Computing Pure Strategy Nash Equilibria in Compact Symmetric Games

Christopher Thomas Ryan
Sauder School of Business
University of British Columbia
2053 Main Mall, Vancouver,
BC, Canada, V6T 1Z2
chris.ryan@sauder.ubc.ca

Albert Xin Jiang
Department of
Computer Science
University of British Columbia
2366 Main Mall, Vancouver,
BC, Canada, V6T 1Z4
jiang@cs.ubc.ca

Kevin Leyton-Brown
Department of
Computer Science
University of British Columbia
2366 Main Mall, Vancouver,
BC, Canada, V6T 1Z4
kevinlb@cs.ubc.ca

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 exists a PSNE and give a polynomial-time algorithm for finding one. We then focus on a specific compact representation: piecewise-linear utility functions. We give polynomial-time algorithms for finding a sample PSNE, counting the number of PSNEs, and also provide an FPTAS for finding social-welfare-maximizing equilibria. We extend our piecewise-linear representation to achieve what we believe to be the first compact representation for parameterized *families* of (symmetric) games. We provide methods for answering questions about a parameterized family without needing to solve each game from the family separately.

Categories and Subject Descriptors

J.4 [**Social and Behavioral Sciences**]: Economics

General Terms

Theory, Economics, Algorithms

Keywords

game theory, symmetric games, rational generating functions

1. INTRODUCTION

In the last decade, there has been much research at the interface of computer science and game theory (see e.g. [26, 31]). One fundamental class of computational problems in game theory is the computation of *solution concepts* of finite games. Much recent effort in the literature has concerned the complexity of computing mixed-strategy Nash [7, 9, 10, 12] and correlated equilibria [19, 27].

In this paper we focus on the problem of computing pure strategy Nash equilibria (PSNE) [6, 11, 16–18]. Unlike mixed-strategy Nash equilibria, which are guaranteed to exist for finite games [25], 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 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 PSNEs: 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 time polynomial 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 of computing PSNE of compactly-represented games is hard in the most general case, when utility functions are arbitrary efficiently-computable functions represented as Turing Machines [1] or circuits [30]. Researchers have also studied compact game representations that exploit various types of structure in utility functions. These include graphical games [20], congestion games [28] and action-graph games [5]. Computing PSNE for each of these representations is hard in general, but polynomial time for certain subclasses of games [11, 13, 16–18].

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 a symmetric mixed Nash equilibrium [25]. 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 number of players choosing each of the actions. As a

EC'10, June 7–11, 2010, Cambridge, Massachusetts, USA.

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 $\binom{n+m-1}{m-1}$. With fixed m, this grows like n^{m-1}, and $\Theta(n^{m-1})$ numbers are required to specify the game. Questions about PSNE can be answered straightforwardly by checking all configurations, which requires polynomial time in the size of the representation, and polynomial time in n when m is fixed. Indeed, Brandt et al. [6] proved that the existence problem for PSNE of symmetric games with constant number of actions is in AC^0.

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 utility functions have additional structure. In symmetric games, the set of players can be specified implicitly by the integer n, requiring only $\log n$ bits to represent. If 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, and an FPTAS for finding social-welfare maximizing equilibria.

Furthermore, we are able to extend this piecewise-linear representation to model parameterized families of symmetric games. While existing literature on equilibrium computation focused on finding equilibria in a single game, in many practical applications of game-theoretic analysis we are interested in questions about Nash equilibria of a *family* of games. For example, in mechanism design, the designer may want to choose from a family of games so that an equilibrium of the chosen game maximizes a given objective. As another example, an econometrician may be given data about agents' observed behavior and wants to estimate parameters of the underlying game. To address such problems, existing equilibrium computation approaches would be forced to solve many individual games from the family, which can be very costly. Our *parameterized symmetric games* are to our knowledge the first compact representation of parameterized families of games. Leveraging this representation, we provide a methodology for solving optimization problems over PSNE of a family of symmetric games, without having to solve each individual game in the family. These include the problem of finding parameters that ensure equilibria that are close to some observed configuration, and the problem of finding parameters that maximize the equilibrium payoff for the worst-off player.

The main challenge in constructing such polynomial-time algorithms is that the set of configurations (and, indeed, the set of PSNEs) can be exponential in the input size. Thus, approaches based on enumerating all configurations require exponential time. Furthermore, the constraints defining PSNE are not generally convex, so the problem cannot be formulated as an integer linear program in fixed dimension. Instead, our approach encodes the set of PSNEs in a compact form that has appealing computational properties. Specifically, we make use of the *rational generating function method* for representing and computing with sets of lattice points, due to Barvinok and Woods [4]. We model configurations as integer lattice points, and formulate the set of equilibrium configurations via operations on sets of 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 recent work coauthored by one of us [21]. This previous 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 actions are lattice points in fixed-dimensional polyhedra and the utilities are given by piecewise linear functions. These results assumed a fixed number of players and made strong restrictions on the piecewise linear functions used to represent utilities. The key conceptual difference between the current paper and [21] is that lattice points are used to represent different things. In [21] lattice points directly represent actions, making the result more direct but the model more restrictive (actions scale in a restricted way and players are fixed). In the current paper's setting of symmetric games, lattice points arise more naturally as configurations, and the number of players scales freely and instead we fix the number of actions. The difference in setting from [21] gives rise to novel technical challenges, requiring different proof techniques based on disjoint set decompositions rather than [21]'s approach using integer projections. Furthermore, computing social welfare maximizing PSNEs (Section 4) presents a novel difficulty because in our setting social welfare becomes a non-linear function. We overcome this difficulty by adapting an FPTAS result of [14].

2. SYMMETRIC GAMES

A strategic game is defined by the tuple $(n, \{A_i\}_{1\le i\le n}, \{U_i\}_{1\le i\le n})$ where n is the number of players, and for each player i, A_i is her set of actions and $U_i : \prod_j A_j \to \mathbb{Z}$ is her utility function. Throughout the paper we assume that utilities are integer-valued.[1]

Symmetric games are a class of strategic games in which each player has an identical set of actions A and for all permutation of players $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$,

$$U_i(a_1, \ldots, a_n) = U_{\pi(i)}(a_{\pi(1)}, \ldots, a_{\pi(n)}).$$

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 count of how many players take each action. A configuration is an m-dimensional vector $\mathbf{x} = (x_a : a \in A)$, where x_a is the number of players taking action a. Let D denote the set of configurations:

$$D = \left\{ \mathbf{x} \in \mathbb{Z}^m : \sum_{a\in A} x_a = n, x_a \ge 0 \text{ for all } a \in A \right\}. \quad (1)$$

[1]Since the set of PSNEs of a game is unchanged under scaling of its utilities by a constant, games with rational-valued utilities can be transformed to equivalent games with integer utilities.

The utility of a given player in a symmetric game depends only on the action played and the overall configuration. For each action $a \in A$, we have a function u_a defined over configurations in which at least one player takes action a. In particular, $u_a(\mathbf{x})$ is the utility of playing action a in configuration $\mathbf{x}$ (provided $x_a \geq 1$). The set of functions u_a for all $a \in A$ is sufficient for specifying the utilities of the game. For the rest of the paper we call these u_a the utility functions and will not refer to the functions U_i.

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 takes action a or the utility of a player playing action a is no less than the utility he would receive from unilaterally deviating to action a'. Symbolically, let N denote the set of PSNE in a symmetric game. Then $\mathbf{x}$ is an element of N if for all $a \in A$ either $x_a = 0$ or for all $a' \in A$, $u_a(\mathbf{x}) \geq u_{a'}(\mathbf{x}+\mathbf{e}_{a'}-\mathbf{e}_a)$, 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'. Similarly, for an integer $\epsilon \geq 0$, a configuration $\mathbf{x}$ is an ϵ-PSNE if for all $a \in A$ either $x_a = 0$ or for all $a' \in A$, $u_a(\mathbf{x}) + \epsilon \geq u_{a'}(\mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a)$. In other words, each player in an ϵ-PSNE can gain at most ϵ by deviating from her current strategy.

2.1 Symmetric games with utilities represented as circuits

We want the utility functions to be compactly represented and efficiently computable. One approach is to model utility functions as circuits, as was done in previous work for general games [30]. In this subsection, 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 $\mathbf{x}$ and output is a binary representation of the corresponding integer utility value. Since circuits can represent arbitrary integer functions of the inputs, this can represent all symmetric games with integer utilities. When utility functions can be specified as circuits with small numbers of gates, the representation size can be as small as $O(\log n)$.

We first consider the case with two actions, $A = \{1, 2\}$. In this case the game always has a PSNE. This follows from the fact that such a game can be formulated as a congestion game, which implies the existence of a PSNE [28]. (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.) The claim was also proven from first principles by [8].

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 for these games that illustrates the structure of the strategy space, and then show how this structure can be exploited for efficient computation.

LEMMA 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 a 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. □

This 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 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.

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. *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.*

The proof follows from a reduction from CIRCUITSAT, and is given in Appendix A.

2.2 Symmetric games with piecewise linear utilities

The hardness result of Theorem 3 suggests that the circuit symmetric game representation is too general to be computationally useful in the worst case: it is powerful enough to encode NP-hard problems. Thus, computation of PSNE is intractable even if the representation is compact.

In this subsection and the rest of the paper, we consider utility functions represented as piecewise-linear functions. Piecewise-linear functions have been widely used as an approximation of

arbitrary continuous functions. A recent example of their use in an economic context is [15], 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 continuous domains, 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 polytope. Thus the definition of piecewise linear functions can be naturally extended to this setting. In particular, we specify a set of 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 polytopes $\{P_{aj}\}_{j \in J_a}$ with index set J_a where the set of configurations D is partitioned by the integer points in these polytopes. In other words,

$$D = \biguplus_{j \in J_a} (P_{aj} \cap \mathbb{Z}^m),$$

where the notation $\biguplus$ signifies disjoint union. Each polytope $P_{aj} = \{\mathbf{x} \in \mathbb{R}^m : M_{aj}\mathbf{x} \leq \mathbf{b}_{aj}\}$ is given by an integer matrix M_{aj} and integer right-hand side vector $\mathbf{b}_{aj}$. Over each cell $P_{aj} \cap \mathbb{Z}^m$ there is an affine function $f_{aj}(\mathbf{x}) = \boldsymbol{\alpha}_{aj} \cdot \mathbf{x} + \beta_{aj}$ with $\boldsymbol{\alpha}_{aj} \in \mathbb{Z}^m$ and $\beta_{aj} \in \mathbb{Z}$, such that

$$u_a(\mathbf{x}) = f_{aj}(\mathbf{x}) \text{ for } \mathbf{x} \in P_{aj} \cap \mathbb{Z}^m. \tag{2}$$

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

We observe that a piecewise linear utility function can be represented as a circuit.[2] 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.

Even if the utility functions cannot be exactly represented as piecewise-linear functions with a small number of pieces, piecewise-linear functions can still be useful as approximations of such utility functions. If for all configurations $\mathbf{x}$ and all actions a the difference between $u_a(\mathbf{x})$ and its approximation $\hat{u}_a(\mathbf{x})$ is at most ϵ, then by a standard argument any Nash equilibrium of the approximate game is a 2ϵ-Nash equilibrium of the original game.

3. MAIN RESULT

Our paper's main positive result follows.

THEOREM 4 (INTUITIVE VERSION). *Consider a symmetric game with piecewise linear utilities given by a binary encoding of the number of players and of the utility functions. Then, when the number of actions m is fixed, there exists*

(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.

The proof of this theorem draws on the theory of rational generating functions for encoding lattice points sets in polyhedra. Below we state the results upon which our proof depends, but offer minimal motivation.[3]

3.1 Rational generating functions

Rational generating functions are a method for compactly representing exponential-cardinality sets of integer points (in our case, configurations), for refining them (to yield only equilibrium configurations) and for efficiently supporting the computational operations of counting and enumerating set members.

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

$$g(P \cap \mathbb{Z}^m; \boldsymbol{\xi}) = \sum_{\mathbf{x} \in P \cap \mathbb{Z}^m} \boldsymbol{\xi}^{\mathbf{x}} = \sum_{\mathbf{x} \in P \cap \mathbb{Z}^m} \xi_1^{x_1} \cdots \xi_n^{x_m}. \tag{3}$$

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

LEMMA 5 (**BARVINOK'S THEOREM** [3]). *Let P be a polytope in $\mathbb{R}^m$ and $S = P \cap \mathbb{Z}^m$ with generating function $g(S, \boldsymbol{\xi})$ given by* (3) *which encodes the lattice points inside P. Then, there exists an algorithm which computes an equivalent representation of the form*

$$g(S; \boldsymbol{\xi}) = \sum_{i \in I} \gamma_i \frac{\boldsymbol{\xi}^{\mathbf{c}_i}}{\prod_{k=1}^{m} (1 - \boldsymbol{\xi}^{\mathbf{d}_{ik}})}, \tag{4}$$

where I is a polynomial-size index set and all data are integer. A formula of this type 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) is m and thus fixed when the dimension is fixed. When a lattice point set S is expressed in the form (4) we refer to $g(S; \boldsymbol{\xi})$ as its *Barvinok encoding*.

A *Boolean combination* of the sets $S_1, \ldots, S_k$ 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.

LEMMA 6 (**BOOLEAN OPERATIONS LEMMA** [COR. 3.7 IN 4]). *Given fixed integers k and ℓ there exists a constant s and a polynomial-time algorithm for the following problem. Given as* input, *in binary encoding,*

(I_1) the dimension m and

(I_2) Barvinok encodings of k finite sets $S_p \subseteq \mathbb{Z}^m$, $g(S_p; \boldsymbol{\xi})$ such that for each rational term the number of binomials in the denominator is at most ℓ,

output, *in binary encoding,*

[2] Addition and multiplication 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.

[3] We provide a more detailed tutorial on rational generating functions at `http://cs.ubc.ca/~kevinlb/papers/gf_tutorial.pdf` for readers unfamiliar with the topic.

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

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

is a rational generating function of the finite set S that is the Boolean combination *of the sets $S_1, \ldots, S_k$, 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.*

Note that if the input sets $S_p \subseteq \mathbb{Z}^m$ are integer points inside polyhedra whose Barvinok encodings $g(S; \boldsymbol{\xi})$ arise from applying Lemma 5) then the condition that the number of binomials ℓ in the denominators are fixed follows under the assumption that the dimension m is fixed.

Disjoint unions are a special case of combining sets. To compute disjoint unions of sets we do not appeal to the Boolean Operations Lemma, and thus the number of sets k in the union may be polynomial in the input size instead of constant.

LEMMA 7 (**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 $S_1, \ldots, S_k$:*

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

where $\biguplus$ denotes disjoint union.

Once a rational generating function of a set S has been computed, various information can be extracted about S, including cardinality and enumeration.

LEMMA 8 (**COUNTING LEMMA** [3]). *Let the dimension m be a fixed constant. Given a lattice point set $S \in \mathbb{Z}^m$ input as its Barvinok encoding $g(S, \boldsymbol{\xi})$, there exists a polynomial-time algorithm for computing $|S|$.*

LEMMA 9 (**ENUMERATION LEMMA**). *Let the dimension m and the maximum number ℓ 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 $g(S, \boldsymbol{\xi})$ 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.*

3.2 Proof of Main Theorem

We now state our main theorem formally and prove it.

THEOREM 4. *Consider a symmetric game with piecewise linear utilities given by the following input:*

(I_1) the binary encoding of the number n of players;

(I_2) for each $a \in A$, the utility function $u_a(\mathbf{x})$ represented as the binary encoding of M_{aj}, $\mathbf{b}_{aj}$, $\boldsymbol{\alpha}_{aj}$ and β_{aj} for each $j \in J_a$.

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

(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. We first show that N, the set of PSNE, can be encoded as a short rational generating function. Let N denote the set of PSNE. A 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 Lemma 6. 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 together, we define the deviation set as

$$Dev(a, a', j, j') = \left\{ \begin{array}{l} \mathbf{x} \in D : x_a \geq 1, \mathbf{x} \in P_{aj}, \\ \mathbf{x}' = \mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a \in P_{a'j'} \\ f_{aj}(\mathbf{x}) \leq f_{a'j'}(\mathbf{x}') - 1\} \end{array} \right\}. \quad (5)$$

Now we can use (5) to describe N. We obtain

$$N = D \setminus \bigcup_{a,a'} \biguplus_{j} \biguplus_{j'} Dev(a, a', j, j'), \quad (6)$$

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 our claim that the second two unions are disjoint) can be verified as follows. Suppose configuration $\mathbf{x} \in D$ *is not* an element of the right-hand side of (6). This implies that $\mathbf{x}$ lies in some deviation set $D(a, a', j, j')$ 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 that $\mathbf{x}$ 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 (6).

Now we substantiate our claims that the second two unions are disjoint. 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 see that the union indexed by j' is disjoint, 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 Lemma 6. 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 Lemma 5.

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

$$Dev(a, a') = \biguplus_j \biguplus_{j'} Dev(a, a', j, j'). \quad (7)$$

$Dev(a, a')$ is a disjoint union, and so by Lemma 7 also admits a Barvinok encoding

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

We can use (7) to rewrite (6) 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, Lemma 6 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 Lemmas 8 and 9. Given that we can compute $g(N, \boldsymbol{\xi})$ in polynomial time, we can compute its cardinality in polynomial time by applying Lemma 8. This establishes (i). Applying Lemma 9 (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 Lemma 9 directly. □

4. SOCIAL-WELFARE MAXIMIZING EQUILIBRIA

When there are many equilibrium configurations in a symmetric game, one may ask which equilibrium is "best" according to some objective function. By applying Lemma 6 and a binary-search argument, we can optimize in polynomial time any linear objective over a set of lattice points in Barvinok encoding, and thus over the set of PSNE. This can be extended to piecewise-linear objectives by dealing with each piece separately. Such objective functions include for example the utility $u_a(\mathbf{x})$ of playing some action a.

A more interesting and natural objective function is the social welfare, which is the sum of all players' utilities. We are thus interested in finding an optimal solution to the optimization problem:

$$\max \left\{ \sum_{a \in A} x_a u_a(\mathbf{x}) : \mathbf{x} \in N \right\}. \quad (8)$$

Denote by $w(\mathbf{x})$ the objective function of this problem and let w^* denote its optimal value. Note that $w(\mathbf{x})$ is not piecewise-linear, but is instead piecewise polynomial. The main result of this section is an FPTAS for this optimization problem.

The result relies on the following lemma from [14] regarding optimizing polynomials over sets encoded as rational generating functions.

LEMMA 10 ([14]). *There exists a algorithm for the following problem. Given as* input *an*

(I_1) two vectors $\mathbf{x}_\mathrm{L}, \mathbf{x}_\mathrm{U} \in \mathbb{Z}^k$,

(I_2) a Barvinok encoding 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[4] encoding, representing a polynomial

$$f = \sum_i f_i \mathbf{x}^{\boldsymbol{\alpha}_i}$$

that is non-negative on S,

(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

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

This algorithm runs in polynomial time when the dimension n and the maximum number ℓ of binomials in the denominator of the Barvinok encoding of S are fixed.

We now state the main theorem of this subsection.

THEOREM 11. *Consider a symmetric game with the same input as in Theorem 4, and in addition a positive rational number $\frac{1}{\epsilon}$ given in unary encoding. Then, for a fixed number of actions m there exists a polynomial-time algorithm to output, in binary encoding, a configuration $\mathbf{x}_\epsilon \in D$ that satisfies $w(\mathbf{x}_\epsilon) \geq (1 - \epsilon) w^*$ where $w(\mathbf{x}) = \sum_{a \in A} x_a u_a(\mathbf{x})$ and $w^* = \max_{\mathbf{x} \in N} w(\mathbf{x})$.*

PROOF. 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 $J = \prod_{a \in A} J_a$. This yields the following fine partition of: $D = \bigcup_{\mathbf{j} \in J} P_\mathbf{j}$ 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

$$\max \left\{ \sum_{a \in A} x_a f_{aj_a}(\mathbf{x}) : \mathbf{x} \in N \cap P_\mathbf{j} \right\}. \quad (9)$$

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 $\sum_{a \in A} x_a u_a(\mathbf{x}^*_\mathbf{j})$ amongst all $\mathbf{j} \in J$. To find an ϵ-optimal solution to inner optimization problem (9) we apply Lemma 10 with the following input:

1. two vectors $\mathbf{x}_\mathrm{L} = (0, 0, \ldots, 0)$ and $\mathbf{x}_\mathrm{U} = (n, \ldots, n)$;
2. The data describing the Barvinok encoding of $N \cap P_\mathbf{j}$ given by the encoding of N from Lemma 4 and the Lemma 6;
3. The polynomial objective $\sum_{a \in A} x_a f_{aj_a}(\mathbf{x})$, where the coefficients are all 1 and either or quadratic exponents; and
4. $\frac{1}{\epsilon}$ in unary encoding

The result then follows. □

Using a similar argument, we can generalize Theorem 10 to construct an FPTAS for maximizing non-negative piecewise-polynomial functions. Such an FPTAS will be useful in Section 5.

[4]In this section we only consider polynomial functions of degree 2 and thus the unary encodings of the exponents are of constant size.

5. PARAMETERIZED SYMMETRIC GAMES

In this section we extend our approach to model families of symmetric games parameterized by integer parameters, and answer interesting questions about a 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.

We allow a fixed number of integer parameters that additively influence each piece of the utility functions, and furthermore allow the *number* of players to be considered as a parameter. The set of actions remains fixed in each member of the parametric family.

Our overall approach is to use rational generating functions as before, except that we augment the configuration vector with the vector of integer parameters. The resulting set we encode as a generating function is the graph of the parameters–equilibria correspondence.

5.1 Parametric families of symmetric games

Since we are considering parametric families of games in which 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*

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

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.

The utilities are specified by 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 that now the division is over F. We also introduce a fixed number of parameters that control the additive constants in the affine functions f_{aj}. We constrain these parameters to be the lattice points inside a polytope of fixed dimension $R \in \mathbb{R}^d$. Formally, let $\mathbf{p}$ be a d-dimensional integer vector inside of $Q = R \cap \mathbb{Z}^d$ and for each $a \in A$ and $j \in J_a$, define

$$f_{aj}(\mathbf{x}, \mathbf{p}) = \boldsymbol{\alpha}_{aj} \cdot \mathbf{x} + \boldsymbol{\beta}_{aj} \cdot \mathbf{p}, \tag{10}$$

where now

$$u_a(\mathbf{x}, \mathbf{p}) = f_{aj}(\mathbf{x}, \mathbf{p}) \text{ for } \mathbf{p} \in Q \text{ and } \mathbf{x} \in P_{aj}. \tag{11}$$

We are interested in encoding the set of *parameterized PSNE* defined as

$$N = \{(\mathbf{x}, n, \mathbf{p}) : \mathbf{p} \in Q, n \in \{0 \ldots B\}, \mathbf{x} \in N(n, \mathbf{p})\}, \tag{12}$$

where $N(n, \mathbf{p})$ is the set of PSNE in the symmetric game with n players, feasible configurations $D_n = \{\mathbf{x} \in F : \sum_{a \in A} x_a = n\}$ and utilities defined as in (11) for the given parameter $\mathbf{p}$.

The approach we use to encode N is similar to that of Section 3 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 the proof of Theorem 11. The expression of N as a Boolean combination of sets is

$$N = H \setminus \bigcup_{a,a'} \biguplus_{j} \biguplus_{j'} Dev(a, a', j, j'), \tag{13}$$

where $H = \{(\mathbf{x}, n, \mathbf{p}) : \mathbf{x} \in D_n, n \in \{0, \ldots, B\},\ \mathbf{p} \in Q\}$ is the overall feasible set, and 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 $Dev(a, a', j, j')$ is the set

$$\left\{ (\mathbf{x}, n, \mathbf{p}) : \begin{array}{l} \mathbf{p} \in Q,\ 0 \leq n \leq B, n \in \mathbb{Z} \\ \mathbf{x} \in D_n \cap P_{aj} \\ \mathbf{x}' = \mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a \in P_{a'j'} \\ f_{aj}(\mathbf{x}) \leq f_{a'j'}(\mathbf{x}') - 1 \} \end{array} \right\}. \tag{14}$$

It is clear that H and each $Dev(a, a', j, j')$ is a polytopal lattice point set in $(\mathbf{x}, n, \mathbf{p})$ and thus a Barvinok encoding of N obtains by applying similar reasoning to that found in Theorem 4.

We have thus established the following.

THEOREM 12. *Consider a parametric family of symmetric games with m actions and n players with utility functions parameterized by $\mathbf{p} \in Q$ as found in* (11) *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 R contained in $\mathbb{R}^d$;*
- *(I_3) 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 $\boldsymbol{\alpha}_{aj} \in \mathbb{Z}^m$, 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 PSNE defined in (12) *has a Barvinok encoding, which can be computed in polynomial time when the number of actions m and the dimension of the parameter space d is fixed.*

5.2 Optimization over parameters

Once we have a Barvinok encoding of the set N of parameterized PSNE, we can use it to answer questions about game-theoretic properties of the parameterized family. For example, if we are interested in finding a value for the parameter such that a PSNE exists, it is just a matter of enumerating from the set N. Similarly we can optimize linear and polynomial objective functions of the parameters, subject to the constraint that a PSNE exists, by optimizing the objective function over the set N.

A more interesting case is when we want to find optimal parameters under some objective function that depends on the PSNE configuration as well as on the parameters themselves. Since each game in the family can have multiple PSNEs, the problem is not well defined until we specify how PSNEs are selected. Depending on the application domain, we might consider selecting the best-case PSNE for each game, or selecting the worst-case PSNE. For example in mechanism design problems, best-case analysis (e.g. recent work on the price of stability [2]) is useful for getting bounds on what we could achieve under equilibrium behavior; while worst-case analysis (e.g. recent work on the price of anarchy [22, 29]) is useful for designing mechanisms with guaranteed performance.

Depending on whether we want to select (e.g.) best-case PSNE or worst-case PSNE, different types of optimization problems arise. For example, if we want to minimize the function $f(\mathbf{x}, n, \mathbf{p})$, selecting the best-case PSNE yields the problem

$$\min\{f(\mathbf{x}, n, \mathbf{p}) : (\mathbf{x}, n, \mathbf{p}) \in N\}. \tag{15}$$

Using similar arguments as in Section 4, we have the following result:

THEOREM 13. *When f is piecewise linear or piecewise polynomial, problem* (15) *can be solved in polynomial time or approximated in polynomial time, respectively.*

We now turn to a trickier problem than (15), selecting the worst-case PSNE. This yields the problem

$$\min_{n,\mathbf{p}} \quad \max_{\mathbf{x}:(\mathbf{x},n,\mathbf{p})\in N} \quad f(\mathbf{x},n,\mathbf{p}). \tag{16}$$

We propose a branch-and-bound approach for solving the min-max optimization (16). Branch and bound is a general optimization method for nonconvex optimization problems, applicable to discrete domains as well as continuous domains [23, 24]. At a high level, the algorithm partitions the space of candidate solutions into regions and iteratively refines the partitioning by bisecting a region along one of its dimensions. The algorithm can prune off regions using bounds on the objective value for each of the regions.

For our problem, a "region" corresponds to the set of integer points in a polytope. The branch-and-bound approach requires that the bounds on the objective value can be efficiently computed for each region. For each region L, we can compute a lower bound on the value of any candidate solution $n, \mathbf{p}$ from the region by solving

$$\min\{f(\mathbf{x},n,\mathbf{p}) : (\mathbf{x},n,\mathbf{p}) \in N, (n,\mathbf{p}) \in L\}.$$

We can compute an upper bound by picking some $\hat{n}, \hat{\mathbf{p}}$ in L and solving the optimization $\max_{\mathbf{x}:(\mathbf{x},\hat{n},\hat{\mathbf{p}})\in N} f(\mathbf{x},\hat{n},\hat{\mathbf{p}})$. Finally, there is a wide choice on the order in which the regions are explored. For discrete domains, a simple and memory-efficient approach is to explore the regions in a depth-first order. The algorithm is outlined as follows.

1. Set of regions $\mathcal{L}$ is initialized with one region $\{(n,\mathbf{p}) : 0 \leq n \leq B, \mathbf{p} \in R\} \cap \mathbb{Z}^{d+1}$.
2. Global upper bound UB is initialized to ∞.
3. Repeat until $\mathcal{L}$ is empty:
   (a) Take the next region L from $\mathcal{L}$ in depth-first order.
   (b) If L contains only one integer point $(\hat{n},\hat{\mathbf{p}})$, solve $\hat{f} = \max_{\mathbf{x}:(\mathbf{x},\hat{n},\hat{\mathbf{p}})\in N} f(\mathbf{x},n,\mathbf{p})$. Update UB if $\hat{f} > UB$.
   (c) Otherwise (i.e., L contains more than one integer point), compute the lower bound LB for L as discussed above. If $LB \geq UB$ discard L. Otherwise bisect L along its longest dimension, and replace L in $\mathcal{L}$ by the resulting two regions.
4. Return UB as the optimal value.

The worse-case running times for branch-and-bound algorithms are exponential in general. Nevertheless, when the algorithm can prune off large regions, it can be much faster than the brute-force approach of trying every parameter instantiation. For branch-and-bound to be effective, upper- and lower-bounds for each region need to be computed in polynomial time. As discussed above, this is the case if f is piecewise linear.

THEOREM 14. *For piecewise-linear f, the branch-and-bound algorithm finds the optimum of* (16) *in finite time; furthermore, steps 3b and 3c of the algorithm can be computed in polynomial time.*

5.3 Fitting a "closest game" to an observed equilibrium

We now consider a problem from econometrics: determining parameters of a utility model that would give rise to observed equilibrium behavior. In particular, we show how to fit an undetermined parameter $\mathbf{p}$ to an observed configuration $\tilde{\mathbf{x}}$ in a game with n players.

Suppose we are analyzing a system modeled as an n-player symmetric game where certain aspects of the game's utility functions are unknown, and furthermore the set of candidate utility functions belongs to the set of parametric (in $\mathbf{p}$) piecewise-linear utilities of the form of (11). We are given the players' observed behavior in the game as some configuration $\tilde{\mathbf{x}}$. Assuming the players are behaving rationally, we would like to know the game's true utility functions, i.e., to estimate the parameter $\mathbf{p}$. Note that we are not parameterizing the number of players n, since given any observed configuration $\tilde{\mathbf{x}}$ we automatically determine the number of players n as $\sum_{a\in A} \tilde{x}_a$. Thus, for simplicity we use the notation $N(\mathbf{p})$ instead of $N(n,\mathbf{p})$ to denote the set of PSNEs with parameter $\mathbf{p}$.

Assuming the players play a PSNE if one exists, we would like to find a $\mathbf{p} \in Q$ such that $\tilde{x}$ is a PSNE of the game with parameter $\mathbf{p}$. Formally, the set of such $\mathbf{p}$ is $N_{\tilde{\mathbf{x}}} = \{\mathbf{p} : \mathbf{p} \in Q, (\tilde{\mathbf{x}},\mathbf{p}) \in N\}$. A Barvinok encoding of $N_{\tilde{\mathbf{x}}}$ can be computed efficiently using a similar construction as in Section 5.1.

However, there might not be such a $\mathbf{p}$, i.e., the set $N_{\tilde{\mathbf{x}}}$ might be empty. In this case we would like to find $\mathbf{p}$ such that $\tilde{\mathbf{x}}$ is close to being an equilibrium. One measure of closeness to being an equilibrium is ϵ-equilibrium. Define $\tilde{N}$ to be the set of $(\mathbf{x},\mathbf{p},\epsilon)$ such that $\mathbf{x}$ is an ϵ-equilibrium of the game with parameter $\mathbf{p}$. We can straightforwardly compute a Barvinok encoding of $\tilde{N}$ using a similar construction as in Section 5.1, with ϵ as an additional parameter. Then finding the best parameter $\mathbf{p}$ amounts to solving the optimization

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

From our discussion at the outset of Section 4 regarding optimizing linear functions over sets with Barvinok encodings, we observe that this can be solved in polynomial time (with a fixed number of actions) since the objective function is linear.

Instead of using ϵ-equilibria, we could try to minimize the distance between $\tilde{\mathbf{x}}$ and the set $N(\mathbf{p})$ of PSNE given $\mathbf{p}$, which is defined as the infimum of distances between $\tilde{\mathbf{x}}$ and points in the set $N(\mathbf{p})$. This yields the optimization problem

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

where d is the distance metric. De Loera et al. [14] analyzed optimization problems of this form. Here we derive some basic results. If d is the ℓ_1 distance, i.e., $d(\mathbf{x},\tilde{\mathbf{x}}) = \sum_a |x_a - \tilde{x}_a|$, the problem can be formulated as

$$\begin{aligned} \min \quad & \sum_a d_a \\ \text{s.t.} \quad & (\mathbf{x},\mathbf{p}) \in N, \\ & x_a - \tilde{x}_a \leq d_a, \forall a \in A, \\ & \tilde{x}_a - x_a \leq d_a, \forall a \in A, \end{aligned}$$

which has a linear objective. Since a Barvinok encoding of the feasible set of the above problem can be computed in polynomial time using a similar construction as in Section 5.1, the problem can be solved in polynomial time. A similar argument shows that the version of the problem with the ℓ_∞ distance can be solved in polynomial time. If d is the ℓ_q distance for $1 < q < \infty$, the problem can be formulated as one with a polynomial objective using similar techniques as above, and thus can be approximated in polynomial time using Theorem 10. See [14] for a more detailed discussion which identifies a more general class of computation-friendly distance metrics.

Another scenario is when the set $N_{\tilde{\mathbf{x}}}$ has multiple points, and we would like to find the parameter $\mathbf{p}$ with the "best fit". The distance

between $\tilde{x}$ and $N(\mathbf{p})$ is 0 for $\mathbf{p} \in N_{\tilde{\mathbf{x}}}$, and is thus not useful here. One approach is to find $\mathbf{p}$ such that $\tilde{x}$ is close to all points in $N(\mathbf{p})$. In other words, we choose the parameter $\mathbf{p} \in N_{\tilde{\mathbf{x}}}$ that minimizes the largest distance from $\tilde{\mathbf{x}}$ to any equilibrium in $\mathbf{x} \in N(\mathbf{p})$. Formally, we need to solve

$$\min_{\mathbf{p} \in N_{\tilde{\mathbf{x}}}} \max_{\mathbf{x}\,\text{s.t.}\,(\mathbf{x},\mathbf{p}) \in N} d(\mathbf{x}, \tilde{\mathbf{x}}). \tag{17}$$

This is an instance of the min-max optimization problem (16). When d is the ℓ_1 distance or the ℓ_∞ distance, $d(\mathbf{x}, \tilde{\mathbf{x}})$ is piecewise linear in x, in which case the problem (17) can be solved using the branch-and-bound algorithm as discussed in Section 5.2. To avoid having to partition the set $N_{\tilde{\mathbf{x}}}$ (which can have a complex shape), we can instead let $\mathbf{p}$ range from Q, and modify Step 3c of the algorithm so that whenever we have a region $L \subset Q$ such that $L \cap N_{\tilde{\mathbf{x}}} = \emptyset$, we set the lower bound for L to be ∞.

5.4 Finding parameters with good equilibrium payoffs

A common goal in mechanism design is to set the parameters of the game so as to achieve some desired properties in the resulting equilibria. Here we consider a mechanism designer choosing the parameters n and $\mathbf{p}$ such that the resulting symmetric game has equilibria with good payoffs. There are several ways to measure the "goodness" of an equilibrium. Suppose the objective we are interested in is the payoff of the worst-off player given an equilibrium $\mathbf{x}$, i.e., $\min_{a:x_a>0} u_a(\mathbf{x}, \mathbf{p})$. As discussed in Section 5.2, we need to specify how PSNE are selected for each game. If we pick the best-case PSNE, this yields the optimization problem

$$\max_{(\mathbf{x},n,p) \in N} \min_{a:x_a>0} u_a(\mathbf{x}, \mathbf{p}). \tag{18}$$

THEOREM 15. *The optimization problem* (18) *can be solved in polynomial time.*

PROOF. The problem can be rewritten as

$$\begin{aligned} \max \quad & w \\ \text{s.t.} \quad & (\mathbf{x}, n, \mathbf{p}, w) \in S \end{aligned}$$

where S is the set

$$\left\{ (\mathbf{x}, n, \mathbf{p}, w) : \begin{array}{l} (\mathbf{x}, n, \mathbf{p}) \in N,\ u_{\min} \le w \le u_{\max} \\ \forall a \in A : w \le u_a(\mathbf{x}, \mathbf{p}) \text{ OR } x_a = 0 \end{array} \right\}.$$

$u_{\min}$ and $u_{\max}$ are the minimum and maximum utilities possible in the game (they can be computed in polynomial time in fixed dimension). We use a similar technique as in Section 5.1 to construct a Barvinok encoding of S in polynomial time. Therefore, problem (18) can be solved in polynomial time. □

If we instead pick the worst-case PSNE, we have the optimization problem

$$\max_{n,p} \min_{\mathbf{x}:(\mathbf{x},n,\mathbf{p}) \in N} \min_{a:x_a>0} u_a(\mathbf{x}, \mathbf{p}). \tag{19}$$

This is an instance of the problem (16), and we can apply the branch-and-bound algorithm as discussed in Section 5.2.

THEOREM 16. *Problem* (19) *can be solved by the branch-and-bound algorithm in worst-case exponential time. Furthermore, steps 3b and 3c of the algorithm (the computation of bounds) require polynomial time.*

PROOF. Here step 3c corresponds to the subproblem

$$\max_{(\mathbf{x},n,\mathbf{p}) \in N:(\mathbf{n},\mathbf{p}) \in L} \min_{a:x_a>0} u_a(\mathbf{x}, \mathbf{p}) \tag{20}$$

for each subregion L of $\{(n, \mathbf{p}) : 0 \le n \le B, n \in \mathbb{Z}, \mathbf{p} \in Q\}$, and step 3b corresponds to the subproblem

$$\min_{\mathbf{x}:(\mathbf{x},\tilde{n},\tilde{\mathbf{p}}) \in N} \min_{a:x_a>0} u_a(\mathbf{x}, \tilde{\mathbf{p}}) \tag{21}$$

for any given $\tilde{n}$ and $\tilde{\mathbf{p}}$. (20) has a similar form as the best-case problem (18) and can be solved using the same approach as above in polynomial time. To solve (21), we break the feasible set into regions: for each subset T of A, we have one region $D(T) = \{x \in D_{\tilde{n}} : x_a > 0 \text{ for all } a \in T, x_a = 0 \text{ for all } a \notin T\}$. There is a constant number of these regions and we solve the problem separately on each region. Formally, we rewrite the problem as

$$\min_{T \subseteq A} \min_{\mathbf{x} \in D(T):(\mathbf{x},\tilde{n},\tilde{\mathbf{p}}) \in N} \min_{a \in T} u_a(\mathbf{x}, \tilde{\mathbf{p}}).$$

We can now exchange the latter two min operators to get

$$\min_{T \subseteq A} \min_{a \in T} \min_{\mathbf{x} \in D(T):(\mathbf{x},\tilde{n},\tilde{\mathbf{p}}) \in N} u_a(\mathbf{x}, \tilde{\mathbf{p}}).$$

The innermost minimization can be solved in polynomial time since $u_a(\mathbf{x}, \tilde{\mathbf{p}})$ is piecewise-linear. Since the other minimizations are over sets of fixed size, this can be solved in polynomial time. □

6. REFERENCES

[1] C. Àlvarez, J. Gabarró, and M. Serna. Pure Nash equilibria in games with a large number of actions. In *MFCS: Proceedings of the International Symposium on Mathematical Foundations of Computer Science*, pages 95–106, 2005.

[2] E. Anshelevich, A. Dasgupta, J. Kleinberg, E. Tardos, T. Wexler, and T. Roughgarden. The price of stability for network design with fair cost allocation. In *FOCS: Proceedings of the Annual IEEE Symposium on Foundations of Computer Science*, volume 45, pages 295–305. IEEE COMPUTER SOCIETY PRESS, 2004.

[3] 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.

[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. Cheng, D. Reeves, Y. Vorobeychik, and M. Wellman. Notes on equilibria in symmetric games. In *AAMAS-04 Workshop on Game-Theoretic and Decision-Theoretic Agents*, 2004.

[9] C. Daskalakis, A. Fabrikant, and C. 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. Papadimitriou. Computing pure Nash equilibria via Markov random fields. In *ACM-EC: Proceedings of the ACM Conference on Electronic Commerce*, pages 91–99, 2006.

[12] C. Daskalakis and C. H. Papadimitriou. Three-player games are hard. In *ECCC: Proceedings of the Electronic Colloquium on Computational Complexity*, 2005, TR05-139.

[13] 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*, 2009.

[14] 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.

[15] 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.

[16] A. Fabrikant, C. Papadimitriou, and K. Talwar. The complexity of pure Nash equilibria. In *STOC: Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 604–612, 2004.

[17] G. Gottlob, G. Greco, and F. Scarcello. Pure Nash equilibria: Hard and easy games. *Journal of Artificial Intelligence Research*, 24:357–406, 2005.

[18] 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*, pages 79–85, 2007.

[19] 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.

[20] M. Kearns, M. Littman, and S. Singh. Graphical models for game theory. In *UAI: Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 253–260, 2001.

[21] M. Köppe, C. Ryan, and M. Queyranne. Rational Generating Functions and Integer Programming Games. *Arxiv preprint arXiv:0809.0689*, 2008.

[22] E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. *Computer Science Review*, 3(2):65–69, 2009.

[23] E. Lawler and D. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719, 1966.

[24] R. Moore. Global optimization to prescribed accuracy. *Computers & Mathematics with Applications*, 21(6-7):25–39, 1991.

[25] J. Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286–295, February 1951.

[26] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, 2007.

[27] C. H. Papadimitriou and T. Roughgarden. Computing correlated equilibria in multi-player games. *Journal of the ACM*, 55(3):1–29, 2008.

[28] R. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory*, 2:65–67, 1973.

[29] T. Roughgarden. *Selfish routing and the price of anarchy*. The MIT Press, 2005.

[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] K. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic and Logical Foundations*. Cambridge University Press, 2009.

APPENDIX

A. PROOF OF THEOREM 3

THEOREM 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. □