On the Verification and Computation of Strong Nash Equilibrium

Nicola Gatti
Politecnico di Milano
Piazza Leonardo da Vinci 32
Milano, Italy
ngatti@elet.polimi.it

Marco Rocco
Politecnico di Milano
Piazza Leonardo da Vinci 32
Milano, Italy
mrocco@elet.polimi.it

Tuomas Sandholm
Carnegie Mellon
Computer Science Dep.
5000 Forbes Avenue
Pittsburgh, PA 15213, USA
sandholm@cs.cmu.edu

ABSTRACT

Computing equilibria of games is a central task in computer science. A large number of results are known for *Nash equilibrium* (NE). However, these can be adopted only when coalitions are not an issue. When instead agents can form coalitions, NE is inadequate and an appropriate solution concept is *strong Nash equilibrium* (SNE). Few computational results are known about SNE. In this paper, we first study the problem of verifying whether a strategy profile is an SNE, showing that the problem is in $\mathcal{P}$. We then design a spatial branch–and–bound algorithm to find an SNE, and we experimentally evaluate the algorithm.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Multi–agent systems

General Terms

Algorithms, Economics

Keywords

Game Theory (cooperative and non–cooperative)

1. INTRODUCTION

Finding solutions of strategic–form games is a central task in computer science. The most studied solution concept is *Nash equilibrium* (NE) [23]. It is appropriate when agents play a game without any form of *a priori* communication, describing a stable state in which no agent can gain more by unilaterally changing her strategy. Any game has at least a mixed–strategy NE, but searching for it is $\mathcal{PPAD}$–complete [9] even with two agents [7]. It is known that $\mathcal{PPAD} \subseteq \mathcal{NP}$ ($\mathcal{PPAD} \notin \mathcal{NP}$–complete unless $\mathcal{NP}$ = co–$\mathcal{NP}$) and it is generally believed that $\mathcal{PPAD} \neq \mathcal{P}$. Thus the worst–case complexity of finding an NE is exponential in the size of the game. Various methods have been adopted to compute NEs, e.g., two–agent games can be solved by linear complementarity mathematical programming (LCP) [18], support enumeration (PNS) [20], or mixed–integer linear programming (MILP) [22].

Appears in: *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013),* Ito, Jonker, Gini, and Shehory (eds.), May 6–10, 2013, Saint Paul, Minnesota, USA.

The NE concept is inadequate when agents can *a priori* communicate, being in a position to form coalitions and deviate multilaterally in a coordinated way. The *strong Nash equilibrium* (SNE) concept strengthens the NE concept by requiring the strategy profile to be resilient also to multilateral deviations, including the grand coalition [2]. However, differently from NE, an SNE may not exist. It is known that searching for an SNE is $\mathcal{NP}$–hard, but, except for the case of two–agent symmetric games, it is not known whether the problem is in $\mathcal{NP}$ [8]. Furthermore, to the best of our knowledge, there is no algorithm to find an SNE in general games. The algorithms known in the literature search only for pure–strategy SNEs with specific classes of games, e.g., congestion games [17, 16, 21], connection games [10], and maxcut games [14].

In this paper, we provide a study of verifying and computing an SNE. Our main contributions are as follows.

- *Verification.* We show that verifying whether a strategy profile in an n–agent game is weakly Pareto efficient is in $\mathcal{P}$. We do this by reducing it to the multi–agent minmax problem where the number of actions of the agent whose minmax value is to be computed is bounded [15]. Thus, verifying whether a strategy profile is an SNE is in $\mathcal{P}$ and finding an SNE is in $\mathcal{NP}$. The same holds for approximate SNEs.
- *Computation.* We exploit the verification problem to design an algorithm to search for an SNE. For simplicity, we focus on two–agent games (in principle, our algorithm can be extended to games with more agents) and we design a spatial branch–and–bound algorithm that iterates between the computation of an NE by an oracle and the verification of an SNE. In addition, we show how our algorithm can be improved when the oracle used to find an NE is MIP Nash [22] (extending this algorithm to polymatrix games is straightforward).
- *Experimental evaluation.* We show that the ubiquitous benchmark testbed for NE, GAMUT [19], is not a suitable testbed for SNE because all the instances are easy: if they admit SNEs, then there is always a pure–strategy SNE. Thus, we design a generator whose instances admit only mixed–strategy SNEs. With these instances we experimentally evaluate our algorithms.
- *Mixed–strategy multilateral deviations.* We show that, differently from what happens with NE, we cannot formulate the conditions of an SNE by a finite set of constraints, one for each pure–strategy uni/multi–lateral

deviation. Rather, mixed–strategy multilateral deviations must be taken into account. This opens the question, left open in this paper, whether or not it is possible to formulate the (necessary and sufficient) conditions of an SNE as a finite set of constraints.

2. GAME–THEORETIC PRELIMINARIES

A strategic–form game is a tuple (N, A, U) where [23]:

- $N = \{1, \ldots, n\}$ is the set of agents (we denote by i a generic agent),
- $A = \{A_1, \ldots, A_n\}$ is the set of agents' actions and A_i is the set of agent i's actions (we denote a generic action by a, and by m_i the number of actions in A_i),
- $U = \{U_1 \ldots, U_n\}$ is the set of agents' utility arrays where $U_i(a_1, \ldots, a_n)$ is agent i's utility when the agents play actions $a_1, \ldots, a_n$.

Without loss of generality, $\max_{a_1,\ldots,a_n}\{U_i(a_1,\ldots,a_n)\} = 1$ and $\min_{a_1,\ldots,a_n}\{U_i(a_1,\ldots,a_n)\} = 0$ for every $i \in N$. We denote by $\mathbf{x}_i$ the strategy (vector of probabilities) of agent i and by $x_{i,a}$ the probability with which agent i plays action $a \in A_i$. We denote by Δ_i the space of strategies over A_i, i.e., vectors $\mathbf{x}_i$ where the probabilities sum to 1.

The central solution concept in game theory is NE. A strategy profile $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ is an NE if, for each $i \in N$, $\mathbf{x}_i^T U_i \prod_{j \neq i} \mathbf{x}_{-j} \geq \mathbf{x}_i'^T U_i \prod_{j \neq i} \mathbf{x}_{-j}$ for every $\mathbf{x}_i' \in \Delta_i$. Every finite game admits at least an NE in mixed strategies. The problem of finding an NE can be expressed as an NLCP:

$$\mathbf{x}_i \geq 0 \qquad \forall i \in N \tag{1}$$

$$\mathbf{1}v_i - U_i \cdot \prod_{j \neq i} \mathbf{x}_j \geq 0 \qquad \forall i \in N \tag{2}$$

$$\mathbf{x}_i^T \cdot (\mathbf{1}v_i - U_i \cdot \prod_{j \neq i} \mathbf{x}_j) = 0 \qquad \forall i \in N \tag{3}$$

$$\mathbf{1}^T \cdot \mathbf{x}_i = 1 \qquad \forall i \in N \tag{4}$$

Here v_i is the expected utility of agent i. Constraints (1) and (4) state that every $\mathbf{x}_i \in \Delta_i$. Constraints (2) state that no pure strategy of agent i gives expected utility greater than v_i. Constraints (3) state that each agent plays only optimal actions. We denote by supp_i the support of the strategy of agent i, i.e., the set of actions played with strictly positive probability by i, while supp denotes the support profile of all the agents. An approximate NE, called ϵ–NE, is a strategy profile $\mathbf{x}$ in which no agent can improve its utility more than $\varepsilon > 0$ by unilaterally deviating.

In [2], Aumann introduced the concept of SNE. An SNE strengthens the NE concept requiring the strategy profile to be resilient also to multilateral deviations by any coalition of agents. That is, in an SNE no coalition of agents can deviate in a way that strictly increases the expected utility of each member of the coalition, again keeping the strategies of the agents outside the coalition fixed. An SNE is an NE and it is weakly Pareto efficient for each possible coalition. Differently from NE, SNE is not assured to exist (even in mixed strategies), as shown in the following example.

Example 2.1. *Consider the game in Fig. 1: we report the bimatrix (left) and the Pareto frontier (right). The unique NE is (a_2, a_4), but it is strictly Pareto dominated by (a_1, a_3).*

Finally, an approximate SNE, called ε–SNE, is a strategy profile $\mathbf{x}$ in which no agent can gain more than $\varepsilon > 0$ by unilaterally or multilaterally deviating, where the only allowed multilateral deviations are those in which all the members of the coalition strictly improve their utility [11].

agent 2

agent 1	a_3	a_4
a_1	$3,3$	$0,5$
a_2	$5,0$	$1,1$

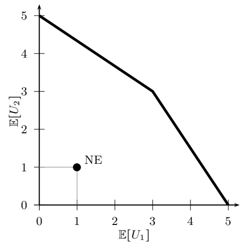

Figure 1: Example of game (prisoner's dilemma) without any SNE (left) and Pareto frontier (right).

3. SNE VERIFICATION

Verifying whether a given solution $\overline{\mathbf{x}}$ is an NE is easy for two reasons:

1. the conditions for a strategy profile to be an NE can be formulated as a set of finite constraints, i.e., (1)–(4);
2. once the strategy variables $\mathbf{x}$ in (1)–(4) have been assigned values of $\overline{\mathbf{x}}$, the constraints are linear in variables v_1 and v_2 and then they are easily solvable.

With SNEs it is not clear whether the two above points hold. Although the NE concept requires that a strategy must be the best w.r.t all the mixed strategies, it is sufficient to require that a strategy is the best w.r.t. all the pure strategies. We show that in the case of SNE, this is not sufficient. For clarity, we focus on two–agent games. In an SNE, in addition to the NE constraints, we need to assure that the agents cannot strictly improve their expected utilities by multilaterally deviating. Limiting the multilateral deviations to pure–strategy deviations is equivalent to requiring that there is no pure outcome that provides both agents with strictly better utilities. We show in the following example that this condition is not sufficient for SNE.

Example 3.1. *Consider the game in Fig. 2. There are three NEs: one pure, (a_3, a_6), and two mixed, $(\frac{1}{2}a_1 + \frac{1}{2}a_2, \frac{1}{2}a_4 + \frac{1}{2}a_5)$ and $(\frac{1}{7}a_1 + \frac{1}{7}a_2 + \frac{5}{7}a_3, \frac{1}{7}a_4 + \frac{1}{7}a_5 + \frac{5}{7}a_6)$. Focus on (a_3, a_6): there is no outcome achievable by pure–strategy multilateral deviations providing both agents a utility strictly greater than 1. For instance, (a_1, a_4) is better for agent 1 than (a_3, a_6), but it is not for agent 2. With (a_2, a_4) we have the reverse. However, (a_3, a_6) is not weakly Pareto efficient, as shown by the Pareto frontier in the figure. Indeed, $(\frac{1}{2}a_1 + \frac{1}{2}a_2, \frac{1}{2}a_4 + \frac{1}{2}a_5)$ strictly Pareto dominates (a_3, a_6). Instead, the former being on the Pareto frontier, it is an SNE.*

agent 2

agent 1	a_4	a_5	a_6
a_1	$5,0$	$0,5$	$0,0$
a_2	$0,5$	$5,0$	$0,0$
a_3	$0,0$	$0,0$	$1,1$

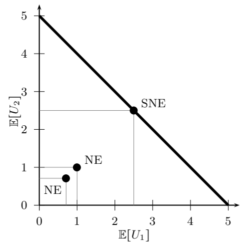

Figure 2: Example of two–agent game (left) and its Pareto frontier (right).

We show that formulating the SNE constraints on the basis of only pure–strategy multilateral deviations is not satisfactory even taking into account the sum of the agents' utilities. (The same holds when a generic function of the agents' utilities is employed in place of the sum.)

Example 3.2. *Consider the game in Fig. 3, where ρ is an arbitrarily small positive value. There is a unique NE, (a_3, a_6), and this NE is weakly Pareto efficient being on the Pareto frontier. Notice that the sum of the agents' utilities at (a_3, a_6) is strictly smaller than the sum at (a_1, a_4).*

		agent 2		
		a_4	a_5	a_6
agent 1	a_1	$5, 0$	$0, 0$	$0, \rho$
	a_2	$0, 0$	$0, 5$	$0, \rho$
	a_3	$\rho, 0$	$\rho, 0$	$2, 2$

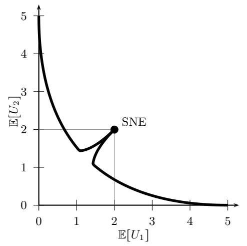

Figure 3: Example of two–agent game (left) and its Pareto frontier (right).

The above examples show the need for taking into account multilateral deviations in mixed strategies. This could prevent one from formulating the SNE conditions as a finite set of constraints where each constraint is related to a pure–strategy (unilateral/multilateral) deviation, and argues for considering the entire Pareto frontier. Finite constraints expressing the membership of a solution to the Pareto frontier can be derived by using Karush–Kuhn–Tucker conditions [5], but, in the case of non–linear non–convex objective functions as is the case with SNE, the set of constraints is only necessary and not sufficient (and therefore we have no guarantee that a strategy is on the Pareto frontier). Nevertheless, although it is not clear whether one can formulate the SNE problem with a set of finite constraints as in the case of NE, we can show that the SNE verification problem is tractable.

We first study the problem of verifying whether a given solution is weakly Pareto efficient.

Theorem 3.3. *Given an n–player game with rational payoffs and a strategy profile $\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_n)$ with rational probabilities, the problem of verifying whether $\mathbf{x}$ is weakly Pareto efficient is in $\mathcal{P}$ when n is constant.*

Proof. Recall that $\mathbf{x}$ is weakly Pareto efficient if

$$\nexists\, \mathbf{x}' = (\mathbf{x}'_1, \dots, \mathbf{x}'_n), \mathbf{x}'_i \in \Delta_i, \nexists\, \phi > 0 : \forall i \in N, \mathbf{x}'_i \cdot U_i \cdot \prod_{j \neq i} \mathbf{x}'_j - \mathbf{x}_i \cdot U_i \cdot \prod_{j \neq i} \mathbf{x}_j \geq \phi \qquad (5)$$

The structure of the proof of theorem is the following:

- (*Step 1*) we show that the problem of verifying whether $\mathbf{x}$ is weakly Pareto efficient is equivalent to the problem of verifying whether the minmax value of a fictitious agent vs. the set N of agents is non–negative;
- (*Step 2*) we show that the problem of verifying whether the minmax value of a fictitious agent vs. the set N of agents is non–negative is in $\mathcal{P}$.

Next we discuss these two steps in detail.

Step 1. Focus on $\mathbf{x}_i \cdot U_i \cdot \prod_{j \neq i} \mathbf{x}_j$: it is a rational value expressing the expected utility of agent i given strategy profile $\mathbf{x}$. Call $\tilde{U}_i = M_1 \cdot (\mathbf{x}_i \cdot U_i \cdot \prod_{j \neq i} \mathbf{x}_j) - U_i$, where M_1 is a multidimensional array of ones. We can show that

$$\begin{aligned} \mathbf{x}'_i \cdot \tilde{U}_i \cdot \prod_{j \neq i} \mathbf{x}'_j &= \mathbf{x}'_i \cdot M_1 \cdot (\mathbf{x}_i \cdot U_i \cdot \prod_{j \neq i} \mathbf{x}_j) \cdot \prod_{j \neq i} \mathbf{x}'_j - \mathbf{x}'_i \cdot U_i \cdot \prod_{j \neq i} \mathbf{x}'_j \\ &= (\mathbf{x}_i \cdot U_i \cdot \prod_{j \neq i} \mathbf{x}_j) \cdot \mathbf{x}'_i \cdot M_1 \cdot \prod_{j \neq i} \mathbf{x}'_j - \mathbf{x}'_i \cdot U_i \cdot \prod_{j \neq i} \mathbf{x}'_j \\ &= \mathbf{x}_i \cdot U_i \cdot \prod_{j \neq i} \mathbf{x}_j - \mathbf{x}'_i \cdot U_i \cdot \prod_{j \neq i} \mathbf{x}'_j \end{aligned}$$

given that $\mathbf{x}'_i \cdot M_1 \cdot \prod_{j \neq i} \mathbf{x}'_j = 1$ because it is a (non–convex) combination of ones. Thus, we can write the condition $\mathbf{x}'_i \cdot U_i \cdot \prod_{j \neq i} \mathbf{x}'_j - \mathbf{x}_i \cdot U_i \cdot \prod_{j \neq i} \mathbf{x}_j \geq \phi$ in (5) as $\mathbf{x}'_i \cdot \tilde{U}_i \cdot \prod_{j \neq i} \mathbf{x}'_j \leq -\phi$.

Now, we consider the following optimization problem

$$\min_{\gamma, \mathbf{x}'_i\ i \in N} \quad \gamma \qquad (6)$$

$$\gamma \geq \mathbf{x}'_i \cdot \tilde{U}_i \cdot \prod_{j \neq i} \mathbf{x}'_j \qquad \forall i \in N \qquad (7)$$

$$\mathbf{1}^T \cdot \mathbf{x}'_i = 1 \qquad \forall i \in N \qquad (8)$$

$$\mathbf{x}'_i \geq \mathbf{0} \qquad \forall i \in N \qquad (9)$$

This minimizes the value of γ subject to the constraint that γ cannot be smaller than $\mathbf{x}'_i \cdot \tilde{U}_i \cdot \prod_{j \neq i} \mathbf{x}'_j$ for every $i \in N$. If the optimal value γ^* is strictly smaller than zero, then $\mathbf{x}'_i \cdot \tilde{U}_i \cdot \prod_{j \neq i} \mathbf{x}'_j < 0$ for every $i \in N$ and therefore there exists $\phi > 0$ such that $\mathbf{x}'_i \cdot \tilde{U}_i \cdot \prod_{j \neq i} \mathbf{x}'_j \leq -\phi$. That is, if $\gamma^* < 0$, then $\mathbf{x}$ is strongly Pareto dominated. Instead, if $\gamma^* = 0$ (it can be observed that γ^* is never strictly positive), we cannot find such a ϕ and therefore $\mathbf{x}$ is weakly Pareto efficient.

Problem (6)–(9) can be interpreted as the minmax problem in which all the agents $i \in N$ minimize (without correlation) the maximum value of a fictitious agent i_f (the $n+1$–th): the fictitious agent i_f has n actions and each action i provides an expected utility of $\mathbf{x}'_i \cdot \tilde{U}_i \cdot \prod_{j \neq i} \mathbf{x}'_j$.

Step 2. Here we consider the problem of verifying whether the minmax value of an agent with n available actions against n opponents, each agent i with $m_i = m$ actions, is (strictly) smaller than a value (in our case zero) when n is given. In [15], the authors show that such a problem can be solved in time $L^{O(1)} \cdot k_1^{O(k_1 \cdot k_2)} \cdot k_3^{k_1 \cdot k_2}$ where:

- L is the bit complexity of the game;
- k_1 is the total number of agents (in our case, $k_1 = n+1$);
- k_2 is the number of actions available to the agent whose minmax value is to be computed (in our case, $k_2 = n$);
- k_3 is the number of actions available to the other agents (in our case, $k_3 = m$).

As a result, we have $L^{O(1)} \cdot (n+1)^{O(n^2)} \cdot m^{n^2+n}$. Given that in our problem $k_1 = k_2 = n$ and n is fixed, the computational complexity is polynomial in the size of the game. To complete the proof of the theorem, we need to show even when each agent $i \in N$ has a potentially different number of actions m_i, the computational complexity stays polynomial. Let $\overline{m} = \max_{i \in N}\{m_i\}$ be the maximum number of actions a (non–fictitious) agent can have, and let $\overline{i} = \arg\max_{i \in N}\{m_i\}$. For each agent $i \in N, i \neq \overline{i}$, we can add $\overline{m} - m_i$ fictitious actions and we can associate each outcomes induced by fictitious actions with 1 for the fictitious agent (recall from Section 2 that the maximum utility of each agent is 1). In this way, each agent $i \in N$ has the same number $\overline{m}$ of actions and the extra actions do not affect the computation of the minmax value, agents never playing the extra ones. □

The algorithm for verifying whether strategy profile $\mathbf{x}$ is weakly Pareto efficient can be obtained by adapting the algorithm presented in [15] to our problem. It is reported in Algorithm 1 where

$\psi(N, \mathbf{x}, \mathsf{supp}) =$

$$\begin{cases} \bigwedge_{i \in N} \left(\sum_{a_1 \in \mathsf{supp}_1} \cdots \sum_{a_n \in \mathsf{supp}_n} (\tilde{U}_i(a_1, \dots, a_n) \cdot \prod_{k \in N} x'_{k, a_k} < 0) \right) & \wedge \\ \bigwedge_{i \in N} \left(\sum_{a_i \in \mathsf{supp}_i} x'_{i, a_i} = 1 \right) & \wedge \\ \bigwedge_{i \in N, a_i \in \mathsf{supp}_i} \left(x'_{i, a_i} \geq 0 \right) \end{cases}$$

The algorithm enumerates (by means of $\mathsf{enumerate}$) all the possible joint supports of size n, because, as shown in [15], there always exists a minmax strategy for the problem (6)–(9) in which all the agents $i \in N$ randomize over at most a number of actions equal to the number of actions of the agent whose minmax value is to be computed (in our case, n). Thus we need to enumerate $\prod_{i \in N} \binom{m_i}{n}$ different joint supports of size n. For each joint support, we need to evaluate formula ψ and we can do that by using the algorithm presented in [4]. If ψ is true, then there is a strategy profile $\mathbf{x}'$ that Pareto dominates the input $\mathbf{x}$.

Algorithm 1 verifyPareto$(N, \{U_i\}_{i \in N}, \mathbf{x})$

```
1: $E \longleftarrow$ enumerate$(\mathsf{supp}_1, \dots, \mathsf{supp}_n : \forall i, |\mathsf{supp}_i| = n)$
2: for all elements of $E$ do
3:     if $\exists(\mathbf{x}'_1, \dots, \mathbf{x}'_n) \in \mathbb{R}^{n^2} : [\psi(N, \mathbf{x}, \mathsf{supp})]$ then
4:         return (false, $\mathbf{x}'$)
5: return (true, $\varnothing$)
```

We leverage Theorem 3.3 to state the following.

Theorem 3.4. *Given an n–player game with rational payoffs and a strategy profile $\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_n)$ with rational probabilities, verifying whether $\mathbf{x}$ is an SNE is in $\mathcal{P}$ when n is constant.*

Proof. Strategy profile $\mathbf{x}$ is an SNE if:

- (*Condition 1*) it is an NE,
- (*Condition 2*) for every $N' \subseteq N$, $(\mathbf{x}_i : i \in N')$ is weakly Pareto efficient once the strategies $\mathbf{x}_j$ of all the other agents $j \in N \setminus N'$ are fixed.

Checking Condition 1 is easy; it is checking whether or not a polynomial number of constraints is satisfied. Checking Condition 2 requires one to verify whether $O(2^n)$ strategy profiles are weakly Pareto efficient where n is given. Checking weak Pareto efficiency is easy as Theorem 3.3 shows and the number of verification problems is a constant in the size of the problem, n being fixed. □

The algorithm to verify whether $(\mathbf{x}_1, \dots, \mathbf{x}_n)$ is an SNE is reported in Algorithm 2. It enumerates all the possible coalitions of agents (except the empty coalition), and for each coalition it verifies whether the coalition strategy is weakly Pareto efficient. If it is not, the algorithm returns the coalition N' for which a multilateral deviation is possible and a strongly Pareto dominant coalition strategy $\mathbf{x}'$ (notice that $\mathbf{x}'$ is not assured to be weakly Pareto efficient).

Algorithm 2 verifySNE$(\mathbf{x})$

```
1: $E \longleftarrow$ enumerate$(N' \subseteq N : N \neq \varnothing)$
2: for all element of $E$ do
3:     $\mathbf{x}'' = (\mathbf{x}_i)_{i \in N'}$
4:     $(\vartheta, \mathbf{x}') \longleftarrow$ verifyPareto$(N', \{U_i \prod_{j \notin N'} \mathbf{x}_j\}_{i \in N'}, \mathbf{x}'')$
5:     if $\neg\vartheta$ then
6:         return (false, $N'$, $\mathbf{x}'$)
7: return (true, $\varnothing$, $\varnothing$)
```

From Theorem 3.4, we can state the following.

Corollary 3.5. *Given an n–player game, the problem of finding an SNE is in $\mathcal{NP}$ when n is constant.*

Combining Corollary 3.5 with the hardness result presented in Corollary 5 of [8], we can state the following.

Corollary 3.6. *Given an n–player game, the problem of finding an SNE is $\mathcal{NP}$–complete when n is constant.*

Corollary 3.6 pushes (unless $\mathcal{P} = \mathcal{NP}$) for the case in which the size of the smallest support of the SNEs rises in $\min_i\{m_i\}$; otherwise the existence of an SNE could be verified by enumerating a number of supports that is polynomial in the size of the game, thus requiring polynomial time. However, the corollary can be proven regardless of whether $\mathcal{P} = \mathcal{NP}$:

Corollary 3.7. *It is possible to construct n–agent game instances with $(m_1, \dots, m_n)$ actions s.t. the size of the smallest support of the SNEs is $\Omega(\min_i\{m_i\})$.*

Proof. It is possible to find instances in which the unique SNE has a support per agent with size $\frac{\min_i\{m_i\}}{2}$, see [1]. □

The above results can be easily extended to the case of ε–SNE. In order to verify whether a strategy profile $\mathbf{x}$ is weakly ε–Pareto efficient, it is enough to substitute < 0 with $< \varepsilon$ in the first row of ψ. The verification of an ε–SNE is the same of SNE except that we need to verify weak ε–Pareto efficiency instead of weak Pareto efficiency. Thus, the same results in Theorems 3.3 and 3.4 and Corollaries 3.5, 3.6, 3.7 stated for SNE hold also for ε–SNE.

4. SNE FINDING WITH 2 AGENTS

With non–degenerate games, a simple algorithm to find an SNE is to enumerate the NEs and to verify whether there is an SNE. However, NE enumeration is $\#\mathcal{P}$–hard [8], while SNE finding is $\mathcal{NP}$–complete. Thus it should likely be possible to design more efficient algorithms. Here we explore alternative better approaches that also work when there is a continuum of equilibria.

4.1 Basic algorithm

Given the difficulties in deriving a finite set of (necessary and sufficient) constraints for the membership of a strategy to be in the Pareto frontier, we propose an algorithm that iterates between the computation of an NE and the verification of an SNE. Basically, the algorithm will compute an NE and verify whether or not it is an SNE. If not, it will compute a new NE excluding the space of strategies that are dominated by strategy profile found during the SNE verification. This process repeats until an SNE is found or it is proven that none exists. Our idea is supported by the experimental evaluation presented in [22], where the authors show that the compute time to find an NE (even optimal) is negligible w.r.t. the compute time to enumerate all the NEs, so calling an NE–finding oracle a number of times can be faster than enumerating all the NEs.

The algorithm we propose is essentially a *spatial branch–and–bound* algorithm. For simplicity, we focus on the case with two agents. A state is denoted by s and the set of states by S. Each state s is associated with a convex subspace V_s of the solution space of the problem (1)–(4). Specifically, V_s is defined by box constraints over v_1 and v_2: $\underline{v}_{1,s} \leq v_1 \leq \overline{v}_{1,s}$ and $\underline{v}_{2,s} \leq v_2 \leq \overline{v}_{2,s}$ (where $\overline{v}_i$ and $\underline{v}_i$ are the upper and lower bounds respectively of the box). We denote by s_0 the state in which $\underline{v}_i = \underline{U}_i$ and $\overline{v}_i = \overline{U}_i$ for every $i \in N$, where $\underline{U}_i$ and $\overline{U}_i$ are respectively the minimum and maximum entries of U_i. Any SNE, if exists, is in V_{s_0}.

Algorithm 3 findingSNE

```
1: $S \longleftarrow$ initialize
2: repeat
3:     $s \longleftarrow$ remove$(S)$
4:     $(\vartheta, \mathbf{x}) \longleftarrow$ findNE$(s)$
5:     if $\vartheta$ = true then
6:         $(\vartheta, \cdot, \mathbf{x}') \longleftarrow$ verifySNE$(\mathbf{x})$
7:         if $\vartheta$ = true then
8:             return $\mathbf{x}$
9:         $S = S \cup$ branch$(s, \mathbf{x}')$
10:        $S \longleftarrow$ filter$(S)$
11: until $S$ is empty
12: return false
```

The algorithm is reported in Algorithm 3 and works as follows. At Step 1, set S is populated with the initial states. Then the algorithm repeats Steps 3–10 while S is not empty. At Step 3 a state s is removed from S and at Step 4 an oracle is called to find an NE in subspace V_s. If there is such an NE $\mathbf{x}$, at Step 6 the algorithm verifies whether $\mathbf{x}$ is an SNE and, in the affirmative case, $\mathbf{x}$ is returned. In the negative case in which $\mathbf{x}$ is Pareto dominated by $\mathbf{x}'$, at Step 9 new states are generated from the current state s in which the subspace dominated by $\mathbf{x}'$ is excluded. At Step 10, redundant states in S are pruned. We now discuss the subroutines that the algorithm calls in more detail.

$S \longleftarrow$ initialize: Generates the initial set S of states. We consider two possible initializations. In the first (*init1*), we assign $S = \{s_0\}$. In the second (*init2*), we compute all the pure–strategy profiles that are resilient to pure–strategy multilateral deviations. This can be done in polynomial time by comparing agents' utilities provided by each outcome w.r.t. the agents' utilities provided by all the other outcomes. Call X the set of pairs $(\hat{v}_{1,h}, \hat{v}_{2,h})$ of the agents' expected utilities given by these strategy profiles. We generate a number of states to exclude the subspace dominated by the elements in X. Order the elements of X in increasing order of $\hat{v}_{1,h}$ and call $\overline{h}$ the number of elements in X. The following states s are generated: for every $h \in \{1, \dots, \overline{h} - 1\}$ a state s is generated with $V_s = [\hat{v}_{1,h}, \hat{v}_{1,h+1}] \times [\hat{v}_{2,h+1}, \overline{U}_2]$; two additional states with zero–measure V_s are generated, the first with $V_s = [\underline{U}_1, \hat{v}_{1,1}] \times [\hat{v}_{2,1}, \hat{v}_{2,1}]$, the second with $V_s = [\hat{v}_{1,\overline{h}}, \hat{v}_{1,\overline{h}}] \times [\underline{U}_2, \hat{v}_{2,\overline{h}}]$. With respect to *init1*, *init2* excludes some dominated subspaces avoiding the algorithm to search for an NE in such subspaces, but, on the other hand, it can introduce a large number of states requiring the algorithm to call the NE–finding oracle many more times.

Example 4.1. *Consider the game reported in Fig. 4. We show the composition of X when* init2 *is used. X is composed by four elements $\omega_1, \omega_2, \omega_3, \omega_4$. The generated states s_1, s_2, s_3 are reported in the figure as boxes delimited by dashed lines: $V_{s_1} = [3, 3.4] \times [3.4, 5]$, $V_{s_2} = [3.4, 5] \times [3, 5]$, $V_{s_3} = [5, 10] \times [0, 5]$ (for simplicity we omit the states with zero–measure V_s in what follows).*

$s \longleftarrow$ remove(S): Receives the set of states S as input, removes a state s from S according to some given strategy (e.g., depth first, breadth first, random), and returns s.

$(\vartheta, \mathbf{x}) \longleftarrow$ findNE(s): Receives a state s as input, searches for an NE in the subspace bounded by V_s, and returns $\vartheta =$ true and an NE $\mathbf{x}$, if there is an NE where agents' utilities are in V_s, and $\vartheta =$ false, otherwise. The introduction of constraints V_s makes some algorithms available in the literature to find an NE not to be applicable, e.g., the Lemke–Howson. NE–finding oracles can be: PNS, LS–PNS [6], MIP Nash.

Example 4.2. *In Fig. 4:* findNE(s_1) *would return the NE with $v_1 = v_2 = 3.4$;* findNE(s_2) *could return either the NE returned with s_1 or the mixed NE with $v_1 = v_2 = 4$;* findNE(s_3) *would return the NE with $v_1 = 8.2$ and $v_2 = 0$;* findNE(s) *with $s \in \{s_4, s_5, s_6\}$ would return no NE;* findNE(s_7) *would return the mixed NE with $v_1 = v_2 = 4$.*

$S' \longleftarrow$ branch($s, \mathbf{x}$): Receives a state s and a strategy profile $\mathbf{x}$ and returns a number of new states in which the subspace dominated by $\mathbf{x}$ is excluded. The generation of the new states is as follows. Call $\hat{v}_{1,s}$ and $\hat{v}_{2,s}$ the expected utilities of agents 1 and 2 respectively provided by $\mathbf{x}$ given as input. A new state s' is generated with $V_{s'} = [\underline{v}_{1,s}, \min\{\overline{v}_{1,s}, \hat{v}_{1,s}\}] \times [\hat{v}_{2,s}, \overline{v}_{2,s}]$ and, if $\min\{\overline{v}_{1,s}, \hat{v}_{1,s}\} \neq \overline{v}_{1,s}$, a new state s'' is generated with $V_{s''} = [\min\{\overline{v}_{1,s}, \hat{v}_{1,s}\}, \overline{v}_{1,s}] \times [\underline{v}_{2,s}, \overline{v}_{2,s}]$.

Example 4.3. *In Fig. 4:* branch($s_3, \mathbf{x}_{\omega_5}$) *produces two states, s_4 and s_5, where $V_{s_4} = [5, 8.2] \times [0.4, 5]$ (dark gray) and $V_{s_5} = [8.2, 10] \times [0, 5]$ (light gray);* branch($s_2, \mathbf{x}_{\omega_6}$) *produces two states, s_6 and s_7, where $V_{s_6} = [3.4, 3.7] \times [3.6, 5]$ (dark gray) and $V_{s_7} = [3.7, 5] \times [3, 5]$ (light gray). ($\mathbf{x}_{\omega_j}$ is the strategy profile associated with ω_j.)*

$S' \longleftarrow$ filter(S): Receives the set of states S as input and returns a set S' subset of S after having pruned states. If there is a pair of states s, s' with $s' \neq s$ and such that $\underline{v}_{1,s} \leq \underline{v}_{1,s'}$ and $\underline{v}_{2,s} \leq \underline{v}_{2,s'}$, then we can remove s and s' and add a new state s'' with $V_{s''} = [\underline{v}_{1,s}, \overline{v}_{1,s'}] \times [\underline{v}_{2,s'}, \overline{v}_{2,s'}]$.

Example 4.4. *In Fig. 4: suppose $S = \{s_1, s_4, s_5, s_6, s_7\}$. States s_1 and s_6 can be removed and state s_8 with $V_{s_8} = [3, 3.7] \times [3.6, 5]$ can be added.*

Theorem 4.5. *Algorithm 3 is sound and complete.*

Soundness is by definition of findNE and of verifySNE. Completeness is due to filter and branch removing only Pareto-dominated solution subspaces. When NEs are finite, Algorithm 3 terminates in finite time. In the next section, we propose a variation able to deal also with games admitting a continuum of NEs (however, while Algorithm 3 can be, in principle, extended to the case with more than two agents, it is not clear whether the algorithm described in the next section can be extended to such case).

4.2 Iterated MIP 2StrongNash

By exploiting MILP tools we can avoid having to spatially branch the solution space into subspaces. We call this new algorithm *iterated MIP 2StrongNash*, given that it exploits iteratively a slightly modified version of MIP Nash for SNE. It is reported in Algorithm 4. With this algorithm, we have a unique state at each iteration and, instead of generating additional states, we add non–convex constraints. The branch and bound process per state is relegated to the MILP solver.

Algorithm 4 Iterated MIP 2StrongNash

```
 1: initialize
 2: while true do
 3:     $(\vartheta, \mathbf{x}) \longleftarrow$ findNE
 4:     if $\vartheta$ = false then
 5:         return false
 6:     $(\vartheta, \cdot, \mathbf{x}') \longleftarrow$ verifySNE($\mathbf{x}$)
 7:     if $\vartheta$ = true then
 8:         return $\mathbf{x}$
 9:     branch
10:     filter
```

The core of this algorithm is the MIP Nash formulation [22] for the computation of an NE (findNE):

constraints (1), (2), (4)

$$\mathbf{1}v_i - U_i \cdot \mathbf{x}_{-i} \leq \overline{U}_i \cdot (\mathbf{1} - \mathbf{b}_i) \qquad \forall i \in N \qquad (10)$$

$$\mathbf{x}_i \leq \mathbf{b}_i \qquad \forall i \in N \qquad (11)$$

$$\mathbf{b}_i \in \{0, 1\}^{m_i} \qquad \forall i \in N \qquad (12)$$

Given that the above program is a MILP, integer and/or linear constraints can be easily added. We now describe the subroutines employed in the algorithm.

initialize. In the case of *init1*, no additional constraint is added. In the case of *init2*, after having found X, we add three constraints for each element $(\hat{v}_{1,k}, \hat{v}_{2,k})$:

		agent 2						
		a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}
agent 1	a_1	$5, 0$	$5, 3$	$3, 5$	$-10, -10$	$-10, -10$	$-10, -10$	$-10, -10$
	a_2	$0, 0$	$3, 5$	$5, 3$	$-10, -10$	$-10, -10$	$-10, -10$	$-10, -10$
	a_3	$10, 0$	$5, 0$	$0, 1$	$-10, 1$	$-10, -10$	$-10, -10$	$-10, -10$
	a_4	$-10, -10$	$-10, -10$	$-10, -10$	$8.2, 0$	$-10, -10$	$-10, -10$	$-10, -10$
	a_5	$-10, -10$	$-10, -10$	$-10, -10$	$-10, -10$	$2, 2$	$-10, -10$	$-10, -10$
	a_6	$-10, -10$	$-10, -10$	$-10, -10$	$-10, -10$	$-10, -10$	$2, 4$	$-10, -10$
	a_7	$-10, -10$	$-10, -10$	$-10, -10$	$-10, -10$	$-10, -10$	$-10, -10$	$3.4, 3.4$

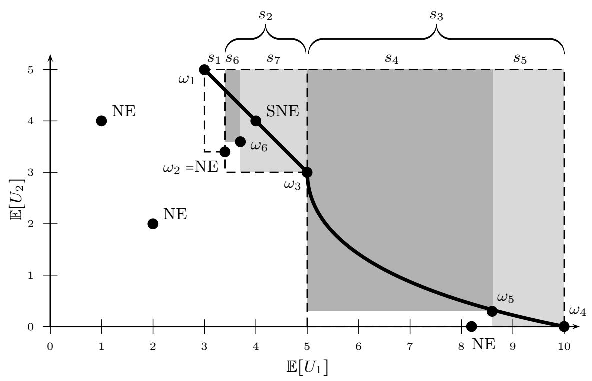

Figure 4: Example of two–agent game (left) and its Pareto frontier (right).

$$v_1 \geq \hat{v}_{1,k} + (\underline{U}_1 - \overline{U}_1) \cdot z_k \tag{13}$$

$$v_2 \geq \hat{v}_{2,k} + (\underline{U}_2 - \overline{U}_2) \cdot (1 - z_k) \tag{14}$$

$$z_k \in \{0, 1\} \tag{15}$$

where constraints (13) and (14) exclude that both v_1 and v_2 are simultaneously smaller than $\hat{v}_{1,k}$ and $\hat{v}_{2,k}$, respectively. A different binary variable z_k is introduced for each k.

branch. After having found an NE $\mathbf{x}$ that is Pareto dominated by $\mathbf{x}'$ with agents' utilities $(\hat{v}_1, \hat{v}_2)$, constraints (13)–(14) are added with a new binary variable z_k and $\hat{v}_{1,k} = \hat{v}_1$ and $\hat{v}_{2,k} = \hat{v}_2$.

filter. If there are two variables $z_k, z_{k'}$ with $z_k \neq z_{k'}$ and such that $\hat{v}_{1,k} \leq \hat{v}_{1,k'}$ and $\hat{v}_{2,k} \leq \hat{v}_{2,k'}$ then variable z_k and the three constraints associated with z_k can be removed.

It is easy to add features to the algorithm. Here we discuss some relevant examples of that.

Social welfare maximization. A way to find Pareto efficient solutions is to maximize the cumulative utility of the agents. We can do it by adding an objective function:

$$\max \quad v_1 + v_2 \tag{16}$$

By employing this feature, Iterated MIP 2Strong–Nash returns an optimal SNE. This feature allows the algorithm to terminate within finite time even on degenerate games that admit a continuum of NEs. This is because, although a continuum of NEs contains infinitely many NEs, there is only one NE of the continuum that maximizes social welfare (in a NE continuum, the utility of only one agent can vary). Thus, either such an NE is an SNE and the algorithm terminates or all the NEs of the continuum are Pareto dominated and therefore they are all discarded in one iteration.

Upper bound over social welfare. We can exploit the social welfare maximization also to add constraints over the solution space. Specifically, call v^* the value of the objective function at iteration k. We can add the constraint

$$v_1 + v_2 \leq \overline{sw} \tag{17}$$

at iteration $k + 1$ where $\overline{sw} = v^*$. Therefore, $\overline{sw}$ decreases monotonically at each iteration.

Lower bound over social welfare. We can find a linear lower bound over the social welfare as follows. We order all the constraints (13)–(15) in increasing order of $\hat{v}_{1,k}$. Let $\overline{k}$ to be the number of constraints (13)–(15). We define $\underline{sw} = \min_{k \in \{1,\ldots,\overline{k}-1\}} \{\hat{v}_{1,k}, \hat{v}_{2,k+1}\}$. It is the lower bound on social welfare. If $\underline{sw} > \overline{sw}$, then there is no SNE. Otherwise, we can call findNE with an additional constraint

$$v_1 + v_2 \geq \underline{sw} \tag{18}$$

Constraints (17) and (18) are redundant. However, they can be used by the MILP solver to speed up the compute time.

5. EXPERIMENTAL EVALUATION

We experimentally evaluate the various configurations of Iterated MIP 2StrongNash, and subsequently, due to limited space, we report our experimental observations obtained from a preliminary comparison of such an algorithm w.r.t. Algorithm 3 and w.r.t. a simple algorithm that combines NE enumeration with SNE verification. We considered four configurations of Iterated MIP 2StrongNash: mode 0 in which *init1* is used and no additional features are active, mode 1 in which *init1* is used with the social welfare maximization and the upper bound over the social welfare, mode 2 is as mode 1 plus *init2*, and mode 3 is as mode 2 without social welfare maximization. Although we showed that it is possible to produce instances such that each configuration is the best and instances in which each configuration is the worst (omitted here due to reasons of space), it is interesting to evaluate them in the average case. We implemented the SNE verification in the C programming language and the NE–finding oracle in AMPL with CPLEX. We used an Intel 2.20GHz processor and Linux kernel 2.6.32.

GAMUT and SNEs. Turns out that the ubiquitous benchmark testbed for NE, GAMUT [19], is not a suitable testbed for SNE finding. We generated 20 instances of 2–agent games per GAMUT game class with a number of actions per agent in $\{25, 50\}$. We report in Tab. 1 the following data for 50 actions per agent: percentage of instances admitting an SNE ('Y'), percentage of instances admitting mixed–strategy SNEs ('mY'), percentage of instances admitting only mixed–strategy SNEs ('omY'), average compute time of Iterated MIP 2StrongNash with mode 0 ('time'), average compute time when there is an SNE ('time|Y'), and average compute time where there is no SNE ('time|N').

Only TravellerDilemma and BertrandOlygopoly instances never admit SNEs, whereas, with the other classes, the percentage of instances admitting an SNE is high on average (with many classes, all the generated instances admit SNEs). Interestingly, only WarOfAttrition instances admit mixed–strategy SNEs, but no instance admits only mixed–strategy SNEs. The results with 25 actions per agent are similar [1]. Therefore, with all the generated GAMUT instances an SNE can be found quickly by enumerating all the pure–strategy NEs and employing SNE verification. The time spent by Iterated MIP 2Strong Nash to find an SNE is much longer than the time spent by MIP Nash to find an NE, see, e.g., [22]. (Computing an SNE requires about 100 times the time needed to find an NE.) As in the case of NE finding, the hardest classes are the Covariant, Graphical, Polymatrix, and Random. Finally, finding an SNE when it exists is significantly faster than certifying that it does not exist.

Novel *ad hoc* game instances. We design a game generator, named *mixed–strategy SNE instance generator* (MISSING), able to produce instances admitting only mixed–strategy SNEs (see [1]). The parameters of the generator are: number of actions per agent, SNE existence or non–existence (in the case of existence, we have only one SNE), size of the support of the SNE (per agent), number of non–strong NEs (generated randomly with a social welfare that can be larger or smaller than the SNE's one). We generated 20 instances per combination of the following parameters: $m_1 = m_2 = m$ from 10 to 100, $|\mathsf{supp}| \in \{0, 2, 4, 6, 8, 10\}$ where $|\mathsf{supp}| = 0$ means that there is no SNE, other NEs $\in \{0, 2, 4, 6, 8, 10, 12\}$. We solved these instances with Iterated MIP 2StrongNash. The main results are in Figs. 5 and 6, complete results are in [1]: the black solid line is with no other NEs, the black dashed line is with 6 non–strong NEs, the gray line is with 12 non–strong NEs.

Initially, we evaluate the different features of the algorithm. We observed that the social welfare maximization leads to a significant reduction of the iterations number (about 3–4 times), but it strongly negatively affect the compute time per iteration spent by the NE–finding oracle (up to 24 times). Interestingly, we observed that the largest increase in NE–finding compute time is in the iteration in which the SNE is returned, while with the other iterations the increase is smaller. This is because the SNE presents a large support and it results harder to be found, instead the other non–strong NEs present a small support. We observed that the lower bound over the social welfare does not provide significant improvements with the used instances. We observed that *init2* reduces the iterations number without increasing significantly the compute time per iteration of the NE–finding oracle. Thus, we focus on mode 2 and mode 3, being the most significative configurations.

It can be observed in the figures that mode 3 has the best compute time (4 times better at $m = 100$). Differently from mode 2, mode 3 does not present an exponential growth around $m = 100$. Surprisingly, as the number of non–strong NEs increases, the compute time does not increase significantly. There are two main reasons. First, a large portion of time is devoted to the verification (50%–80% with mode 3 and 10%–30% with mode 2) and almost all the verification time is spent on the verification of the SNE (this time is the same for all the instances and modes), while the verification of the NEs that are not SNEs requires much less time. Second, the NE–finding oracle rapidly finds the NEs that are not SNEs (because they have a small support of 1 or 2), while finding the SNE (that has a much larger support) is much harder. Thus the presence of additional small–supported non–strong NEs produces non–significative effects. Instead, compute time rises exponentially in $|\mathsf{supp}|$ of SNE. This shows the need for developing techniques to speed up the SNE verification algorithm. It shows also that in the average case mode 3 is the best and this is because with mode 2 the NE–finding oracle is much more expensive. We believe that it would be interesting to develop new game classes combining hard GAMUT classes with MISSING instances to evaluate for what classes mode 3 would still be best and for what classes welfare maximizing is useful.

Comparison to other algorithms. We used MISSING instances in the following comparisons.

We compared Iterated MIP 2StrongNash and Algorithm 3 when MIP Nash is the NE–finding oracle in terms of the compute time. Iterated 2StrongNash dramatically outperforms Algorithm 3—by more than one order of magnitude. The main reason is that Algorithm 3 solves each NE–finding subproblem on V_s independently from the others and there is no a clear heuristic to select the next subproblem to solve. It is interesting to observe that the relationship between Iterated MIP 2StrongNash and Algorithm 3 with MIP Nash as the NE–finding oracle is close to the relationship between MIP Nash and PNS. MIP Nash deals with all the supports together, while PNS works with each support separately. PNS adopts a specific heuristic, scanning the supports from smallest to largest (this holds in almost all the GAMUT instances). When there is no NE with small support, PNS is dramatically worse than MIP Nash. In our case, we did not find a good heuristic for Algorithm 3 because for each heuristic it is possible to design a worst–case game instance in which the number of calls to the NE–finding oracle is equal to the number of NEs. Thus, we selected randomly the next state s to which apply the NE–finding oracle.

class	Y	mY	omY	time	time|Y	time|N
BertrandOlygopoly	0%	0%	0%	1963.6 s	–	1963.6 s
BidirectionalLEG–RG	95%	0%	0%	29.6 s	29.6 s	8.6 s
CovariantGame–Rand	55%	0%	0%	17,701.5 s	53.9 s	39,270.9 s
CovariantGame–Zero	40%	0%	0%	3,734.1 s	1,184.2 s	5,434.1 s
GraphicalGame–Road	35%	0%	0%	15,247.7 s	495.0 s	23,191.4 s
GraphicalGame–SG	60%	0%	0%	10,217.3 s	4,029.5 s	19,498.9 s
MinimumEffortGame	100%	0%	0%	22.3 s	22.3 s	–
PolymatrixGame–CG	45%	0%	0%	3,860.0 s	892.9 s	6,287.6 s
PolymatrixGame–SW	25%	0%	0%	6,950.3 s	3,478.7 s	8,107.5 s
RandomGame	45%	0%	0%	6,998.0 s	6,229.9 s	7,628.2 s
TravelersDilemma	0%	0%	0%	101.7 s	–	101.7 s
UniformLEG–CG	75%	0%	0%	33.2 s	35.9 s	25.2 s
UniformLEG–SG	100%	0%	0%	8.5 s	8.5 s	–
WarOfAttrition	100%	35%	0%	13.8 s	13.8 s	–

Table 1: Results with 50x50 GAMUT instances: instances that admit at least one SNE (Y), instances that admit at least one mixed–strategy SNE (mY), instances that admit only mixed–strategy SNE (omY), compute time (time), compute time when there is an SNE (time|Y), compute time when there is no SNE (time|N).

The adoption of PNS as the NE–finding oracle in Algorithm 3 is not satisfactory. It beats Iterated MIP 2StrongNash only when Algorithm 3 finds an SNEs by a sequence of NE–finding subproblems, each admitting an NE with small support. Otherwise, if some subproblem does not admit any NE or admits one with large support, PNS must enumerate a very large number of supports, making it slow.

Finally, we report experimental results on the employment of SNE verification with an NE enumeration algorithm [3] that is stopped every time an NE is found and then verified. Enumerating all the NEs, necessary when an SNE does not exist, requires more than 300 s with 20x20 games and does not terminate in one day with 50x50 games, while Iterated MIP 2StrongNash terminates within 150 s with 100x100 games. Even when an SNE exists, we did not observe any termination in one day with 100x100 games.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we studied the verification (with n agents) and computation (with two agents) of a strong Nash equilibrium (SNE). A number of results for Nash equilibrium (NE) are known, but that concept is inappropriate when coalitions are an issue. Unlike in NE finding, here mixed–strategy deviations must be taken into account, which makes the problem significantly more difficult. We showed that the verification problem is nevertheless in $\mathcal{P}$ given an arbitrary n–agent game, and therefore the problem of finding an SNE is in $\mathcal{NP}$.

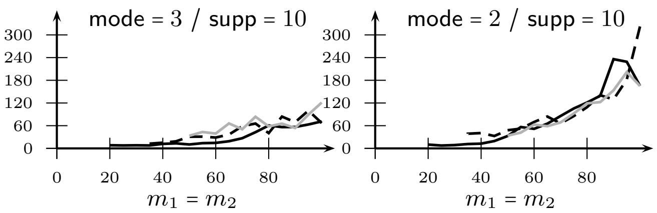

Figure 5: Average compute time (s).

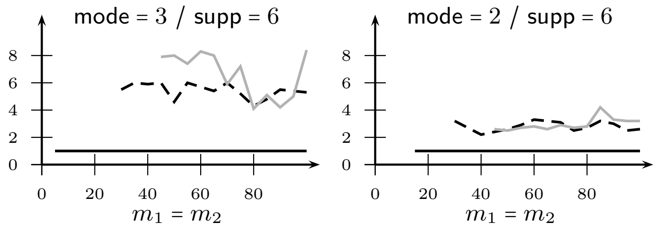

Figure 6: Average iterations.

Then, we exploited our algorithm for SNE verification to design an algorithm for finding an SNE. It is based on spatial branch–and–bound and iterates between a NE–finding oracle and the verification algorithm. Finally, we experimentally evaluated our algorithm. We showed that the instances from the ubiquitous NE benchmark testbed, GAMUT, are not suitable for testing SNE–finding algorithms because all the instances either admit pure SNEs or do not admit any SNE. Then we compared different configurations of our algorithm using a new instance generator to identify the best one. With these instances, it turns out that (our algorithm for) SNE finding takes about 100 times as long as NE finding. In addition, our algorithm dramatically outperforms the approach of enumerating NEs and verifying them for SNE; therefore it is the current state of the art for finding mixed–strategy SNEs even when NEs are finite.

Remaining open questions include the following:

Question 6.1. *How can Algorithms 3 and 4 be extended to the case with an arbitrary number of agents [12]?*

Question 6.2. *Given a game with two agents and with m actions per agent, what is the worst case number of calls to the NE–finding oracle?*

Question 6.3. *Given a game with an arbitrary number of agents, is it possible to formulate the SNE–finding problem with a finite set of (necessary and sufficient) constraints?*

Question 6.4. *What is the smoothed complexity of finding an SNE [13]?*

Acknowledgments

The authors thank Zongque Xu for his collaboration. Nicola Gatti was supported by the Italian Ministry for Instruction, University and Research (MIUR) under Protocol N. 2009BZM837 (PRIN PeopleNet). Tuomas Sandholm was supported by the National Science Foundation under grants IIS–0964579, CCF–1101668, and IIP–1127832.

7. REFERENCES

[1] Appendix. https://sites.google.com/site/aamas13sne/home/onlineappendix.pdf.

[2] R. Aumann. Acceptable points in games of perfect information. *PAC J MATH*, 10:381–417, 1960.

[3] D. Avis, G. D. Rosenberg, R. Savani, and B. von Stengel. Enumeration of Nash equilibria for two–player games. *ECON THEORY*, 42(1):9–37, 2010.

[4] S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry.* Springer, 2003.

[5] D. P. Bertsekas. *Nonlinear Programming.* Athena Scientific, 1999.

[6] S. Ceppi, N. Gatti, G. Patrini, and M. Rocco. Local search techniques for computing equilibria in two–player general–sum strategic–form games. In *AAMAS*, pages 1469–1470, 2010.

[7] X. Chen, X. Deng, and S.-H. Teng. Settling the complexity of computing two–player Nash equilibria. *J ACM*, 56(3):14:1–14:57, 2009.

[8] V. Conitzer and T. Sandholm. New complexity results about Nash equilibria. *GAME ECON BEHAV*, 63(2):621–641, 2008.

[9] C. Daskalakis, P. Goldberg, and C. Papadimitriou. The complexity of computing a Nash equilibrium. In *STOC*, pages 71–78, 2006.

[10] A. Epstein, M. Feldman, and Y. Mansour. Strong equilibrium in cost sharing connection games. In *ACM EC*, pages 84–92, 2007.

[11] M. Feldman and T. Tamir. Approximate strong equilibrium in job scheduling games. *J ARTIF INTELL RES*, 36:387–414, 2009.

[12] N. Gatti, M. Rocco, and T. Sandholm. Algorithms for strong Nash equilibrium with more than two agents. *Mimeo*, 2013.

[13] N. Gatti, M. Rocco, and T. Sandholm. On the complexity of strong Nash equilibrium: Hard–to–solve instances and smoothed complexity. *Mimeo*, 2013.

[14] L. Gourvès and J. Monnot. On strong equilibria in the max cut game. In *WINE*, pages 608–615, 2009.

[15] K. A. Hansen, T. D. Hansen, P. B. Miltersen, and T. B. Sørensen. Approximability and parameterized complexity of minmax values. In *WINE*, pages 684–695, 2008.

[16] A. Hayrapetyan, E. Tardos, and T. Wexler. The effect of collusion in congestion games. In *STOC*, pages 89–98, 2006.

[17] R. Holzman and N. Law-Yone. Strong equilibrium in congestion games. *GAME ECON BEHAV*, 21:85–101, 1997.

[18] C. Lemke and J. Howson. Equilibrium points of bimatrix games. *SIAM J APPL MATH*, 12(2):413–423, 1964.

[19] E. Nudelman, J. Wortman, K. Leyton-Brown, and Y. Shoham. Run the GAMUT: A comprehensive approach to evaluating game–theoretic algorithms. In *AAMAS*, pages 880–887, 2004.

[20] R. Porter, E. Nudelman, and Y. Shoham. Simple search methods for finding a Nash equilibrium. *GAME ECON BEHAV*, 63:642–662, 2009.

[21] O. Rozenfeld and M. Tennenholtz. Strong and correlated strong equilibria in monotone congestion games. In *WINE*, pages 74–86, 2006.

[22] T. Sandholm, A. Gilpin, and V. Conitzer. Mixed–integer programming methods for finding Nash equilibria. In *AAAI*, pages 495–501, 2005.

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