

# COMPUTING PURE STRATEGY NASH EQUILIBRIA IN COMPACT, SYMMETRIC GAMES WITH A FIXED NUMBER OF ACTIONS

ALBERT XIN JIANG, CHRISTOPHER THOMAS RYAN, AND KEVIN LEYTON-BROWN

ABSTRACT. We analyze the complexity of computing pure strategy Nash equilibria (PSNE) in symmetric games with a fixed number of actions. We restrict ourselves to “compact” representations, meaning that the number of players can be exponential in the representation size. We show that in the general case, where utility functions are represented as arbitrary circuits, the problem of deciding the existence of PSNE is NP-complete. For the special case of games with two actions, we show that there always exist a PSNE and give a polynomial-time algorithm for finding one. We then focus on a specific compact representation: piecewise-linear functions. We give polynomial-time algorithms for finding a sample PSNE and for counting the number of PSNE. Our approach makes use of Barvinok and Wood’s rational generating function method [3], which enables us to encode the set of PSNE as a generating function of polynomial size.

## 1. INTRODUCTION

In the last decade, there has been much research at the interface of computer science and game theory (see e.g. [27, 31]). One fundamental class of computational problems in game theory is the computation of *solution concepts* of a finite game. Much recent effort in the literature has concerned the complexity of computing mixed-strategy Nash [7, 9–11] and correlated equilibria [22, 28].

In this paper we focus on the problem of computing pure strategy Nash equilibria (PSNE). Unlike mixed-strategy Nash equilibria, which are guaranteed to exist for finite games [26], in general PSNE are not guaranteed to exist. Nevertheless, in many ways PSNE is a more attractive solution concept than mixed-strategy Nash equilibrium. First, PSNE can be easier to justify because it does not require the players to randomize. Second, it can be easier to analyze because of its discrete nature (see, e.g., [6]). There are several versions of the problem of computing PSNE: deciding if a PSNE exists, finding one, counting the number of PSNEs, enumerating them, and finding the optimal equilibrium according to some objective (e.g., social welfare). The latter problems are game-theoretically more useful, but often computationally harder.

The complexity of each of these problems very much depends on the *representation* used. *Normal form* is the traditional choice. In this representation, each player’s utilities are specified explicitly for each pure strategy profile. Questions about PSNE can be answered in polynomial time in the input size, by checking every pure strategy profile. However, the size of the normal form representation grows exponentially in the number of players. This is problematic in practice, especially since many games of interest involve large numbers of players.

Fortunately, most large games of practical interest have highly-structured payoff functions, and thus it is possible to represent them compactly. A line of research thus exists looking for *compact game representations* that are able to succinctly describe structured games, and efficient algorithms for finding equilibria that run in time polynomial in the size of the representation. The problem is hard in the most general case, when utility functions are arbitrary efficiently computable functions represented as circuits [30] or Turing Machines [1]. Researchers have also studied compact game representations that exploit various types of structure in utility functions. These include graphical games [23], congestion games [29] and action-graph games [5]. Computing PSNE for these representations is hard in general, but polynomial time for certain subclasses of games [12, 14, 19–21].

One important type of structure is symmetry. A game is *symmetric* when all players are identical and interchangeable. Symmetric games have been studied since the beginning of noncooperative game theory. For example, Nash proved that symmetric games always have symmetric mixed Nash equilibria [26]. In a symmetric game, a player’s utility depends only on the player’s chosen action and the *configuration*, which is the vector of integers specifying the numbers of players choosing each of the actions. As a result, symmetric games can be represented more compactly than games in normal form: we only need to specify a utility value for each action and each configuration. For a symmetric game with  $n$  players and  $m$  actions per player, the number of configurations is  $\binom{n+m-1}{m-1}$ . For fixed  $m$ , this grows like  $n^{m-1}$ , in which case  $\Theta(n^m)$  numbers are required to specify the game. Questions about PSNE can be computed straightforwardly by checking all configurations, which requires polynomial time in the size of the representation, and polynomial time in  $n$  when the number of actions is fixed. Indeed, [6] proved that the existence problem for PSNE of symmetric games with constant number of actions is in  $AC^0$ . There has also been research on a generalization of symmetric games called *anonymous games*, in which a given player’s utility depends on his identity as well as the action chosen and the configuration [6, 13].

Existing work on symmetry in games focuses on utility functions that explicitly enumerate utility values for each configuration. However, more concise representations are possible when the utility functions have additional structure. In symmetric games, the set of players can be specified implicitly by the number  $n$ , requiring only  $\log n$  bits to represent. If the utility functions can be represented in size polynomial in the number of bits needed to represent the configuration vector, the game can be represented in size polynomial in  $\log n$ . Thus, such a “compact” representation is able to specify games with a number of players exponential in the input size.

In this paper, we consider the complexity of computing PSNE for symmetric games with compactly-represented utility functions. We first look at the most general setting, where the utility functions are represented as circuits whose inputs are binary representations of the configuration vector. We show that even with a fixed number of actions, the problem of deciding the existence of PSNE is NP complete. The only exception is the case of two actions, for which we show that there always exists a PSNE and present an algorithm that identifies such an equilibrium in polynomial time.

Our main positive result is the identification of a compact representation of utility with nice computational properties—piecewise linear functions of the configuration vector. Piecewise linear functions are a natural and convenient way of representing utilities. For this setting, we present novel algorithms for finding a sample PSNE and for counting the number of PSNEs. When the number of actions is fixed, these algorithms run in polynomial time. In particular, if the total number of pieces is bounded by a polynomial of  $\log n$  then we achieve an exponential improvement over the algorithm of Brandt et. al. [6], which scales polynomially with  $n$ . Our techniques also yield a *polynomial-space polynomial-delay* output-sensitive algorithm for enumerating the set of PSNE.

The main challenge in constructing such polynomial-time algorithms is that the set of configurations and the set of PSNE configurations can be exponential in the input size. Thus, approaches based on enumerating all configurations require exponential time. Instead, our approach encodes the set of PSNE in a compact representation that has appealing computational properties. Specifically, we make use of the *rational generating function method* due to Barvinok and Woods [3]. (We give a brief overview of the rational generating function techniques that we use in Section 4.) We formulate the set of equilibrium configurations via operations on lattice points in polyhedra; the resulting set of points can be encoded as a rational generating function of polynomial size.

The current paper relates to some recent work by one of the authors [24]. This work introduced rational generating function methods to the algorithmic study of games, showing that they can be used to compute pure-strategy Nash equilibria of games in which the action sets are represented by fixed-dimensional polyhedra and the utilities are given by piecewise linear functions. These results assumed a fixed number of players and made restrictions on the piecewise linear functions.

By contrast, the current paper allows for a non-fixed number of players and instead restricts the number of actions; it also allows a much more general family of piecewise linear functions.

## 2. SYMMETRIC GAMES AND CONFIGURATIONS

Symmetric games are a class of strategic games in which each player has an identical set of actions  $A$  and utility function  $u$ . We consider  $n$ -player symmetric games in which the number of actions  $m$  is a fixed constant.

The outcomes of the game are sufficiently described by *configurations* of players; that is, a record of how many players play each action. A configuration is an  $m$ -dimensional vector  $\mathbf{x} = (x_a : a \in A)$ , where  $x_a$  is the number of players playing action  $a$ . Let  $D$  denote the set of configurations:

$$D = \left\{ \mathbf{x} \in \mathbb{Z}^m : \sum_{a \in A} x_a = n, x_a \geq 0 \text{ for all } a \in A \right\}. \quad (2.1)$$

Since each player has the same utility function, the utility of a given player depends only on the action played and the overall configuration. For each action  $a \in A$ , we have a utility function defined on all the configurations where at least one player plays actions  $a$ . In particular,  $u_a(\mathbf{x})$  is the utility of playing action  $a$  in configuration  $\mathbf{x}$  provided  $x_a \geq 1$ .

A configuration  $\mathbf{x} \in D$  is a *pure strategy Nash equilibrium configuration* (or simply a PSNE) if for all actions  $a$  and  $a'$  either no player plays action  $a$  or the utility of a player playing action  $a$  exceeds the utility he would receive from unilaterally deviating to action  $a'$ . Symbolically, let  $N$  denote the set of PSNE in a symmetric game. Then

$$\mathbf{x} \in N \iff (\forall a \in A : x_a = 0) \text{ OR } (\forall a' \in A, u_a(\mathbf{x}) \geq u_{a'}(\mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a)), \quad (2.2)$$

where  $\mathbf{e}_a$  is the  $a$ th unit vector with components  $e_{aa} = 1$  and  $e_{aa'} = 0$  for  $a' \neq a$ . Note that  $\mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a$  is the same configuration as  $\mathbf{x}$  except that one player has deviated from playing action  $a$  to action  $a'$ .

## 3. SYMMETRIC GAMES WITH UTILITIES AS CIRCUITS

In this section, we consider *circuit symmetric games*, a representation in which each utility function  $u_a$  is represented as a circuit whose input is a binary representation of the configuration vector  $\mathbf{x}$ . The representation size can thus be as small as  $O(\log n)$ .

We first consider the case with two actions,  $A = \{1, 2\}$ . Cheng et. al. [8] proved that a symmetric game with two actions always has at least one PSNE. This also follows from the fact that such a game can be formulated as a congestion game,<sup>1</sup> which implies the existence of a PSNE [29]. However, even when a PSNE provably exists (or when a game is a congestion game), PSNEs can still be difficult to find. We give an alternative proof of the existence of PSNE that illustrates the structure of the strategy space for these games, and then show how this structure can be exploited for efficient computation.

**Lemma 3.1.** *Any symmetric game with two actions has a PSNE.*

*Proof.* Given such a symmetric game, we construct the *deviation graph*, whose vertices are the configurations  $\mathbf{x} \in D$ . There is an directed edge from  $\mathbf{x}$  to  $\mathbf{x}'$  if and only if a deviation by a single player from  $\mathbf{x}$  results in  $\mathbf{x}'$ . Since each  $\mathbf{x} = (x_1, n - x_1)$ , where  $x_1$  is the number of agents playing action 1, we can identify each configuration by its first component. Under this mapping, the set of configurations corresponds to the set of integers  $\{0, \dots, n\}$ . It is straightforward to see that the only edges in the deviation graph are between adjacent integers:  $i, j \in \{0, \dots, n\}$  such that  $|i - j| = 1$ .

<sup>1</sup>To see this, observe that  $u_1(\mathbf{x}) = u_1(x_1, n - x_1)$  is a function of only  $x_1$ ; similarly  $u_2(\mathbf{x}) = u_2(n - x_2, x_2)$  is a function of only  $x_2$ .

We then consider the *profitable deviation graph* (PDG), whose vertices are the same configurations and directed edges represent strictly profitable deviations. For example, if a deviation by one player in configuration  $\mathbf{x}$  from action  $a$  to action  $3 - a$  results in configuration  $\mathbf{x}'$ , and furthermore if  $u_{3-a}(\mathbf{x}') > u_a(\mathbf{x})$ , then the PDG has an edge from  $\mathbf{x}$  to  $\mathbf{x}'$ . Observe that the PDG is a subgraph of the deviation graph, and that if there is an edge from  $\mathbf{x}$  to  $\mathbf{x}'$  in the PDG, then there cannot be an edge from  $\mathbf{x}'$  to  $\mathbf{x}$ .

A sink of the PDG has no profitable deviations, which means that it is a PSNE. We claim that the PDG must have a sink. To see this, we can start at vertex 0 and follow the directed edges. Because the PDG is a subgraph of the deviation graph, each edge on this path must increase the vertex's index (in fact, by exactly one). Thus, the path must eventually stop at a sink.  $\square$

The above proof suggests a straightforward algorithm for finding a PSNE: start at configuration 0 and follow the edges in the PDG. In fact by a similar argument any starting configuration would lead to a sink. Unfortunately this approach can take  $\Omega(n)$  steps before reaching a sink, which can be exponential in the representation size. Instead, we present a divide-and-conquer approach that exploits the structure of the PDG.

**Theorem 3.2.** *For circuit symmetric games with two actions, a PSNE can be found in polynomial time.*

*Proof.* Given such a game with  $n$  players, consider the configurations  $\lfloor \frac{n}{2} \rfloor$  and  $\lfloor \frac{n}{2} \rfloor + 1$ . There are three cases:

- (1) If there is an edge from  $\lfloor \frac{n}{2} \rfloor$  to  $\lfloor \frac{n}{2} \rfloor + 1$  in the PDG, then there must exist a PSNE in the subset  $\{\lfloor \frac{n}{2} \rfloor + 1, \dots, n\}$ . This is because a path from  $\lfloor \frac{n}{2} \rfloor + 1$  must be increasing and eventually stop at a sink.
- (2) Likewise, if there is an edge from  $\lfloor \frac{n}{2} \rfloor + 1$  to  $\lfloor \frac{n}{2} \rfloor$ , there must exist a PSNE in the subset  $\{0, \dots, \lfloor \frac{n}{2} \rfloor\}$ , since a path from  $\lfloor \frac{n}{2} \rfloor$  must be decreasing and stop at a sink.
- (3) If there is no edge between the two configurations, then there must exist a PSNE in each of the subsets  $\{0, \dots, \lfloor \frac{n}{2} \rfloor\}$  and  $\{\lfloor \frac{n}{2} \rfloor + 1, \dots, n\}$ .

Our algorithm picks a subset that contains a PSNE, and then recursively bisects that subset. This process terminates at a PSNE after  $O(\log n)$  iterations. For each iteration, checking the existence of edges between two configurations requires evaluation of utility at the two configurations, which can be done in linear time for utility functions represented as circuits. Therefore the running time of this algorithm is  $O(|\Gamma| \log n)$ , where  $|\Gamma|$  is the size of the circuits.  $\square$

Our next result shows that the problem of finding a PSNE in a circuit symmetric game becomes intractable once we go beyond two actions.

**Theorem 3.3.** *For circuit symmetric games in which the number of actions is a fixed constant of at least three, the problem of determining the existence of PSNE is NP complete.*

*Proof.* The problem is in NP because to determine whether a configuration  $\mathbf{x}$  is a PSNE, there are only  $O(m^2)$  possible deviations to check.

We show NP-hardness by reduction from CIRCUITSAT. Given a CIRCUITSAT problem instance  $C(y_1, \dots, y_m)$ , we construct a circuit symmetric game with  $n = 2^m - 1$  players and 3 actions  $\{1, 2, 3\}$  such that the game has a PSNE if and only if there exists a satisfying assignment of  $y_1, \dots, y_m$ .

Given a configuration  $\mathbf{x} = (x_1, x_2, x_3)$ , the utility functions  $u_1(\mathbf{x}), u_2(\mathbf{x})$  and  $u_3(\mathbf{x})$  are defined as follows:

- (1) If the binary representation of  $x_1$  correspond to a satisfying assignment for  $C$ , i.e.  $C(x_1^0, \dots, x_1^m) = 1$  where  $x_1^i$  is the  $i$ th bit of  $x_1$ , then  $u_1(\mathbf{x}) = u_2(\mathbf{x}) = u_3(\mathbf{x}) = 2$ .
- (2) Otherwise:
  - (a) if  $x_1 > 0, x_2 > 0, x_3 > 0$ , then  $u_1(\mathbf{x}) = u_2(\mathbf{x}) = 1, u_3(\mathbf{x}) = -2$ ;
  - (b) if  $x_1 > 0, x_2 > 0, x_3 = 0$ , then  $u_1(\mathbf{x}) = -1, u_2(\mathbf{x}) = 1$ ;

- (c) if  $x_1 = 0, x_2 > 0, x_3 > 0$ , then  $u_2(\mathbf{x}) = -1, u_3(\mathbf{x}) = 1$ ;
- (d) if  $x_1 > 0, x_2 = 0, x_3 > 0$ , then  $u_1(\mathbf{x}) = 1, u_3(\mathbf{x}) = -1$ ;
- (e) if  $x_a = n$  for some action  $a$ , i.e. all players are playing  $a$ , then  $u_a(\mathbf{x}) = 0$ .

If there exists a satisfying assignment for  $C$ , then any configuration with the corresponding  $x_1$  is a PSNE because each player receives the maximum utility of the game. If there does not exist a satisfying assignment, then the game's utilities are defined by condition 2. We claim that this subgame under condition 2 does not have a PSNE. Intuitively, the game can be thought of as a generalization of the 2-player Rock-Paper-Scissors game. Formally, given a configuration of case 2a, a deviation from action 3 (with utility -2) to 1 or 2 is profitable. Given a configuration of case 2b, a profitable deviation is from action 1 (utility -1) to 2 (utility 1 if the resulting configuration is of case 2b, utility 0 if the resulting configuration is of case 2e). Similarly, given a configuration of case 2c, a profitable deviation is from action 2 to 3; and given a configuration of case 2d, a profitable deviation is from action 3 to 1. Given a configuration of case 2e with e.g.  $x_1 = n$ , a profitable deviation is to action 2, resulting in a configuration of case 2b. Therefore all configurations have profitable deviations, thus the subgame does not have a PSNE.

Finally, we observe that the utility functions described above can be formulated as circuits of the binary representation of  $\mathbf{x}$ . The size of the circuit symmetric game is linear in the size of the given CIRCUITSAT problem instance, and these utility functions can be constructed in polynomial time. This concludes the reduction proof.  $\square$

#### 4. RATIONAL GENERATING FUNCTIONS

In Section 5 we will describe a class of utility functions that yield efficient computation of PSNE. This result relies heavily on results from the literature on rational generating functions and in particular on the method of Barvinok and Woods [3]. Generating functions have been applied in an analogous fashion in a variety of other contexts, including discrete optimization [17], combinatorics [15], social choice theory [25] and compiler optimization [32]. We here provide a brief and selective overview of this theory for two reasons: to introduce some machinery that is used in proving our main theorem, and to invite other researchers to use these methods in future work. Readers familiar with these methods can skip ahead to Section 5.

The main impetus for considering generating functions comes from our desire to compactly represent exponential-cardinality sets of integer points, and to efficiently support the computational operations of counting and enumerating points in the set. We demonstrate the essence of the approach in the following simple example. Consider the set of integers on the line between 0 and  $n$ . We can represent these points as follows: with every integer  $x \in [0, n]$ , we associate an exponent of real variable  $\xi$ . Using this encoding we can represent the integers in the interval  $[0, n]$  as the exponents in the polynomial expression

$$\sum_{x=0}^n \xi^x, \tag{4.1}$$

called a *generating function representation*. So far we have not gained much: there are exponentially many terms in (4.1) (in terms of the binary encoding size of  $n$ ), just like an explicit listing of the numbers  $0, 1, \dots, n$ . However, we can also write the expression as

$$\sum_{x=0}^n \xi^x \frac{1 - \xi^{n+1}}{1 - \xi} = \frac{1}{1 - \xi} - \frac{\xi^{n+1}}{1 - \xi}. \tag{4.2}$$

Thus, we have written the *long* sum (4.1) as a *short* sum of two rational functions, called a *rational generating function representation*. The encoding length of this new representation is now *polynomial* in the encoding length of  $n$ . Not only does this representation appeal because of its compact size, but also when we input different values for  $\xi$  in the exponential-sized sum (4.1), we need only do a small number of calculations in its rational representation in order to evaluate the

sum. For instance, if we substitute  $\xi = 1$  in (4.1) we compute the cardinality of  $[0, n] \cap \mathbb{Z}$ , albeit with an exponential number of arithmetic operations; however, the same substitution (letting  $\xi \rightarrow 1$  and using L'Hôpital's rule) can be done in (4.2) with exponentially fewer arithmetic operations.

We are often interested in sets arising from unions, intersections and differences of simpler sets of integers. A basic illustration is as follows: suppose we have the set  $\{0, \dots, n\}$  represented by the rational generating function  $\frac{1-\xi^{n+1}}{1-\xi}$ , as well as the set  $\{n+1, \dots, 2n\}$  represented by the rational generating function  $\frac{\xi^{n+1}-\xi^{2n+1}}{1-\xi}$ . A rational generating function representing the *union* of these two *disjoint* sets can be found by *summing* the representations of  $\{0, \dots, n\}$  and  $\{n+1, \dots, 2n\}$ :

$$\frac{1-\xi^{n+1}}{1-\xi} + \frac{\xi^{n+1}-\xi^{2n+1}}{1-\xi} = \frac{1-\xi^{2n+1}}{1-\xi}.$$

It is straightforward to verify that  $\frac{1-\xi^{2n+1}}{1-\xi}$  is a rational generating function encoding of the union  $\{0, \dots, 2n\}$ . A more general result obtains when combining sets that are not disjoint (Theorem 4.2).

Operating on rational generating functions may seem unnecessary in the simple setting we have discussed so far, but it becomes useful when extended to deal with more general sets of integer points in higher dimensions. We do not detail the full progression of this theory but instead refer the reader to the excellent textbook by Beck and Robbins [4].

Our starting point is the work of Barvinok [2], who introduced a polynomial-time algorithm to represent as a rational generating function the integer points inside of a *rational polytope*  $P \subseteq \mathbb{R}^m$  given by an inequality system  $\{\mathbf{x} \in \mathbb{R}^m : M\mathbf{x} \leq \mathbf{b}\}$ , provided the dimension  $m$  is fixed. Note that representing the integer set  $P \cap \mathbb{Z}^m$  by the inequality description of  $P$  gives little hint as to its exact cardinality. We shall see below that a generating function representation allows us to *count* the number of integer points in  $P$  exactly in polynomial time (Theorem 4.4).

Now we briefly describe Barvinok's algorithm and its useful extensions and applications. Consider the *generating function* of the lattice point set  $P \cap \mathbb{Z}^m$ , which is defined as

$$g(P \cap \mathbb{Z}^m; \boldsymbol{\xi}) = \sum_{\mathbf{x} \in P \cap \mathbb{Z}^m} \boldsymbol{\xi}^{\mathbf{x}} = \sum_{\mathbf{x} \in P \cap \mathbb{Z}^m} \xi_1^{x_1} \dots \xi_m^{x_m} \in \mathbb{Z}[\xi_1^{\pm 1}, \dots, \xi_m^{\pm 1}]. \quad (4.3)$$

Note that each lattice point  $\mathbf{x}$  in  $P$  is mapped to the exponent of a monomial  $\boldsymbol{\xi}^{\mathbf{x}}$  in  $g(P \cap \mathbb{Z}^m; \boldsymbol{\xi})$ .

**Theorem 4.1 (Barvinok's Theorem [2]).** *Let  $P$  be a polytope in  $\mathbb{R}^m$  with generating function  $g(P \cap \mathbb{Z}^m, \boldsymbol{\xi})$  given by (4.3) which encodes the lattice points inside  $P$ . Then, there exists an algorithm which computes an equivalent representation of the form:*

$$g(P \cap \mathbb{Z}^m; \boldsymbol{\xi}) = \sum_{i \in I} \gamma_i \frac{\boldsymbol{\xi}^{\mathbf{c}_i}}{(1 - \boldsymbol{\xi}^{\mathbf{d}_{i1}})(1 - \boldsymbol{\xi}^{\mathbf{d}_{i2}}) \dots (1 - \boldsymbol{\xi}^{\mathbf{d}_{im}})}, \quad (4.4)$$

where  $I$  is a polynomial-size index set and all data are integer. A formula of the type (4.4) is called a short rational generating function. The algorithm runs in polynomial time when the dimension  $m$  is fixed.

Note that the number of binomial terms in the denominator of each rational term in (4.4) is  $m$  and thus fixed when the dimension is fixed. When a lattice point set  $S$  is expressed in the form (4.4) we refer to  $g(S; \boldsymbol{\xi})$  as its *Barvinok encoding*. In the algorithms that follow, when a set  $S$  is given as input or output by its Barvinok encoding  $g(S, \boldsymbol{\xi})$ , the encoding size is the binary encoding of the integer vectors  $\gamma, \mathbf{c}_i, \mathbf{d}_{i1}, \dots, \mathbf{d}_{im}$  for  $i \in I$ .

It is important to note that Theorem 4.1 only encodes sets of integer points inside of polytopes. The key result that makes this theory useful in our setting is that some more general lattice point sets arising from simple operations on polytopal lattice point sets admit short rational generating function encodings. Barvinok and Woods [3] developed powerful algorithms that apply to these more general settings. For our purposes the most important algorithm concerns constant-length

Boolean combinations of polyhedra. A *Boolean combination* of the sets  $S_1, \dots, S_k$  is any combination of unions, intersections and set differences of those sets. For instance,  $(S_1 \cap S_2) \setminus S_3$  is a Boolean combination of the sets  $S_1, S_2$  and  $S_3$ .

**Theorem 4.2 (Boolean Operations Theorem).** (Corollary 3.7 in [3]) *Given fixed integers  $k$  and  $\ell$  there exists a constant  $s$  and a polynomial-time algorithm for the following problem. Given as input, in binary encoding,*

- (I<sub>1</sub>) *the dimension  $m$  and*
- (I<sub>2</sub>) *Barvinok encodings of  $k$  finite sets  $S_p \subseteq \mathbb{Z}^m$ ,  $g(S_p; \xi)$  such that for each rational term the number of binomials in the denominator is at most  $\ell$ ,*

*output, in binary encoding,*

- (O<sub>1</sub>) *rational numbers  $\gamma_i$ , integer vectors  $\mathbf{c}_i, \mathbf{d}_{ij}$  for  $i \in I, j = 1, \dots, s_i$ , where  $s_i \leq s$ , such that*

$$g(S; \xi) = \sum_{i \in I} \gamma_i \frac{\xi^{\mathbf{c}_i}}{(1 - \xi^{\mathbf{d}_{i1}}) \dots (1 - \xi^{\mathbf{d}_{is_i}})}$$

*is a rational generating function of the finite set  $S$  that is the Boolean combination of the sets  $S_1, \dots, S_k$ , and where each rational term in the expression has at most  $s$  terms in its denominator and where  $I$  is a polynomial-sized index set.*

We remark that if  $k$  were allowed to vary, the number of binomials in the denominators would become exponential (essentially doubling with each Boolean operation); this explains the requirement that  $k$  be fixed in order to achieve a polynomial run time. (For a more precise statement see the statement of Lemma 3.4 and the proof of Corollary 3.7 in [3].) We also note that if the input sets  $S_p \subseteq \mathbb{Z}^m$  are integer points inside of polyhedra whose Barvinok encodings  $g(S; \xi)$  arise from applying Barvinok's Theorem (Theorem 4.1) then the condition that the number of binomials  $\ell$  in the denominators are fixed follows under the assumption that the dimension  $m$  is fixed.

Disjoint unions are a special case of combining sets.

**Lemma 4.3 (Disjoint Unions).** *If two lattice point sets  $S$  and  $T$  are disjoint then the generating function for  $S \cup T$  is the sum of generating functions for  $S$  and  $T$ . More generally, for disjoint lattice point sets  $S_1, \dots, S_k$ :*

$$g\left(\biguplus_{i=1}^k S_i, \xi\right) = \sum_{i=1}^k g(S_i, \xi),$$

where  $\biguplus$  denotes disjoint union.

Note that to compute disjoint unions of sets we do not appeal to the Boolean Operations Theorem, and thus the number of sets  $k$  in the union may be polynomial in the input size instead of a fixed number.

Once a rational generating function of a set  $S$  has been computed, various pieces of information can be extracted from it. As in our example, a useful operation is to *compute the cardinality* of  $S$ :

**Theorem 4.4 (Counting Theorem).** [2] *Let the dimension  $m$  be a fixed constant. Given a lattice point set  $S \in \mathbb{Z}^m$  input as its Barvinok encoding  $g(S, \xi)$ , there exists a polynomial time algorithm for computing  $|S|$ .*

The idea behind the proof of this theorem is analogous to the basic example at the beginning of this section. Given a Barvinok encoding of a lattice point set  $S$  as in (4.4), each of the basic rational functions has poles (the point  $\xi = \mathbf{1}$  in particular is a pole of all the basic rational functions), but after summing up only removable singularities remain. Obtaining the exact number of lattice points of lattice point set  $S$  is easy in (4.3), since clearly  $|S| = g(S; \mathbf{1})$ . Since (4.4) is a formula for the same function (except for removable singularities), we also have  $|S| = \lim_{\xi \rightarrow \mathbf{1}} g(S; \xi)$ , which can be

evaluated in polynomial time by performing a residue calculation with each basic rational function in the sum (4.4).

Finally, we can *explicitly enumerate* all elements of  $S$ . We note that the cardinality of  $S$  can be exponential in the encoding length. Nevertheless there exists a *polynomial-space polynomial-delay enumeration algorithm*. The following result is a version of Theorem 7 of [16].

**Theorem 4.5 (Enumeration Theorem).** *Let the dimension  $m$  and the maximum number  $\ell$  of binomials in the denominators be fixed. Then there exists a polynomial-space polynomial-delay enumeration algorithm for the following enumeration problem. Given as input, in binary encoding, a bound  $M$  and the Barvinok encoding  $g(S, \xi)$  of a lattice point set  $S \in [-M, M]^m \cap \mathbb{Z}^n$ , output, in binary encoding, all points in  $S$  in lexicographic order.*

## 5. SYMMETRIC GAMES WITH PIECEWISE LINEAR UTILITIES

Now we present our key positive result. Specifically, we show that when each utility function is a piecewise linear function of the configuration, the PSNE of compact, symmetric games with fixed numbers of actions can be computed in polynomial time. Piecewise-linear functions have been widely used as an approximation of arbitrary continuous functions. A recent example of their use is [18], which considered piecewise-linear utilities in the computation of market equilibria in the Arrow-Debreu model, and presented a polynomial-time algorithm when utilities are piecewise-linear concave functions and the number of goods is constant.

Piecewise linear functions are normally defined over a continuous domain, in which case it is sufficient to specify a polytopal subdivision of the domain and an affine function for each cell. The domain for our utility functions is the set of configurations  $D$ , a discrete set. Nevertheless we observe that  $D$  is the set of integer points in a rational polytope. Thus the definition of piecewise linear functions can be naturally extended to this setting. In particular, we specify a set of rational polytopes that induces a partition of the integer points  $D$ , and an affine function for each cell.

Formally, for each action  $a \in A$ , the piecewise linear utility function  $u_a(\mathbf{x})$  is given as follows. There is a finite set of rational polytopes  $\{P_{aj}\}_{j \in J_a}$  where each  $P_{aj} = \{\mathbf{x} \in \mathbb{R}^m : M_{aj}\mathbf{x} \leq \mathbf{b}_{aj}\}$  is given by an integer matrix  $M_{aj}$  and integer right-hand side vector  $\mathbf{b}_{aj}$ , and whose integer points partition the set of configurations  $D$ ; that is,  $D = \bigsqcup_{j \in J_a} (P_{aj} \cap \mathbb{Z}^m)$ . Over each cell  $P_{aj} \cap \mathbb{Z}^m$  in the partition of  $D$  there is an affine function  $f_{aj}(\mathbf{x}) = \alpha_{aj} \cdot \mathbf{x} + \beta_{aj}$ , where  $\alpha_{aj} \in \mathbb{Z}^m$  and  $\beta_{aj} \in \mathbb{Z}$ , such that

$$u_a(\mathbf{x}) = f_{aj}(\mathbf{x}) \text{ for } \mathbf{x} \in P_{aj} \cap \mathbb{Z}^m. \quad (5.1)$$

The piecewise linear utility function  $u_a(\mathbf{x})$  is input as the binary encoding of  $M_{aj}$ ,  $\mathbf{b}_{aj}$ ,  $\alpha_{aj}$  and  $\beta_{aj}$  for each  $j \in J_a$ .

We observe that a piecewise linear utility function can be represented as a circuit.<sup>2</sup> Also, given an arbitrary utility function, it can be described exactly by a piecewise linear function, although the number of pieces required may be  $\Theta(n^m)$  in general, in which case it is no longer compact in the sense we defined at the beginning of the paper.

The main positive result of the paper follows.

**Theorem 5.1.** *Consider a symmetric game with piecewise linear utilities given by the following input:*

- (I<sub>1</sub>) *the binary encoding of the number  $n$  of players;*
- (I<sub>2</sub>) *for each  $a \in A$ , the utility function  $u_a(\mathbf{x})$  represented as the binary encoding of  $M_{aj}$ ,  $\mathbf{b}_{aj}$ ,  $\alpha_{aj}$  and  $\beta_{aj}$  for each  $j \in J_a$ .*

*Then, when the number of actions  $m$  is fixed, there exists*

<sup>2</sup>Additions and multiplications can be carried out by circuits. To determine which piece a configuration is in, we can go through each piece and test the inequalities. The size of the resulting circuit is polynomial in the number of bits needed to describe the piecewise linear function.

- (i) a polynomial time algorithm to compute the number of pure Nash equilibrium configurations;
- (ii) a polynomial time algorithm to find a sample pure strategy Nash equilibrium, when at least one exists; and
- (iii) a polynomial-space polynomial-delay enumeration algorithm to enumerate all the pure Nash equilibrium configurations of the game.

*Proof.* All three results follow from an argument that  $N$ , the set of PSNE, can be encoded as a short rational generating function. We defined  $N$  in (2.2), but for convenience repeat it here:

$$N = \{ \mathbf{x} \in D : \forall a \in A : (x_a = 0) \text{ OR } (x_a \geq 1, \forall a' \in A : u_a(\mathbf{x}) \geq u_{a'}(\mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a)) \}.$$

The major difficulty in applying generating functions to encoding  $N$  is the nonlinearity of the objectives  $u_a$ . However, since these objectives are piecewise linear we simply consider the partitions of  $D$  into regions in which the objectives *are* linear. We use these partitions of  $D$  (and hence of  $N$ ) to express  $N$  as a Boolean combination of polytopal lattice point sets, and thus will ultimately be able to apply Theorem 4.2. The overall idea is to define subsets of configurations that have strictly profitable deviations, then remove these subsets from  $D$ , leaving only the set of PSNE.

Define the *deviation set*  $Dev(a, a', j, j')$  as the set of configurations  $\mathbf{x}$  in which a player currently playing action  $a$  lying in region  $P_{aj}$  has a *strictly* profitable deviation to playing action  $a'$ , thereby yielding a new configuration  $\mathbf{x}' \in P_{a'j'}$ . Such a profitable deviation will exist whenever  $f_{aj}(\mathbf{x}) < f_{a'j'}(\mathbf{x}')$ . Since the affine functions have integer coefficients, we can rewrite this condition as  $f_{aj}(\mathbf{x}) \leq f_{a'j'}(\mathbf{x}') - 1$  and thereby avoid strict inequalities. Putting this all together, we define the deviation set as

$$Dev(a, a', j, j') = \left\{ \mathbf{x} \in D : x_a \geq 1, \mathbf{x} \in P_{aj}, \begin{array}{l} \mathbf{x}' = \mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a \in P_{a'j'} \\ f_{aj}(\mathbf{x}) \leq f_{a'j'}(\mathbf{x}') - 1 \end{array} \right\}. \quad (5.2)$$

Now we can use (5.2) to rewrite (2.2). We obtain

$$N = D \setminus \bigcup_{a, a'} \bigcup_j \bigcup_{j'} Dev(a, a', j, j'), \quad (5.3)$$

where the first union is over all  $a, a' \in A$ , the second union is over  $j \in J_a$ , and the third union is over  $j' \in J_{a'}$ . This identity (ignoring for now why the second two unions are disjoint) can be verified as follows. Suppose configuration  $\mathbf{x} \in D$  *does not* lie in the right-hand side of (5.3). This implies that  $\mathbf{x}$  lies in some deviation set  $Dev(a, a', j, j')$  for some  $a, a' \in A$  and  $(j, j') \in J_a \times J_{a'}$  and hence there is a profitable unilateral deviation away from  $\mathbf{x}$  implying  $\mathbf{x}$  is not in  $N$ . Conversely, suppose  $\mathbf{x} \in D$  is not in  $N$ . This implies that there exists a profitable unilateral deviation, say from playing action  $a$  to  $a'$ . This implies  $u_a(\mathbf{x}) < u_{a'}(\mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a)$ . Now,  $\mathbf{x}$  and  $\mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a$  lie in cells  $P_{aj}$  for some  $j \in J_a$  and  $P_{a'j'}$  for some  $j' \in J_{a'}$  respectively. The condition on the utilities then implies that  $f_{aj}(\mathbf{x}) \leq f_{a'j'}(\mathbf{x}') - 1$ . It follows that  $\mathbf{x}$  is in the deviation set  $Dev(a, a', j, j')$  and thus not contained in the righthand side of (5.3).

The union indexed by  $j$  is disjoint because the sets  $\{P_{aj}\}_{j \in J_a}$  form a partition of  $D$ , and  $Dev(a, a', j, j') \subseteq P_{aj}$ . To show that the union indexed by  $j'$  is disjoint, we consider an arbitrary element  $\mathbf{x}$  of  $Dev(a, a', j, j')$ . This implies that  $\mathbf{x}' = \mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a \in P_{a'j'}$ . Because  $\{P_{a'j'}\}_{j' \in J_{a'}}$  are disjoint sets, for all  $j'' \in J_{a'} \setminus \{j'\}$  we have  $\mathbf{x}' \notin P_{a'j''}$  and thus  $\mathbf{x} \notin Dev(a, a', j, j'')$ . Therefore  $Dev(a, a', j, j')$  is disjoint from  $Dev(a, a', j, j'')$  for any  $j', j'' \in J_{a'}, j' \neq j''$ .

We took particular care in describing which unions in our expression for  $N$  are disjoint and which are not. This is because the second and third unions are not of fixed length as would be required for the application of Theorem 4.2. However, since the unions are disjoint we can use simple addition of generating functions to handle this part of the overall expression of  $N$ . To make this precise, note that each of the  $Dev(a, a', j, j')$  terms are polytopal lattice point sets and thus admit Barvinok encodings  $g(Dev(a, a', j, j'), \boldsymbol{\xi})$  that can be computed in polynomial time by Theorem 4.1.

For each  $a, a' \in A$  define

$$Dev(a, a') = \bigsqcup_j \bigsqcup_{j'} Dev(a, a', j, j'). \quad (5.4)$$

$Dev(a, a')$  is a disjoint union, and so by Lemma 4.3 also admits a Barvinok encoding

$$g(Dev(a, a'), \xi) = \sum_j \sum_{j'} g(Dev(a, a', j, j'), \xi).$$

We can use 5.4 to rewrite 5.3 as  $N = D \setminus \bigcup_{a, a' \in A} Dev(a, a')$ . Since  $D$  and each of the sets  $Dev(a, a')$  have Barvinok encodings, Theorem 4.2 tells us that we can also derive such an encoding for  $N$ . This Boolean combination of sets describing  $N$  is of constant length since  $m$  is fixed.

Now that we have shown that  $N$  can be encoded as a short rational generating function, our PSNE computation results follow by applying Theorems 4.4 and 4.5. Given that we can compute  $g(N, \xi)$  in polynomial time, we can compute its cardinality in polynomial time by applying Theorem 4.4. This establishes (i). Applying Theorem 4.5 (noting the bound  $N \subseteq [0, n]^m$ ) we need only wait polynomial time to output the first element of  $N$ , establishing (ii). The enumeration scheme (iii) derives from Theorem 4.5 directly.  $\square$

## 6. CONCLUSIONS

In this paper we explored the computational complexity of computing PSNE of symmetric games with succinctly represented utilities and a fixed number of actions, detailing the problem's hardness in general and presenting a promising form of compactly represented utilities as piecewise linear functions. The methods explored here allow us to compute PSNE of such games with algorithms that scale polynomially with the representation size of the number of players  $\log n$  and the binary encoding size of the linear pieces which define the utilities. This result can also be understood as a statement about the relation between structure in a game's utility functions and structure in the set of PSNE: if the utility functions have a certain type of structure (can be represented as piecewise linear functions with a polynomial number of pieces), then the set of PSNE of the game has a corresponding type of structure (can be encoded as a rational generating function of polynomial size).

In the full version of this paper,<sup>3</sup> we consider two main extensions of our approach. First, the power of the rational generating function representation of PSNE allows us to go beyond basic questions such as existence and enumeration, and provide a more informative picture of the set of PSNE. One quantity of interest is social welfare. The social welfare is not a piecewise-linear function of the configuration; instead it can be formulated as a piecewise-polynomial function. We give a FPTAS for finding a PSNE with the best (or worst) social welfare. We also give a polynomial-time algorithm for computing the mean and variance of the social welfare over all PSNE of the game. Second, we are able to consider a family of games parameterized by a fixed number of parameters, and answer interesting questions about the family of games without having to solve each individual game in the family. This has interesting applications, such as estimating parameters so that PSNE match observed player behavior, or setting the parameters so that the resulting set of PSNE satisfies some objective. Our overall approach is to augment the configuration vector with a fixed number of integer parameters. We then construct a generating function for a set of points in this configuration space, with the property that if we fix the parameters, it gives the set of PSNE for the game with that particular instantiation of parameters. In other words, it is the graph of the parameters-equilibria correspondence.

---

<sup>3</sup>A draft of the full paper is available at <http://www.cs.ubc.ca/~jiang/papers/symmetric.pdf>.

## REFERENCES

- [1] C. Alvarez, J. Gabarro, and M. Serna. Pure Nash equilibria in a game with large number of actions. In *Mathematical Foundations of Computer Science*. 2005.
- [2] A. Barvinok. A polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed. *Mathematics of Operations Research*, 19(4):769–779, 1994.
- [3] A. Barvinok and K. M. Woods. Short rational generating functions for lattice point problems. *Journal of the American Mathematical Society*, 16(957-979), 2003.
- [4] M. Beck and S. Robbins. *Computing the Continuous Discretely: Integer-point Enumeration in Polyhedra*. Springer, 2007.
- [5] N. A. R. Bhat and K. Leyton-Brown. Computing Nash equilibria of action-graph games. In *UAI: Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 35–42, 2004.
- [6] F. Brandt, F. Fischer, and M. Holzer. Symmetries and the complexity of pure Nash equilibrium. *Journal of Computer and System Sciences*, 75(3):163–177, 2009.
- [7] X. Chen and X. Deng. Settling the complexity of two-player Nash equilibrium. In *FOCS: Proceedings of the Annual IEEE Symposium on Foundations of Computer Science*, pages 261–272, 2006.
- [8] S.F. Cheng, D.M. Reeves, Y. Vorobeychik, and M.P. Wellman. Notes on equilibria in symmetric games. In *AAMAS: Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 2004.
- [9] C. Daskalakis, A. Fabrikant, and C.H. Papadimitriou. The game world is flat: The complexity of Nash equilibria in succinct games. In *ICALP: Proceedings of the International Colloquium on Automata, Languages and Programming*, pages 513–524, 2006.
- [10] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou. The complexity of computing a Nash equilibrium. In *STOC: Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 61–70, 2006.
- [11] C. Daskalakis and C. H. Papadimitriou. Three-player games are hard. In *ECCC: Electronic Colloquium on Computational Complexity*, 2005.
- [12] C. Daskalakis and C.H. Papadimitriou. Computing pure Nash equilibria in graphical games via Markov random fields. In *ACM-EC: Proceedings of the ACM Conference on Electronic Commerce*, pages 91–99, 2006.
- [13] C. Daskalakis and C.H. Papadimitriou. Computing equilibria in anonymous games. In *FOCS: Proceedings of the Annual IEEE Symposium on Foundations of Computer Science*, pages 83–93, 2007.
- [14] C. Daskalakis, G. Schoenebeck, G. Valiant, and P. Valiant. On the complexity of Nash equilibria of Action-Graph Games. In *SODA: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 710–719, 2009.
- [15] J. A. De Loera. The many aspects of counting lattice points in polytopes. *Mathematische Semesterberichte*, 52(2):175–195, August 2005.
- [16] J. A. De Loera, R. Hemmecke, and M. Köppe. Pareto optima of multicriteria integer linear programs. *INFORMS Journal on Computing*, 21(1):39–48, 2009.
- [17] J. A. De Loera, R. Hemmecke, J. Tauzer, and R. Yoshida. Effective lattice point counting in rational convex polytopes. *Journal of Symbolic Computation*, 38(4):1273–1302, October 2004.
- [18] N. R. Devanur and R. Kannan. Market equilibria in polynomial time for fixed number of goods or agents. In *FOCS: Proceedings of the Annual IEEE Symposium on Foundations of Computer Science*, pages 45–53, 2008.
- [19] A. Fabrikant, C. Papadimitriou, and K. Talwar. The complexity of pure Nash equilibria. *STOC: Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 604–612, 2004.
- [20] G. Gottlob, G. Greco, and F. Scarcello. Pure Nash equilibria: Hard and easy games. *Journal of Artificial Intelligence Research*, 24:357–406, 2005.
- [21] A. Jiang and K. Leyton-Brown. Computing pure Nash equilibria in symmetric action graph games. In *AAAI: Proceedings of the AAAI Conference on Artificial Intelligence*, 2007.
- [22] S. Kakade, M. Kearns, J. Langford, and L. Ortiz. Correlated equilibria in graphical games. In *ACM-EC: Proceedings of the ACM Conference on Electronic Commerce*, pages 42–47, 2003.
- [23] M. Kearns, M. Littman, and S. Singh. Graphical models for game theory. In *UAI: Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2001.
- [24] M. Köppe, C. T. Ryan, and M. Queyranne. Rational generating functions and integer programming games (<http://arxiv.org/abs/0809.0689>). August 2008.
- [25] D. Lepelley, A. Louichi, and H. Smaoui. On Ehrhart polynomials and probability calculations in voting theory. *Social Choice and Welfare*, 30(3):363–383, 2008.
- [26] J. Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286–295, February 1951.
- [27] N. Nisan, T. Roughgarden, E. Tardos, and V.V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press New York, NY, USA, 2007.
- [28] C. H. Papadimitriou and T. Roughgarden. Computing correlated equilibria in multi-player games. *Journal of the ACM*, 55(3):1–29, 2008.

- [29] R.W. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory*, 2:65–67, 1973.
- [30] G. Schoenebeck and S. Vadhan. The computational complexity of Nash equilibria in concisely represented games. In *ACM-EC: Proceedings of the ACM Conference on Electronic Commerce*, pages 270–279, 2006.
- [31] Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, New York, 2009.
- [32] S. Verdoolaege, R. Seghir, K. Beyls, V. Loechner, and M. Bruynooghe. Analytical computation of Ehrhart polynomials: Enabling more compiler analyses and optimizations. In *Proceedings of the ACM International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 248–258, 2004.

A. X. JIANG: UNIVERSITY OF BRITISH COLUMBIA, DEPARTMENT OF COMPUTER SCIENCE, 201-2366 MAIN MALL, VANCOUVER, B.C., CANADA, V6T 1Z4  
*E-mail address:* `jiang@cs.ubc.ca`

C. T. RYAN: UNIVERSITY OF BRITISH COLUMBIA, SAUDER SCHOOL OF BUSINESS, 2053 MAIN MALL, VANCOUVER, BC, CANADA, V6T 1Z2  
*E-mail address:* `chris.ryan@sauder.ubc.ca`

K. LEYTON-BROWN: UNIVERSITY OF BRITISH COLUMBIA, DEPARTMENT OF COMPUTER SCIENCE, 201-2366 MAIN MALL, VANCOUVER, B.C., CANADA, V6T 1Z4  
*E-mail address:* `kevinlb@cs.ubc.ca`