally deal with extensions of Boolean games such as constraints and costs. We present detailed experimental results in which we compare the proposed method against a number of existing methods for solving specific classes of Boolean games, as well

This research is funded by the Research Foundation Flanders (FWO).

S. De Clercq · M. De Cock
Dept. Applied Mathematics, CS & Statistics, Ghent Universtiy, Ghent, Belgium
E-mail: sofier.declercq@ugent.be,martine.decock@ugent.be

K. Bauters
School of Electronics, Electrical Engineering and CS, Queen's University, Belfast, UK
E-mail: k.bauters@qub.ac.uk

S. Schockaert
School of Computer Science and Informatics, Cardiff University, Cardiff, UK
E-mail: s.schockaert@cs.cardiff.ac.uk

M. Mihaylov · A. Nowé
Artificial Intelligence Lab, Vrije Universiteit Brussel, Brussel, Belgium
E-mail: mmihaylo@vub.ac.be,ann.nowe@vub.ac.be

M. De Cock
Center for Data Science, University of Washington, Tacoma, USA
E-mail: mdecock@uw.edu

as adaptations of methods that were initially designed for normal-form games. We found that the heuristic methods that do not require all payoff matrix entries performed well for smaller Boolean games, while our ASP based technique is faster when the problem instances have a higher number of agents or action variables.

Keywords Boolean games · Heuristic methods · Answer set programming

1 Introduction

Boolean games are a compact game-theoretic framework that uses propositional formulas to represent agents' goals [24]. In contrast, in normal form games, these goals are encoded implicitly in the form of a payoff or utility function, which might complicate the understanding of the goal in games with many possible outcomes. We illustrate the concept of a Boolean game with the following example [8].

Example 1 (Boolean game G_1) Consider a set of agents $N = \{1, 2, 3\}$ and a set of action variables $V = \{p_1, p_2, p_3\}$. Agent 1 controls p_1, agent 2 controls p_2 and agent 3 controls p_3. Agent 1's goal is to make the formula $\varphi_1 = \neg p_1 \vee (p_1 \wedge p_2 \wedge \neg p_3)$ true, whereas agents 2 and 3 respectively aim to make $\varphi_2 = (p_1 \leftrightarrow (p_2 \leftrightarrow p_3))$ and $\varphi_3 = ((p_1 \wedge \neg p_2 \wedge \neg p_3) \vee (\neg p_1 \wedge p_2 \wedge p_3))$ true.

A possible intuition underlying this game could be the following: three persons 1, 2 and 3 can individually decide to go to a bar (set their action variable to true) or to stay at home (set their action variable to false). These persons have no a priori preference for going to the bar or staying at home, as long as their goal is fulfilled (the propositional formula is true). Person 1 either wants to meet person 2 without person 3 or wants to stay at home. Person 2 either wants to meet both the first and third person or wants just one person to go to the bar. The goal of person 3 is to either only meet the second person or to let the first person be alone in the bar. Note that the equivalent normal form game would require two 2×2 matrices or one 3-dimensional $2 \times 2 \times 2$ matrix of triplets to represent the payoff of all agents for every outcome (1 if the goal is fulfilled, 0 otherwise). In general, agents in Boolean games aim to satisfy their individual goal, which is formulated as a propositional combination of the action variables in the game.

In this paper we tackle the problem of computing solutions of Boolean games. Two well-known solution concepts in game theory are pure Nash equilibria (PNEs) and core elements. A PNE is characterized by the fact that no agent is better off by individually deviating from the PNE. A core element is characterized by the fact that no coalition of agents can jointly deviate and all be strictly better off. In Example 1 a core element – and thus also a PNE – is reached when person 3 is the only one to go to the bar: in that case, no group of agents can jointly change actions and all be better off. The two main problems we focus on in our experiments are (i) the computation of a sample PNE and (ii) the computation of a sample core element, with a Boolean game as the problem input. Additionally, we also investigate the computation of Pareto optimal PNEs and core elements. An outcome is Pareto optimal if there exists no outcome such that all agents are at least as well off and at least one agent is strictly better off.

The strength of Boolean games lies in their transparent and compact representation, since Boolean games mention the goal of each agent but do not require

utility functions to be explicitly mentioned for every strategy profile. Indeed, the utility of a strategy can be derived from the agents' goals. This advantage in the form of a more compact notation also has a downside: computing a PNE or a core element is harder in Boolean games than in most other game representations, as we explain in the following paragraph. This higher complexity is undoubtedly part of the reason why, to the best of our knowledge, no general methods for computing a PNE or a core element of a Boolean game have yet been proposed.

In normal form games, the problem of 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 agent, (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 [21,22]. Deciding whether a Boolean game has a PNE, on the other hand, is Σ_2^{P}-complete[1], even for 2-player zero-sum games [8]. In zero-sum games, the utility of all agents sums up to 0 for every outcome. Deciding whether there exists a core element of a Boolean game is also Σ_2^{P}-complete [15].

Although no methods for solving general Boolean games have yet been proposed, two categories of existing techniques can be applied to this computational problem. First, given that translations from Boolean games to normal form games are available, we can readily use solvers for normal form games. However, such a translated game is exponential in the number of action variables. Especially methods that require the computation of the entire payoff matrix are likely to only be suitable when the number of agents and actions is sufficiently small. Methods that avoid the usage of all payoff matrix entries, such as tabu best-response search [28], might be more suitable for Boolean games. A detailed description of techniques in this category is given in Section 4.2. Second, a number of authors have proposed methods for special sub-classes of Boolean games, e.g. to find PNEs [5, 6] or to find Pareto optimal outcomes for a certain class of Boolean games [15]. It is important to note that none of the techniques in this category can compute a PNE of general Boolean games. More details on these techniques and the differences with the methods proposed in this paper are discussed in Section 4.1.

The aim of this paper is twofold. First, we introduce a novel method for computing solutions of general Boolean games that does not require an exponential translation to normal-form games. Our technique is based on disjunctive answer set programming (ASP) and is able to compute a wide variety of solution concepts for arbitrary Boolean games due to its flexibility. Moreover, it can for instance easily be extended to take constraints [7], costs [15] or prioritized goals [9] into account.

The second aim of this paper is to experimentally assess the strengths and weaknesses of the three aforementioned classes of solution methods: the ASP method we introduce in this paper, exact methods for particular classes of Boolean games and (mainly heuristic) methods that were initially designed for normal-form games. To the best of our knowledge, no such evaluation has been done yet. All methods and data discussed in this paper have been made available online [13]. As such, our work yields the first benchmarks and implementations for solving Boolean games.

[1] The class of Σ_2^{P}-complete or $\mathrm{NP}^{\mathrm{NP}}$-complete problems – at the 2^{nd} level of the Polynomial Hierarchy – consists of all decision problems that can be solved in polynomial time by a non-deterministic Turing machine with the help of an NP oracle having unitary cost [27].

The paper is organized as follows. We first give some background on Boolean games. In Section 3, we discuss our method to solve Boolean games. In addition, we prove the correctness and discuss several extensions, enlarging the set of problems that our technique can tackle. In Section 4 we discuss several methods that can alternatively be used to address the considered problems. All these techniques are evaluated in our experiments in Section 5. Finally, we conclude the paper with a discussion in Section 6.

This paper extends our previous work [12,11]. In particular, we introduce a new variant of the WSLpS algorithm to extend its range of application to general Boolean games. Moreover, we have extended our experimental set-up, by adding new benchmarks and implementing additional methods for comparison.

2 Background on Boolean Games

We write L_V for the propositional language associated with a set of atomic propositions V in the usual way, i.e. L_V contains the following formulas:

- every atomic proposition of V,
- the logical constants $\bot$ and $\top$ to denote contradiction and tautology, respectively, and
- the formulas $\varphi \to \psi$, $\neg\varphi$, $\varphi \leftrightarrow \psi$, $\varphi \wedge \psi$ and $\varphi \vee \psi$ for every φ, $\psi \in L_V$.

An interpretation of V is defined as a subset ξ of V, with the convention that all atoms in ξ are considered to be true ($\top$) and all atoms in $V \setminus \xi$ are considered to be false ($\bot$). An interpretation can be used to derive the truth-value of any $\varphi \in L_V$ in the usual way. If a formula φ in L_V is satisfied by an interpretation ξ, we denote this as $\xi \models \varphi$. A formula $\varphi \in L_V$ is independent of $p \in V$ if there exists a logically equivalent formula ψ in which p does not occur. The set of variables on which φ depends, is denoted as $DV(\varphi)$. In this paper, we use the following definition of Boolean games [8].

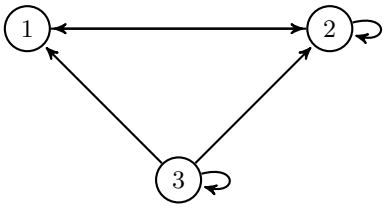
Definition 1 (Boolean Game) A Boolean game is a 4-tuple $G = (N, V, \pi, \Phi)$ with $N = \{1, \ldots, n\}$ a set of agents, V a set of atomic propositions, $\pi : N \to 2^V$ a control assignment function such that $\{\pi(1), \ldots, \pi(n)\}$ forms a partition of V, and $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ a collection of formulas in L_V.

The set V consists of all action variables. When we say that agent i undertakes action p, this means that the action variable $p \in \pi(i)$ is set to true. Similarly, not undertaking an action is considered as setting the variable to false. We adopt the notation π_i for $\pi(i)$, i.e. the set of action variables under agent i's control [8]. Since π defines a partition, every action variable is controlled by exactly one agent. The formula φ_i is the goal of agent i. For every $p \in V$ we define $\pi^{-1}(p) = i$ iff $p \in \pi_i$, hence π^{-1} maps every action variable to the agent controlling it.

Definition 2 (Relevant agents, neighbors) Let $G = (N, V, \pi, \Phi)$ be a Boolean game. The set of relevant variables for agent i is defined as $DV(\varphi_i)$. The set of relevant agents $RA(i)$ for agent i is defined as $\{\pi^{-1}(p) \in N \mid p \in DV(\varphi_i)\}$. The neighborhood of agent i is the set of agents for which i is relevant, i.e. $\mathit{Neigh}(i) = \{j \in N \mid i \in RA(j)\}$.

Note that the relevant agents for agent i are all agents controlling a variable on which agent i's goal depends [5,6]. The neighbors of i are all agents whose goal depends on a variable controlled by agent i. The dependency graph of a Boolean game is the graph (N, E), where the set of vertices N corresponds to the set of agents and where E contains an edge (i, j) for every agent j in $RA(i)$ [5,6]. We illustrate these concepts in the following example.

Example 2 (Boolean game G_2) Let G_2 be a 3-player Boolean game with $\pi_i = \{p_i\}$, $\varphi_1 = p_2$, $\varphi_2 = p_1 \vee \neg p_2$ and $\varphi_3 = p_1 \wedge p_2 \wedge p_3$. Then 1 is a relevant agent for 2, but not for itself. The neighborhoods in the game are $Neigh(1) = \{2, 3\}$, $Neigh(2) = \{1, 2, 3\}$ and $Neigh(3) = \{3\}$. The dependency graph of G_2 is:



Since there is an arrow from agent 3 to agent 1, 3 depends on 1, but not the other way around. Or similarly, 1 is relevant for 3, i.e. 3 is a neighbor of 1.

Definition 3 (Strategy profile) Let $G = (N, V, \pi, \Phi)$ be a Boolean game. An interpretation s_i of π_i is called a strategy of agent $i \in N$. A strategy profile of the Boolean game G is an n-tuple $S = (s_1, \ldots, s_n)$, with s_i a strategy of agent i for every $i \in N$.

Because π partitions V and $s_i \subseteq \pi_i$, for every $i \in N$, we can unambiguously use the set notation $\cup_{i=1}^{n} s_i \subseteq V$ for a strategy profile $S = (s_1, \ldots, s_n)$. For example, in the Boolean game G_2 from Example 2 the strategy profile $\{p_1\}$ corresponds to agent 1 setting p_1 to true, agent 2 setting p_2 to false and agent 3 setting p_3 to false. With s_{-i} we denote the projection of the strategy profile $S = (s_1, \ldots, s_n)$ 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 $(s_1, \ldots, s_{i-1}, s_i', s_{i+1}, \ldots, s_n)$. We illustrate these concepts for the Boolean game G_2 from Example 2.

Example 2 (Boolean game G_2 – continued) The strategy profile $S = (\{p_1\}, \emptyset, \{p_3\})$ $= \{p_1, p_3\}$ corresponds to the outcome in which agents 1 and 3 set their action variables to true and agent 2 sets its action variable to false. We have $s_{-1} = (\emptyset, \{p_3\})$, $s_{-2} = (\{p_1\}, \{p_3\})$ and $s_{-3} = (\{p_1\}, \emptyset)$.

The definition of a Boolean game can be extended with constraints [7]: with each agent $i \in N$ a satisfiable formula $\gamma_i \in L_{\pi_i}$ is associated. A strategy of agent i is then required to be a model of γ_i.
A Boolean utility function can be defined from the goals of the agents in a natural way.

Definition 4 (Utility function) Let $G = (N, V, \pi, \Phi)$ be a Boolean game. For every agent $i \in N$ and every strategy profile S of G, the utility function u_i is defined as $u_i(S) = 1$ iff $S \models \varphi_i$ and $u_i(S) = 0$ otherwise.

This binary utility is often seen as an obvious limitation of Boolean games. Several proposals have been made in the literature to overcome this restriction. First, Boolean games can be extended with costs, imposed on agents depending on their actions. We denote $\{\neg p \mid p \in V\}$ as $\neg V$. If G includes a cost function $c : V \cup \neg V \rightarrow \mathbb{R}^+$, then playing a certain strategy s_i imposes a cost $c_i(s_i) = \sum_{p \in s_i} c(p) + \sum_{p \in \pi_i \setminus s_i} c(\neg p)$ on agent i. The utility function of agent i is then a function which values the utility of a strategy profile S higher than a strategy profile S' iff (i) S satisfies agent i's goal and S' does not, or (ii) S and S' both satisfy agent i's goal but the cost for agent i is lower in S than in S' ($c_i(S) < c_i(S')$), or (iii) neither S nor S' satisfy agent i's goal but $c_i(S) < c_i(S')$. Some definitions impose no cost for not undertaking an action, i.e. $c(\neg p) = 0$ for every $p \in V$ [15]. A second approach to overcome the limitation of binary utility degrees is the use of compactly represented preference relations on the set of strategy profiles [9]. We demonstrate Boolean games with costs in the following example.

Example 3 (Boolean game G_3) Consider the Boolean game G_3. Two students can work on two projects: $V = \{p_1^1, p_1^2, p_2^1, p_2^2\}$. If p_i^j is true, student i works on project j. Not working on the project requires no energy (cost of 0). For the first student, working on the first project is harder than working on the second: $c(p_1^1) = 2$ and $c(p_1^2) = 1$. For the second student, the costs are the other way around. Note that, in contrast to the Boolean game in Example 1, the agents have an a priori preference on the actions, although this preference is subordinated to the satisfaction of their goal. The students can work on at most one project, i.e. $\gamma_i = \neg(p_i^1 \wedge p_i^2)$. Student 1 wants to work on either one of the projects i.e. $\varphi_1 = p_1^1 \vee p_1^2$. Student 2 wants to cooperate on any project if student 1 joins him, i.e. $((p_2^1 \rightarrow p_1^1) \wedge (p_2^2 \rightarrow p_1^2)$.

We now discuss solution concepts in Boolean games. A common, intuitive and straightforward solution concept in game theory is the notion of pure Nash equilibrium.

Definition 5 (Pure Nash Equilibrium) A strategy profile $S = (s_1, \ldots, s_n)$ for a Boolean game G is a pure Nash equilibrium (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 $s_i' \subseteq \pi_i$.

Deciding whether a Boolean game has a PNE is Σ_2^{P}-complete [8]. An alternative solution concept for Boolean games is the core, which is related to *strong Nash equilibria* [2]. Checking whether the core is non-empty is also Σ_2^{P}-complete [15].

Definition 6 (Core) 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
- all agents in C strictly prefer S' to S, i.e. they have a strictly higher utility in S' than in S.

The core of a Boolean game G is defined as the set $\mathit{Core}(G)$ of strategy profiles S that are not blocked by any non-empty coalition $C \subseteq N$.

Note that in the context of Definition 6, a coalition can be assumed to be such that all members change their actions. Indeed, in case a coalition does not satisfy this property, we can remove all agents that do not change actions and still satisfy the

conditions of Definition 6. Definition 6 also entails that, in particular, an element of $Core(G)$ is not blocked by a single agent, implying that every core element is a PNE. Another desirable property of strategy profiles is Pareto optimality.

Definition 7 (Pareto optimality) A strategy profile S is Pareto optimal if there exists no strategy profile S' such that $u_i(S) \leq u_i(S')$ for every $i \in N$ and $u_i(S) < u_i(S')$ for at least one $i \in N$.

As we show further on, the PNEs and core elements of G_1, G_2 and G_3 coincide. To demonstrate that this is not generally true, we introduce another example of Boolean games.

Example 4 (Boolean game G_4) A project is set up and 4 students have the opportunity to cooperate. Student 1 wants to join iff all other students also join, i.e. $\varphi_1 = (p_1 \wedge p_2 \wedge p_3 \wedge p_4) \vee \neg p_1$. Student 2 wants to cooperate with at least one partner, i.e. $\varphi_2 = p_2 \wedge (p_1 \vee p_3 \vee p_4)$. Student 3 also wants to join, but he wants the second or fourth student as a partner, i.e. $\varphi_3 = p_3 \wedge (p_2 \vee p_4)$. The fourth student wants to do the project by himself or does not want to do the project, i.e. $\varphi_4 = (p_4 \wedge \neg p_1 \wedge \neg p_2 \wedge \neg p_3) \vee \neg p_4$. The resulting Boolean game is denoted as G_4.

In Table 1, the PNEs and core elements of the Boolean games of Example 1, 2, 3 and 4 are listed, using set notation.

Table 1: The PNEs and core elements of the Boolean games $G_{\{1,2,3,4\}}$.

Boolean game	G_1	G_2	G_3	G_4
PNEs	$\{p_3\}$	$\emptyset, \{p_1\}, \{p_3\}, \{p_1, p_3\}, \{p_1, p_2, p_3\}$	$\{p_1^2\}$	$\emptyset, \{p_2, p_3\}$
Core elements	$\{p_3\}$	$\emptyset, \{p_1\}, \{p_3\}, \{p_1, p_3\}, \{p_1, p_2, p_3\}$	$\{p_1^2\}$	$\{p_2, p_3\}$

Recall that the goals of the agents in Example 1 are respectively $\varphi_1 = \neg p_1 \vee (p_1 \wedge p_2 \wedge \neg p_3)$, $\varphi_2 = (p_1 \leftrightarrow (p_2 \leftrightarrow p_3))$ and $\varphi_3 = ((p_1 \wedge \neg p_2 \wedge \neg p_3) \vee (\neg p_1 \wedge p_2 \wedge p_3))$. The action variable p_i represents whether agent i goes to the bar. The fact that $S = \{p_3\}$ is a PNE of G_1 means that, if the third person goes to the bar, no individual person can change his action to improve the outcome for himself. Indeed, the third person cannot improve his own situation by leaving. Similarly, the first and second person could decide (individually) to come to the bar but this isolated decision will not lead to a strictly better outcome for them, since they already reach their goal in S. For G_1, G_2 and G_3, the PNEs and core elements coincide, but in G_4 there is a unique core element, whereas there are two PNEs. The core element is the only solution that is also Pareto optimal and in which every agent reaches its goal. The PNE $\emptyset$ is a less satisfactory solution than $\{p_2, p_3\}$, since only two agents succeed in fulfilling their goal in the outcome $\emptyset$, versus four satisfied agents in $\{p_2, p_3\}$.

Alternative solution concepts in Boolean games include the weak and strong core [7], k-bounded Nash equilibria [16] and stable sets [15], although this latter term is also used to describe certain coalitions [5,6]. In this paper, we focus the discussion on PNEs, cores and Pareto optimality, since these are the most common solution concepts in the context of Boolean games. However, our reduction to

disjunctive ASP can readily be generalized to these alternative solution concepts. Still other solution concepts have been studied recently, such as verifiable equilibria [1], which require the Boolean game framework to be extended with a visibility set for every agent i. These visibility sets restrict the agents' knowledge of the game by specifying the action variables whose values can be observed by agent i. Verifiable equilibria differ from normal PNEs because, when playing the strategies corresponding to a verifiable equilibrium, it is guaranteed that the agents are able to know whether they have reached an equilibrium. Although we have not implemented these solution concepts, it is plausible that, based on the complexity of the associated decision problems, an ASP encoding can be found to compute these solutions as well.

3 Computing Solutions using Answer Set Programming

We first recall answer set programming or ASP, which is a form of declarative programming [10]. Its transparency and non-monotonic character, as well as the existence of efficient ASP solvers, make it an attractive method for solving optimization and combinatorial search problems. In recent years, a range of Σ_2^{P} problems have been identified on which ASP solvers outperform other state-of-the-art methods [18].

A ground disjunctive ASP program has *atoms*, *literals* and *rules* as building blocks. The most elementary are *atoms*, which are atomic propositions from a fixed set $\mathcal{A}$, that can be true or false. A *literal* is an atom a or a negated atom $\neg a$. Apart from strong negation, denoted as $\neg$, ASP uses a special kind of negation, called *negation-as-failure* and denoted with '*not*'. Intuitively, $\neg a$ is true when there is proof that a is not true, whereas $not\, a$ is true when there is no proof that a is true. A ground *disjunctive rule* has the following form

$$a_1 \vee \ldots \vee a_k \leftarrow b_1, \ldots, b_m, not\, c_1, \ldots, not\, c_n$$

where $a_1, \ldots, a_k, b_1, \ldots, b_m, c_1, \ldots, c_n$ are literals. We call $a_1 \vee \ldots \vee a_k$ the head of the rule while $b_1, \ldots, b_m, not\, c_1, \ldots, not\, c_n$ is called the *body*. The rule above intuitively encodes that at least one of $a_1, \ldots, a_k$ is true when all of $b_1, \ldots, b_m$ are known to be true and none of $c_1, \ldots, c_n$ is known to be true. When a rule has an empty body, we call it a *fact*; when the head is empty, it is called a *constraint*. A rule without occurrences of *not* is called a *simple disjunctive rule*. A *simple disjunctive* ASP program is a finite collection of simple disjunctive rules and similarly, a *disjunctive* ASP program $\mathcal{P}$ is a finite collection of disjunctive rules.

Example 5 Let $\mathcal{P}$ be the ASP program with the following 5 rules:

$$\begin{aligned}
man(john) &\leftarrow \\
person(john) &\leftarrow \\
person(fiona) &\leftarrow \\
woman(john) \vee child(john) &\leftarrow person(john), not\ man(john) \\
woman(fiona) \vee child(fiona) &\leftarrow person(fiona), not\ man(fiona)
\end{aligned}$$

The first 3 rules are facts, hence their heads will be in any answer set. The last two rules encode that if *john* (respectively *fiona*) are known to be persons, but not known to be men, then they are assumed to be women or children. This program contains two constants, namely *john* and *fiona*. An example of a literal in $\mathcal{P}$ is *man(john)*.

An *interpretation* I of a ground disjunctive ASP program is a consistent subset of $\mathcal{L} = \mathcal{A} \cup \neg\mathcal{A}$, with $\neg\mathcal{A} = \{\neg a \mid a \in \mathcal{A}\}$. A simple disjunctive rule $a_1 \vee \ldots \vee a_k \leftarrow b_1, \ldots, b_m$ is *satisfied* by an interpretation I when $\{a_1, \ldots, a_k\} \cap I \neq \emptyset$ or $\{b_1, \ldots, b_m\} \not\subseteq I$. A *model* of a simple disjunctive program $\mathcal{P}$ is an interpretation satisfying all rules of $\mathcal{P}$. An interpretation I is an *answer set* of a simple disjunctive program $\mathcal{P}$ iff it is a minimal model of $\mathcal{P}$, i.e. I is a model and there does not exist a strict subset of I that is also a model of $\mathcal{P}$ [19,20]. The *reduct* $\mathcal{P}^I$ of a ground disjunctive ASP program $\mathcal{P}$ w.r.t. an interpretation I is the simple disjunctive ASP program defined by:

$$\mathcal{P}^I = \{a_1 \vee \ldots \vee a_k \leftarrow b_1, \ldots, b_m \mid \{c_1, \ldots, c_n\} \cap I = \emptyset, \\ (a_1 \vee \ldots \vee a_k \leftarrow b_1, \ldots, b_m, not\, c_1, \ldots, not\, c_n) \in \mathcal{P}\}.$$

Intuitively, we guess an interpretation I and use it to remove negation-as-failure from $\mathcal{P}$, which results in the reduct $\mathcal{P}^I$. An interpretation I of a ground disjunctive ASP program $\mathcal{P}$ is an answer set of $\mathcal{P}$ iff I is an answer set of $\mathcal{P}^I$. In Example 5, the interpretation $I = \{$*man(john)*, *woman(fiona)*, *person(john)*, *person(fiona)*$\}$ is an answer set of $\mathcal{P}$. Indeed, if we compute the reduct, the fourth rule is deleted since *man(john)* is in I, i.e. the body is false. The reduct $\mathcal{P}^I$ is:

$$\begin{aligned} man(john) &\leftarrow \\ person(john) &\leftarrow \\ person(fiona) &\leftarrow \\ woman(fiona) \vee child(fiona) &\leftarrow person(fiona) \end{aligned}$$

It is clear that I is a minimal model of this simple program, hence I is an answer set of $\mathcal{P}$. Note that $I \setminus \{$*woman(fiona)*$\} \cup \{$*child(fiona)*$\}$ is another answer set of $\mathcal{P}$. Although $I \cup \{$*child(fiona)*$\}$ also satisfies the rules of $\mathcal{P}$, it is not an answer set since the minimality condition is not fulfilled.

The ASP syntax also uses variables, denoted in upper case, to write related rules in a more compact or general fashion. For instance, the last two rules in Example 5 are used to encode that any person who is not known to be a man, is assumed to be a woman or child. This can be written as:

$$woman(X) \vee child(X) \leftarrow person(X), not\, man(X)$$

with X a variable. Hence a (disjunctive) ASP rule is written as:

$$A_1 \vee \ldots \vee A_k \leftarrow B_1, \ldots, B_m, not\, C_1, \ldots, not\, C_n$$

where $A_1, \ldots, A_k, B_1, \ldots, B_m, C_1, \ldots, C_n$ are literals which may contain variables. The semantics are defined by the ground version of the rule, i.e. all ground instantiations of the rule w.r.t. the constants that appear in the program – see e.g. [10] for a thorough overview.

We now associate disjunctive ASP programs with Boolean games to compute either PNEs or to find strategy profiles in their core. Additionally, we can require these PNEs or core elements to be Pareto optimal. In our case, the answer sets correspond to strategy profiles. Intuitively, the idea is to create a program with 3 parts: (i) a part generating a strategy profile and checking the satisfaction of the agents' goals, (ii) a part generating 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 or core elements by saturation, a powerful technique for implementing optimization problems in ASP [17]. We will demonstrate our approach in detail next.

3.1 Computing PNEs with Disjunctive ASP

Let $G = (N, V, \pi, \Phi)$ be a Boolean game. We define the first program part $\mathcal{P}_1$ and start by adding the facts $agent(1..n) \leftarrow$ and $action(p) \leftarrow$ for every $p \in V$. The 'rule' $agent(1..n) \leftarrow$ is an abbreviation for the n facts $agent(1) \leftarrow; \ldots; agent(n) \leftarrow$. Next, we add a rule that expresses that every $p \in V$ is either undertaken or not:

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

For every agent $i \in N$, we add a 'rule' that checks whether its goal is satisfied or not:

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

The notation $\varphi_i(act(V))$ represents the formula φ_i in which every occurrence of $p \in V$ is replaced by $act(p)$. Note that (2) is not a valid ASP rule, since ASP does not allow an arbitrary formula in the body. However, as we explain next, we can easily translate these 'rules' into valid ASP rules. Our solver first transforms the goals of the agents into negation-normal form (NNF). 'Rules' of the form $goal \leftarrow \varphi$, with φ a propositional formula in NNF, can then be recursively translated to ASP rules by introducing new atoms. The objective is to write rules that derive $goal$ whenever the formula φ is true. For instance, a 'rule' $goal \leftarrow ((a \wedge b) \vee \neg c) \wedge d$ can be replaced with the set of rules $\{goal \leftarrow x, d; x \leftarrow a, b; x \leftarrow \neg c\}$, with x a newly introduced atom. To this end, a recursive method transforms a 'rule' $goal \leftarrow f_1(y_1, f_2(y_2, \ldots, f_{k-1}(y_{k-1}, f_k(y_k, y_{k+1}))\ldots))$ into a set of valid ASP rules, with f_i a binary logical operator $\in \{\wedge, \vee\}$ for every $i \in \{1, \ldots, k\}$ and y_i a literal for every $i \in \{1, \ldots, k+1\}$. The first step of this recursive method yields the 'rule' $goal \leftarrow f_1(y_1, f_2(y_2, \ldots, f_{k-1}(y_{k-1}, x_k)\ldots))$ together with the rules in the middle or right column of Table 2, depending on f_k. Here x_k is a newly introduced atom.
We now recursively apply this approach for each obtained 'rule' of the form

$$goal \leftarrow f_1(y_1, f_2(y_2, \ldots, f_{k-1}(y_{k-1}, x_k)\ldots))$$

until the remaining rule with $goal$ in the head is a valid ASP rule. Note that when $f_i = f_{i+1} = \ldots f_j$, we can take advantage of the associativity of $\wedge$ and $\vee$ to obtain a slightly more compact translation, as we illustrate in Example 6 below. It is easy to see that every 'rule' of the form (2) with k binary operators ($\vee$ and $\wedge$) is

Table 2: Translation rules

logical operator	$f_k = \wedge$	$f_k = \vee$
ASP rules	$x_k \leftarrow y_k, y_{k+1}$	$x_k \leftarrow y_k$ $x_k \leftarrow y_{k+1}$

translated to a set of at most $2k$ ASP rules and that the translation involves at most k newly introduced atoms. With X we denote the set of all atoms that were introduced during the translation of the rules of the form (2).

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

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

Note that, in this program part, we simulate the strong negation with a prefix 'n'. The intuitive idea is to make all the literals of $\mathcal{P}_2$ true in the final part of the program, but this would lead to contradictions if we used $\neg act'(P)$ instead of $nact'(P)$. The literals of $\mathcal{P}_2$ correspond to an alternative strategy for every agent in the Boolean game. Further on, we will explain how we use saturation in the third program part to ensure that not one, but all possible alternative strategies are taken into account.

The definition of PNEs states that the utility of every agent that 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 the alternative strategy of i in the second program part. To this end, we add the following rules to $\mathcal{P}_2$ for every $i \in N$:

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

with $\sim\varphi_i$ the notation for a formula, equivalent to $\neg\varphi_i$, in NNF. We introduce the notation $\sim\varphi_i(act(\pi_{-i}), nact'(\neg\pi_i), act'(\pi_i))$ for the formula $\sim\varphi_i$ in which every occurrence of $p \in \pi_{-i}$ is replaced by $act(p)$, every occurrence of $\neg p$ with $p \in \pi_i$ is replaced by $nact'(p)$ and every other occurrence of $p \in \pi_i$ is replaced by $act'(p)$. Note that (4) implicitly encodes the control assignment function of the Boolean game. 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 $\mathcal{P}_3$ is used to check whether the strategy of agent i chosen in $\mathcal{P}_1$ is a better response than its alternative strategy defined in $\mathcal{P}_2$, given the strategies of the other agents chosen in $\mathcal{P}_1$. If so, we derive $pleased(i)$ with the following rules of $\mathcal{P}_3$:

$$\begin{aligned} pleased(I) &\leftarrow goal(I) \\ pleased(I) &\leftarrow ngoal'(I) \end{aligned} \tag{5}$$

These rules are added only once since I is a variable, whereas e.g. rule (4) is added for every agent $i \in N$. If all agents have a better response in $\mathcal{P}_1$ than in $\mathcal{P}_2$, we

derive *sat* using the following rule of $\mathcal{P}_3$, where we identify the body with a set of literals for the ease of presentation:

$$sat \leftarrow \{pleased(i) \mid i \in N\} \tag{6}$$

Moreover, we use the following rule to exclude answer sets in which *sat* is not derived:

$$\leftarrow not\ sat \tag{7}$$

As part of the saturation technique, which we describe next, 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 $x \in X'$:

$$\begin{aligned} act'(P) &\leftarrow sat, action(P) \\ nact'(P) &\leftarrow sat, action(P) \\ x &\leftarrow sat \\ ngoal'(I) &\leftarrow sat, agent(I) \end{aligned} \tag{8}$$

Together with (5), (8) implies that all literals of the form *pleased* will also be made true if *sat* is derived. The intuition of the saturation is to impose that *sat* should be in every answer set with rule (7). Since rule (6) is the only rule in the program that can derive *sat*, *pleased*(i) should also be in every answer set, which is only the case if all agents play a better response in $\mathcal{P}_1$ than in $\mathcal{P}_2$, due to (5). The rules (8) ensure that every possible alternative strategy encoded by $\mathcal{P}_2$ is in every answer set. By definition of the reduct of an ASP program, *sat* needs to be in the minimal model of the reduct in order to obtain an answer set. Intuitively, there will exist an answer set if and only if there exists no alternative strategy for an agent such that it plays a better response than in $\mathcal{P}_1$.

Remark 1 Note that while the heads of the last two rules in (8) will generally follow from the saturation of the literals $act'(.)$ and $act'(.)$, these rules are not redundant. In particular consider a game with an agent i whose goal is $p_2 \vee (p_3 \wedge p_4)$, where none of the occurring action variables is controlled by i itself. The negation of the goal is $\neg p_2 \wedge (\neg p_3 \vee \neg p_4)$. We would introduce an extra variable x to model the disjunction, i.e. $x \leftarrow \neg act(p_3)$ and $x \leftarrow \neg act(p_4)$. Clearly, x would not follow from the atoms $act'(.)$ and $act'(.)$, and neither would $ngoal'(i)$.

We denote the entire program $\mathcal{P}_1 \cup \mathcal{P}_2 \cup \mathcal{P}_3$ as $\mathcal{P}$ and call it the *PNE-program induced by* G. Note that the Σ_2^{P}-complexity of deciding whether there is a PNE in a Boolean game matches the Σ_2^{P}-complexity of our grounded disjunctive ASP programs [3]. If m is the maximum number of binary operators ($\vee$ and $\wedge$) appearing in the NNF of the goals in G, and p is the maximum number of action variables per agent, i.e. $p = \max_{i \in N} |\pi_i|$, then the number of ASP rules in the ungrounded program $\mathcal{P}$ is bounded by $n + pn + 9 + 5mn = (p + 5m + 1)n + 9$. Hence the size of the translation to ASP is polynomial in the size of the original problem description as a Boolean game.

Example 6 Reconsider the Boolean game of Example 1. The associated program (syntactically not yet a valid ASP program) consists of $\mathcal{P}_1$:

$$agent(1..3) \leftarrow; \qquad action(p_1) \leftarrow; \qquad action(p_2) \leftarrow; \qquad action(p_3) \leftarrow$$
$$act(P) \vee \neg act(P) \leftarrow action(P)$$
$$goal(1) \leftarrow \neg act(p_1) \vee (act(p_1) \wedge act(p_2) \wedge \neg act(p_3))$$
$$goal(2) \leftarrow (act(p_1) \wedge ((act(p_2) \wedge act(p_3)) \vee (\neg act(p_2) \wedge \neg act(p_3)))) \vee (\neg act(p_1) \wedge (\neg act(p_2) \vee \neg act(p_3)) \wedge (act(p_2) \vee act(p_3)))$$
$$goal(3) \leftarrow (act(p_1) \wedge \neg act(p_2) \wedge \neg act(p_3)) \vee (\neg act(p_1) \wedge act(p_2) \wedge act(p_3))$$

After the translation into valid ASP rules, the rules with heads $goal(i)$ from $\mathcal{P}_1$ become:

$$goal(1) \leftarrow \neg act(p_1); \qquad goal(1) \leftarrow act(p_1), act(p_2), \neg act(p_3)$$
$$x_3 \leftarrow act(p_2), act(p_3); \qquad x_3 \leftarrow \neg act(p_2), \neg act(p_3)$$
$$x_2 \leftarrow \neg act(p_2); \qquad x_2 \leftarrow \neg act(p_3)$$
$$x_1 \leftarrow act(p_2); \qquad x_1 \leftarrow act(p_3)$$
$$goal(2) \leftarrow act(p_1), x_3; \qquad goal(2) \leftarrow \neg act(p_1), x_1, x_2$$
$$goal(3) \leftarrow act(p_1), \neg act(p_2), \neg act(p_3); \qquad goal(3) \leftarrow \neg act(p_1), act(p_2), act(p_3)$$

Program part $\mathcal{P}_2$ contains the following rules:

$$act'(P) \vee nact'(P) \leftarrow action(P)$$
$$ngoal'(1) \leftarrow act'(p_1) \wedge (nact'(p_1) \vee \neg act(p_2) \vee act(p_3))$$
$$ngoal'(2) \leftarrow (\neg act(p_1) \vee ((nact'(p_2) \vee \neg act(p_3)) \wedge (act'(p_2) \vee act(p_3)))) \wedge (act(p_1) \vee (act'(p_2) \wedge act(p_3)) \vee (nact'(p_2) \wedge \neg act(p_3)))$$
$$ngoal'(3) \leftarrow (\neg act(p_1) \vee act(p_2) \vee act'(p_3)) \wedge (act(p_1) \vee \neg act(p_2) \vee nact'(p_3))$$

The rules with heads $ngoal'(i)$ from $\mathcal{P}_2$ become:

$$x'_1 \leftarrow nact'(p_1); \qquad x'_1 \leftarrow \neg act(p_2); \qquad x'_1 \leftarrow act(p_3)$$
$$ngoal'(1) \leftarrow act'(p_1), x'_1; \qquad x'_3 \leftarrow nact'(p_2); \qquad x'_3 \leftarrow \neg act(p_3)$$
$$x'_2 \leftarrow act'(p_2); \qquad x'_2 \leftarrow act(p_3); \qquad x'_5 \leftarrow act(p_1)$$
$$x'_5 \leftarrow nact'(p_2), \neg act(p_3); \qquad x'_5 \leftarrow act'(p_2), act(p_3); \qquad x'_4 \leftarrow \neg act(p_1)$$
$$x'_4 \leftarrow x'_2, x'_3; \qquad ngoal'(2) \leftarrow x'_4, x'_5; \qquad x'_6 \leftarrow \neg act(p_1)$$
$$x'_6 \leftarrow act(p_2); \qquad x'_6 \leftarrow act'(p_3); \qquad x'_7 \leftarrow act(p_1)$$
$$x'_7 \leftarrow \neg act(p_2); \qquad x'_7 \leftarrow nact'(p_3); \qquad ngoal'(3) \leftarrow x'_6, x'_7$$

The final program part $\mathcal{P}_3$ is given by:

$$pleased(I) \leftarrow goal(I); \qquad pleased(I) \leftarrow ngoal'(I)$$
$$sat \leftarrow pleased(1), pleased(2), pleased(3); \qquad \leftarrow not\ sat$$

$$act'(P) \leftarrow sat, action(P) \qquad nact'(P) \leftarrow sat, action(P)$$
$$x'_1 \leftarrow sat; \quad x'_2 \leftarrow sat; \quad x'_3 \leftarrow sat$$
$$x'_4 \leftarrow sat; \quad x'_5 \leftarrow sat; \quad x'_6 \leftarrow sat$$
$$x'_7 \leftarrow sat; \qquad ngoal'(I) \leftarrow sat, agent(I)$$

The only answer set of the induced PNE-program is the one containing $\neg act(p_1)$, $\neg act(p_2)$, $act(p_3)$, $goal(1)$, $goal(2)$, $\neg goal(3)$, corresponding to the unique PNE in which agent 1 sets action variable p_1 to false, agent 2 sets p_2 to false and agent 3 sets p_3 to true. Consequently agent 3 is the only agent that does not reach its goal.

Remark 2 Many ASP based approaches are set up in such a way that the only rules depending on the particular problem instance (in our case, a Boolean game) are facts. When tackling the problem of finding PNEs in Boolean games, however, translating the goals of the agents to plain facts is not feasible.

Proposition 1 *Let $G = (N, V, \pi, \Phi)$ be a Boolean game and $\mathcal{P}$ its induced PNE-program. For every answer set I of $\mathcal{P}$ the strategy profile $S_I = \{p \mid act(p) \in I\}$ is a PNE of G. Moreover, the goal of agent $i \in N$ is satisfied for S_I iff $goal(i) \in I$. Conversely, for every PNE $S = (s_1, \ldots, s_n)$ of G there exists an answer set I_S of $\mathcal{P}$ such that*

1. *for every $p \in V$ it holds that $act(p) \in I_S$ iff $p \in S$ and $\neg act(p) \in I_S$ otherwise, and*
2. *for every $i \in N$ it holds that $goal(i) \in I_S$ iff $u_i(S) = 1$.*

The proof is given in Appendix A.

Remark 3 Our approach can easily be extended to Boolean games with constraints [7]. Assuming without loss of generality that the constraint γ_i and its negation $\sim\gamma_i$ are in negation normal form, we add the 'rule' $\leftarrow \sim\gamma_i(act(\pi_i))$ to $\mathcal{P}_1$ for every $i \in N$ and $sat \leftarrow \sim\gamma_i(nact'(\neg\pi_i), act'(\pi_i))$ to $\mathcal{P}_2$. Again $\sim\gamma_i(nact'(\neg\pi_i), act'(\pi_i))$ is the formula $\sim\gamma_i$ in which every occurrence of $p \in \pi_i$ preceded by $\neg$ is replaced by $nact'(p)$ and every other occurrence of $p \in \pi_i$ by $act'(p)$.

Remark 4 Similarly, our method can also take into account cost functions [15] in a natural way. Let us write $\{p^i_1, \ldots, p^i_{|\pi_i|}\}$ for the actions in π_i, for every agent i. To define the ASP program induced by a Boolean games with costs, we add the following rules[2] to $\mathcal{P}_1$:

$$cost(i, j, c(p^i_j)) \leftarrow act(p^i_j)$$
$$cost(i, j, c(\neg p^i_j)) \leftarrow \neg act(p^i_j)$$
$$sum(i, |\pi_i|, X) \leftarrow cost(i, |\pi_i|, X)$$
$$sum(I, J, X + Y) \leftarrow sum(I, J + 1, X), cost(I, J, Y)$$
$$costs(I, Z) \leftarrow sum(I, 1, Z)$$

[2] The ASP solver clingo can use these rules as such, but the WASP implementation requires the built-in aggregate functions $\#int$ and $\#succ$, see the solver implementation for details.

for every agent $i \in N$ and every $j \in \{1, \ldots, |\pi_i|\}$. We add similar rules to $\mathcal{P}_2$:

$$\begin{aligned} cost'(i, j, c(p_j^i)) &\leftarrow act'(p_j^i) \\ cost'(i, j, c(\neg p_j^i)) &\leftarrow nact'(p_j^i) \\ sum'(i, |\pi_i|, X) &\leftarrow cost'(i, |\pi_i|, X) \\ sum'(I, J, X + Y) &\leftarrow sum'(I, J + 1, X), cost'(I, J, Y) \\ costs'(I, Z) &\leftarrow sum'(I, 1, Z) \end{aligned}$$

The rules above compute the total cost per agent by going through its actions one by one and add exactly one cost per action to the subtotal. Note that saturating the new literals $cost'$, sum' and $costs'$ is unnecessary since they will already be set to true because of the literals of the form $act'(p)$ and $nact'(p)$. Indeed, if sat is true, which is the case in every answer set due to (7), then $act'(p)$ and $nact'(p)$ will be in any answer set, for every p, due to (8). Hence the rules stated above imply that all possible costs associated with an action will be in every answer set, for every action. Consequently, all possible total costs per agent that can occur in the game will be in every answer set automatically, without using extra saturation rules. Finally, we replace rules (5) by:

$$\begin{aligned} &pleased(I) \leftarrow goal(I), ngoal'(I) \\ &pleased(I) \leftarrow goal(I), costs(I, X), costs'(I, Y), X \leq Y \\ &pleased(I) \leftarrow ngoal'(I), costs(I, X), costs'(I, Y), X \leq Y \end{aligned}$$

These rules encode the best response condition for Boolean games with costs.

Remark 5 Pareto optimality is an important property of strategies. We can easily extend the previous ASP encoding to find Pareto optimal PNEs. It suffices to add rules expressing that for an arbitrary alternative strategy profile, defined by the literals act' and $nact'$, there is either an agent which is strictly better off in the original strategy profile or all agents are at least as well off in the original strategy profile. We therefore add the following rules to $\mathcal{P}_2$ and $\mathcal{P}_3$:

$$\begin{aligned} ngoal''(i) &\leftarrow \sim\varphi_i(nact'(\neg V), act'(V)) \\ cond &\leftarrow goal(I), ngoal''(I) \\ pleased'(I) &\leftarrow goal(I) \\ pleased'(I) &\leftarrow ngoal''(I) \end{aligned}$$

with $\sim\varphi_i(nact'(\neg V), act'(V))$ the formula $\sim\varphi_i$ in which every occurrence of $\neg p$ (with $p \in V$) is replaced with $nact'(p)$ and every occurrence of p with $act'(p)$. These rules derive $cond$ iff one agent is strictly better off in the strategy profile encoded by $\mathcal{P}_1$ and they derive $pleased'(i)$ iff agent i is at least as good off in the strategy profile encoded by $\mathcal{P}_1$. Furthermore, we add rules of the form $x \leftarrow sat$ to saturate every literal x introduced when transforming the 'rules' with head *ngoal"(.)* into valid ASP rules. We also saturate all literals of the form $ngoal''$ by adding $ngoal''(I) \leftarrow sat, agent(I)$. Finally, we replace rule (6) with:

$$\begin{aligned} &sat \leftarrow \{pleased(i), pleased'(i) \mid i \in N\} \\ &sat \leftarrow \{pleased(i) \mid i \in N\}, cond \end{aligned}$$

Intuitively, we saturate whenever (i) all agents play a best response and are at least as well off in the strategy profile encoded by $\mathcal{P}_1$; or (ii) all agents play a best response and at least one agent is strictly better off in the strategy profile encoded by $\mathcal{P}_1$. The correctness of this encoding can be proven in a similar way as Proposition 1.

3.2 Computing Core Elements with Disjunctive ASP

We now describe an ASP program with the property that its answer sets correspond to the core elements of a Boolean game. To this end, we adjust the induced PNE-program to define the *core-program induced by* the Boolean game $G = (N, V, \pi, \Phi)$. This program contains all rules from the PNE program, with the exception of (4) and (6). In addition, we add the facts $control(i, p) \leftarrow$ for every $i \in N$ and every $p \in \pi_i$ to $\mathcal{P}_1$, to express that agent i controls action p. We add the following rules to divide the agents into coalition members and non-coalition members, demanding non-empty coalitions:

$$coalition(I) \vee ncoalition(I) \leftarrow agent(I) \tag{9}$$

$$sat \leftarrow \{ncoalition(i) \mid i \in N\} \tag{10}$$

We add the following rules to express that non-coalition members do not alter their actions:

$$\begin{aligned} act'(P) &\leftarrow control(I, P), ncoalition(I), act(P) \\ nact'(P) &\leftarrow control(I, P), ncoalition(I), \neg act(P) \end{aligned} \tag{11}$$

We replace rule (4) with:

$$ngoal'(i) \leftarrow \sim \varphi_i(nact'(\neg V), act'(V)) \tag{12}$$

and rule (6) with:

$$sat \leftarrow pleased(I), coalition(I) \tag{13}$$

i.e. we saturate whenever a coalition member does not strictly prefer the alternative strategy of the coalition. To $\mathcal{P}_3$ we add the following rules:

$$coalition(I) \leftarrow sat, agent(I) \tag{14}$$

$$ncoalition(I) \leftarrow sat, agent(I) \tag{15}$$

Example 7 Let us reconsider the Boolean game of Example 1. The associated program consists of $\mathcal{P}_1$, which contains exactly the same rules as in Example 6, with the addition of the following facts:

$$control(1, p_1) \leftarrow; \qquad control(2, p_2) \leftarrow; \qquad control(3, p_3) \leftarrow$$

Program part $\mathcal{P}_2$ contains the following rules:

$$act'(P) \vee nact'(P) \leftarrow action(P)$$
$$coal(I) \vee ncoal(I) \leftarrow agent(I)$$

$$
\begin{aligned}
act'(P) &\leftarrow control(I,P), ncoal(I), act(P)\\
nact'(P) &\leftarrow control(I,P), ncoal(I), \neg act(P)\\
ngoal'(1) &\leftarrow act'(p_1) \wedge (nact'(p_1) \vee \neg act(p_2) \vee act(p_3))\\
ngoal'(2) &\leftarrow (\neg act(p_1) \vee ((nact'(p_2) \vee \neg act(p_3)) \wedge (act'(p_2) \vee act(p_3))))\\
&\quad \wedge (act(p_1) \vee (act'(p_2) \wedge act(p_3)) \vee (nact'(p_2) \wedge \neg act(p_3)))\\
ngoal'(3) &\leftarrow (\neg act(p_1) \vee act(p_2) \vee act'(p_3))\\
&\quad \wedge (act(p_1) \vee \neg act(p_2) \vee nact'(p_3))
\end{aligned}
$$

The rules with heads $ngoal'(i)$ from $\mathcal{P}_2$ become the same as in Example 6. The final program part $\mathcal{P}_3$ is given by:

$$
\begin{aligned}
pleased(I) &\leftarrow goal(I); & & & pleased(I) &\leftarrow ngoal'(I)\\
sat &\leftarrow pleased(I), coal(I); & & & &\leftarrow not\ sat\\
sat &\leftarrow ncoal(1), ncoal(2), ncoal(3)\\
act'(P) &\leftarrow sat, action(P) & & & nact'(P) &\leftarrow sat, action(P)\\
coal(I) &\leftarrow sat, agent(I) & & & ncoal(I) &\leftarrow sat, agent(I)\\
x'_1 &\leftarrow sat; & x'_2 &\leftarrow sat; & x'_3 &\leftarrow sat\\
x'_4 &\leftarrow sat; & x'_5 &\leftarrow sat; & x'_6 &\leftarrow sat\\
x'_7 &\leftarrow sat; & & & ngoal'(I) &\leftarrow sat, agent(I)
\end{aligned}
$$

The only answer set of the induced core-program is the one containing $\neg act(p_1)$, $\neg act(p_2)$, $act(p_3)$, $goal(1)$, $goal(2)$, $\neg goal(3)$, corresponding to the unique core element.

Proposition 2 *Let $G = (N, V, \pi, \Phi)$ be a Boolean game and $\mathcal{P}$ be its induced core-program. For every answer set I of the program $\mathcal{P}$ the strategy profile $S_I = \{p \mid act(p) \in I\}$ is an element of $Core(G)$. Moreover, the goal of agent $i \in N$ is satisfied for S_I iff $goal(i) \in I$. Conversely, for every element $S = (s_1, \ldots, s_n)$ of $Core(G)$ there exists an answer set I_S of $\mathcal{P}$ such that*

1. *for every $p \in V$ it holds that $act(p) \in I_S$ iff $p \in S$ and $\neg act(p) \in I_S$ otherwise, and*
2. *for every $i \in N$ it holds that $goal(i) \in I_S$ iff $u_i(S) = 1$.*

The proof is given in Appendix A.

Remark 6 As in the case of PNEs, we can easily extend the core-program induced by a Boolean game to take constraints, weights, or Pareto optimality into account. For example, to impose Pareto optimality, we add the following rules:

$$
\begin{aligned}
act''(P) \vee nact''(P) &\leftarrow action(P)\\
ngoal''(I) &\leftarrow \sim \varphi_i(nact''(\neg V), act''(V))\\
cond &\leftarrow goal(I), ngoal''(I)\\
pleased'(I) &\leftarrow goal(I)\\
pleased'(I) &\leftarrow ngoal''(I)
\end{aligned}
$$

together with saturation rules for act, $nact$, $ngoal''$ and newly introduced atoms in $\mathcal{P}_2$. Furthermore, rule (13) must be replaced with:

$$sat \leftarrow pleased(I), coalition(I), \{pleased'(i) \mid i \in N\}$$
$$sat \leftarrow pleased(I), coalition(I), cond$$

We refer to the implementation for details on the extension of our encoding of core strategies to Boolean games with constraints and costs.

The implementation of our ASP encodings to compute PNEs or core elements of Boolean games with costs and constraints, with the option of demanding Pareto optimality, has been made available online [13]. Our solver takes a Boolean game as input, generates the ASP encoding of the problem for either WASP or clingo, then uses the corresponding ASP solver to find an answer set, and finally distills the solution from this answer set.

4 Alternative Methods to Solve Boolean Games

In this section, we give an overview of alternative methods which take a Boolean game as input and return a PNE or a core element as output, assuming one exists. Since there are currently no methods to solve general Boolean games, every approach in this section is either designed for a specific class of Boolean games or designed for normal form games. In Section 5, we will experimentally compare the efficiency of these techniques with the method discussed in Section 3.

4.1 Alternative Methods Designed for Boolean Games

4.1.1 Win-Stay Lose-probabilistic Shift

The Win-Stay Lose-probabilistic Shift (WSLpS) algorithm was originally introduced to tackle coordination and anti-coordination games [25,26], with the purpose of coordinating agents towards a solution in a distributed manner. Recently, we have proposed WSLpS to coordinate agents towards a Pareto optimal PNE for a specific class of Boolean games [12]. In this paper, the algorithm is modified to enable the computation of a PNE in general Boolean games. Algorithm 1 describes one iteration of WSLpS. The strategy profile in the current iteration is S^t, the strategy profile in the next iteration is S^{t+1}. The function $rand\_subset(uniform, Neigh(i), k_i)$ computes a uniformally random subset of the set of neighbors $Neigh(i)$ of agent i (see Definition 2) of size k_i.

The function $i.shiftActions(\beta_i(S^t, r))$ is defined as stated in Algorithm 2. The notation $s_i^t[a]$ stands for the truth value of action variable a, which is controlled by agent i, in the strategy profile S^t.

WSLpS can be viewed as an iterative solver for Boolean games. Initially, every agent randomly sets each of the variables under its control to true or false, without knowledge of the actions or goals of the other agents. The strategy profile corresponding with this initial choice is denoted as S^0, with 0 the iteration number. In every iteration, it is checked whether the current strategy profile S^t is a solution or not. If so, we are finished; if not, the agents choose new actions, depending on

Algorithm 1 One iteration of WSLpS

Input: S^t
Output: S^{t+1}
1: **if** S^t is PNE **then**
2: $S^{t+1} = S^t$
3: **else**
4: **for** agent $i \in N$ **do**
5: $r \leftarrow$ rand_subset(uniform,$Neigh(i), k_i$)
6: **if** $success(S^t, r) = 1$ **then**
7: $s_i^{t+1} \leftarrow s_i^t$
8: **else**
9: $s_i^{t+1} \leftarrow i$.shiftActions($\beta_i(S^t, r)$)
10: **end if**
11: **end for**
12: **end if**

Algorithm 2 i.shiftActions($\beta_i(S^t, r)$)

Input: s_i^t, $\beta_i(S^t, r)$
Output: s_i^{t+1}
1: **for** each variable a in π_i **do**
2: $p \leftarrow$ rand_double(uniform,[0,1])
3: **if** $p \leq \beta_i(S^t, r)$ **then**
4: $s_i^{t+1}[a] \leftarrow 1 - s_i^t[a]$
5: **else**
6: $s_i^{t+1}[a] \leftarrow s_i^t[a]$
7: **end if**
8: **end for**

a random subset of k_i neighbors and a function *success*, which the agents try to maximize. In this paper, we define the binary function $success : 2^V \times 2^N \rightarrow \{0, 1\}$ for a Boolean game $G = (N, V, \pi, \Phi)$ as

$$success(S, r) = \begin{cases} 1 & \text{iff } u_j(S) = 1, \forall j \in r \\ 0 & \text{otherwise} \end{cases}$$

Intuitively, the success function returns 1 if and only if all agents in the subset r of N reach their goal. Note that if for every agent i it holds that $success(S, r) = 1$ for every subset r of neighbors of i, then every agent's goal ($\neq \bot$) is fulfilled in S.

Given a certain strategy profile S^t in iteration $t + 1$, each agent evaluates its success function. If its value is 0 (i.e. the agent gets negative feedback), the agent independently flips each of the variables under its control with probability β_i. Flipping a variable means setting the corresponding variable to true if it was false and vice versa. In case of positive feedback, the agent does not alter its strategy choice.

WSLpS uses a parameter k, ranging from 1 to n, which determines the maximum number of neighbors we take into account to evaluate the *success* function. If an agent i has fewer than k neighbors, we just take all neighbors of i into account. The actual number of neighbors taken into account is thus defined as $k_i = \min(k, |Neigh(i)|)$.

Remark 7 If we omit lines 1, 2, 3 and 12 from Algorithm 1, we obtain a WSLpS variant that is able to coordinate agents in a distributed manner towards a Pareto

optimal pure Nash equilibrium, under the condition that the game has an outcome such that every agent reaches its goal [12]. Note that the decentralized aspect here is crucial, as methods which rely on central entities, such as SAT solvers, are obviously more efficient to compute these solutions, yet fail to work in a distributed manner.

To execute line 1, it is checked whether every agent plays a best response in S^t. In the worst case, this requires $\sum_{i \in N} 2^{|\pi_i|}$ goal evaluations.

The shift probability β_i depends on a parameter $\alpha \in]0, 1[$, the subset r of neighbors of agent i and the strategy profile S that was chosen in the current iteration:

$$\beta_i(S, r) = \max\left(\alpha - \frac{|\{j \in r \mid u_j(S) = 1\}|}{k_i}, 0\right)$$

Thus the probability of an agent changing its strategy increases with the number of unsatisfied neighbors. Note that α is the probability with which an agent shifts actions in case none of its neighbors in r has reached their goal. The stochastic algorithm WSLpS has converged if the condition on line 1 is fulfilled.

Analogously as in [12], the convergence of WSLpS can be proven using Markov chains. A finite Markov chain [23] is a random process that transitions from one state to another, where the total number of possible states is finite, and that satisfies the Markov property. The Markov property states that the probability of transitioning from one state S to another state S' depends on S and S', but not on the states the system was previously in. If the transition probabilities between states do not alter over time, in other words, if the transition probabilities are time-independent, then the Markov chain is called homogeneous. It is easy to see that WSLpS, applied to a Boolean game, induces a homogeneous Markov chain, in which states are strategy profiles and each transition between states corresponds to an iteration of WSLpS.

An absorbing state S of a Markov chain is a state such that the transition probability from S to itself is 1, i.e. once the system reaches this state, it cannot escape from it. An absorbing Markov chain satisfies two conditions: (i) there is at least one absorbing state and (ii) for each non-absorbing state there exists an accessible absorbing state, where a state S' is called accessible from a state S if there exists a positive $m \in \mathbb{N}$ such that the probability of transitioning from state S to state S' in m steps is strictly positive. Absorbing Markov chains have an interesting property [23]: regardless of the initial state, the Markov chain will eventually end up in an absorbing state with probability 1. Therefore, the theory and terminology of Markov chains offer an alternative formulation for convergence of WSLpS: when applied to a Boolean game G, WSLpS converges if and only if the induced random process is an absorbing Markov chain. When α is chosen larger than $\frac{k-1}{k}$, the induced random process is an absorbing Markov chain if and only if G has a PNE.

Proposition 3 *Let $G = (N, V, \pi, \Phi)$ be a Boolean game and $\alpha > \frac{k-1}{k}$. WSLpS applied to G converges to a PNE if and only if G has a PNE.*

The proof is given in Appendix A. Note that the algorithm is not able to determine whether a Boolean game has a PNE or not. Moreover, if the game does not have a PNE, the algorithm will not converge. It has therefore been implemented with an adjustable time-out function.

Remark 8 Note that constraints can easily be incorporated in WSLpS by avoiding that an agent changes its actions to an excluded strategy, i.e. a strategy violating the agent's constraint. Concretely, the agent keeps changing strategies until an allowed strategy is obtained. However, through this adaptation the algorithm loses the advantage of shifting actions in constant time. For instance, when the constraint excludes many strategies, the computation time can rapidly increase.
It is easy to see that the convergence of WSLpS is no longer theoretically guaranteed when we add costs to the games. Indeed, in such a case a Boolean game might have outcomes such that every agent reaches its goal, but which are not PNEs because the costs of at least one agent is not minimal. If WSLpS reaches such an outcome, no agent will change its strategy with the current success function, and the algorithm will be stuck. For Boolean games with costs, alternative success functions can be used to address this issue. One could, for instance, define a success function that counts the number of neighbors playing a best response. However, investigating these computationally more complex variants lies beyond the scope of this paper.

Remark 9 Very recently, new notions of dependencies between agents in Boolean games have been introduced and investigated, based on postulates on the dependency function [4]. These dependency notions can be used to define alternative notions of neighborhoods, which lead to new variants of WSLpS. Investigating these variants is beyond the scope of this paper.

Alternatively, one can employ WSLpS as a heuristic algorithm to compute a core element of Boolean games by replacing the condition in line 1 by *S is a core element*. In that case we check whether any coalition blocks the strategy profile. Note that this is computationally more challenging than the original condition: in the worst case, we have to iterate over every coalition C of agents $(2^n - 1)$ and over all their alternative strategies $\left(2^{\sum i \in C|\pi_i|} - 1\right)$ to see if one blocks the current strategy profile.

We have implemented WSLpS to compute either PNEs or core elements, allowing Boolean games with costs and constraints (available online [13]).

4.1.2 CompPNEAcycl

The algorithm CompPNEAcycl has been provided to compute PNEs of Boolean games for which the irreflexive part of the dependency graph is acyclic [5,6]. The dependency graph of a Boolean game connects every agent with its relevant agents. A Boolean game for which the irreflexive part of the dependency graph is acyclic has at least one PNE [5,6]. Moreover, the authors show that PNEs of Boolean games can also be found by computing the PNEs of subgames of the Boolean game. More specifically, a Boolean game is decomposed using a collection of stable sets which covers the total set of agents. It is shown that if there exists a collection of PNEs of the subgames – with exactly one PNE for every subgame – such that the strategies of agents belonging to multiple stable sets of the covering agree, then the strategy profile obtained by combining these strategies is a PNE of the original Boolean game. It is, however, important to note that a decomposition based on stable sets cannot remove or break cycles in the dependency graph. Indeed, the decomposition is only used to speed up the computation

of the PNEs by dividing the problem in smaller problems (divide-and-conquer). Therefore, the usage of the algorithm combined with the decomposition is still restricted to Boolean games for which the irreflexive part of the dependency graph is acyclic. In the pseudocode of Algorithm 3, the dependency graph of the Boolean game is denoted as $\langle V, RA \rangle$, where V is the set of nodes and RA the set of edges of the graph. Note that this pseudocode is slightly different from the original al-

Algorithm 3 CompPNEAcycl

Input: Boolean game G with acyclic irreflexive part of dependency graph
Output: PNE S
1: $\langle V, RA \rangle \leftarrow G$.dependencygraph(), $T \leftarrow V$, $I \leftarrow \emptyset$, $S \leftarrow \emptyset$
2: **while** $T \neq \emptyset$ **do**
3: $PI \leftarrow \emptyset$
4: **for** each agent $i \in T$ **do**
5: **if** $RA(i) \subseteq I \cup \{i\}$ **then**
6: $PI \leftarrow PI \cup \{i\}$
7: **end if**
8: **end for**
9: **for** each agent $i \in PI$ **do**
10: $s_i \leftarrow i$.bestresponse(S)
11: $I \leftarrow I \cup \{i\}$, $T \leftarrow T \setminus \{i\}$
12: **end for**
13: **end while**

gorithm [5,6] in the sense that the dependency graph is not part of the input of CompPNEAcycl. Moreover, the output is one PNE (instead of all of them), but we have implemented both variants of the algorithm (available online [13]).

4.2 Alternative Methods Designed for Normal Form Games

An obvious way to obtain PNEs of Boolean games is to translate a Boolean game to an equivalent normal form game. However, the size of the resulting normal form game may be exponential in the size of the initial Boolean game. In normal form games, agents choose one action out of a (finite) set. When translating Boolean games to normal form games, agent i will have $2^{|\pi_i|}$ possible actions. Therefore, as we show in Section 5.2, normal form game techniques that require the computation of the entire payoff matrix – such as the ASP approach for normal form games [14] – are unsuitable for larger problem instances of Boolean games. However, there are methods that have been specifically proposed for solving normal form games whose payoff matrix is expensive to compute, such as tabu best-response search. Such techniques may thus be more appropriate for solving Boolean games.

4.2.1 Answer Set Programming for Normal Form Games

This technique is a standard approach to compute PNEs of normal form games in ASP [14]. To use this method for Boolean games, we need to translate the Boolean game to a normal form game, generate the ASP encoding to compute the PNEs of the normal form game and translate these PNEs back to Boolean game format. The strategies of every agent in the Boolean game are numbered

from 0 to $2^{|\pi_i|}-1$, corresponding to actions of the agent in the normal form game. Intuitively, the method describes which strategies are best responses (the head of the ASP rule) given the strategies of the other agents (the body of the ASP rule). Thus for every joint strategy of the relevant agents of agent i (without i itself), we need to compute the best responses of i and write an ASP rule to capture this information. For example, suppose the set $\{a^i_{k_1},\ldots,a^i_{k_m}\}$ contains the actions of agent i in the normal form game corresponding to best response strategies of agent i to a joint strategy $s_{i_1},\ldots,s_{i_n}$ of the relevant agents of i, i.e. $i_1,\ldots,i_n$, in the Boolean game. Suppose that the actions $a^{i_1}_{l_1},a^{i_2}_{l_2},\ldots,a^{i_n}_{l_n}$ of the normal form game are the ones corresponding with this joint strategy. Then the ASP encoding will contain the rule

$$1\{a^i_{k_1},\ldots,a^i_{k_m}\}1 \leftarrow a^{i_1}_{l_1},a^{i_2}_{l_2},\ldots,a^{i_n}_{l_n}$$

The head of the body $1\{a^i_{k_1},\ldots,a^i_{k_m}\}1$ denotes a choice: exactly one of the literals in the set should be true. Obviously, the number of rules in the ASP program will be exponential in the number of action variables in the game. In the worst case, every agent is relevant for every agent (i.e. the dependency graph is complete) and there are $\sum_{i\in N} 2^{\sum_{j\neq i}|\pi_j|}$ rules in the ASP encoding. We have implemented this method to compute PNEs of Boolean games (available online [13]).

4.2.2 Tabu Best-Response Search

Tabu best-response search (TBRS) has been introduced for normal form games [28]. This heuristic algorithm combines best-response dynamics and tabu search, i.e. its moves through the search space by letting agents choose best response strategies and it keeps track of the last l examined solutions in a tabu list to avoid infinite looping on a small part of the search space. TBRS can easily be applied to Boolean games, without requiring the computation of the entire payoff matrix. However, the algorithm is still exponential, since it uses the correspondence between Boolean and normal form games to enumerate the exponential number of strategies and strategy profiles.

Algorithm 4 One iteration of TBRS

Input: S, tabu list L, number of best responses b
Output: S, tabu list L, number of best responses b

```
for agent i ∈ N do
    if i plays no best response in S then
        b ← 1
        L.addStrategyProfile(S)
        s_i ← i.bestResponse(S, L)
    else
        b ← b + 1
    end if
end for
```

The function L.addStrategyProfile(S) adds S to the tabu list L and deletes the oldest item in the list in case the length is $l+1$. The function i.bestResponse(S,L) searches a best response strategy of i such that the newly obtained strategy profile

is not on the tabu list L. The algorithm is initialized with a random strategy profile. In a fixed order, every agent changes its strategy to a best response (best-response dynamics), if it did not yet play a best response. In case no costs are involved, checking whether a best response is played requires that an agent evaluates its utility for each of its $2^{|\pi_i|}$ strategies, in the worst-case scenario, i.e. when it plays a best response but its goal is not satisfied. In the best-case scenario, the goal is satisfied and no other evaluations are required to assure that a best response is played. For Boolean games with costs, an agent who plays a best response must always evaluate its utility for each of its $2^{|\pi_i|}$ strategies to be certain of playing a best response. In order to avoid getting stuck in a cycle of non-optimal solutions, a tabu list stores the last l strategy profiles (tabu search) and agents are not allowed to change their strategy such that the resulting strategy profile is among the l strategy profiles in the tabu list. It is straightforward to see that, if a PNE exist, choosing l sufficiently large guarantees the convergence of TBRS. In the worst-case scenario, one must choose $l = 2^{|V|} - 1$, but in practice a smaller l will usually suffice. Note that a PNE is found as soon as $b = n$.

Additionally checking the necessary condition of a core element allows TBRS to search for core elements instead of PNEs. This condition is only checked when all agents play a best response, since every core element is a PNE. If the investigated PNE is not a core element, it is added to the tabu list and the algorithm continues its search.

We have implemented TBRS to compute either PNEs or core elements of Boolean games, allowing costs and constraints (available onine [13]).

5 Experiments

In this section, we set up experiments to evaluate and compare the performance of several solution methods for Boolean games. Specifically, the methods that we compare are the following:

- the disjunctive ASP based solver of Section 3, abbreviated to dASP;
- WSLpS (see Section 4.1.1);
- CompPNEAcycl (see Section 4.1.2), referred to as BONZON;
- naive random search, referred to as NAIVE;
- the ASP based technique for normal form games (see Section 4.2.1), abbreviated to NFG; and
- TBRS (described in Section 4.2.2).

The naive random search is a baseline iterative technique that randomly assigns truth-values to the action variables and checks whether the obtained strategy profile is a solution (PNE or core element). We have made an implementation of this heuristic method for Boolean games with costs and constraints. The stochastic algorithms (WSLpS, NAIVE and TBRS) are run 25 times per game. For TBRS, we have empirically determined appropriate values for the length of the tabu list; $l = 20$ turned out to be a suitable tabu list length. For WSLpS, we use $k = 5$ neighbors, combined with shift probabilities $\alpha = \frac{1}{100}\lfloor 100 \cdot \frac{k-1}{k} \rfloor + 0.01$, as suggested in [12].

We test the dASP approach using two state-of-the-art ASP solvers, WASP[3] and clingo.

Next we describe the Boolean game generators of the experiments in Section 5.1. Finally, we present our results concerning the performance of the different techniques in Section 5.2.

All problem instances, problem generators and solvers have been made available online [13]. The measurements have been made using a dual CPU system with two 2.4GHz Intel Xeon six core E5-4610 processors and 8GB RDIMM.

5.1 Boolean Game Generators

We evaluate the performance of the different methods on three classes of problems: a class of random Boolean games, a class of project Boolean games and a class of Boolean games for which the irreflexive part of the dependency graph is acyclic.

First, the class of *random Boolean games* has 5 parameters:

1. the number of agents;
2. the number of action variables per agent;
3. the maximum number m of binary operators ($\wedge$ or $\vee$) in a goal;
4. the probability of a binary operator in a goal being a conjunction (as opposed to a disjunction); and
5. the probability of an atom occurring in a goal being negated.

Note that the only binary operators that we consider are conjunction and disjunction. To introduce some diversity in the class of random games, we let the length of the goals of different agents vary. To this end, we use the parameter m and, for each agent, choose the number of binary operators in its goal uniformly between 1 and m. The generated goals are in negation normal form.

Second, the class of *project Boolean games* models the following problem. A group of people can work on several projects. Depending on the project, each person has his preferences concerning the people he might collaborate with – called partners – and the people he does not want to collaborate with – called anti-partners. Someone who is not a partner is called a non-partner. Note that all anti-partners are non-partners. Partnership and anti-partnership relations are anti-reflexive, not necessarily symmetric and project dependent. The project Boolean games are inspired by a Boolean game about people being invited to a party [5, 15]. We consider this second class of problems, in addition to the first, because they represent an example of a structured problem, which are usually harder to solve than random problems, and may thus give a better indication of how the solvers would perform in applications. We include 3 parameters for project Boolean games: (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 Boolean game, every agent i controls an action variable p_i^m per project m. Setting it to true means joining the project. For every project, every agent is randomly assigned one of 13 types, which are determined by three parameters:

[3] We also ran the experiments with WASP's 'predecessor' DLV, but those results are omitted since WASP was always faster than DLV.

1. personal preference: (a) join the project, (b) do not join, and (c) no preference;
2. positive partnership (only relevant if level 1 is not (b)): (a) no condition, (b) only join a project if at least one partner joins the project, and (c) only join a project if all partners join the project;
3. negative partnership (only relevant if level 1 is not (b)): (a) no condition, and (b) only join the project if no anti-partners join the project.

For each agent, we start by (uniformly) choosing the personal preference, and based on this choice we (uniformly) choose among the allowed positive and negative partnerships, i.e. the conditions w.r.t. the partners and anti-partners. The type of an agent determines its goal. Suppose for example that agent i is of type (c,b,b) and 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. Additionally, costs and constraints can be added to the project Boolean games. The constraints enforce that an agent can join at most 1 project. For every project and every agent, a cost in $\{0, 1, \ldots, m\}$ is randomly assigned to joining the project. Not joining involves no costs.

Third, we have developed a generator for Boolean games with an acyclic irreflexive part of the dependency graph. This class of Boolean games can be used to evaluate the performance of the algorithm in [5,6], since the usage of this algorithm is limited to Boolean games with an acyclic irreflexive part of the dependency graph. To the best of our knowledge, the algorithm of Bonzon et al. has not been experimentally evaluated yet. This third Boolean game generator starts by generating a directed acyclic graph (DAG), whose nodes represent the agents. Next, we use this DAG to construct a Boolean game such that the irreflexive part of the dependency graph is exactly the generated DAG. To this end, we make sure the action variables in the goal of an agent i are controlled by agents who agent i depends on. We refer to this class of games as *DAG Boolean games*. The generator has 6 parameters:

1. the number of agents;
2. the number of action variables per agent;
3. the maximum number m of operators ($\wedge$, $\vee$ or $\neg$) in a goal;
4. the probability of an operator in a goal being a conjunction (as opposed to a disjunction or a negation);
5. the probability of an operator in a goal being a disjunction (as opposed to a conjunction or a negation); and
6. the probability that agent i depends on agent j ($j \geq i$).

5.2 Results

Experiment 1 (PNE Computation in Boolean Games)
In this experiment, we investigate the performance of the different algorithms to compute a PNE in general Boolean games. First we use the random Boolean game generator with 50% of atom occurrences negated and an equal chance on $\wedge$ and $\vee$. We generate 100 Boolean games and compute 1 PNE, with a 5 minutes timeout. We let the number of agents vary from 5 to 100. The number of action

variables per agent and the maximum number of binary operators in the goal are respectively $(2, 3), (2, 6), (4, 6), (6, 10)$ and $(8, 10)$, gradually making the problem instances more difficult. Due to the use of a timeout, the distribution of our timing data is likely to be skewed. Therefore the median is a suitable measure to aggregate the computation times corresponding to each set of 100 games. The median computation times and the 95% confidence intervals are shown in Figure 1.
In our experiments, we see that clingo and WASP are comparable in terms of computation time. We observe that the NFG method is only suitable for the problem instances with a small number of action variables per agent and few binary operators. As soon as the problem instances involve larger goals or more action variables per agent, NFG consistently times out. We can expect similar results for other PNE computing techniques developed for normal form games that require to compute the entire payoff matrix of all agents. This is a consequence of the exponential translation of Boolean games to normal form games. If the maximum number of binary operators in a goal increases, the number of rules in the NFG encoding increases exponentially. Indeed, the goals become lengthier and for every possible strategy combination w.r.t. the action variables occurring in the agent's goal we need an ASP rule to determine its best responses. Our experimental results confirm that TBRS can indeed take advantage of the fact that it does not need to compute all payoffs. As the problem instances become more difficult, TBRS has a similar performance as dASP. The fastest method for all the investigated problem instances is NAIVE. This can be explained by the fact that, in general, these random Boolean games have a lot of PNEs. Indeed, if we look at the setting with 100 agents, 8 action variables and a maximum of 10 binary operators in a goal, the median number of iterations to convergence of NAIVE is $1[1, 1]$. If we think about how the random games are set up, we see that the probability of an agent controlling its own goal is very small. If an agent does not influence its own goal, then all its strategies are best responses to all possible strategies of the other agents. Due to the high number of PNEs relative to the number of strategy profiles in these random Boolean games, every strategy profile, guessed by NAIVE, has a high chance of being a PNE.

In the second part of this experiment, we overcome this issue of having too many PNEs. We now compute PNEs for problem instances with more structure than the random Boolean games, whose solutions might therefore be harder to compute. Specifically, we compute 1 PNE for project Boolean games with a timeout of 5 minutes. We generate 100 Boolean games with 1, 3 and 5 projects and a 50% probability that an agent is another agent's partner and that a non-partner agent is an anti-partner. This time, the number of agents varies from 5 to 50, since we expect these structured problems to be more difficult than the random Boolean games. The results are shown in Figure 2.
The performance of WASP and clingo is again comparable. For project Boolean games the number of PNEs is smaller than for random Boolean games. Therefore, the higher the number of projects, the smaller the number of agents at which dASP starts outperforming WSLpS and NAIVE. Note that for the entire range of parameters, dASP (with clingo) still succeeds in finding a solution for all problem instances, whereas both NFG, WSLpS and NAIVE consistently time out for the larger settings. If we look at the setting with 5 projects and 20 agents, then the median number of iterations to convergence of NAIVE – in the 50% cases for which convergence is reached before timing out – is $430000[360000, 590000]$. This

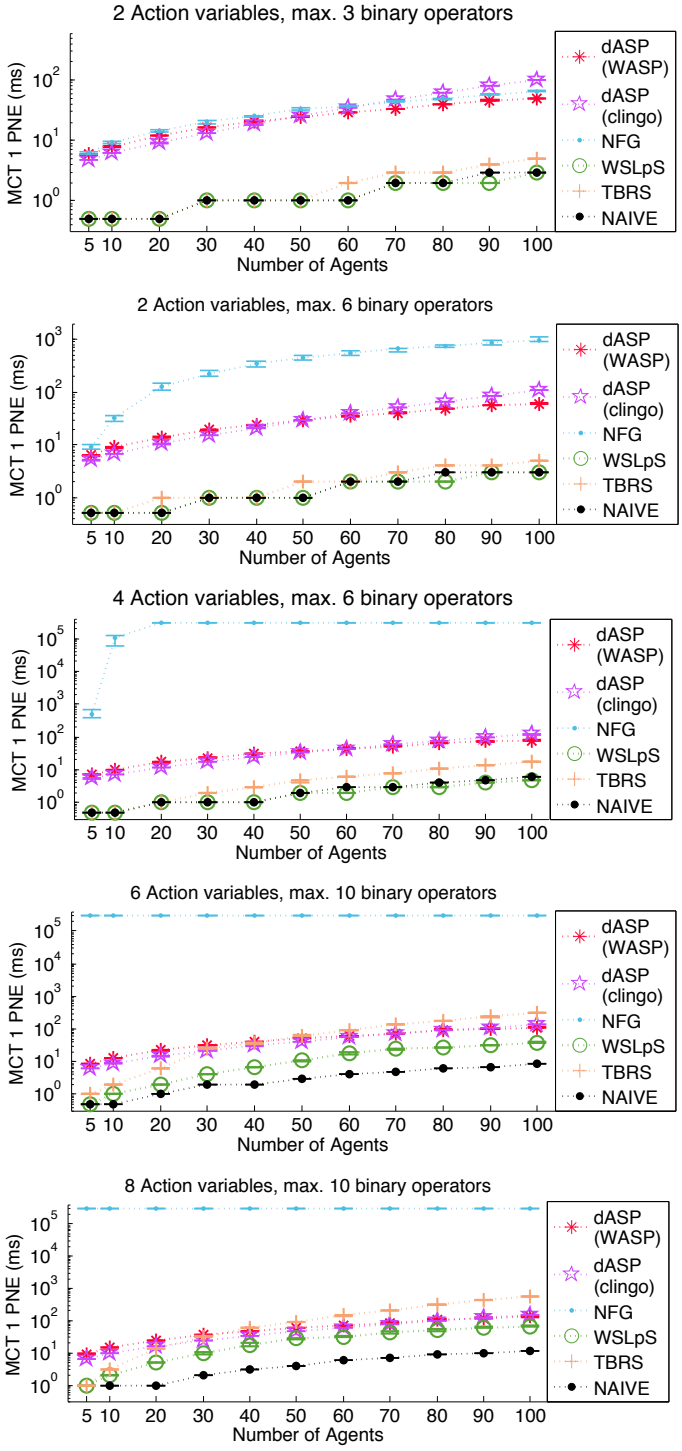


Fig. 1: Median computation time (ms) of 1 PNE in random Boolean games.

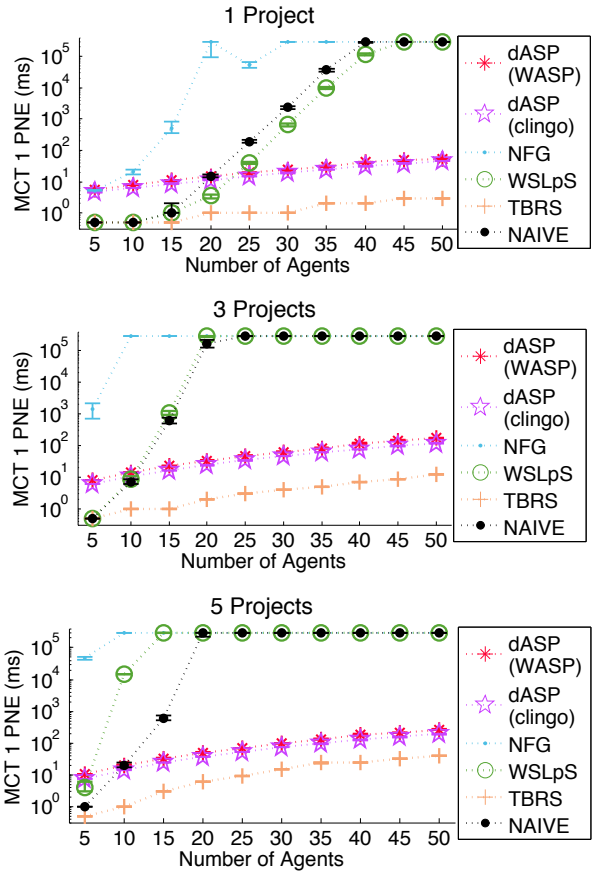


Fig. 2: Median computation time (ms) of 1 PNE in project Boolean games.

illustrates that the number of PNEs is indeed much smaller for these structured games. The best performing method is now TBRS, taking advantage of the fact that its heuristic, as opposed to the one of WSLpS, is not random, but searches for best responses. To see whether this trend continues, we now fix the number of agents at 15, 20, 25, 30 and 35. The number of projects varies from 4 to 20 by steps of size 4. The results are shown in Figure 3.

Note that NFG and WSLpS consistently timeout for all parameter settings. We now clearly see the exponential trend in the median computation time of TBRS, which is to be expected due to the exponential translation from Boolean games to normal form games. So for games with a high number of action variables, the dASP method outperforms all other methods. In particular, the NFG method and WSLpS consistently time out for all problem instances. Note that NAIVE behaves peculiarly, as it needs less computation time for problem instances with 4 projects than for those with 8 or more projects. This might be due to the fact that the goal of an agent i in case of a large number of projects is more likely to e.g. be unsatisfied no matter what agent i does. This means that agent i is likely to play a best response for many strategies s_{-i}. Overall, a project Boolean game with

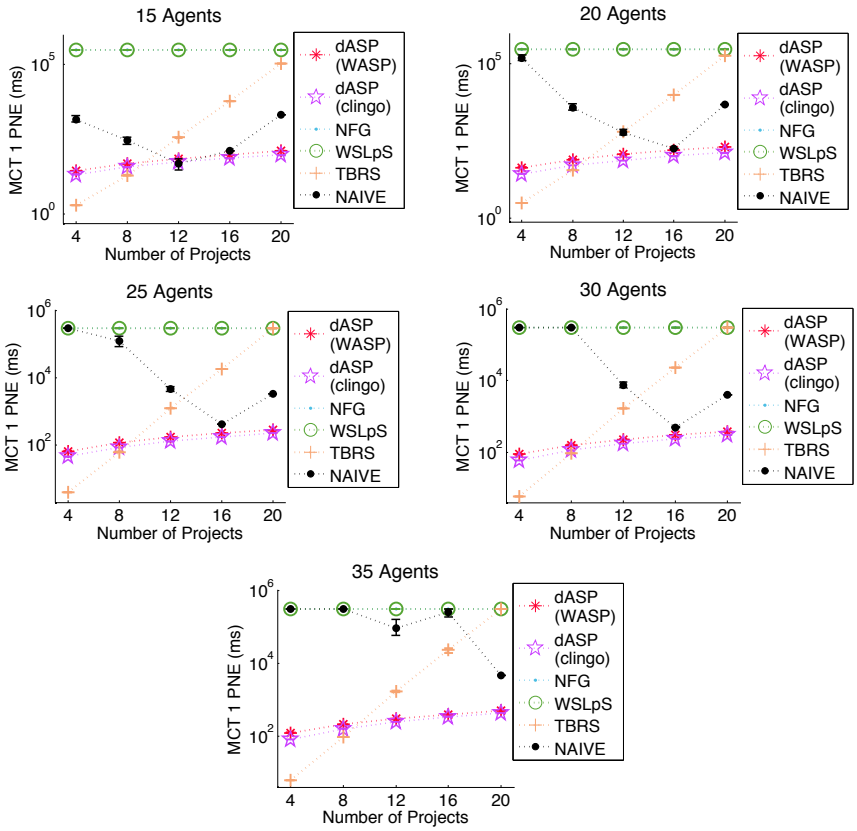


Fig. 3: Median computation time (ms) of 1 PNE in project Boolean games.

such a high number of projects will have relatively more 'bad' PNEs (i.e. PNEs in which many agents do not reach their goal). If we for instance take a closer look at the method NAIVE for 35 agents and 20 projects, we see that only 1% of the runs results in a PNE in which exactly one agent reaches its goal. In 89%, a PNE is reached in which no agent reaches its goal and in the remaining 10%, a timeout occurs.

Experiment 2 (Centralized Computation of PNEs in DAG Boolean games) In this experiment, we include the algorithm CompPNEAcycl of Bonzon et al. to compute PNEs. Therefore, we must limit this experiment to Boolean games with an acyclic irreflexive part of the dependency graph. We generate 100 Boolean games and compute 1 PNE, with a 5 minutes timeout. We let the number of agents vary from 5 to 50. Each operator ($\neg$, $\wedge$ and $\vee$) has an equal chance of appearing in the goal. The probability that agent i is depedent of agent j ($j \geq i$) is 75%. The number of action variables per agent and the maximum number of operators in the goal are respectively $(2, 4), (2, 7), (4, 7), (6, 10)$ and $(8, 10)$, gradually making the problem instances more difficult. The median computation times, within a 95% confidence interval, are shown in Figure 4.

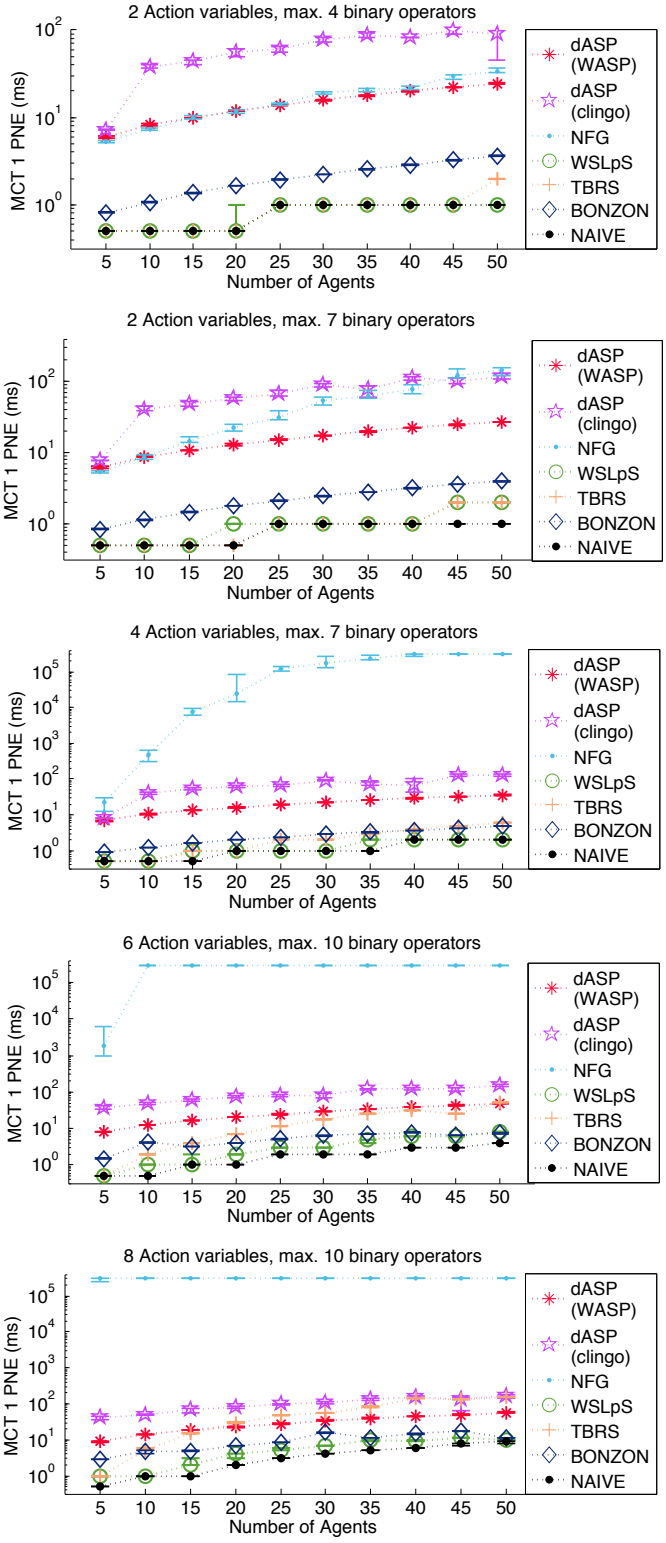


Fig. 4: Median computation time (ms) of 1 PNE in DAG Boolean games.

As expected, the NFG method again turns out to be unsuitable due to its exponential translation and the fact that it computes all payoffs. As before, the heuristic techniques (WSLpS and TBRS) perform well for the smaller problems. As the problem instances gradually become more difficult, CompPNEAcycl (BONZON) tends to outperform the other approaches, with the exception of the NAIVE approach. Presumably, this is caused by a high number of PNEs, as the median number of iterations to convergence of NAIVE is 12 [12, 13] in the setting with 8 action variables, 50 agents and a maximum of 10 operators in the goals. However, this random method is less powerful than CompPNEAcycl, since the latter is able to compute all PNEs and the former can only compute a sample PNE. Therefore, if there is a priori knowledge that the irreflexive part of the dependency graph is actually acyclic, we would advise to use CompPNEAcycl.

To conclude our experiments on the centralized computation of PNEs in Boolean games, we note that for smaller problems, the heuristic methods TBRS, NAIVE and WSLpS are generally the fastest approaches to compute a sample PNE. For Boolean games with an acyclic irreflexive part of the dependency graph, CompPNEAcycl is to be recommended. For larger Boolean games with no specific structure, the dASP approach is to be recommended. Moreover, as opposed to heuristic approaches as WSLpS and TBRS, dASP is able to figure out whether, e.g., some agent undertakes a certain action in every PNE or whether there does not exist a solution. Additionally, the disjunctive ASP based solver is more general since it can enforce desirable properties on solutions such as Pareto optimality. In addition and opposed to TBRS, the disjunctive ASP technique was developed especially for Boolean games, which yields the advantage that it can still handle problems with a larger number of action variables.

In the following part of the experiments, we use centralized techniques to compute core elements instead of PNEs. Since the NFG method cannot be used to compute core elements, we limit the experiments to dASP, WSLpS, TBRS and NAIVE.

Experiment 3 (Centralized Computation of Core Elements in Boolean Games) In this experiment, we investigate the performance of the different centralized algorithms to compute a core element in general Boolean games. For the same range of parameters as in Experiment 1, we compute a core element in 100 random Boolean games. As the results are very similar for all parameters settings, we restricted Figure 5 to the parameter settings $(2, 4)$, $(4, 6)$ and $(8, 10)$.
As expected, the heuristic methods WSLpS, TBRS and NAIVE are only suitable for small problem instances, i.e. games with a small number of action variables or agents. Indeed, core elements are more rare than PNEs, as they have stronger conditions. Therefore, there are likely much more PNEs than core elements, so a heuristic method will have a harder time finding one. Note that WSLpS performs slightly better than TBRS and NAIVE, since TBRS times out for all games with more than 50 agents, and WSLpS is the only heuristic method that still finds a solution for the parameter setting $(4, 6)$ with 50 agents. Moreover, the computation times of NAIVE are slightly higher than those of WSLpS in case of the smaller problem instances.

We also investigate the computation of a core element for project Boolean games, with the same parameter range as in Experiment 1, adding also the case of 2 and 4 projects. The results are shown in Figure 6. Again, we observe that

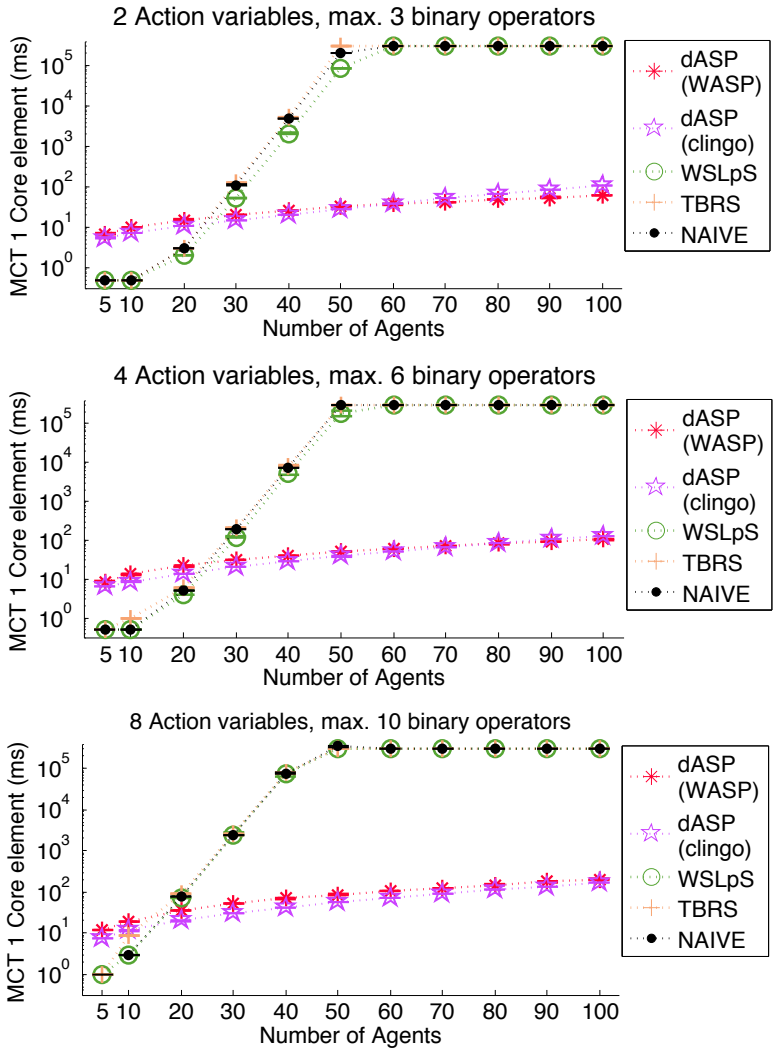


Fig. 5: Median computation time (ms) of 1 core element in random Boolean games.

dASP consistently outperforms WSLpS, TBRS and NAIVE for the more difficult problem instances. We also see that TBRS is now the best performing heuristic approach. For smaller problem instances, WSLpS performs better than NAIVE, and vice versa for larger problem instances.

Experiment 4 (Centralized Computation of Different Solution Concepts)
We now investigate the scalability of the disjunctive ASP approach on project Boolean games with 2, 4 and 6 projects with costs and constraints. The number of agents ranges between 5 and 50 and we consider 100 Boolean games for each parameter setting. We use a timeout of 10 minutes and respectively compute 1 PNE, 1 Pareto optimal PNE (POPNE), 1 core element and 1 Pareto optimal core el-

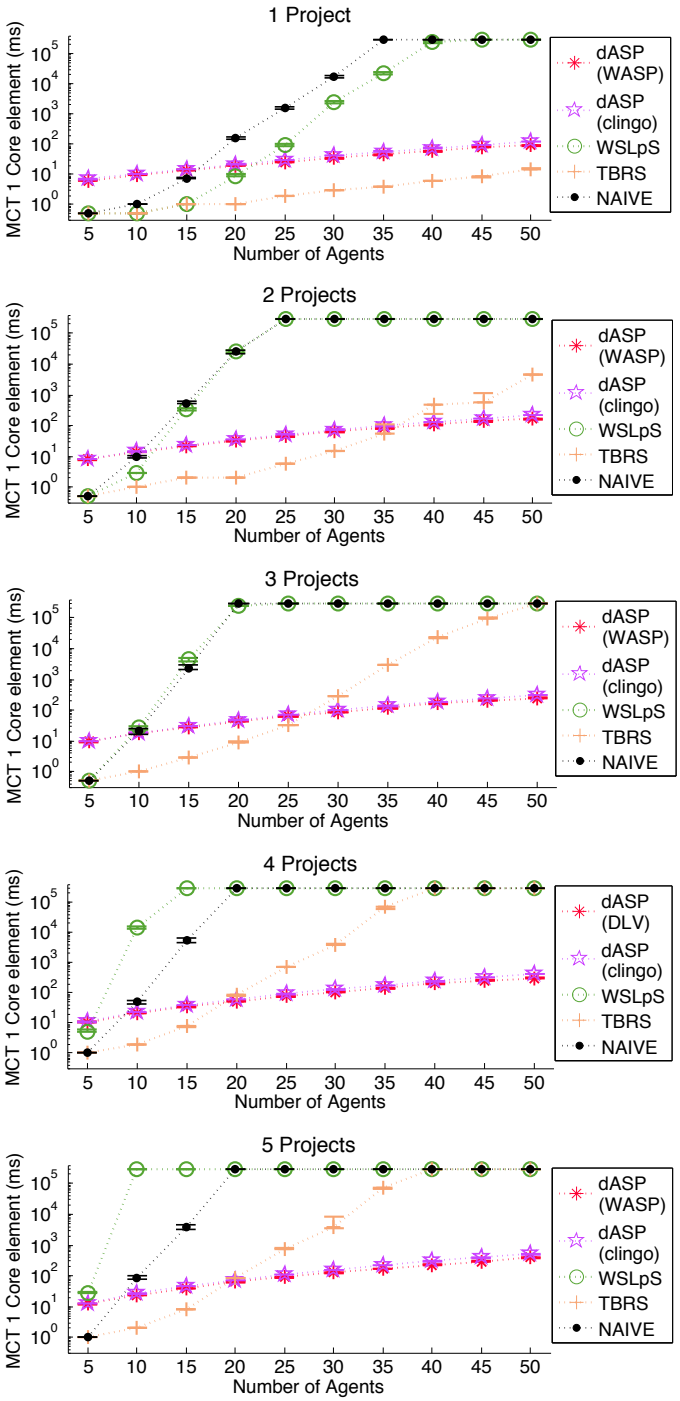


Fig. 6: Median computation time (ms) of 1 core element in project Boolean games.

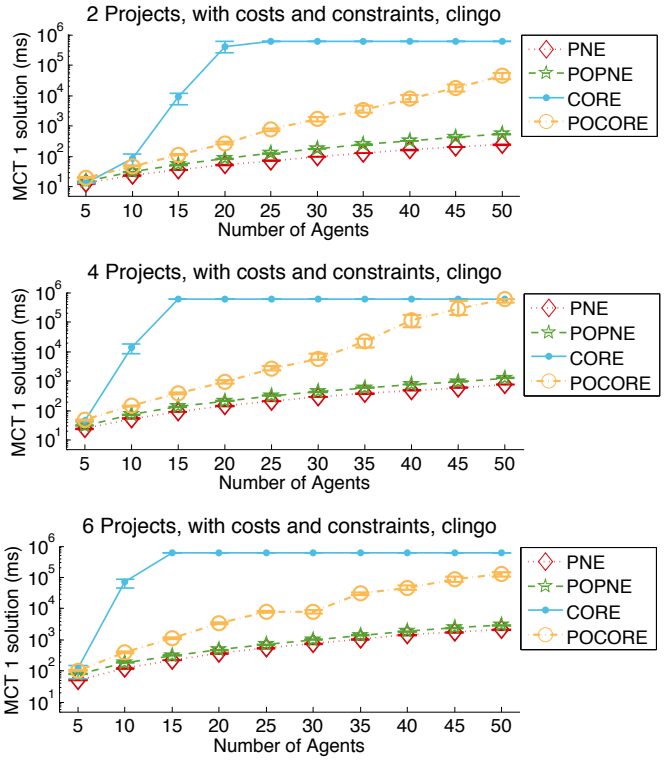


Fig. 7: Median computation time (ms) of 1 solution in project Boolean games using dASP with clingo. Note that out of the techniques discussed in this paper, dASP is the only one that can compute a Pareto optimal PNE and a Pareto optimal core element.

ement (POCORE). Note that in this experiment, we do not compare the dASP method with the other techniques, because those are not designed for computing Pareto optimal solutions. The median computation time per solution concept is shown in Figures 7 and 8, using a logarithmic scale. For the clarity of the figures, we show the time results for clingo and WASP on separate graphs.

Computing a PNE takes less time than computing a core element, and enforcing Pareto optimality further increases the computation time of a PNE. This is to be expected, as Pareto optimal PNEs are rarer than regular PNEs. Note that we do not see something similar when we compare the computation times of clingo for regular and Pareto optimal core elements. This is a consequence of the common easy-hard-easy pattern: problems with very few or very much constraints are easier to solve than those problems lying in between. We can conclude from the experiment that WASP tends to be faster for computing core elements and clingo tends to perform better for computing Pareto optimal core elements.

Although the computational cost is generally higher for stronger solution concepts or for Boolean games involving costs and constraints, we conclude that the dASP method remains suitable for medium-sized problems.

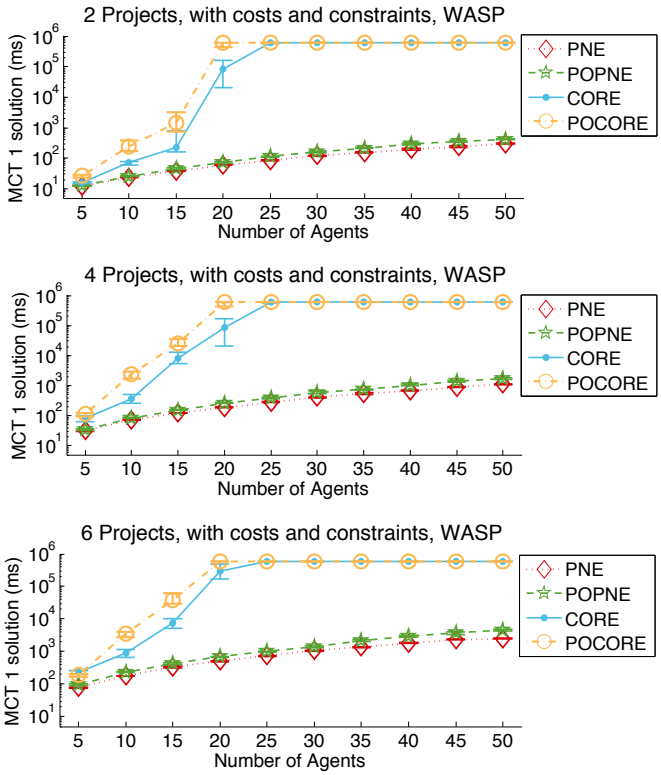


Fig. 8: Median computation time (ms) of 1 solution in project Boolean games using dASP with WASP.

6 Discussion

In this paper we mainly focused on the time efficiency of methods to solve Boolean games. It is important to note that there exist other significant aspects that might be taken into account when deciding which method is appropriate. One could for instance be looking for the most flexible or general method, in which case the technique based on disjunctive answer set programming would be the best option. One can also distinguish between methods based on their ability to compute all solutions (instead of just one sample solution) or prove that no solution exists. In that case, the heuristic methods (TBRS, WSLpS, NAIVE) are not suitable. Also note that, as shown in our experiments, the size of the game – the number of agents and the number of action variables – is important when deciding which method to use, as the heuristic methods tend to perform better than the disjunctive ASP approach for small Boolean games.

Another distinction between the different methods can be made based on whether the problem allows for a central entity and whether such an entity is desirable. This is important because some applications are inherently decentralized, i.e. central control is simply unavailable or too costly to set up. One might also argue that a centralized algorithm presents a single point of failure and is more

vulnerable to manipulation, as one person has all the information of the game at his disposal. This could also raise privacy concerns, because self-interested agents might not be willing to share information, e.g. about their individual goal, with a central entity. In contrast, in decentralized approaches (such as the WSLpS variant in [12]) point-to-point communication can be encrypted and shared with only few peers. Note that one can also create a decentralized variant of TBRS by using individual tabu lists, i.e. one per agent instead of one global tabu list. Similarly, one can adapt the naive random search algorithm to obtain a decentralized variant, in which the agents individually keep changing their strategies randomly unless they play a best response. Another example of a decentralized approach is the bargaining protocol for Boolean games developed by Dunne et al. [15]. Using this bargaining protocol, Pareto optimal outcomes can be obtained for a specific class of Boolean games. In the protocol, agents negotiate in rounds by successively proposing a strategy profile, which the others can accept or reject. Under the quite severe restriction that the goals of the agents are positive (i.e. only $\wedge$ and $\vee$ may be used), the negotiations end during the first round and any strategy profile resulting from the negotiations is Pareto optimal if the agents follow specific negotiation strategies. If the restriction is not met, the negotiations can last n rounds – with n the number of agents – and obtaining a solution is not guaranteed. Note that the condition of positive goals trivially implies the existence of a Pareto optimal PNE and that one can be obtained by computing a model of the conjunction of the goals.

Another aspect that might be important is the amount of communication required to compute a solution, in contexts where communication is expensive or communication channels are limited. For centralized approaches, the communication is limited to the information transmission between the central entity and the agents, i.e. the input and the output of the algorithm. For decentralized approaches, the communication is every intra-agent transmission of information. Investigating which method would be the most suitable w.r.t. the amount of required communication lies beyond the scope of this paper.

Alternative aspects that could be taken into account are, for instance, the spatial complexity of the different methods, or the 'quality' of the obtained solution. In the case of Boolean games, one could for instance measure the quality of a solution through the number of agents that reach their goal. Setting up an analogous experimental evaluation to investigate such criteria is an interesting direction for future work.

Instead of computing the solutions of a Boolean game, one might also be interested in altering the Boolean game such that its solutions satisfy some propositional formula φ. This can be obtained through taxation schemes, which have been investigated for Boolean games with cost functions [29]. These Boolean games impose costs on the agents, depending on which actions they undertake [15]. The taxation scheme in [29] consists of an external agent, called the principal, which imposes additional costs to incentivize the agents to rationally choose a strategy profile that satisfies some propositional formula φ. For example, one can define a taxation scheme such that the resulting Boolean game has at least one PNE and all PNEs satisfy φ. In contrast to WSLpS, this approach is centralized: a central entity uses global information to find a taxation scheme. Moreover, these taxation schemes are not developed with the aim of computing solutions of the original Boolean game, but they are used to incentivize the agents to choose cer-

tain strategies. The scheme alters the original solutions such that the agents are coordinated to new, more desirable solutions.

7 Conclusion

We proposed a method based on disjunctive answer set programming for finding the solutions of a Boolean game. To the best of our knowledge, this is to date the only solver for general Boolean games which does not rely on an exponential representation of normal form games. Depending on the chosen solution concept, the problem of finding a solution of the Boolean game is encoded as a disjunctive answer set program, using the saturation technique. Next, the answer sets of these programs are computed using a state-of-the-art ASP solver. Finally, these answer sets are translated to strategy profiles of the Boolean game. The ASP based solver can compute (Pareto optimal or arbitrary) PNEs and core elements, even in the presence of costs and constraints.

Experimental results using several classes of Boolean games have shown the strengths of the different investigated methods. As expected, methods for normal form games that require all payoff matrix entries turned out to be suitable only for small problem instances. However, even for small problem instances, such methods generally perform worse than other techniques. We have also investigated the heuristic TBRS algorithm, which has also originally been introduced for normal form games but does not require computing all payoff matrix entries. This method proved to be fast to compute PNEs of small Boolean games. Similarly, an adapted version of TBRS can quickly compute a core element for small Boolean games. As soon as the problem instances have a higher number of agents or action variables, the ASP based solver outperforms all other tested techniques. In addition, we showed that, in general, the performance of the disjunctive ASP based technique is comparable for clingo and WASP. Furthermore, the heuristic methods WSLpS, TBRS and NAIVE, that are less flexible, are unable to enforce Pareto optimality. It has also been shown that CompPNEAcycl is indeed very effective to compute PNEs for the specific class of Boolean games it is equipped for.

Finally, we would like to point out that our work not only provides several command-line tools for solving Boolean games, but also provides numerous benchmark data for future work. From this point of view, we believe our work forms an important basis for further improvements and extensions of methods for solving Boolean games.

A Proofs

Proposition 1. To improve readability, we will first briefly sketch the intuition behind the proof. Every strategy profile of the game G corresponds to an answer set of the ASP program $\mathcal{P}_1$. Program part $\mathcal{P}_2$ encodes the possible alternative strategies of the agents. On the one hand, if I is an answer set of $\mathcal{P}$, the minimal model of the reduct must contain *sat*. On the other hand, if there exists an agent whose alternative strategy (encoded by $\mathcal{P}_2$) yields a better response to the actions of the other agents (encoded by $\mathcal{P}_1$) than the strategy profile encoded by $\mathcal{P}_1$, the literal *sat* in $\mathcal{P}_3$ cannot be derived. Hence every strategy profile S_I that corresponds to an answer set I of $\mathcal{P}$ must be a PNE. Conversely, given a PNE S, it is straightforward to write down the corresponding answer set of $\mathcal{P}$ and prove that no smaller model of the reduct

exists, since *sat* must be in every minimal model as a consequence of the assumption that S is a PNE.

Answer set ⇒ PNE Let I be an arbitrary answer set of $\mathcal{P}$. Since for all $p \in V$, either the corresponding literal $act(p)$ or its negation $\neg act(p)$ is contained in I due to rule (1), it holds that S_I is a well-defined strategy profile for G. We denote $S_i = \pi_i \cap S_I$ and $S_{-i} = S_I \setminus \pi_i$, for every agent $i \in N$. It remains to prove that for every agent i: $u_i(S_I) \geq u_i(S_{-i}, s'_i)$, $\forall s'_i \subseteq \pi_i$. Let i be an arbitrary agent in N. If $u_i(S_I) = 1$, then this condition is fulfilled since u_i only takes values from $\{0, 1\}$. If $u_i(S_I) = 0$, then the condition is fulfilled iff $u_i(S_{-i}, s'_i) = 0$, $\forall s'_i \subseteq \pi_i$. Suppose by contradiction that there exists an alternative set of actions $s'_i \subseteq \pi_i$ such that $u_i(S_I) < u_i(S_{-i}, s'_i)$, i.e. $u_i(S_I) = 0$ and $u_i(S_{-i}, s'_i) = 1$. We can now construct a model J of the reduct $\mathcal{P}^I$, which is strictly contained in I. To this end, we make the following observations:

1. For every agent $j \in N$: $goal(j) \in I$ iff $u_j(S_I) = 1$: this follows from rule (2) and the fact that φ_j is true iff $u_j(S_I) = 1$.
2. Since I is an answer set of $\mathcal{P}$, rule (7) implies that $sat \in I$. This rule is not contained in the reduct $\mathcal{P}^I$ and (8) implies that I contains all variables of the form $act'(p)$ and $nact'(p)$ for all $p \in V$ and $ngoal'(j)$ for every agent $j \in N$. In turn rule (8) implies that I also contains all atoms of X' defined in $\mathcal{P}_2$ as components of formulas. The previous observation together with rules (5) imply that $pleased(j)$ is in I for every $j \in N$, since $ngoal'(j)$ is also in I. Obviously all the facts $action(p)$ and $agent(j)$ of $\mathcal{P}$ also belong to I for every $p \in V$ and $j \in N$.
3. If we denote all literals occurring in program $\mathcal{P}_1$ as $\mathcal{B}_1$ and we define $J_1 = I \cap \mathcal{B}_1$, then clearly J_1 satisfies all the rules of $\mathcal{P}_1$ since I is an answer set of $\mathcal{P}$. Moreover, J_1 is a model of the reduct $\mathcal{P}_1^I$, since $\mathcal{P}_1^I = \mathcal{P}_1^{J_1}$. Note that J_1 also contains all the facts $action(p)$ and $agent(j)$ for every $p \in V$ and $j \in N$.
4. Define the interpretation $J_2 = A \cup NA \cup X'_2 \cup NG$ with

$$\begin{aligned} A &= \{act'(p) \mid p \in (S_{-i}, s'_i)\}, \\ NA &= \{nact'(p) \mid p \notin (S_{-i}, s'_i)\}, \\ X'_2 &= \{a \in X' \mid \exists r : a \leftarrow B \in \mathcal{P} : \forall b \in B : b \in A \cup NA \cup X'_2\}, \\ NG &= \{ngoal'(j) \mid j \in N \setminus \{i\}, goal(j) \notin I\}. \end{aligned}$$

   The interpretation J_2 represents a valid strategy profile $S_{J_2} = \{p \mid act(p) \in J_2\}$ and the actions of all agents but i coincide with their actions in S_I. Hence rule (4) will also be satisfied by J_2 for every agent $j \neq i$. Agent i's strategy in J_2 corresponds to s'_i, hence the assumption $u_i(S_{-i}, s'_i) = 1$ implies that $\sim\varphi_i(act(\pi_{-i}), nact'(\neg\pi_i), act'(\pi_i))$ is false and $ngoal'(i)$ cannot be derived, hence J_2 satisfies rule (4) for i as well. Since $\mathcal{P}_2^I = \mathcal{P}_2$, J_2 is a model of $\mathcal{P}_2^I$.
5. Define $J_3 = \{pleased(j) \mid j \in N \setminus \{i\}\}$ and $J = J_1 \cup J_2 \cup J_3$. Our assumption $u_i(S_I) = 0$ and the first observation imply that $goal(i) \notin I$. By definition, we also have $goal(i) \notin J_1$ for $J_1 \subset J$. By definition $ngoal'(i) \notin J_2$ for $J_2 \subset J$. Hence the bodies of the rules (5) are false, which means that *pleased(i)* cannot be derived for any i. Hence, J satisfies these rules for agent i. For every other agent $j \neq i$ such that $goal(j) \in I$, it follows that $goal(j) \in J_1$ for $J_1 \subset J$, and by definition $ngoal'(j) \in J_2$ for $J_2 \subset J$ for every $j \neq i$ such that $goal(j) \notin I$. In any case the rules (5) are fulfilled by J. Since $pleased(i)$ and *sat* are not in J, rule (6) is also fulfilled by J. Since the bodies of the rules in (8) are not satisfied, these rules are all satisfied by J.

Combining these subresults, we see that J is a model of $\mathcal{P}^I$. Moreover, we have $J \subset I$ and $sat \in I \setminus J$, which implies that I is not a minimal model of $\mathcal{P}^I$, contradicting the assumption that I is an answer set of $\mathcal{P}$.

PNE ⇒ Answer set Suppose $S = (s_1, \ldots, s_n)$ is a PNE. Let the sets of atoms X and X' be defined as in the construction of the induced PNE program. We define the sets A, NA and Z as follows:

$$\begin{aligned} A &= \{act(p) \mid p \in S\}, \\ NA &= \{\neg act(p) \mid p \notin S\}, \\ Z &= \{a \in X \mid \exists r : a \leftarrow B \in \mathcal{P} : \forall b \in B : b \in A \cup NA \cup Z\}. \end{aligned}$$

We define an interpretation I_S and prove that it is a minimal model of $\mathcal{P}^{I_S}$:

$$I_S = A \cup NA \cup Z \cup \{goal(i) \mid i \in N, u_i(S) = 1\} \cup \{sat\}$$
$$\cup \{act'(p) \mid p \in V\} \cup \{nact'(p) \mid p \in V\} \cup X' \cup \{ngoal'(i) \mid i \in N\} \cup \{pleased(i) \mid i \in N\}$$
$$\cup \{agent(i) \mid i \in N\} \cup \{action(p) \mid p \in V\}$$

Any literal $act(p)$ in A cannot be omitted without requiring that $\neg act(p)$ is in NA, due to rule (1). Hence it suffices to prove that I_S is minimal among the models of $\mathcal{P}^{I_S}$ containing $A \cup NA$. Clearly the rules of the form (2) uniquely determine which variables of X should be in a minimal model of $\mathcal{P}^{I_S}$ that already contains A and NA. The same reasoning holds for the variables of the form $goal(i)$ for every agent i, i.e. the variables of the form $goal(i)$ with $u_i(S) = 1$ should be in every model of $\mathcal{P}^{I_S}$ already containing A and NA. Since S is a PNE, the actions in S are best responses for every agent i to s_{-i}. We check whether it is possible that there exists a model J of $\mathcal{P}^{I_S}$ that contains $A \cup NA$ but does not contain sat. If we define $S' = (s'_1, \ldots, s'_n)$ with $s'_i = \{p \mid act'(p) \in J\}$, rules (3) imply that S' is a valid strategy profile of the game: sat is not derived hence $\{p \mid act'(p) \in J\}$ and $\{p \mid nact'(p) \in J\}$ form a partition of V. Moreover, for every agent i, it holds that $ngoal'(i)$ belongs to J iff $u_i(S_{-i}, s'_i) = 0$ due to rule (4). But then the fact that S is a PNE implies that $pleased(i)$ is derived by the rules (5) for every agent i, and in turn rule (6) derives sat. Hence sat should be in any model of $\mathcal{P}^{I_S}$ which contains A and NA. But rules (8) imply that such a model of $\mathcal{P}^{I_S}$ should contain $\{act'(p) \mid p \in V\} \cup \{nact'(p) \mid p \in V\}$. Moreover, the rules of the form (4) imply that $X' \cup \{ngoal'(i) \mid i \in N\}$ should be contained as well in the presence of $\{act'(p) \mid p \in V\} \cup \{nact'(p) \mid p \in V\}$. The rules (5) imply that $pleased(i)$ should be in such a minimal model as well for every agent i, since for every agent i we proved that it contains $ngoal'(i)$. Finally note that the facts $agent(1..n)$ and $action(p)$ $(p \in V)$ should be in every model of $\mathcal{P}^{I_S}$. We have proved that every variable of I_S is part of every minimal model of $\mathcal{P}^{I_S}$ containing $A \cup NA$. Moreover, I_S satisfies all rules of $\mathcal{P}^{I_S}$, hence I_S is an answer set of $\mathcal{P}$. □ □

Proposition 2. This proof is analogous to the previous proof, except that the induced core program has some additional literals: $control(.,.)$, $coalition(.)$ and $ncoalition(.)$. The intuition, however, is very similar, except that program part $\mathcal{P}_2$ encodes a coalition and an alternative strategy profile such that non-coalition members play the same strategy as in $\mathcal{P}_1$.

Answer set ⇒ Core element Let I be an arbitrary answer set of $\mathcal{P}$. Since for all $p \in V$, either the corresponding literal $act(p)$ or its negation $\neg act(p)$ is contained in I due to rule (1), it holds that S_I is a well-defined strategy profile for G. We write $\pi_C = \bigcup_{i \in C} \pi_i$, $S_C = \pi_C \cap S_I$ and $S_{-C} = S_I \setminus \pi_C$, for every coalition $C \subseteq N$. It remains to prove that for every coalition $C \subseteq N$ and for all $s'_C \subseteq \pi_C$: $\exists i \in C$: $u_i(S_I) \geq u_i(S_{-C}, s'_C)$. Let C be an arbitrary coalition in N. If there exists a coalition player $i \in C$ such that $u_i(S_I) = 1$ then the condition is fulfilled since u_i takes values in $\{0, 1\}$. If $u_i(S_I) = 0$ for all $i \in C$ then the condition is fulfilled iff $\forall s'_C \subseteq \pi_C : \exists i \in C$: $u_i(S_{-C}, s'_C) = 0$. Suppose by contradiction that there exists an alternative set of actions $s'_C \subseteq \pi_C$ such that $\forall i \in C$: $u_i(S_I) < u_i(S_{-C}, s'_C)$ i.e. $u_i(S_I) = 0$ and $u_i(S_{-C}, s'_C) = 1$ for every $i \in C$. We can then construct a model J of the reduct $\mathcal{P}^I$, which is strictly contained in I. To this end, we make the following observations:

1. For every agent $j \in N$: $goal(j) \in I$ iff $u_j(S_I) = 1$: this follows from rule (2) and the fact that φ_j is true iff $u_j(S_I) = 1$.
2. Since I is an answer set of $\mathcal{P}$, rule (7) implies that $sat \in I$. This rule is not contained in the reduct $\mathcal{P}^I$ and (8) implies that I contains all variables of the form $act'(p)$, $nact'(p)$ (for all $p \in V$), $coalition(i)$, $ncoalition(i)$ and $ngoal'(i)$ (for all $i \in N$). In turn rule (8) implies that I also contains all atoms of X' defined in $\mathcal{P}_2$ as components of formulas. The previous observation together with rules (5) imply that $pleased(j)$ is in I for every $j \in N$, since $ngoal'(j)$ is also in I. Obviously all the facts $action(p)$, $agent(i)$ and $control(i, p_i)$ of $\mathcal{P}$ also belong to I for every $p \in V$, $i \in N$ and $p_i \in \pi_i$.
3. If we denote all literals occurring in program $\mathcal{P}_1$ as $\mathcal{B}_1$ and we define $J_1 = I \cap \mathcal{B}_1$, then clearly J_1 satisfies all the rules of $\mathcal{P}_1$ since I is an answer set of $\mathcal{P}$. Moreover, J_1 is a model of the reduct $\mathcal{P}_1^I$, since $\mathcal{P}_1^I = \mathcal{P}_1^{J_1}$. Note that J_1 also contains all the facts $action(p)$, $agent(i)$ and $control(i, p_i)$ of $\mathcal{P}$ for every $p \in V$, $i \in N$ and $p_i \in \pi_i$.

4. Define the interpretation $J_2 = A \cup NA \cup X'_2 \cup CL \cup NCL \cup NG$ with

$$\begin{aligned} A &= \{act'(p) \mid p \in (S_{-C}, s'_C)\}, \\ NA &= \{nact'(p) \mid p \notin (S_{-C}, s'_C)\}, \\ X'_2 &= \{a \in X' \mid \exists r : a \leftarrow B \in \mathcal{P} : \forall b \in B : b \in A \cup NA \cup X'_2\}, \\ CL &= \{coalition(i) \mid i \in C\}, \\ NCL &= \{ncoalition(i) \mid i \notin C\}, \\ NG &= \{ngoal(i) \mid i \notin C, u_i(S_{-C}, s'_C) = 0\}. \end{aligned}$$

   Since J_2 represents a valid strategy profile $S_{J_2} = \{p \mid act(p) \in J_2\}$, *sat* cannot be derived through rule (3). Similarly, rule (10) cannot derive *sat* since J_2 corresponds to a valid coalition of N (every agent is either a member or not and the coalition is non-empty). Moreover, the actions in J_2 of all non-coalition players coincide with their actions in S_I, hence rules (11) will be satisfied. The actions of coalition members in J_2 corresponds to s'_C, hence $\sim\varphi_i(nact'(\neg V), act'(V))$ is true for $i \notin C$ iff $u_i(S_{-C}, s'_C) = 0$, implying that J_2 satisfies rule (12) for non-coalition members. Furthermore, the assumption $u_i(S_{-C}, s'_C) = 1$ for every $i \in C$ implies that $\sim\varphi_i(nact'(\neg V), act'(V))$ is false and $ngoal'(i)$ cannot be derived, hence J_2 satisfies rule (12) for coalition members as well. Since $\mathcal{P}_2^I = \mathcal{P}_2$, J_2 is a model of $\mathcal{P}_2^I$.
5. Define $J_3 = \{pleased(i) \mid i \notin C, u_i(S_I) \geq u_i(S_{-C}, s'_C)\}$ and $J = J_1 \cup J_2 \cup J_3$. In case of a non-coalition member i, the rules (5) will be satisfied by J_3 since one of the bodies is true iff $u_i(S_I) \geq u_i(S_{-C}, s'_C)$. Our assumption $u_i(S_I) = 0, \forall i \in C$ and the first step imply that $goal(i) \notin I$ for every $i \in C$. By definition the same holds for $J_1 \subset J$. By definition it holds that $ngoal'(i) \notin J_2$ with $J_2 \subset J$ for every $i \in C$. Therefore, the bodies of (5) are not satisfied for coalition members i and $pleased(i)$ cannot be derived, hence J satisfies these rules for every agent $i \in C$. If $i \notin C$, then either $u_i(S_I) \geq u_i(S_{-C}, s'_C)$ or $u_i(S_I) < u_i(S_{-C}, s'_C)$. In the first case, $pleased(i) \in J$ holds, hence rules (5) are satisfied. In the second case, $u_i(S_{-C}, s'_C) \neq 0$ and $u_i(S_I) = 0$, hence the first step and the definition of J imply that $goal(j) \notin J$ and $ngoal(i) \notin J$. Therefore, rules (5) are satisfied. Since $pleased(i)$ is not in J for every $i \in C$, rule (13) is also satisfied by J. The bodies of rules (8) and (15) are falsified by J, hence the rules are satisfied by J.

Combining these subresults, we see the J is a model of $\mathcal{P}^I$. Moreover, $J \subset I$ and $sat \in I \setminus J$. This implies that I is not a minimal model of $\mathcal{P}^I$, contradicting the assumption that I is an answer set of $\mathcal{P}$.

Core element ⇒ Answer set Suppose $S = (s_1, \ldots, s_n)$ is an element of $Core(G)$. Let the sets of atoms X and X' be defined as in the construction of the induced PNE program. Define the following sets of literals:

$$\begin{aligned} A &= \{act(p) \mid p \in S\}, \\ NA &= \{\neg act(p) \mid p \notin S\}, \\ Z &= \{a \in X \mid \exists r : a \leftarrow B \in \mathcal{P} : \forall b \in B : b \in A \cup NA \cup Z\}. \end{aligned}$$

and an interpretation I_S:

$$\begin{aligned} I_S = {} & A \cup NA \cup Z \cup \{goal(i) \mid i \in N, u_i(S) = 1\} \cup \{sat\} \\ & \cup \{act'(p) \mid p \in V\} \cup \{nact'(p) \mid p \in V\} \cup \{coalition(i) \mid i \in N\} \cup \{ncoalition(i) \mid i \in N\} \\ & \cup X' \cup \{ngoal'(i) \mid i \in N\} \cup \{pleased(i) \mid i \in N\} \\ & \cup \{agent(i) \mid i \in N\} \cup \{action(p) \mid p \in V\} \cup \{control(i, p) \mid i \in N, p \in \pi_i\} \end{aligned}$$

Any atom $act(p)$ in A cannot be omitted without requiring that $\neg act(p)$ is then in NA, due to rule (1). Hence we have to prove that I_S is minimal among the models of $\mathcal{P}^{I_S}$ containing $A \cup NA$. Clearly the rules of the form (2) uniquely determine which atoms of X should be in a minimal model of $\mathcal{P}^{I_S}$ which already contains A and NA. The same reasoning holds for the literals of the form $goal(i)$ for every agent i, i.e. the literals of the form $goal(i)$ with $u_i(S) = 1$ should be in every model of $\mathcal{P}^{I_S}$. This is fulfilled by definition of I_S. Since S is a core element, S is not blocked by any coalition. We show that there does not exist a minimal model J of

$\mathcal{P}^{I_S}$ which contains $A \cup NA$ but does not contain *sat*. If we define $S' = (s'_1, \ldots, s'_n)$ with $s'_i = \{p \mid act'(p) \in J\}$, rules (3) imply that S' is a valid strategy profile of the game: *sat* is not derived hence $\{p \mid act'(p) \in J\}$ and $\{p \mid nact'(p) \in J\}$ form a partition of V. Moreover for every agent i it holds that $ngoal'(i)$ belongs to J iff $u_i(S_{-C}, s'_C) = 0$ due to rule (12). But then S being a core element implies that $pleased(i)$ will be derived for a coalition player i by the rules (5) for at least one agent i, and in turn rule (6) derives *sat*. So *sat* should be in any minimal model of $\mathcal{P}^{I_S}$ which contains A and NA. But rules (8) and (15) imply that such a model of $\mathcal{P}^{I_S}$ should contain $\{act'(p) \mid p \in V\} \cup \{nact'(p) \mid p \in V\} \cup \{coalition(i) \mid i \in N\} \cup \{ncoalition(i) \mid i \in N\}$. Moreover, the rules of the form (4) imply that $X' \cup \{ngoal'(i) \mid i \in N\}$ should be contained as well in the presence of $\{act'(p) \mid p \in V\} \cup \{nact'(p) \mid p \in V\}$. The rules (5) imply that $pleased(i)$ should be in such a minimal model as well for every agent i, since for every agent i we proved that $ngoal'(i)$ is in any minimal model. Finally note that the facts $agent(1..n)$, $action(p)$ $(p \in V)$ and $control(i, p)$ $(i \in N, p \in \pi_i)$ should be in every minimal model of $\mathcal{P}^{I_S}$. We have proved that every variable of I_S is part of every minimal model of $\mathcal{P}^{I_S}$ containing $A \cup NA$. Moreover, I_S satisfies all rules of $\mathcal{P}^{I_S}$, hence I_S is an answer set of $\mathcal{P}$. □ □

Proposition 3. First note that S is an absorbing state iff S is a PNE.

WSLpS converges $\Rightarrow \exists S : S$ is a PNE

Convergence of WSLpS applied to G boils down to the fact that the induced Markov chain ends up in an absorbing state. In particular, there must exist an absorbing state S. Due to the observation above, it holds that $\exists S : S$ is a PNE.

WSLpS converges $\Leftarrow \exists S : S$ is a PNE

Assume that there exists a PNE $S^g = (s^g_1, \ldots, s^g_n)$ of G. Then S^g is an absorbing state of the induced Markov chain. For the chain to be absorbing, it is sufficient to prove that for each non-absorbing state there exists an accessible absorbing state. We now prove that S^g is accessible from every non-absorbing state.

Let $S^1 = (s^1_1, \ldots, s^1_n)$ be an arbitrary non-absorbing state, then S^1 is no PNE. In particular there must exist at least one agent i_1 which does not reach its goal, since reaching your goal is a best response by definition. We now show that a strategy profile S^2 exists such that

1. $s^2_j = s^g_j, \forall j \in RA(i_1)$ and $s^2_j = s^1_j, \forall j \in N \setminus RA(i_1)$,
2. the probability of a transition from S^1 to S^2 is non-zero,

For every $j \in RA(i_1)$ there exists a subset of neighbors r_j with $i_1 \in r_j$ and $success(S^1, r_j) = 0$. We define $s^2_j = s^g_j$. For every $j \in N \setminus RA(i_1)$ we define $s^2_j = s^1_j$. Obviously, this S^2 fulfills the first condition. Note that this implies that $u_{i_1}(S^2) = 1$, since $u_{i_1}(S^g) = 1$ and $s^2_j = s^g_j, \forall j \in RA(i_1)$. The fact that the transition probability from S^1 to S^2 is non-zero is due to the following reasons. First, all agents with positive feedback from S^1 stick to their current strategy. Second, all agents with negative feedback can change to any strategy with a non-zero probability; in particular, the probability that all agents j with negative feedback from S^1 'switch' to s^1_j is non-zero. To see this, note that the probability of agent j flipping a variable is $\beta_j(S^1, r_j) = \max(\alpha - \frac{|\{j' \in r_j \mid {u_j}'(S^1)=1\}|}{k_j}, 0)$. If agent j got negative feedback from S^1, then $success(S^1, r_j) = 0$ and there must be at least one neighbor j' of j with ${u_j}'(S^1) \neq 1$. Considering the fact that $\alpha > \frac{k-1}{k}$ and $k_j = \min(k, |Neigh(j)|)$, it follows that $\alpha > \frac{k_j - 1}{k_j}$. Therefore it holds that $\beta_j(S^1, r_j)$ is in $]0, 1[$ if agent j got negative feedback from S^1. Hence, irrespective of whether the transition from s^1_j to s^2_j requires flipping variables or not, the transition probability is non-zero.

Either $\forall i \in N$: $u_i(S^2) = 1$, hence S^2 is an accessible absorbing state, or $\exists i_2 \in N$: $u_{i_2}(S^2) = 0$. As long as there exists an $i_l \in N$ such that $u_{i_l}(S^l) = 0$ we can find a state $S^{l+1} = (s^{l+1}_1, \ldots, s^{l+1}_n)$ such that the transition probability from S^l to S^{l+1} is non-zero, $s^{l+1}_j = s^g_j, \forall j \in RA(i_l)$, and $s^{l+1}_j = s^l_j, \forall j \in N \setminus RA(i_l)$. Moreover, for every i_m with $1 \leq m \leq l$ it holds that $u_{i_m}(S^{l+1}) = 1$ because $\forall j \in RA(i_m) : s^{l+1}_j = s^g_j$. To see this, note that we assumed that every $j \in RA(i_m)$ switched to s^g_j in the transition from S^m to S^{m+1}. In all the next transitions, j kept the previous strategy, hence $s^{l+1}_j = s^g_j$. Clearly $l = n$ is the maximum value

for which S^{l+1} will be an absorbing state accessible from S^1, as a strategy profile for which every agent reaches its goal is a PNE by definition. □ □

References

1. Ågotnes, T., Harrenstein, P., van der Hoek, W., Wooldridge, M.: Verifiable equilibria in Boolean games. In: Proc. IJCAI, pp. 689–695 (2013)
2. Aumann, R.: Acceptable points in games of perfect information. Pacific Journal of Mathematics **10**(2), 381–417 (1960)
3. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press, New York (2003)
4. Ben-Naim, J., Lorini, E.: Evaluating power of agents from dependence relations in boolean games. In: Proc. AAMAS, pp. 853–860 (2014)
5. Bonzon, E., Lagasquie-Schiex, M.C., Lang, J.: Dependencies between players in Boolean games. In: Proc. ECSQARU, pp. 743–754. Springer (2007)
6. Bonzon, E., Lagasquie-Schiex, M.C., Lang, J.: Dependencies between players in Boolean games. Int. J. Approx. Reasoning **50**(6), 899–914 (2009)
7. Bonzon, E., Lagasquie-Schiex, M.C., Lang, J.: Effectivity functions and efficient coalitions in Boolean games. Synthese **187**, 73–103 (2012)
8. Bonzon, E., Lagasquie-Schiex, M.C., Lang, J., Zanuttini, B.: Boolean games revisited. In: Proc. ECAI, pp. 265–269. ACM (2006)
9. Bonzon, E., Lagasquie-Schiex, M.C., Lang, J., Zanuttini, B.: Compact preference representation and Boolean games. Autonomous Agents and Multi-Agent Systems **18**(1), 1–35 (2009)
10. Brewka, G., Eiter, T., Truszczyński, M.: Answer set programming at a glance. Communications of the ACM **54**(12), 92–103 (2011)
11. De Clercq, S., Bauters, K., Schockaert, S., De Cock, M., Nowé, A.: Using answer set programming for solving Boolean games. In: Proc. KR, pp. 602–605 (2014)
12. De Clercq, S., Bauters, K., Schockaert, S., Mihaylov, M., De Cock, M., Nowé, A.: Decentralized computation of Pareto optimal pure Nash equilibria of Boolean games with privacy concerns. In: Proc. ICAART, pp. 50–59 (2014)
13. De Clercq, S., (contact person): `http://www.cwi.ugent.be/BooleanGamesSolver.html`
14. De Vos, M., Vermeir, D.: Choice logic programs and Nash equilibria in strategic games. In: Proc. CSL, pp. 266–276. Springer (1999)
15. Dunne, P., van der Hoek, W., Kraus, S., Wooldridge, M.: Cooperative Boolean games. In: Proc. AAMAS, vol. 2, pp. 1015–1022. IFAAMAS (2008)
16. Dunne, P.E., Wooldridge, M.: Towards tractable Boolean games. In: Proc. AAMAS, pp. 939–946. IFAAMAS (2012)
17. Eiter, T., Gottlob, G.: On the computational cost of disjunctive logic programming: Propositional case. Annals of Mathematics and Artificial Intelligence **15**(3-4), 289–323 (1995)
18. Faber, W., Leone, N., Ricca, F.: Solving hard problems for the 2nd level of the Polynomial Hierarchy: Heuristics and benchmarks. Intelligenza Artificiale **2**(3), 21–28 (2005)
19. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Proc. ICLP/SLP, pp. 1070–1080 (1988)
20. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New generation computing **9**(3-4), 365–385 (1991)
21. Gottlob, G., Greco, G., Scarcello, F.: Pure Nash equilibria: hard and easy games. In: Proc. TARK, pp. 215–230. ACM (2003)
22. Gottlob, G., Greco, G., Scarcello, F.: Pure nash equilibria: Hard and easy games. Journal of Artificial Intelligence Research **24**, 357–406 (2005)
23. Grinstead, C., Snell, J.: Introduction to Probability. American Mathematical Society (1997)
24. Harrenstein, P., van der Hoek, W., Meyer, J.J., Witteveen, C.: Boolean games. In: Proc. TARK, pp. 287–298. Morgan Kaufmann Publishers Inc. (2001)
25. Mihaylov, M.: Decentralized coordination in multi-agent systems. Ph.D. thesis, Vrije Universiteit Brussel, Brussels (2012)
26. Mihaylov, M., Tuyls, K., Nowé, A.: A decentralized approach for convention emergence in multi-agent systems. Autonomous Agents and Multi-Agent Systems **28**(5), 749–778 (2014)

27. Papadimitriou, C.: Computational complexity. Addison-Wesley, Reading, Massachusetts (1994)
28. Sureka, A., Wurman, P.R.: Using tabu best-response search to find pure strategy Nash equilibria in normal form games. In: Proc. AAMAS, pp. 1023–1029 (2005)
29. Wooldridge, M., Endriss, U., Kraus, S., Lang, J.: Incentive engineering for Boolean games. Artificial Intelligence **195**, 418–439 (2013)