SYMMETRIC GAMES WITH PIECEWISE LINEAR UTILITIES

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, where the utilities are compactly represented. Such a representation is able to describe symmetric games whose number of players is 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, there always exist a PSNE and we give a polynomial algorithm for finding one. We then focus on a natural representation of utility as piecewise-linear functions, and show that such a representation has nice computational properties. In particular, we give polynomial-time algorithms to count the number of PSNE (thus deciding if such an equilibrium exists) and to find a sample PSNE, when one exists. Our approach makes use of Barvinok and Wood's rational generating function method [4], 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. [30, 34]). 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, 31].

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 [29], 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 [33] or Turing Machines [1]. Researchers have also studied compact game representations that exploit various types of structure in utility functions. These include graphical

Date: April 2, 2009.

games [23], congestion games [32] 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 [29]. 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 [4]. (We give a brief overview of the rational generating function techniques that we use in Section 5.) We formulate the set of equilibrium configurations via operations on lattice points in polyehdra; 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 [26]. 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.

The paper is organized as follows. In Section 2 we define symmetric games and configurations. In Section 3 we discuss the complexity of computing PSNE of symmetric games when the utility functions are represented as circuits. We then turn our attention to piecewise linear utility functions. In Section 4 we discuss two representations of piecewise linear utility. In Section 5 we introduce rational generating functions. This section is for the most part self-contained and hopefully can be appreciated by those with little or no exposure to generating functions.

Section 6 contains the main result of the paper, that the set of pure equilibrium configurations in symmetric games can be encoded *short rational generating function*. An immediate corollary is that there are efficient algorithms to count and enumerate equilibria, when the number of actions k is fixed. We present two alternate methods to derive such a generating function encoding, both of which yield fresh insights into the general method. Section 7 highlights how generating function can be used to answer interesting questions about the structure of equilibria in our games. An approximation algorithm is developed for finding social welfare maximizing equilibria in the game, as well as exact algorithms to find the mean and variance of social welfare across all equilibria. Section 8 details a direct generalization of our methods to the case of m-symmetric games with a fixed number of player classes m. Finally, Section 9 explores the situation where the number of players and utilities of each player in a symmetric game are influenced by parametric choices of a leader or designer. We are able to answer questions regarding how the designer might choose these parameters to optimize some polynomial objective, or minimize error in having an equilibrium close to some target equilibrium, $\tilde{\mathbf{x}}$.

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:

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

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

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

where $\mathbf{e}_a$ is the ath 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,[1] which implies the existence of a PSNE [32]. However, even when a PSNE provably exists (or when a game is a congestion game), PSNEs can still be difficulty 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 $\{0, \ldots, n\}$. It is straightforward to see that the only edges in the deviation graph are between adjacent integers: $i, j \in \{0, \ldots, n\}$ such that $|i - j| = 1$.

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. □

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 $\{\lfloor \frac{n}{2} \rfloor + 1, \ldots, n\}$. 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 $\{0, \ldots, \lfloor \frac{n}{2} \rfloor\}$, since a path from $\lfloor \frac{n}{2} \rfloor$ must be decreasing and stop at a sink.

[1] 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.

(3) If there is no edge between the two configurations, then there must exist a PSNE in each of the subsets $\{0, \ldots, \lfloor \frac{n}{2} \rfloor\}$ and $\{\lfloor \frac{n}{2} \rfloor + 1, \ldots, n\}$).

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. □

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 CIRUITSAT problem instance $C(y_1, \ldots, y_m)$, 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 an satisfying assignment of $y_1, \ldots, y_m$.

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. $C(x_1^0, \ldots, x_1^m) = 1$ where x_1^i is the ith 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. □

4. Symmetric games with piecewise linear utilities

We consider a setting where each utility function is piecewise linear. Two different representations of piecewise linear functions are considered and parallel results are developed for both representations. The first is an *explicit representation* which has as a polytopal subdivision of D

into region over which utility is linear. As input for each action $a \in A$ there is a finite set of polytopes $\{P_{aj}\}_{j \in J_a}$ which partition the set of configurations D, and a finite set of affine functions with integer coefficients $\{f_{aj}\}_{j \in J_a}$ such that

$$u_a(\mathbf{x}) = f_{aj}(\mathbf{x}) \text{ for } \mathbf{x} \in P_{aj}. \tag{4.1}$$

In other words, the utility functions are affine functions over specified polytopal subdivisions of D. For simplicity we will henceforth refer to utility functions in this representation as PWL utility functions.

Secondly, we consider an implicit form of piecewise linear representation called *difference of piecewise linear convex* (DPLC) utilities. The input here is a pair of sets of affine functions with integer coefficients $\{u_{ak}\}_{j \in J_a}$ and $\{v_{al}\}_{j \in J_a}$ such that

$$u_a(\mathbf{x}) = \max_{k \in K_a} u_{ak}(\mathbf{x}) - \max_{l \in L_a} v_{al}(\mathbf{x}). \tag{4.2}$$

Where convenient we label the elements of J_a by $\{1, \ldots, |J_a|\}$, and similarly for K_a and L_a. Without loss of generality we assume that the utility functions are all positive over the domain D.

Theoretically our two representations of piecewise linear objectives are equivalent, as stated in the following proposition:

Proposition 4.1. *A utility function over D can be represented as an explicit piecewise linear utility function of the form (4.1) if and only if it can be represented as a* DPLC *utility function of the form (4.2).*

Proof. We first prove the if-direction. Given a DPLC function of the form (4.2) we consider partitioning the set of configurations D into regions where the objectives are linear. For each action $a \in A$ partition D as follows

$$D = \left(\biguplus_{(k,l) \in K_a \times L_a} D^a_{k,l} \right)$$

where

$$D^a_{k,l} = \left\{ \mathbf{x} \in D : x_a \geq 1, \quad \begin{array}{ll} u_{ak}(\mathbf{x}) \geq u_{aj}(\mathbf{x}),\ j > k, & u_{ak}(\mathbf{x}) \geq u_{aj}(\mathbf{x}) + 1,\ j < k \\ v_{al}(\mathbf{x}) \geq v_{aj}(\mathbf{x}),\ j > l, & v_{al}(\mathbf{x}) \geq v_{aj}(\mathbf{x}) + 1,\ j < l \end{array} \right\}.$$

is the set of elements in D where $\max_{j \in K_a} u_{aj}(\mathbf{x}) = u_{ak}(\mathbf{x})$ and $\max_{j \in L_a} v_{aj}(\mathbf{x}) = v_{al}(\mathbf{x})$ breaking ties to make the sets disjoint (see Lemma **??**. Now if we let $J_a = K_a \times L_a$ then our explicit representation of u will have $P_{aj} = D^a_{k,l}$ and $f_{aj}(\mathbf{x}) = u_{ak}(\mathbf{x}) - v_{al}(\mathbf{x})$ for $j = (k,l) \in J_a$.

Conversely, given an explicitly represented piecewise linear function a result by Zalgaller (Theorem 4.2 in [**?**]) shows that it can be represented by a DPLC function. In his proof Zalgaller shows that our explicitly represented piecewise-linear function u_a can be expressed as the difference $u_a = g - h$ where $h = \sum_{i=1}^{m} h_i$ with each h_i a piecewise linear function with two pieces and $g = u_a + h$, which is also shown to be a piecewise linear convex function. The number m of functions h_i in the sum defining h depends on the nature of the polytopal division $\{P_{aj}\}_{j \in J_a}$ of D and is bounded above by the total number of facets in the polytopes P_{aj}. The proof in [**?**] is not explicit enough for us to easily show how to convert g and h into "maximum of affine functions" form. Indeed, there is some flexibility in how to choose the h_i. It is clear, however, that such a representation exists as every piecewise linear convex function has a "maximum of affine functions" representation. □

In practice, the conversion from an explicitly represented piecewise linear function into a DPLC function seems of little value, but the opposite conversion, which is given explicitly in the above proposition may be of interest depending on the context. Although in theory these two representations of utilities are equivalent, from a size of representation point of view they can be quite

different. The benefit of the DPLC representation is that there is no need to explicitly represent the polytopal subdivision of D, which may yield savings in terms of input size. Although the explicit representation of a piecewise linear utility is typical, there are setting where a DPLC formulation is more natural. For instance, it is common in economic applications for cost functions to be expressed as piecewise linear concave functions in "minimum of affine functions" form. The affine functions represent production processes with different start-up fixed costs (intercepts) and per-unit costs of production (slopes). A manufacturer then produces using the production process with minimum total cost.

Since configurations are *integer* vectors, one needs only consider values of the utility functions $u_a(\mathbf{x})$ at *integer* points. Hence, the relevant values of an arbitrary utility function can be described exactly by a piecewise linear function, although again a large number of pieces may be required in general. Thus piecewise linear payoff functions are expressive of general utility functions, the only limitation being in the number of pieces that may be required, and in particular if the number of pieces grows in n as opposed to $\log n$ the efficiency savings will disappear.

5. Introduction to rational generating functions

In Section 6 we will describe efficient algorithms for computing PSNE in symmetric games with piecewise linear utilities. This result relies heavily on results from the literature on rational generating functions and in particular on the method of Barvinok and Woods [4]. Generating functions have been applied in an analogous fashion in a variety of other contexts, including discrete optimization [18], combinatorics [15], social choice theory [28] and compiler optimization [36].

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.

Our starting point is is the work of Barvinok [2] who introduced an algorithm for determining the exact number of lattice points in a rational polytope $P = \{\, \mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \leq \mathbf{b} \,\}$, that runs in polynomial time for every fixed dimension n using a rational generating functions representation of P. The major contribution is the polynomial running time of this algorithm, which is the workhorse of almost all of the complexity results in this thesis. What follows is a a brief description of this algorithms and its useful extensions and applications.

Consider the *generating function* of the lattice point set $P \cap \mathbb{Z}^n$, which is defined as

$$g(P \cap \mathbb{Z}^n; \boldsymbol{\xi}) = \sum_{\mathbf{x} \in P \cap \mathbb{Z}^n} \boldsymbol{\xi}^{\mathbf{x}} = \sum_{\mathbf{x} \in P \cap \mathbb{Z}^n} \xi_1^{x_1} \cdots \xi_n^{x_n} \in \mathbb{Z}[w_1^{\pm 1}, \ldots, \xi_n^{\pm 1}]. \tag{5.1}$$

Note that each lattice point $\mathbf{x}$ in P is mapped to the exponent of a monomial $\boldsymbol{\xi}^{\mathbf{x}}$ in $g(P; \boldsymbol{\xi})$, and thus we have simply encoded lattice points in a large Laurent polynomial. The use of this is demonstrated in Barvinok's remarkable algorithm which computes a compact representation of the function $g(P; \boldsymbol{\xi})$ as a *short* sum of basic rational functions:

Theorem 5.1 (**Barvinok's Theorem** [2])**.** *Let P be a polytope in $\mathbb{R}^n$ with generating function $g(P, \boldsymbol{\xi})$ given by (5.1) which encodes the lattice points inside P. Then, there exists an algorithm which computes an equivalent representation of the form:*

$$g(P \cap \mathbb{Z}^n; \boldsymbol{\xi}) = \sum_{i \in I} \gamma_i \frac{\boldsymbol{\xi}^{\mathbf{c}_i}}{(1 - \boldsymbol{\xi}^{\mathbf{d}_{i1}})(1 - \boldsymbol{\xi}^{\mathbf{d}_{i2}}) \ldots (1 - \boldsymbol{\xi}^{\mathbf{d}_{in}})}, \tag{5.2}$$

where I is a polynomial-size index set and all data are integer. A formula of the type (5.2) is called a short rational generating function*. The algorithm runs in* polynomial time *when the dimension d is fixed.*

When a lattice point set S is expressed in this form we refer to $g(S;\boldsymbol{\xi})$ as its *short rational generating function encoding*.

Note that each of the basic rational functions has poles (the point $\boldsymbol{\xi} = \mathbf{1}$ 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 P is easy in (5.1), since clearly $|P \cap \mathbb{Z}^n| = g(P;\mathbf{1})$. Since (5.2) is a formula for the same function (except for removable singularities), we also have $|P \cap \mathbb{Z}^n| = \lim_{\boldsymbol{\xi}\to\mathbf{1}} g(P;\boldsymbol{\xi})$, which can be evaluated in polynomial time by performing a residue calculation with each basic rational function in the sum (5.2). An important point to note is that this evaluation is possible with arbitrary rational generating functions that correspond to finite lattice point sets. In other words, if we can compute in polynomial time a rational generating function of a finite lattice point set $S \subseteq \mathbb{Z}^n$, we can also compute in polynomial time its cardinality:

Theorem 5.2 (**Counting Theorem** [2])**.** *Let the dimension n be a fixed constant. Given a lattice point set $S \in \mathbb{Z}^n$ and a short rational generating function $g(S,\boldsymbol{\xi})$ for S there exists a polynomial time algorithm for computing $|S|$.*

The key result that makes this theory useful in our setting are due to the fact that more general lattice point sets admit Barvinok encodings, in our case sets of pure equilibria. The contributions found in [4] develops powerful algorithms that apply to these more general settings.

The first of these algorithms concerns constant-length Boolean combinations of finite lattice point sets that are already encoded by rational generating functions.

Theorem 5.3 (**Boolean Operations Theorem**)**.** (Corollary 3.7 in [4]) *Let m and ℓ be fixed integers. Then there exists a constant $s = s(\ell, m)$ and a polynomial-time algorithm for the following problem. Given as* input*, in binary encoding,*

(I_1) *the dimension n and*

(I_2) *rational generating functions*

$$g(S_p;\boldsymbol{\xi}) = \sum_{i\in I_p} \gamma_{pi} \frac{\boldsymbol{\xi}^{\mathbf{c}_{pi}}}{(1-\boldsymbol{\xi}^{\mathbf{d}_{pi1}})\dots(1-\boldsymbol{\xi}^{\mathbf{d}_{pis}})},$$

of m finite sets $S_p \subseteq \mathbb{Z}^n$, represented by the rational numbers γ_{pi}, integer vectors $\mathbf{c}_{pi}$ and $\mathbf{d}_{pij}$ for $p = 1,\dots,m$, $i \in I_p$, $j = 1,\dots,\ell_{mp}$ such that the numbers ℓ_{mp} of terms in the denominators are at most ℓ,

output*, in binary encoding,*

(O_1) *rational numbers γ_i, integer vectors $\mathbf{c}_i$, $\mathbf{d}_{ij}$ for $i \in I$, $j = 1,\dots,s_i$, where $s_i \le s$, such that*

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

is a rational generating function of the finite set S that is the Boolean combination *of the sets $S_1,\dots,S_m$; that is, S is a combination of unions, intersections and set differences of the sets $S_1,\dots,S_m$.*

The essential part of the construction of Theorem 5.3 is the implementation of set intersections, which are based on the *Hadamard product* [4, Definition 3.2], which is the bilinear extension of the operation defined on monomials as

$$\alpha\mathbf{x}^{\boldsymbol{\xi}} * \alpha'\mathbf{x}^{\boldsymbol{\xi}'} = \begin{cases} \alpha\alpha'\mathbf{x}^{\boldsymbol{\xi}} & \text{if } \boldsymbol{\xi} = \boldsymbol{\xi}', \\ 0 & \text{otherwise.} \end{cases}$$

With this definition, clearly

$$g(S_1 \cap S_2) = g(S_1;\boldsymbol{\xi}) * g(S_2;\boldsymbol{\xi}).$$

Remark 5.4 (**Disjoint unions**). If two lattice point sets S and T are disjoint then the generating function for $S \cup T$ is simply the sum of generating functions for S and T, since each monomial in the generating function will correspond to exactly one point in $S \cup T$. Note that if S and T are not disjoint then some monomial terms would have multiplicities and thus not fit into the desired form (5.1). More generally, for disjoint lattice point sets $S_1, \ldots, S_m$:

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

where $\biguplus$ is notation for disjoint union. In this case we do not appeal to the Boolean Operations Theorem, and thus m does not need to be fixed in size.

Another powerful method to define lattice point sets is by integer projections. Let $S \subseteq \mathbb{Z}^n$ be a finite lattice point set, given by its rational generating function $g(S; \boldsymbol{\xi})$. Let $\psi \colon \mathbb{Z}^n \to \mathbb{Z}^k$ be a linear function and denote by $T = \psi(S)$ the image (projection) of S. Then it is easy to compute a *positively weighted generating function*

$$g(T, \alpha; \boldsymbol{\xi}) = \sum_{\mathbf{y} \in T} \alpha_{\mathbf{y}} \boldsymbol{\xi}^{\mathbf{y}} \quad \text{where} \quad \alpha_{\mathbf{y}} = |\{\, \mathbf{x} \in S : \psi(\mathbf{x}) = \mathbf{y} \,\}| \tag{5.3}$$

in the form of a rational function, by making a *monomial substitution*. If $g(P; \mathbf{z}) = \sum_{\mathbf{x} \in P \cap \mathbb{Z}^n} \mathbf{z}^{\mathbf{x}}$ is the generating function of P then substituting $\mathbf{z}$ with $\boldsymbol{\xi}^{\psi(\mathbf{x})}$ yields the generating function $g(T, \boldsymbol{\xi})$ for the image of P.

This monomial substitution can also be done directly in the short rational generating function representation of $g(P; \mathbf{w})$ and the outcome computed in polynomial time (Theorem 2.6 in [4]).

Remark 5.5 (**One-to-one mappings**). We remark that if the map ψ is one-to-one (injective) from S, then $g(T, \alpha; \boldsymbol{\xi})$ is the same as the ordinary (i.e., unweighted) generating function $g(T; \boldsymbol{\xi})$ of the projection T.

When a set T that is represented by pwgf's we no longer have the ability to count the number of points in T exactly, as we would if T is represented by an unweighted generating function. Nonetheless, by evaluating a pwgf $g(T, \alpha; \boldsymbol{\xi})$ at $\boldsymbol{\xi} = \mathbf{1}$, we can decide whether $T = \emptyset$. Moreover, the pwgf of an intersection of sets can be determined from the pwgf's of the given sets using Hadamard products. However, the full set of Boolean operations of Theorem 5.3 is *not* available. This restriction will be detrimental to the kind of analysis done in this thesis and thus we only consider unweighted generating functions (henceforth just referred to as generating functions) in what follows. The implication is that we must isolate our attention to integer projections which do not introduce any weights. This trivially includes one-to-one projections but also projections of polytopal lattice point sets, which we now describe.

In the special case where S is the set of lattice points in a polytope P, the integer projection method of [4] can be employed to construct an ordinary (unweighted) rational generating function of the projection T. This allows us to use the full set of Boolean operations of Theorem 5.3 on the resulting generating function.

Theorem 5.6 (**Projection Theorem**). (Theorem 1.7 in [4]) *Let the dimension n be a fixed constant. Then there exists a constant $s = s(n)$ and a polynomial-time algorithm for the following problem. Given as* input*, in binary encoding,*

(I_1) *an inequality description of a rational polytope $P \subset \mathbb{R}^n$;*
(I_2) *a positive integer k; and*
(I_3) *a linear map $\psi \colon \mathbb{Z}^n \to \mathbb{Z}^k$ given by an integral matrix;*

output*, in binary encoding,*

(O_1) *rational numbers* γ_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*

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

is a rational generating function of the set $T = \psi(P \cap \mathbb{Z}^n)$*.*

Once a rational generating function of a set S has been computed, various pieces of information can be extracted from it. Finding the cardinality has already been discussed, further computations are available, many of which are based on cardinality calculations. As a first application, we can *explicitly enumerate* all elements of S. Since the cardinality of S can be exponential in the encoding length of the input, we use output-sensitive complexity analysis, i.e., to measure the complexity of the enumeration algorithm in terms of both the input and the output. The strongest notion of an output-sensitive polynomial-time enumeration algorithm is that of a *polynomial-space polynomial-delay enumeration algorithm*. Such an algorithm only uses space that is polynomial in the encoding length of the input data. In addition, the time spent between outputting two items, and before outputting the first item and after outputting the last item, is bounded by a polynomial in the encoding length of the input data. The following result is a version of Theorem 7 of [16].

Theorem 5.7 (**Enumeration Theorem**)**.** *Let the dimension* n *and the maximum number* ℓ *of binomials in the denominator be fixed. Then there exists a polynomial-space polynomial-delay enumeration algorithm for the following enumeration problem. Given as* input*, in binary encoding,*

(I_1) *a number* $M \in \mathbb{N}$*;*

(I_2) *rational numbers* γ_i*, integer vectors* $\mathbf{c}_i$*,* $\mathbf{d}_{ij}$ *for* $i \in I$*,* $j = 1, \dots, \ell_i$*, where* $\ell_i \leq \ell$ *such that*

$$\sum_{i \in I} \gamma_i \frac{\boldsymbol{\xi}^{\mathbf{c}_i}}{(1 - \boldsymbol{\xi}^{\mathbf{d}_{i1}})(1 - \boldsymbol{\xi}^{\mathbf{d}_{i2}}) \dots (1 - \boldsymbol{\xi}^{\mathbf{d}_{is}})}$$

is a rational generating function of a set $S \subseteq \mathbb{Z}^n$ *of lattice points with* $S \subseteq [-M, M]^n$*;*

output*, in binary encoding,*

(O_1) *all points in* S *in lexicographic order.*

In addition, binary search can be used to optimize a linear function over a lattice point set encoded as a generating function:

Theorem 5.8 (**Linear Optimization Theorem**)**.** *Let the dimension* n *and the maximum number* ℓ *of binomials in the denominator be fixed. Then there exists a polynomial-time algorithm for the following problem. Given as* input*, in binary encoding,*

(I_1): *and (I_2) as in Theorem 5.7*

(I_3): *a vector* $\mathbf{f} \in \mathbb{Z}^n$*,*

output*, in binary encoding,*

(O_1) *an optimal solution* $\mathbf{x}^* \in \mathbb{Z}^k$ *of the optimization problem* $\max\{ \langle \mathbf{f}, \mathbf{x} \rangle : \mathbf{x} \in S \}$*.*

When the objective function is an arbitrary polynomial function (without any assumptions on convexity), then it is still possible to use a fully polynomial time approximation scheme (FPTAS). The following result is a more general formulation of Theorem 1.1 from [17]; it first appeared in this form in [16].

Theorem 5.9 (FPTAS for maximizing non-negative polynomials over finite lattice point sets)**.** *For all fixed integers* k *(dimension) and* ℓ *(maximum number of binomials in the denominator), there exists a polynomial-time algorithm for the following problem. Given as* input

(I_1) *two vectors* $\mathbf{x}_L$*,* $\mathbf{x}_U \in \mathbb{Z}^k$*,*

(I$_2$) *rational numbers* γ_i*, integer vectors* $\mathbf{c}_i$*,* $\mathbf{d}_{ij}$ *for* $i \in I$*,* $j = 1, \ldots, \ell_i$*, where* $\ell_i \leq \ell$ *such that*

$$g(X; \boldsymbol{\xi}) = \sum_{i \in I} \gamma_i \frac{\boldsymbol{\xi}^{\mathbf{c}_i}}{(1 - \boldsymbol{\xi}^{\mathbf{d}_{i1}})(1 - \boldsymbol{\xi}^{\mathbf{d}_{i2}}) \ldots (1 - \boldsymbol{\xi}^{\mathbf{d}_{i\ell_i}})}$$

is a rational generating function of a finite set $S \subseteq \mathbb{Z}^k$ *of lattice points that is contained in the box* $\{ \mathbf{x} : \mathbf{x}_{\mathrm{L}} \leq \mathbf{x} \leq \mathbf{x}_{\mathrm{U}} \}$*,*

(I$_3$) *a list of coefficients* $f_i \in \mathbb{Q}$*, encoded in binary encoding, and exponent vectors* $\boldsymbol{\alpha}_i \in \mathbb{Z}_+$*, encoded in unary encoding, representing a polynomial*

$$f = \sum_i f_i \mathbf{x}^{\boldsymbol{\alpha}_i} \in \mathbb{Q}[x_1, \ldots, x_n]$$

that is non-negative on X*,*

(I$_4$) *a positive rational number* $1/\epsilon$ *encoded in unary encoding,*

output*, in binary encoding,*

(O$_1$) *a point* $\mathbf{x}_\epsilon \in S$ *that satisfies*

$$f(\mathbf{x}_\epsilon) \geq (1 - \epsilon) f^* \quad \textit{where} \quad f^* = \max_{\mathbf{x} \in S} f(\mathbf{x}).$$

An ingredient in the proof of the FPTAS which will be used in other contexts below is the following result due to Barvinok [3] and generalized in [16].

Lemma 5.10. *Let* $g(S, \boldsymbol{\xi})$ *denote the short rational generating function encoding of* $S \subseteq \mathbb{Z}^n$*. Let* f *denote a polynomial in* $\mathbb{Z}[x_1, \ldots, x_k]$ *(i.e. a polynomial with integer coefficients) of maximum degree* q*. We can compute, in polynomial time in* q *and the input size, a short rational generating function representation of the function* $g_f(S, \boldsymbol{\xi}) = \sum_{\mathbf{x} \in S} f(\mathbf{x}) \boldsymbol{\xi}^{\mathbf{x}}$*.*

We can use the generating function $g_f(S, \boldsymbol{\xi})$ to evaluate the sum of values of the function f on all the points in S, a generalization of the Count Theorem, as follows:

Theorem 5.11. *Let* $S \in \mathbb{Z}^k$ *be a finite lattice point set,* f *a polynomial in* $\mathbb{Z}[x_1, \ldots x_k]$ *of degree* q*. Given a short rational generating function representation of the function* f *over* S*,* $g_f(S, \boldsymbol{\xi})$ *as input, there exists an algorithm that runs in in polynomial time in* q *and the input size to evaluate the sum* $\sum_{\mathbf{x} \in S} f(\mathbf{x})$*.*

6. Encoding equilibrium configurations

The main result of this section will be a rational generating function description of the set N of pure Nash equilibrium configurations of symmetric games with piecewise linear utility functions.

6.1. **A decomposition-based encoding.** To start we will consider explicitly represented piecewise linear utility functions of the form (4.1). To establish the result we take advantage of the polyhedral subdivisions of D which define the utility functions and view this in terms of boolean operations involving lattice points of polyhedra.

Theorem 6.1. *Consider an symmetric integer programming game with* PWL *payoffs of the form 4.1 given by the following input in binary encoding:*

(I$_1$) *the number* n *of players;*

(I$_2$) *action set* A*;*

(I$_3$) *for each* $a \in A$ *a nonnegative integer* $|J_a|$*, and for each* $j \in J_a = \{1, \ldots, |J_a|\}$ *an inequality description of the polytope* P_{aj} *given by an integer matrix* M_{aj} *and integer right-hand side vector* $\mathbf{b}_{aj}$ *such that* $P_{aj} = \{\mathbf{x} \in \mathbb{R}^{|A|} : M_{aj}\mathbf{x} \leq \mathbf{b}_{aj}\}$ *and integer vectors* $\boldsymbol{\alpha}_{aj} \in \mathbb{Z}^{|A|}$ *and integers* β_{aj} *defining the affine functions* $f_{aj}(\mathbf{x}) = \boldsymbol{\alpha}_{aj} \cdot \mathbf{x} + \beta_{aj}$*.*

Then, the set N of pure Nash equilibrium configurations has a short rational generating function encoding, which can be computed in polynomial time when the number of actions is fixed.

Proof. From (2.2) we have the following definition of N:

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

The only major difficulty in applying generating functions to encode 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 where the objectives are linear; that is, the polytopal subdivisions $\{P_{aj}\}_{j \in J_a}$ for all $a \in A$.

We use these partitions of D to express N as a Boolean combination of polytopal lattice point sets, in view of an eventual application of the Boolean Operations Theorem. Here is the resulting expression:

$$N = D \setminus \bigcup_{a,a'} \biguplus_{j} \biguplus_{j'} Dev(a, a', j, j') \tag{6.1}$$

where the first union is over all $a, a' \in A$, the second union is over $j \in J_a$, the third union over $j' \in J_{a'}$ and the *deviation set*

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

is the set of configurations where a player currently playing action a in a configuration lying in region P_{aj} has a profitable deviation to playing action a' yielding a new configuration $\mathbf{x}' \in P_{a'j'}$.

It is straightforward to verify that (6.1) is indeed a valid description of N, we only point out the reasoning behinds why the second two unions are disjoint. The union indexed by j is disjoint because the sets P_{aj} form a partition of D. The union indexed by j' is disjoint because once actions a and a' are chosen, $\mathbf{x}'$ is determined uniquely by $\mathbf{x}$ and lies in precisely one set $P_{a'j'}$.

The discussion in Remark 5.4 highlights the reason why we took such care in describing which unions in our expression for N are disjoint and which are not. Note that the second and third unions are not of fixed length as would be required in applying the Boolean Operations Theorem. 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, first note each of the $Dev(a, a', j, j')$ are polytopal lattice point sets and thus admit short rational generating function encodings $g(Dev(a, a', j, j'), \boldsymbol{\xi})$ that can be computed in polynomial time.

Noting Remark 5.4, for all $a, a' \in A$ the disjoint union

$$Dev(a, a') = \biguplus_{j} \biguplus_{j'} Dev(a, a', j, j')$$

can also be encoded by a short rational generating in polynomial time:

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

Thus since $N = D \setminus \bigcup_{a,a' \in A} Dev(a, a')$ and D and each of the sets $Dev(a, a')$ have short rational generating function encodings, we derive such an encoding for N using the Boolean Operations Theorem. It is important to note in applying the theorem that this Boolean combination of sets describing N are of constant length, since $|A|$ is fixed. □

Corollary 6.2. *Consider a symmetric game with* PWL *utilities given by the same input as in Theorem 6.1. Then there is*

(1) a polynomial time algorithm to compute the number of pure Nash equilibrium configurations,

(2) *a polynomial time algorithm to find a sample pure strategy Nash equilibrium, when at least one exists, and*
(3) *a polynomial-space polynomial-delay algorithm to enumerate all the pure Nash equilibrium configurations of the game,*

when the number of actions $|A|$ is fixed.

Proof. These are direct applications of the Counting Theorem and Enumeration Theorem to N. From the previous theorem we can find a short rational generating function encoding $g(N, \boldsymbol{\xi})$ of N in polynomial time and hence: a direct application of the Counting Theorem yields (1), applying the Enumeration Theorem (noting the bound $N \subseteq [0, n]^k$) we need only wait polynomial time to output the first element of N thus implying (2), and applying the Enumeration Theorem directly establishes (3). □

Of course this result trivially answers the problem of deciding if pure Nash equilibria exist in symmetric games with DPLC payoffs, simply count the number of equilibria by the above algorithm and observe if the number is nonzero. This can be done in polynomial time when the number of actions $|A|$ is fixed. When we contrast this result with that of Brandt et. al. [6] we see that the input of a utility value for each action and configuration has been replaced with an input of the binary encoding of each piece of the PWL function which describes the utilities. In particular, since the input is the number n and its encoding size is $\log n$ the number of players is *exponential* in the input size if the other input data, including the number of pieces, scales with $\log n$. By contrast the algorithm of Brandt et. al. [6] scales with n.

6.2. **Encoding extended equilibria.** In the case of DPLC functions we could apply Proposition 4.1 to convert DPLC utility functions into PWL utility functions and then apply Theorem 6.1 to the to yield an encoding of N. Despite this, we present an alternate approach to encoding N, which follows a different argument and takes as input the DPLC description of utilities given in (4.2), which is more compact than its PWL equivalent, it has fewer pieces to encode and no need for describing a polytopal subdivision of D directly. In addition, we believe the following approach to encoding N is of interest in its own and is thus included. The idea of the encoding is to introduce auxiliary variables $\mathbf{y}$ and $\mathbf{z}$ which capture information about the utilities. The relationship we want to establish between these new variables, the configurations $\mathbf{x}$ and the utilities are defined in the following set of *extended equilibrium configurations*:

$$\hat{N} = \left\{ (\mathbf{x}, \mathbf{y}, \mathbf{z}) \in D \times \mathbb{Z}^{A \times A} \times \mathbb{Z}^{A \times A} : \forall a, a' \in A : \begin{array}{ll} y_{aa'} = \max_{k \in K_{a'}} u_{a'k}(\mathbf{x} + \mathbf{e}'_a - \mathbf{e}_a), & \\ z_{aa'} = \max_{l \in L_{a'}} v_{a'k}(\mathbf{x} + \mathbf{e}'_a - \mathbf{e}_a), & \\ y_{aa} - z_{aa} \geq y_{aa'} - z_{aa'} & \text{when } x_a \geq 1 \end{array} \right\}. \tag{6.2}$$

Proposition 6.3. *The exists a bijection between the sets N and $\hat{N}$, where N is the set of pure Nash equilibrium configurations in a symmetric game defined by (2.2) and $\hat{N}$ is the set of extended equilibrium configurations defined in (6.2).*

Proof. Consider the mapping $\varphi : N \longrightarrow \hat{N}$ defined by $\mathbf{x} \longmapsto (\mathbf{x}, \mathbf{y}, \mathbf{z})$ where for all $a, a' \in A$, $y_{aa'} = \max_{k \in K_{a'}} u_{a'k}(\mathbf{x} + \mathbf{e}'_a - \mathbf{e}_a)$ and $z_{aa'} = \max_{l \in L_{a'}} v_{a'k}(\mathbf{x} + \mathbf{e}'_a - \mathbf{e}_a)$. In particular the mapping yields: $u_a(\mathbf{x}) = y_{aa} - z_{aa}$ and more generally, $u_{a'}(x + \mathbf{e}_{a'} - \mathbf{e}_a) = y_{aa'} - z_{aa'}$.

We first claim that φ is well-defined. If $\mathbf{x} \in N$ then for all $a \in A$ either $x_a = 0$ or for all $a' \in A$, $u_a(\mathbf{x}) \geq u_{a'}(\mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a)$. Thus, this implies that if $x_a \geq 1$ then $y_{aa} - z_{aa} \geq y_{aa'} - z_{aa'}$ and hence $(\mathbf{x}, \mathbf{y}, \mathbf{z})$.

The fact that that φ is a bijection is then straightforward to verify. □

Theorem 6.4. *Consider a symmetric game with* DPLC *utilities given by (4.2) with input:*

(I$_1$) *the number n of players;*
(I$_2$) *action set A;*
(I$_3$) *for each $a \in A$, nonnegative integers $|K_a|$ and $|L_a|$, and for all integers $k \in K_a$ and $l \in L_a$, integer vectors $\boldsymbol{\alpha}_{ik} \in \mathbb{Z}^{|A|}$, $\boldsymbol{\gamma}_{il} \in \mathbb{Z}^{|A|}$ and integers β_{ak}, δ_{ak} defining the affine functions u_{ak} and v_{al} by $u_{ak}(\mathbf{x}) = \boldsymbol{\alpha}_{ik} \cdot \mathbf{x} + \beta_{ik}$ and $v_{al}(\mathbf{x}) = \boldsymbol{\gamma}_{al} \cdot \mathbf{x} + \delta_{il}$ for all $\mathbf{x} \in D$.*

Then there exists a polynomial time algorithm which computes a short rational generating function encoding of $\hat{N}$ when the number of actions $|A|$ is fixed.

Proof. Let

$$\hat{D} = \left\{ (\mathbf{x}, \mathbf{y}, \mathbf{z}) \in D \times \mathbb{Z}^{A \times A} \times \mathbb{Z}^{A \times A} : \begin{array}{c} \forall a, a' \in A : \quad \forall k \in K_{a'},\ u_{a'k}(\mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a) \leq\ y_{aa'} \leq U'_a, \\ \forall l \in L_{a'},\ v_{a'k}(\mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a) \leq z_{aa'} \leq V'_a, \end{array} \right\}.$$

where U'_a and V'_a are upper bounds on the value of the utilities, as defined below. The reason for including these bounds is in anticipation of applying Barvinok's algorithm for encoding the set $\hat{D}$ as a short rational generating function, which applies only to polytopes.

We define the bound $U_{a'}$ for $y_{aa'}$ as follows, $V_{a'}$ is defined analogously. Let $U_{a'k}$ denote the optimal value of the following linear program:

$$\max\{u_{a'k}(\mathbf{x}) : \mathbf{x} \in \operatorname{conv}(D)\}.$$

The feasible set $\operatorname{conv}(D)$ is the convex hull of the set of feasible configurations D. Thus, the value $U_{a'k}$ can be found in polynomial time (say by the ellipsoid method) and is of size polynomial in the binary encoding of the input. Note that $U_{a'k}$ is an upper bound on the corresponding integer program: $\max\{u_{a'k}(\mathbf{x}) : \mathbf{x} \in D\}$.

Let $U_{a'} = \max_k U_{a'k}$. Again, this can be computed in polynomial time, since K_a is part of the input. This bound does not overly restrict the choice of $y_{aa'}$ since we have the relation:

$$y_{aa'} \leq \max_k u_{a'k}(\mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a) \leq \max_k \max_{\mathbf{x}} u_{a'k}(\mathbf{x}) = \max_k U_{a'k} = U_{a'}.$$

Having described $\hat{D}$, the next step is to use express $\hat{N}$ as a Boolean combination of polytopal lattice point sets. Here is one possible way of expressing $\hat{N}$ in this form:

$$\hat{N} = \hat{D} \setminus \left(\bigcup_{a,a' \in A} Y^{>}_{a,a'} \right) \setminus \left(\bigcup_{a,a' \in A} Z^{>}_{a,a'} \right) \setminus \left(\bigcup_{a,a' \in A} W^{>}_{a,a'} \right)$$

where

$$Y_{a,a'} = \{(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \hat{D} : \forall k \in K_{a'} : y_{aa'} \geq u_{a'k}(\mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a) + 1\},$$

$$Z_{a,a'} = \{(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \hat{D} : \forall l \in L_{a'} : z_{aa'} \geq v_{a'k}(\mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a) + 1\},$$

and

$$W_{a,a'} = \{(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \hat{D} : x_a \geq 1, y_{aa} - z_{aa} \leq y_{aa'} - z_{aa'} - 1\}.$$

We can view $Y_{a,a'}$ as the set of $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ where for $a, a' \in A$ the values of $y_{aa'}$ is too large for the set $\hat{N}$. We can similarly interpret the sets $Z_{a,a'}$ and $W_{a,a'}$. It is straightforward to verify that this is indeed a valid description of $\hat{N}$.

Each of $Y_{a,a'}$, $Z_{a,a'}$ and $W_{a,a'}$ are polytopal lattice points sets and therefore can be encoded by short rational generating functions in polynomial time. The Boolean expression for $\hat{N}$ thus consists of a fixed number of set operations over sets with short rational generating function encodings and thus $\hat{N}$ itself can be encoded by a short rational generating function in polynomial time, by the Boolean Operations Theorem. □

Corollary 6.5. *Consider an symmetric integer programming game with* DPLC *payoffs of the form 4.2 with input as in the previous theorem. Then, the set N of pure Nash equilibrium configurations has a short rational generating function encoding, which can be computed in polynomial time when the number of actions $|A|$ is fixed.*

Proof. With an encoding of $\hat{N}$ we get an encoding of N by simply noting the bijection in Proposition 6.3 and applying Remark 5.5. □

As in the PWL case we can use this encoding to count and enumerate equilibria.

6.3. **What's next?** In the remainder of this paper we explore extensions and applications of the encoding results found in this section for both the PWL and DPLC utility function setting. A proof in one setting leads naturally to an analogous ideas for a proof in the other. The first of these results, however, has complete proofs for both settings to give the reader the taste for the arguments. Later on we only establish results carefully for the PWL case, where the extension to the DPLC case should be clear.

7. Exploring the structure and distribution of equilibrium configurations

This section examines how, once a generating function encoding of the set of pure Nash equilibria has been attained, this encoding can be used to further analyze the structure of equilibria. In so doing, we demonstrate the power of the rational generating functions in allowing for efficient computation.

First we derive algorithms to compute approximate social welfare maximizing equilibria. Next we demonstrate how generating functions can be used to learn about the distributions of welfare amongst the set of equilibria, in terms of calculating the mean and variance of social welfare over the set N. Lastly, we explore the distributions of equilibrium payoffs by deriving histograms, in addition to mean and variance, for the distribution of payoffs earned from playing a particular action a in equilibrium. All of this gives us further insight into the structure and characteristics of alternate equilibria in symmetric games.

7.1. **Social-welfare maximizing equilibria.** When there are many equilibrium configurations in a symmetric game, one may ask which configuration is "best" from a social-welfare point of view. That is, we are interested in finding an optimal solution to the optimization problem:

$$\max\{\sum_{a\in A} x_a u_a(\mathbf{x}) : \mathbf{x} \in N\}; \tag{7.1}$$

that is, find an equilibrium which maximizes the sum of utilities. Denote by $w(\mathbf{x})$ the objective function of this problem and let w^* denote its optimal value. The next result describes an FPTAS for this optimization problem.

Theorem 7.1. *Consider a symmetric game with either* PWL *given by the same input as in Theorem 6.1, and in addition a positive rational number $\frac{1}{\epsilon}$ encoded in unary encoding. Then, for a fixed number of actions $|A|$ there exists a polynomial time algorithm to output, in binary encoding, a configuration $\mathbf{x}_\epsilon \in D$ that satisfies*

$$w(\mathbf{x}_\epsilon) \geq (1-\epsilon)w^* \text{ where } w(\mathbf{x}) = \sum_{a\in A} x_a u_a(\mathbf{x}) \text{ and } w^* = \max_{\mathbf{x}\in N} w(\mathbf{x})$$

Proof. We establish the result in both the PWL and DPLC settings. We begin with considering PWL utility functions. A major difficulty in solving for w^* is in understanding the objective function $w(\mathbf{x})$. To aid in this we partition the feasible region N into subregions where $u_a(\mathbf{x})$ is linear simultaneously for all $a \in A$. This is achieved by considering the problem separately within

each cell of the the common refinement of each partition $\{P_{aj}\}_{j\in J_a}$ of D. Let $J = \prod_{a\in A} J_a$, then we have the following fine partition of D:

$$D = \bigcup_{\mathbf{j}\in J} P_{\mathbf{j}} \tag{7.2}$$

where $\mathbf{j} = (j_a)_{a\in A} \in J$ and $P_{\mathbf{j}} = \bigcap_{a\in A} P_{aj_a}$. Note that this is also a partition of N. It is then clear that for $\mathbf{x} \in P_{\mathbf{j}}$ each action's utility is $u_a(\mathbf{x}) = f_{aj_a}(\mathbf{x})$ and thus,

$$\begin{aligned} w^* &= \max\left\{\sum_{a\in A} x_a u_a(\mathbf{x}) : \mathbf{x} \in N\right\} \\ &= \max_{\mathbf{j}\in J} \max\left\{\sum_{a\in A} x_a f_{aj_a}(\mathbf{x}) : \mathbf{x} \in N \cap P_{\mathbf{j}}\right\} \end{aligned}$$

Our approach will be to find an ϵ-optimal solution $\mathbf{x}^*_{\mathbf{j}}$ for each inner optimization problem

$$\max\left\{\sum_{a\in A} x_a f_{aj_a}(\mathbf{x}) : \mathbf{x} \in N \cap P_{\mathbf{j}}\right\} \tag{7.3}$$

There are $|J| = \prod_{a\in A} |J_a|$ such subproblems and note that $|J|$ is polynomially bounded in the input size. An overall ϵ-optimal solution $\mathbf{x}^*$ is simply the $\mathbf{x}^*_{\mathbf{j}}$ which maximizes $\sum_{a\in A} x_a u_a(\mathbf{x}^*_{\mathbf{j}})$. To find an ϵ-optimal solution an inner optimization problem (7.3) we apply Theorem 5.9 with the following input:

(1) two vectors $\mathbf{x}_{\mathrm{L}} = \mathbf{0}$ and $\mathbf{x}_{\mathrm{U}} = (n, \ldots, n)$
(2) The data describing the short rational generation function encoding of $N \cap P_{\mathbf{j}}$ given by the encoding of N from Theorem 6.1 and the Boolean Operations Lemma.
(3) The polynomial objective $\sum_{a\in A} x_a f_{aj_a}(\mathbf{x})$, where the coefficients are all 1 and the exponent vectors are binary.
(4) positive rational number $\frac{1}{\epsilon}$

The result then follows.

For the DPLC case, by noting the bijection in Proposition 6.3 we can view the optimization problem (7.1) as an optimization problem over $\hat{N}$ since:

$$\begin{aligned} w^* &= \max\left\{\sum_{a\in A} x_a u_a(\mathbf{x}) : \mathbf{x} \in N\right\} \\ &= \max\left\{\sum_{a\in A} x_a u_a(\mathbf{x}) : \boldsymbol{\varphi}(\mathbf{x}) = (\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \hat{N}\right\} \\ &= \max\left\{\sum_{a\in A} x_a (y_{aa} - z_{aa}) : (\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \hat{N}\right\} \end{aligned}$$

The latter optimization problem to maximize the polynomial $\hat{w}(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \sum_{a\in A} x_a (y_{aa} - z_{aa})$ over the lattice point set $\hat{N}$. Since we assumed that utility are always positive, this is clearly a positive polynomial over the set $\hat{N}$.

Now, apply Theorem 5.9 with the following input:

(1) two vectors $(\mathbf{x}^{\mathrm{L}}, \mathbf{y}^{\mathrm{L}}, \mathbf{z}^{\mathrm{L}}) = \mathbf{0}$ and $(\mathbf{x}^{\mathrm{U}}, \mathbf{y}^{\mathrm{U}}, \mathbf{z}^{\mathrm{U}})$ where for all $a, a' \in A$, $x^{\mathrm{U}}_a = n$, $y^{\mathrm{U}}_{aa'} = \max_a U_a$ and $z^{\mathrm{U}}_a = \max_a V_a$ (where U_a and V_a are defined as in the proof of Theorem 6.5.

(2) The data describing the short rational generation function encoding of $\hat{N}$ given by Theorem 6.4, where clearly $\hat{N} \in \{(\mathbf{x}, \mathbf{y}, \mathbf{z}) : (\mathbf{x}^{\mathrm{L}}, \mathbf{y}^{\mathrm{L}}, \mathbf{z}^{\mathrm{L}}) \leq (\mathbf{x}, \mathbf{y}, \mathbf{z}) \leq (\mathbf{x}^{\mathrm{U}}, \mathbf{y}^{\mathrm{U}}, \mathbf{z}^{\mathrm{U}})$.
(3) The polynomial objective $\sum_{a \in A} x_a(y_{aa} - z_{aa})$, where the coefficients are all 1 and the exponent vectors are binary.
(4) positive rational number $\frac{1}{\epsilon}$

The output is a point $(\mathbf{x}_\epsilon, \mathbf{y}_\epsilon, \mathbf{z}_\epsilon) \in \hat{N}$ that satisfies $\hat{w}(\mathbf{x}_\epsilon, \mathbf{y}_\epsilon, \mathbf{z}_\epsilon) \geq (1 - \epsilon)w^*$. By the bijection we have $\varphi^{-1}(\mathbf{x}_\epsilon, \mathbf{y}_\epsilon, \mathbf{z}_\epsilon) = \mathbf{x}_\epsilon$ and it follows that $w(\mathbf{x}_\epsilon) \geq (1 - \epsilon)w^*$, as required. □

One might ask why we developed a separate argument for the DPLC case, when by Proposition 4.1 we could simply convert a DPLC instance into a PWL instance and then use the algorithm in that setting. note that the two settings, PWL and DPLC utilities, have two very different approaches. In the PWL case we have to find many ϵ-optimal solutions to subproblems via our partitioning, and in the DPLC case we need to expand the dimension of sets under consideration using an extended formulation. As our methods are sensitive to dimension the latter seems unfavorable, but a single FPTAS is required in this setting and so computational advantage of one method overall another is somewhat ambiguous. Further considerations of this nature are discussed in Section 10 and are a potential avenue for future investigations.

7.2. **Mean and variance of social welfare.** A related question to social welfare maximization is that of calculating the mean and variance of social welfare over all equilibrium configurations. At first glance these calculations would appear to be computationally expensive since in the worst case there may be an exponential number of equilibrium configurations. Nonetheless, we show that our polynomial-sized short rational generating function encoding of the set of equilibrium configurations can be used to answer this question efficiently. We isolate discussion to the PWL case, the details for the DPLC case are similar.

Given a symmetric game we let μ_w and σ_w^2 denote the mean and variance of social welfare over all its equilibrium configurations N which can be expressed as:

$$\mu_w = \frac{\sum_{\mathbf{x} \in N} w(\mathbf{x})}{|N|} \tag{7.4}$$

and

$$\sigma_w^2 = \frac{\sum_{\mathbf{x} \in N} (w(\mathbf{x}) - \mu_w)^2}{|N|}.$$

We claim that μ_w and σ_w^2 can be computed in polynomial time when the number is actions is fixed.

Theorem 7.2. *Consider a symmetric game with either* PWL *given by the same input as in Theorem 6.1. Then, when the number of actions is fixed, there exists a polynomial time algorithms to find the mean social welfare μ_w and variance of social welfare σ_w^2 over all equilibrium configurations.*

Proof. We show how to compute μ_w in polynomial time by establishing algorithms to compute the numerator and denominator of the right-hand side of (7.4). In Corollary 6.2 we showed how to calculate the numerator $|N|$ using a generating function encoding of N in polynomial time. The only task is to calculate the numerator. This again can be achieved using the partition of D given in equation (7.2) and using the simple fact that we can break up the sum into smaller sums over the cells in the partition; that is, by computing

$$\sum_{\mathbf{x} \in N} \hat{w}(\mathbf{x}) = \sum_{\mathbf{j} \in J} w_{\mathbf{j}} \tag{7.5}$$

where

$$w_{\mathbf{j}} = \sum_{\mathbf{x} \in N \cap P_{\mathbf{j}}} \sum_{a \in A} x_a f_{aj_a}(\mathbf{x}) \tag{7.6}$$

is the partial sum of utilities in cell $N \cap P_{\mathbf{j}}$ using the same notation as in the proof of Theorem 7.1.

To compute this sum we compute each $w_{\mathbf{j}}$ separately. Note that $\sum_{a \in A} x_a f_{aj_a}(\mathbf{x})$ is a polynomial function over the set $N \cap P_{\mathbf{j}}$ (which as have shown in the proof of Theorem 6.1 has a short rational generating functioning encoding) and thus by Lemma 5.11 we can evaluate the sum. This in turn yields an overall polynomial time algorithm to compute μ_w.

A similar argument holds for finding σ_w^2 by noting that the function $f(\mathbf{x}) = (w(\mathbf{x}) - \mu_w)^2$ is a polynomial function over each cell $N \cap P_{\mathbf{j}}$ of the partition of N and then appealing to Lemma 5.11. □

Note that we can evaluate the mean and standard deviation of social welfare without recourse to enumeration of any equilibrium configurations, which again underscores the compactness provided by rational generating function encodings.

7.3. **Distribution of payoffs in equilibrium.** It should be noted that Theorem 7.2 can be generalized to hold for finding the mean and variance of evaluating an arbitrary polynomial function $f(\mathbf{x})$ on the elements of N.

One such function of interest is simply the payoff $u_a(\mathbf{x})$ for players playing action a in configuration $\mathbf{x}$. In this subsection we explore the distribution of payoffs $u_a(\mathbf{x})$ for playing action a amongst all equilibrium configurations where actions a is played, i.e. for $\mathbf{x} \in N$ with $x_a \geq 1$. We let N_a denote this set of equilibria. We immediately get the following result.

Proposition 7.3. *Consider a symmetric game with either* PWL *or* DPLC *utilities given by the same input as in Theorems 6.1 and 6.4 respectively. Then, when the number of actions is fixed, for each specified action a there exists a polynomial time algorithm to compute the mean μ_a and variance σ_a^2 of payoffs for playing action a, taken over all equilibrium configurations $\mathbf{x} \in N_a$.*

The proof is analogous to the proof of Theorem 7.2, with the only major difference being that the we exchange N_a for N. Details are omitted.

However, since $u_a(\mathbf{x})$ corresponds to a *affine* functions over the cells $N_a \cap P_{aj}$ there is still more that we can say about how its values are distributed. As will be seen in our next result, we can construct a frequency histogram to describe the distribution of payoffs $u_a(\mathbf{x})$ over N_a.

The key step is to *count* equilibria in N_a where the payoffs to players playing action a falls in some range $u_i \leq u_a(\mathbf{x}) < u_{i+1}$ where $u_i < u_{i+1}$ are integers. The histogram $H_a^{u_0,u_1,\ldots,u_T}$ will consist of counts of equilibria lying in consecutive intervals $[u_i, u_i + 1)$ for $i = 0, \ldots T-2$ and $[u_{T-1}, u_T]$ with

$$u_{\min} = u_0 < u_1 < \cdots < u_{T-1} < u_T = u_{\max}$$

where

$$u_{\min} = \min\{u_a(\mathbf{x}) : \mathbf{x} \in N_a\},$$
$$u_{\max} = \max\{u_a(\mathbf{x}) : \mathbf{x} \in N_a\}$$

Let $c_{i,i+1}$ denote the number of equilibria in N_a where $u_a(\mathbf{x})$ lies in the interval $[u_i, u_{i+1})$ for $i = 0, \ldots T-2$ and the interval $[u_{T-1}, u_T]$ for $i = T-1$.

The next result shows such a histogram can be produced in polynomial time when the number of intervals T is not too large.

Theorem 7.4. *Given a symmetric game with* PWL *payoffs with the same input as Theorem 6.1 and a fixed number of actions there exists a polynomial time algorithm to construct histogram $H_a^{u_0,u_1,\ldots,u_T}$ provided the number of consecutive intervals T for payoffs is of size polynomial in the binary encoding of the input.*

Proof. As in the previous theorem our approach is to compute counts for the cells $N_a \cap P_{aj}$ then to sum the results over all $j \in J_a$. Thus, for $i \neq 0, T-1$ we have

$$c_{i,i+1} = \sum_{j \in J_a} |\{\mathbf{x} \in N \cap P_{aj} : u_i \leq f_{aj}(\mathbf{x}) \leq u_{i+1}\}|.$$

To compute the cardinalities of the sets in the sum we apply the Counting Theorem to the rational generating function of $\{\mathbf{x} \in N \cap P_{aj} : u_i \leq f_{aj}(\mathbf{x}) \leq u_{i+1}\}$ which itself can be found in polynomial time using the Boolean Operations Theorem.

It remains to show how to compute $c_{0,1}$ and $c_{T-1,T}$, and this is achieved as above once we know $u_{\min}$ and $u_{\max}$. We only give details for finding $u_{\max}$. Again using our partition of N_a it suffices to compute $\max\{f_{aj}(\mathbf{x}) : \mathbf{x} \in N \cap P_{aj}$ for each $j \in J_a$. Given our short rational generating function encoding of $N \cap P_{aj}$ this can be found in polynomial time by an appeal to the Linear Optimization Theorem with the appropriate input. □

8. M-symmetric games

A game is M-symmetric, when the set of players can be partitioned into M classes, labeled $\{1, \ldots, M\}$, such that within each class, players are interchangeable. This means that players in each class have the same number of actions and the same utility functions. Formally, each class i has n_i players and A_i actions with $A = \bigcup_{i=1}^{M} A_i$.

Define an M-configuration to be a vector $\mathbf{x} \in Z^A$ such that for each $i \in \{1, \ldots, M\}$, for each $a \in A_i$, x_{ia} is the number of players in class i that choose action a. The set of M-configurations is $D = \prod_{i=1}^{M} D_i$ where

$$D_i = \{\mathbf{x}_i \in Z^{|A_i|} : \mathbf{x}_i \geq 0, \sum_{a \in A_i} x_{ia} = n_i\}$$

For a player from class i, her utility of playing action $a \in A_j$ is a function of the configuration $\mathbf{x}$, denoted $u_{ia}(\mathbf{x})$.

We are interested in M-symmetric games with piecewise linear utility functions. The input is as follows: for each class i there is a polyhedral subdivision $D_i = \bigcup_{j \in J_{ia}} P_{iaj}$ where J_{ia} is a finite index set, over affine functions f_{iaj} for $j \in J_{ia}$ defining utility functions

$$u_{ia}(\mathbf{x}) = f_{iaj}(\mathbf{x}) \text{ for } \mathbf{x} \in P_{iaj}.$$

An M-configuration $\mathbf{x}$ is a pure Nash equilibrium M-configuration if and only if for all $i \in \{1, \ldots, M\}$ and for all $a, a' \in A_j$, $x_{ia} = 0$ or $u_{ia}(\mathbf{x}) \geq u_{ia'}(\mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a)$. Using similar arguments as before, we can encode the set of pure equilibria as a short rational generating function. This procedure works in polynomial time in the input size (that is, the binary encoding of n and the description of the objective functions) as long as $|A|$ and M are bounded constants.

9. Parameterized symmetric games

Another advantage of our approach is that it is now possible 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. Many problems in mechanism design have such a flavor; e.g. designing a game (i.e. setting the parameters) such that the resulting equilibria satisfy certain properties or optimize a certain objective.

Our overall approach is to augment the configuration vector with a fixed number of integer parameters and consider the *number* of players as a parameter. The actions remain the same in each member of the parametric family. We continue our construction of rational generating function for pure equilibria as before. The resulting set of points has the property that if we fix the parameters, it gives the set of pure equilibria for the game with that particular instantiation of parameters. In other words, it is the graph of the parameters-equilibria correspondence.

A precondition of our approach is that after parametrization, the constraints that specify pure equilibria are still linear. This implies that certain aspects of the game can be efficiently parameterized while certain other aspects are not.

Since we are considering a parametric family of games where the number of players is a changing parameter, we define our piecewise linear utilities over configurations of different numbers of players. That is, we define the utilities over the set of *feasible configurations*:

$$F = \{\mathbf{x} \in \mathbb{Z}^{|A|} : 0 \leq \sum_{a \in A} x_a \leq B, x_a \geq 0 \forall a \in A\}$$

where B is a bound on the total number of players. In other words, the number of players in the game can be between 0 and B and utilities can be defined on any configuration involving up to B players.

In the PWL setting the input is a polyhedral subdivision $\{P_{aj}\}_{j \in J_a}$ of F, for each action a, and affine functions f_{aj} defined over the cells P_{aj} as in Section 2 except now the division is over F. We can also now introduce a fixed number of parameters that controls the additive constants in the affine functions f_{aj}. We constrain these parameters to be the lattice points inside a polytope of fixed dimension $Q \in \mathbb{R}^d$. Formally, let $\mathbf{p}$ be d-dimensional integer vector inside of P and for each a and each $j \in J_a$, define

$$f_{aj}(\mathbf{x}, \mathbf{p}) = \boldsymbol{\alpha}_{aj}\mathbf{x} + \boldsymbol{\beta}_{aj}\mathbf{p} \tag{9.1}$$

where now

$$u_a(\mathbf{x}, \mathbf{p}) = f_{aj}(\mathbf{x}, \mathbf{p}) \text{ for } \mathbf{p} \in Q \text{ and } \mathbf{x} \in P_{aj}$$

In other words, the parameter allows for an additive change in each piece of the piecewise linear utility function u_a.

We are interested in encoding the set of *parameterized pure Nash equilibria* defined as follows:

$$N = \{(\mathbf{x}, n\mathbf{p}) : \mathbf{p} \in Q,\ 0 \leq n \leq B,\ \mathbf{x} \in N(n, \mathbf{p})\} \tag{9.2}$$

where $N(n, \mathbf{p})$ is the set of pure Nash equilibria in the symmetric game with n players, feasible configurations $\{\mathbf{x} \in F : \sum_{a \in A} = n\}$ and utilities are defined as in (9.1) for a given parameter $\mathbf{p}$.

The approach we use to encode N is similar to that of Section 6.1 in that we describe N as a Boolean combination of sets of lattice points contained in polyhedra that in part derive from a partitioning of N into cells of the form $N \cap P_{\mathbf{j}}$ using the same notation as in that section. The expression of N as a Boolean combination of sets is as follows:

$$N = F \setminus \bigcup_{a,a'} \biguplus_{j} \biguplus_{j'} Dev(a, a', j, j')$$

where the first union is over all $a, a' \in A$, the second union is disjoint over $j \in J_a$, the third union is disjoint over $j' \in J_{a'}$ and the *deviation set*

$$Dev(a, a', j, j') = \left\{ (\mathbf{x}, n, \mathbf{p}) : \begin{array}{l} \mathbf{p} \in Q,\ 0 \leq n \leq B, \\ \sum_{a \in A} x_a = n,\ x_a \geq 1,\ \mathbf{x} \in P_{aj} \\ \mathbf{x}' = \mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a \in P_{a'j'} \\ f_{aj}(\mathbf{x}) \leq f_{a'j'}(\mathbf{x}') - 1 \} \end{array} \right\}$$

It is clear that each $Dev(a, a', j, j')$ is a polytopal lattice point set in $(\mathbf{x}, \mathbf{p})$ and thus the encoding of N then obtains by applying the Boolean Operations Theorem and the content of Remark 5.4 just as in the proof of Theorem 6.1.

We have thus established the following:

Theorem 9.1. *Consider a parametric symmetric integer programming game with* PWL *payoffs of the form 4.1 given by the following input in binary encoding:*

(I_1) *an integer bound B on the number of players;*

(I_2) *an inequality description of a rational polytope Q contained in $\mathbb{R}^d$.*
(I_3) *an action set A;*
(I_4) *for each $a \in A$, a nonnegative integer $|J_a|$ and each $j \in J_a = \{1, \ldots, |J_a|\}$ an inequality description of the polytope P_{aj} and integer vectors $\boldsymbol{\alpha}_{aj} \in \mathbb{Z}^{|A|}$, and integers $\boldsymbol{\beta}_{aj}$ defining the affine functions $f_{aj}(\mathbf{x}) = \boldsymbol{\alpha}_{aj} \cdot \mathbf{x} + \boldsymbol{\beta}_{aj} \cdot \mathbf{p}$.*

Then, the set N of parameterized Nash equilibrium configurations defined in (9.2) has a short rational generating function encoding, which can be computed in polynomial time when the number of actions $|A|$ and the dimension of the parameter space d is fixed.

Once we have encoded the set N of parameterized pure equilibria in (9.2) as a short rational generating function we can optimize linear and polynomial objectives over this set efficiently in view of the Linear Optimization Theorem and Theorem 5.9. This can be seen as a form of mechanism design, where the underlying configurations of players F remain the same but a designer can adjust the number of players and utilities through $\mathbf{p}$ so as to optimize (or approximately optimize) some objective. Formally, we solve

$$\begin{aligned} \min \quad & f(\mathbf{x}, n, \mathbf{p}) \\ \text{s.t.} \quad & (\mathbf{x}, n, \mathbf{p}) \in N \end{aligned}$$

to ϵ-optimality for a general nonnegative polynomial objective f and exactly when f is linear, both in polynomial time when the number of actions is fixed. This implicitly assumes that the designer can choose a given equilibrium $\mathbf{x}$ from the set $N(n, \mathbf{p})$ once n and $\mathbf{p}$ are chosen. This can be seen as a generalization of the "optimistic assumption" common in the bilevel optimization literature, for details see [**?**].

10. Computational issues

We want to highlight a need for caution when considering the practical implications of complexity results using the above-stated results for computing equilibria in symmetric games. Although the algorithms presented have nice theoretical run-times in practice the situation in less promising. Many of these algorithms have been implemented in two software packages: `LattE machiatto` [24] and `barvinok` [35]. The algorithm of Barvinok for encoding polyhedra is quite sensitive to dimension and problems with up to dimension 30 are the largest instances that have been done in practice [25]. The algorithms for boolean operations also requires costly operations and current implementations in essentially doubling the dimension, and problems with only up to 15. Potentially more disconcerting is that the only working implementation of the Projection Theorem works has performed well on problems with only very few variables (up to four) [27]. This is an active area of research and it is hoped that future gains can be made in the practical implementations of these algorithms.

The method of this paper shares some important similarities with that of the previous study by Köppe et. al. [26] on using rational generating functions to find equilibria in integer programming games, in particular both build on the theory of rational generating functions, largely from the 2003 paper by Barvinok and Woods [4]. There are, however, some important differences. In our symmetric case, because there are a finite number of actions, it is much easier to capture deviations in this setting. Indeed, as we saw, unilateral deviations will only affect a configuration vector by 1 in two entries and thus there are few deviations one needs to consider. By contrast, in integer programming games a unilateral deviation can go from the current action profile to a large, in fact exponential, number of different alternate profiles. This is the major reason behind the fact that in this paper we can avoid the use of the Projection Theorem which seems quite is necessary in the integer programming games case. The fact that we can avoid the Projection Theorem altogether in the symmetric case makes the algorithms of this paper more likely to be practical than in the previous study.

11. Conclusions and directions for further research

In this paper we explored a promising form of compactly represented symmetric games – those with piecewise linear objectives. Given enough pieces such a setting is completely general, but the methods explored here allow us to compute pure NE with the complexity scaling in the number of pieces and not the number of configurations which define the game. This is a major advantage over previous methods. We believe no existing compact representations of symmetric games combine both the generality and powerful approach to computation offered here.

Also of note is the novel approach for studying pure strategy Nash equilibria – rational generating functions. The simplicity by which they were used to compute important information on the structure of these games, again reinforces the power of generating functions as an analytical and computational tool for game theory. This confirms the hope of the previous study which used generating functions in games [26] had that these methods can be used to enlighten further investigations in algorithmic game theory, and only adds weight to the case for future use of the method.

This work also puts more impetus on searching for improved implementations of Barvinok's Theorem and the Boolean Operations Theorem, an important area for future research. The results on parametric games also hold promise as a methodology for further investigations on the nature and structure of symmetric games. Future studies might do well to explore practical computations to glean more insights in this direction.

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 inpolyhedra when the dimension is fixed. *Mathematics of Operations Research*, 19(4):769–779, 1994.

[3] A. Barvinok. Computing the Ehrhart quasi-polynomial of a rational simplex. *Mathematics of Computation*, 75(255):1449–1466, 2006.

[4] A. Barvinok and K. M. Woods. Short rational generating functions for lattice point problems. *Journal of the American Mathematical Society*, 16(957-979), 2003.

[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, M. Köppe, and R. Weismantel. Integer polynomial optimization in fixed dimension. *Mathematics of Operations Research*, 31(1):147–153, February 2006.
[18] 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.
[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. `LattE machiatto`. Available from URL http://www.math.ucdavis.edu/ mkoeppe/latte/, 2006.
[25] M. Köppe. LattE machiatto benchmarks. http://www.math.ucdavis.edu/ mkoeppe/latte/latte-benchmark-2007-11-01.html, 2007.
[26] M. Köppe, C. T. Ryan, and M. Queyranne. Rational generating functions and integer programming games (http://arxiv.org/abs/0809.0689). August 2008.
[27] M. Köppe, S. Verdoolaege, and K. M. Woods. An implementation of the Barvinok-Woods integer projection algorithm. In *International Conference on Information Theory and Statistical Learning (June 2008)*, pages 53–59, 2008.
[28] 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.
[29] J. Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286–295, February 1951.
[30] N. Nisan, T. Roughgarden, E. Tardos, and V.V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press New York, NY, USA, 2007.
[31] C. H. Papadimitriou and T. Roughgarden. Computing correlated equilibria in multi-player games. *Journal of the ACM*, 55(3):1–29, 2008.
[32] R.W. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory*, 2:65–67, 1973.
[33] 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.
[34] Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, New York, 2009.
[35] S. Verdoolaege. `barvinok`. Available from URL http://freshmeat.net/projects/barvinok/, 2006.
[36] 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`