Theoretical Computer Science 410 (2009) 101–107

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Note

The complexity of power-index comparison☆

Piotr Faliszewski*, Lane Hemaspaandra

Department of Computer Science, University of Rochester, Rochester, NY 14627, United States

ARTICLE INFO

Article history:
Received 30 January 2008
Received in revised form 30 August 2008
Accepted 12 September 2008
Communicated by X. Deng

Keywords:
Power index
Closure property
Computational complexity
#P
Completeness

ABSTRACT

We study the complexity of the following problem: Given two weighted voting games G' and G'' that each contain a player p, in which of these games is p's power index value higher? We study this problem with respect to both the Shapley–Shubik power index and the Banzhaf power index. Our main result is that for both of these power indices the problem is complete for probabilistic polynomial time (i.e., is PP-complete). We apply our results to partially resolve some recently proposed problems regarding the complexity of weighted voting games. We also study the complexity of the raw Shapley–Shubik power index. Deng and Papadimitriou showed that the raw Shapley–Shubik power index is #P-metric-complete. We strengthen this by showing that the raw Shapley–Shubik power index is many–one complete for #P. And our strengthening cannot possibly be further improved to parsimonious completeness, since we observe that, in contrast with the raw Banzhaf power index, the raw Shapley–Shubik power index is not #P-parsimonious-complete.

1. Introduction

In an abstract, direct democracy, each member in a certain sense has equal potential for impact on the decisions that the society makes. However, in many practical decision-making scenarios it is reasonable to give up this noble idea and consider weighted voting instead. Here are a few motivating examples. In a country divided into districts it makes sense to give each district voting power proportional to its population (consider, e.g., the US House of Representatives or various decision making processes within the European Union). In fact, the power that various apportionment methods give to the US states in its House of Representatives has been studied in terms of how well it is proportional to the sizes of the states [9]. In a business setting, stockholders in a company might hope to have voting power proportional to the amount of stock they own. Within computer science, Dwork et al. [4] suggested building a meta search engine for the web via treating other search engines as voters in an election. It would only be natural to weigh the participating search engines with their (quantified in some way) quality. Naturally, one can provide many other examples.

The focus of this paper is on the computational complexity of the following issue: Given an individual and two weighted voting scenarios (in each of them our individual might have different weight and each scenario might involve different sets of voters with different weights), in which one of them is our individual more influential? (We provide a formal definition of this problem in Section 1.1.) This problem has a very natural motivation. For example, consider a company that wishes to join some business consortium and has a choice among several consortia (e.g., consider an airline deciding which airline alliance to join). It is natural to assume that within each consortium companies make decisions via weighted voting, with companies weighted, e.g., via their size or revenue or some combination thereof. In a political context, members of the European Union

☆ A preliminary version of this paper appears in the Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management, June 2008, Shanghai, China, LNCS, 5034.

* Corresponding address: Department of Computer Science, University of Rochester, 734 Computer Studies Bldg, Rochester, NY 14627, United States.
E-mail address: pfali@cs.rochester.edu (P. Faliszewski).
URLs: http://www.cs.rochester.edu/u/pfali/ (P. Faliszewski), http://www.cs.rochester.edu/u/lane/ (L. Hemaspaandra).

0304-3975/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2008.09.034

sometimes try to promote new schemes of distributing vote weights among EU members. It is important for the countries involved to see which scheme is better for them. One can easily give many other applications of the issue we study.

Formally, we model the above problem via comparing the values of power index functions – in our case those of Shapley and Shubik [15] and of Banzhaf ([1], see also [6]) – of a particular player within two given weighted voting games. Our main result is that this problem is PP-complete for both the Shapley–Shubik power index and the Banzhaf power index. Let us now define our problem formally.

1.1. The power-index comparison problem

We model weighted voting via so-called weighted voting games. An n-player weighted voting game is a sequence of n nonnegative integer weights, $w_1, \ldots, w_n$, together with a quota q. We denote it as $(w_1, \ldots, w_n; q)$. We refer to the player with weight w_i as the i'th player. Weighted voting games model the following scenario: The players are given a yes/no question (e.g., should we lower the taxes? should we buy out our competitors?) and each player either agrees (answers *yes*) or disagrees (answers *no*). If the total weight of the voters who agree is at least as high as the quota then the result of the game is *yes* and otherwise it is *no*.

Let G be a voting game $(w_1, \ldots, w_n; q)$. Any subset of $\{1, \ldots, n\}$ is a coalition in G. We say that a coalition S is successful if $\sum_{i \in S} w_i \geq q$. We define $\mathrm{succ}_G(S)$ to be 1 if S is a successful coalition for G and to be 0 otherwise.

Interestingly, the relation between the effective power of a player within a voting game and his or her weight is not as simple as one might think. Consider game $G = (8, 7, 2; 9)$, i.e., a game with quota $q = 9$ and three players with weights 8, 7, and 2, respectively. It is easy to see that in this game any coalition of at least two players is successful. In effect, each of the players can influence the final result of the game to exactly the same degree, regardless of the fact that their weights differ significantly. Thus when analyzing weighted voting games it is standard to measure players' power using, e.g., the Shapley–Shubik power index [15] or the Banzhaf power index [1,6].

In essence, these power indices measure the probability that, assuming some coalition formation model, our designated player is critical for the forming coalition. By *critical* we mean here that the coalition is successful with our designated player but is not successful without him or her.

Let $G = (w_1, \ldots, w_n; q)$ be a voting game, let i be a player in this game, and let $N = \{1, \ldots, n\}$ be the set of all players of G. The value of the Banzhaf power index of i in G is defined as $\mathrm{Banzhaf}(G, i) = \frac{\mathrm{Banzhaf}^*(G,i)}{2^{n-1}}$, where $\mathrm{Banzhaf}^*(G, i)$ is the *raw* version of the index, $\mathrm{Banzhaf}^*(G, i) = \sum_{S \subseteq N - \{i\}} (\mathrm{succ}_G(S \cup \{i\}) - \mathrm{succ}_G(S))$. The Shapley–Shubik power index of player i in game G is defined as $\mathrm{SS}(G, i) = \frac{\mathrm{SS}^*(G,i)}{n!}$, where $\mathrm{SS}^*(G, i)$ is the raw version of the index, $\mathrm{SS}^*(G, i) = \sum_{S \subseteq N - \{i\}} \|S\|!(n - \|S\| - 1)!(\mathrm{succ}_G(S \cup \{i\}) - \mathrm{succ}_G(S))$.

Intuitively, $\mathrm{Banzhaf}(G, i)$ gives the probability that a randomly chosen coalition of players in $N - \{i\}$ is not successful but would become successful had player i joined in. The intuition for the Shapley–Shubik index is that we count the proportion of permutations for which a given player is pivotal. Given a permutation π of $\{1, \ldots, n\}$, the $\pi(i)$'th player is pivotal if it holds that the coalition $\{\pi(1), \pi(2), \ldots, \pi(i)\}$ is successful and the coalition $\{\pi(1), \pi(2), \ldots, \pi(i-1)\}$ is not. This permutation-based intuition is motivated by the view of the successful-coalition formation as the process of players joining in in random order. Naturally, the first player that makes the coalition successful is crucial and so the idea is to measure power via counting how often our player-of-interest is pivotal.

The focus of this paper is on the computational complexity analysis of the following problem.

Definition 1.1. Let f be either the Shapley–Shubik or the Banzhaf power index. By $\mathrm{PowerCompare}_f$ we mean the problem where the input (G', G'', i) contains two weighted voting games, $G' = (w'_1, \ldots, w'_n, q')$ and $G'' = (w''_1, \ldots, w''_n, q'')$, and an integer i, $1 \leq i \leq n$, and where we ask whether $f(G', i) > f(G'', i)$.

Note that in the above definition we assume that both games have the same number of players. At first this might seem to be a weakness but it is easy to see that given two games with different numbers of players we can easily pad the smaller one with weight-0 players. On the other hand, the assumption that both games have the same number of players allows us to solve the problem via comparing the raw values of the index: The scaling factor for both games is the same and thus it does not affect the result of the comparison.

1.2. Computational complexity

We briefly review some notions and notations. We fix the alphabet $\Sigma = \{0, 1\}$, and we assume that all the problems we consider are encoded in a natural, efficient manner over Σ. By $|\cdot|$ we mean the length function. We assume $\langle \cdot, \cdot \rangle$ to be a standard, natural pairing function such that $|\langle x, y \rangle| = 2(|x| + |y|) + 2$.

The main result of this paper, Theorem 2.1, says that the power-index comparison problem is PP-complete both for the Shapley–Shubik power index and for the Banzhaf power index. The class PP, probabilistic polynomial time, was defined by Simon [14] and Gill [7]. A language $L \subseteq \Sigma^*$ belongs to PP if and only if there exists a polynomial p and a polynomial-time computable relation R such that $x \in L \iff \|\{w \in \Sigma^{p(|x|)} \mid R(x, w) \text{ holds}\}\| > 2^{p(|x|)-1}$. PP captures the set of languages having a probabilistic Turing machine that on precisely the elements of the set has strictly more than 50% probability of acceptance. Let us mention that PP is a very powerful class. For example, it is well-known that NP is a subset of PP (as are even various larger classes). Via Toda's Theorem [16], we know that $\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{PP}}$. That is, PP is at least as powerful as

polynomial-time hierarchy, give or take the flexibility of polynomial-time Turing reductions. Many other properties of PP have been established in the literature.

Let us now recall the definition of the class #P [17]. For each NP machine N (i.e., for each nondeterministic polynomial-time machine N), by $\#\mathrm{acc}_N(x)$ we mean the number of accepting computation paths of N running with input x. A function $f, f : \Sigma^* \to \mathbb{N}$, belongs to #P if and only if there is an NP machine N such that $(\forall x \in \Sigma^*)[f(x) = \#\mathrm{acc}_N(x)]$. #P is, in a very loose sense, a functional counterpart of PP. For example, $\mathrm{P}^{\#\mathrm{P}} = \mathrm{P}^{\mathrm{PP}}$ [2]. More typically, #P is described as the counting analogue of NP.

As is usual, we say that a language L is hard for a complexity class $\mathcal{C}$ if every language in $\mathcal{C}$ polynomial-time many–one reduces to L. If in addition L belongs to $\mathcal{C}$ then we say that L is $\mathcal{C}$-complete. A language A polynomial-time many–one reduces to a language B if there exists a polynomial-time computable function f such that for each string $x \in \Sigma^*$ it holds that $x \in A \iff f(x) \in B$. On the other hand, there is no one agreed-upon notion of completeness for function classes. For example, Valiant [17] in his seminal paper used Turing reductions but other people have preferred notions such as Krentel's metric reductions [10], Zankó's many–one reductions (for functions) [19], and Simon's [14] parsimonious reductions.

In the context of power index functions, Prasad and Kelly [13] (implicitly) showed that the (raw) Banzhaf power index is #P-parsimonious-complete and Deng and Papadimitriou [5] established that the (raw) Shapley–Shubik power index is #P-metric-complete (regarding the complexity analysis of power indices, we also mention the paper of Matsui and Matsui [11]). We now review parsimonious and metric reductions, as those underpin the notions of parsimonious-completeness and metric-completeness.

Definition 1.2. (i) [10] A function $f : \Sigma^* \to \mathbb{N}$ metric reduces to a function $g : \Sigma^* \to \mathbb{N}$ if there exist two polynomial-time computable functions, φ and ψ, such that $(\forall x \in \Sigma^*)[f(x) = \psi(x, g(\varphi(x)))]$.

(ii) [19] A function $f : \Sigma^* \to \mathbb{N}$ many–one reduces to a function $g : \Sigma^* \to \mathbb{N}$ if there exist two polynomial-time computable functions, φ and ψ, such that $(\forall x \in \Sigma^*)[f(x) = \psi(g(\varphi(x)))]$.[1]

(iii) [14] f parsimoniously reduces to g if there is a polynomial-time computable function φ such that $(\forall x \in \Sigma^*)[f(x) = g(\varphi(x))]$.

Note that (a) if f parsimoniously reduces to g, then f many–one reduces to g, and (b) if f many–one reduces to g, then f metric reduces to g. Given a function class $\mathcal{C}$, we say that a function f is $\mathcal{C}$-parsimonious-complete if $f \in \mathcal{C}$ and each function in $\mathcal{C}$ parsimonious reduces to f. $\mathcal{C}$-metric-completeness and $\mathcal{C}$-many–one-completeness are defined analogously. Typically, parsimonious-complete functions are easier to work with than functions that are merely metric-complete or many–one-complete. In particular, our proof of Theorem 2.11 is more involved than our proof of Theorem 2.4 because, as we note, the raw Shapley–Shubik power index is not parsimoniously complete.

2. Main results

Our main result, Theorem 2.1, says that the power index comparison problem is PP-complete. This section is devoted to building the infrastructure for Theorem 2.1's proof and giving that proof. We also show that the raw Shapley–Shubik power index is #P-many–one-complete but not #P-parsimonious-complete.

Theorem 2.1. *Let f be either the Banzhaf or the Shapley–Shubik power index. The problem* $\mathrm{PowerCompare}_f$ *is* PP*-complete.*

We start via showing PP-membership of a problem closely related to our $\mathrm{PowerCompare}_{\mathrm{Banzhaf}}$ and $\mathrm{PowerCompare}_{\mathrm{SS}}$ problems. Let f be a #P function and let $\mathrm{Compare}_f$ be the language $\{\langle x, y\rangle \mid x, y \in \Sigma^* \wedge f(x) > f(y)\}$. ($\mathrm{PowerCompare}_{\mathrm{Banzhaf}}$ and $\mathrm{PowerCompare}_{\mathrm{SS}}$ are essentially, up to a minor definitional issue, incarnations of $\mathrm{Compare}_f$ for appropriate functions f.)

Lemma 2.2. *Let f be a* #P *function. The language* $\mathrm{Compare}_f$ *is in* PP.

Proof. Let f be an arbitrary #P function and let N be an NP machine such that $f = \#\mathrm{acc}_N$. Without the loss of generality, we assume that there is a polynomial q such that for each input $x \in \Sigma^*$ all computation paths of N make exactly $q(|x|)$ binary nondeterministic choices. Thus each computation path of N on input x can be represented as a string w in $\Sigma^{q(|x|)}$.

In order to show that $\mathrm{Compare}_f$ is in PP we need to provide a polynomial-time computable relation R and a polynomial p such that for each string $z = \langle x, y\rangle$ it holds that: $z \in \mathrm{Compare}_f \iff \|\{w \in \Sigma^{p(|z|)} \mid R(z, w) \text{ holds}\}\| > 2^{p(|z|)-1}$. We now define such R and p. Let us fix two strings, x and y, and let $z = \langle x, y\rangle$ and $n = |z|$. We define $p(n) = q(n) + 1$ and, for each string $w = w_0 w_1 \ldots w_{p(n)-1} \in \Sigma^{p(n)}$, we define $R(z, w)$ as follows:

Case 1. If $w_0 = 0$ then $R(z, w)$ is *true* exactly if the string $w_1, \ldots, w_{q(|x|)}$ denotes an accepting computation path of N on x and the symbols $w_{q(|x|)+1}$ through $w_{p(n)-1}$ are all 0. $R(z, w)$ is false otherwise.

Case 2. If $w_0 = 1$ then $R(z, w)$ is *false* exactly if the string $w_1, \ldots, w_{q(|y|)}$ denotes an accepting computation path of N on y and the symbols $w_{q(|x|)+1}$ through $w_{p(n)-1}$ are all 0. $R(z, w)$ is true otherwise.

[1] Note that Zankó's many–one reduction is an analogue for functions of the standard many–one reduction notion for sets. To avoid confusion, we mention to the reader that the term "functional many–one reduction" (which we do not use here) is sometimes used in the literature [18] as a synonym for "parsimonious reductions".

Via analyzing the above two cases it is easy to see that there are exactly $f(x) + (2^{p(n)-1} - f(y)) = f(x) - f(y) + 2^{p(n)-1}$ strings $w \in \Sigma^{p(n)}$ for which $R(z, w)$ is true. This value is greater than $2^{p(n)-1}$ if and only if $f(x) > f(y)$. Thus the relation R and the polynomial p jointly witness that Compare_f belongs to PP. □

Lemma 2.2 gives an upper bound on the complexity of Compare_f (assuming that $f \in$ #P). We now prove a matching lower bound, PP-completeness, for the case that f is #P-parsimonious-complete.

Lemma 2.3. *Let f be a* #P*-parsimonious-complete function. The language* Compare_f *is* PP*-complete.*

Proof. Let f be a #P-parsimonious-complete function. Via Lemma 2.2 we know that Compare_f is in PP and thus to show PP-completeness it remains to show PP-hardness. We do so via reducing an arbitrary PP language L to Compare_f. Let L be an arbitrary PP language. By definition, there exists a polynomial-time relation R and a polynomial p such that for each string $x \in \Sigma^*$ it holds that $x \in L \iff \|\{y \in \Sigma^{p(|x|)} \mid R(x, y) \text{ holds}\}\| > 2^{p(|x|)-1}$. We define two functions, g_1 and g_2, such that $g_1(x) = \|\{y \in \Sigma^{p(|x|)} \mid R(x, y) \text{ holds}\}\|$ and $g_2(x) = 2^{p(|x|)-1}$. It is easy to see that both g_1 and g_2 are in #P. g_1 can be computed via an NP machine that on input x guesses a binary string y of length $p(|x|)$ and accepts if and only if $R(x, y)$ holds. g_2 can be computed via a machine that on input x guesses a binary string of length $2^{p(|x|)-1}$ and then accepts. Naturally, $x \in L$ if and only if $g_1(x) > g_2(x)$.

Since f is #P-parsimonious-complete, both g_1 and g_2 parsimoniously reduce to f. Let φ_1 be the reduction function for g_1 and let φ_2 be the reduction function for g_2. We have that for each string x it holds that $g_1(x) = f(\varphi_1(x))$ and $g_2(x) = f(\varphi_2(x))$.

Our reduction from L to Compare_f works as follows. On input x we output the string $z = \langle \varphi_1(x), \varphi_2(x) \rangle$. Clearly, this can be done in polynomial time. To show correctness it is enough to recall that $x \in L$ if and only if $g_1(x) > g_2(x)$, which is equivalent to testing whether z is in Compare_f. Since L was chosen as an arbitrary PP language, this proves PP-completeness. □

We are almost ready to show that $\text{PowerCompare}_{\text{Banzhaf}}$ is PP-complete. However, in order to do so, we need to justify the claim that the raw version of the Banzhaf power index is #P-parsimonious-complete. (This was shown implicitly in the work of Prasad and Kelly [13], but we feel that it is important to explicitly outline the proof.)

One of our important tools here (and later on) is the function #X3C. The input to the X3C problem is a set $B = \{b_1, \ldots, b_{3k}\}$ and a family $\mathcal{S} = \{S_1, \ldots, S_n\}$ of 3-element subsets of B. The X3C problem asks whether there exists a collection of exactly k sets in $\mathcal{S}$ whose union is B. #X3C$(B, \mathcal{S})$ is the number of solutions of the X3C instance $(B, \mathcal{S})$.

Hunt et al. [8] showed that #X3C is parsimonious complete for #P. This is very useful for us as the standard reduction from #X3C to #SubsetSum (see, e.g., [12, Theorem 9.10]; #SubsetSum is the function that accepts as input a vector of nonnegative integers $(s_1, \ldots, s_n; q)$ and returns the number of subsets of $\{s_1, \ldots, s_n\}$ that sum up to q) is parsimonious and Prasad and Kelly's reduction from #SubsetSum to Banzhaf* (the raw version of Banzhaf's power index) is parsimonious as well. Since Banzhaf* is in #P, Banzhaf* is #P-parsimonious-complete. Thus the following theorem is, essentially, a direct consequence of Lemma 2.3.

Theorem 2.4. $\text{PowerCompare}_{\text{Banzhaf}}$ *is* PP*-complete.*

Proof. The raw version of the Banzhaf power index is #P-parsimonious-complete and so, via Lemma 2.3, $\text{Compare}_{\text{Banzhaf}^*}$ is PP-complete. Via a slight misuse of notation, we can say that $\text{Compare}_{\text{Banzhaf}^*}$ accepts as input two weighted voting games, G' and G'', and two players, p' and p'', such that p' participates in G' and p'' participates in G'' and accepts if and only if $\text{Banzhaf}^*(G', p') > \text{Banzhaf}^*(G', p'')$. We give a reduction from $\text{Compare}_{\text{Banzhaf}^*}$ to $\text{PowerCompare}_{\text{Banzhaf}}$.

Let G', p' and G'', p'' be our input to the $\text{Compare}_{\text{Banzhaf}^*}$ problem. We can assume that G' and G'' have the same number of players. If G' and G'' do not have the same number of players then it is easy to see that the game with fewer players can be padded with players whose weight is equal to this game's quota value. Such a padding leaves the raw Banzhaf power index values of the game's original players unchanged. (The reason for this is that any coalition that includes any of the padding candidates is already winning and so none of the original players is critical to the success of the coalition, and so the coalition does not contribute to the original players' power index values.)

We form two games, K' and K'', that are identical to games G' and G'', respectively, except that K' lists player p' as first and G'' lists player p'' as first. Our reduction's output is $(K', K'', 1)$. Naturally, $\text{Banzhaf}(K', 1) > \text{Banzhaf}(K'', 1)$ if and only if $\text{Banzhaf}^*(G', p') > \text{Banzhaf}^*(G'', p'')$. Also, clearly, K' and K'' can be computed in polynomial time. Thus we have successfully reduced $\text{Compare}_{\text{Banzhaf}^*}$ to $\text{PowerCompare}_{\text{Banzhaf}}$. This shows PP-hardness of $\text{PowerCompare}_{\text{Banzhaf}}$. PP-membership of $\text{PowerCompare}_{\text{Banzhaf}}$ is, essentially, a simple consequence of Lemma 2.2. This completes the proof. □

Let us now focus on the computational complexity of the power index comparison problem for the case of Shapley–Shubik. It would be nice if the raw Shapley–Shubik power index were #P-parsimonious-complete. If that were the case then we could establish PP-completeness of $\text{PowerCompare}_{\text{SS}}$ in essentially the same way as we did for $\text{PowerCompare}_{\text{Banzhaf}}$. Thus it is natural to ask whether the Shapley–Shubik power index (i.e., its raw version) is #P-parsimonious-complete. Prasad and Kelly [13] at the end of their paper, after – in effect – showing #P-parsimonious-completeness of the raw Banzhaf power index (their Theorem 4), write: "Such a straightforward approach does not seem possible with the Shapley–Shubik [power index]". We reinforce their intuition by now proving that the raw Shapley–Shubik power index in fact is *not* #P-parsimonious-complete.

Theorem 2.5. *The raw Shapley–Shubik power index (i.e., SS*) is not* #P*-parsimonious-complete.*

Proof. For the sake of contradiction, let us assume that SS* is #P-parsimonious-complete. Thus for each natural number k there is a weighted voting game G and a player i within G such that $\mathrm{SS}^*(G, i) = k$. This is the case because the function $f(x) = x$ belongs to #P (we assume that the "output x" is an integer obtained via a standard bijection between Σ^* and $\mathbb{N}$) and if SS* is #P-parsimonious-complete then there has to be a parsimonious reduction from f to SS*.

Let G be an arbitrary voting game with $n \geq 4$ players and let i be a player in G. By definition, $\mathrm{SS}^*(G, i)$ is a sum of terms of the form $k!(n-k-1)!$, where k is some value in $\{0, \ldots, n-1\}$. Since $n \geq 4$, each such term is even and thus $\mathrm{SS}^*(G, i)$ is even. The raw Shapley–Shubik power index of any player in a game with at most 3 players is at most $3! = 6$ and thus there is no input on which SS* yields the value 7. This contradicts the assumption that SS* is #P-parsimonious-complete and completes the proof. □

So the well-known result of Deng and Papadimitriou [5] that the raw Shapley–Shubik power index is #P-metric-complete cannot be strengthened to #P-parsimonious-completeness. Theorem 2.5 prevents us from directly using Lemma 2.3 to show that $\mathrm{PowerCompare}_{\mathrm{SS}}$ is PP-complete. Nonetheless, via the following set of results not only do we establish that $\mathrm{PowerCompare}_{\mathrm{SS}}$ is PP-complete, but we also strengthen the result of Deng and Papadimitriou via showing that the raw Shapley–Shubik power index is #P-many–one-complete (i.e., is #P-complete w.r.t. Zankó's many–one reductions [19]).

To establish our results we need to be able to build X3C instances that satisfy certain properties. Fact 2.6 below lists three basic transformations that we use to enforce these properties.

Fact 2.6. *Let $(B, \mathcal{S})$ be an instance of* X3C *and let $b_1, b_2, \ldots, b_6$ be elements that do not belong to B. Let $B_1 = \{b_1, b_2, b_3\}$, $B_2 = \{b_4, b_5, b_6\}$, $B_3 = \{b_1, b_4, b_5\}$ and $B_4 = \{b_1, b_4, b_6\}$. The following transformations preserve the number of solutions of the input instance:*

(i) $g(B, S) = (B \cup B_1, \mathcal{S} \cup \{B_1\})$,
(ii) $h'(B, S) = (B \cup B_1 \cup B_2, \mathcal{S} \cup \{B_1, B_2, B_3\})$,
(iii) $h''(B, S) = (B \cup B_1 \cup B_2, \mathcal{S} \cup \{B_1, B_2, B_3, B_4\})$.

In the following lemma we use these transformations to, in some sense, normalize X3C instances.

Lemma 2.7. *There is a polynomial-time algorithm that given an* X3C *instance $X = (B, \mathcal{S})$ outputs instance $X'' = (B'', \mathcal{S}'')$ such that $\#\mathrm{X3C}(X'') = \#\mathrm{X3C}(X)$ and $\frac{\frac{1}{3}\|B''\|}{\|\mathcal{S}''\|} = \frac{2}{3}$.*

Proof. Let $X = (B, \mathcal{S})$ be our input X3C instance and let $3k = \|B\|$ and $m = \|\mathcal{S}\|$. Let g and h'' be the transformations as in Fact 2.6. The idea of our algorithm is to apply transformation g to X so many times as to achieve the $\frac{2}{3}$ ratio. Let t be some nonnegative integer and let $(B_t, \mathcal{S}_t) = g^{(t)}(B, \mathcal{S})$. We observe that $\frac{\frac{1}{3}\|B_t\|}{\|\mathcal{S}_t\|} = \frac{k+t}{m+t}$ and that if $t = 2m - 3k$ (assuming this value is nonnegative) then $\frac{k+t}{m+t} = \frac{2}{3}$.

Our algorithm works as follows. First, we form instance $X' = (B', \mathcal{S}')$ such that $2\|\mathcal{S}'\| - 3 \cdot \frac{1}{3}\|B'\| \geq 0$. If $2m - 3k \geq 0$ then we set $X' = X$ and otherwise we repeatedly apply transformation h'', until this condition is met. (It is easy to see that $\lceil \frac{3k-2m}{2} \rceil$ applications are sufficient.) Then we derive the instance X'' from X' via $2\|\mathcal{S}'\| - 3 \cdot \frac{1}{3}\|B'\|$ applications of g. That is, $X'' = g^{(2\|\mathcal{S}'\| - 3 \cdot \frac{1}{3}\|B'\|)}(X')$.

Naturally, the algorithm runs in polynomial time. The correctness follows via the observation in the first paragraph and the fact that transformations g and h'' preserve the number of solutions. □

Finally, we are ready to show that the raw Shapley–Shubik power index is #P-many–one-complete.

Theorem 2.8. *The raw Shapley–Shubik power index (i.e.,* SS***$)$ is* #P*-many–one-complete.*

Proof. The raw Shapley–Shubik power index is in #P and thus it remains to show that it is #P-many–one-hard. To do so, we give a many–one reduction from #X3C′ to SS*. #X3C′ is a restriction of #X3C to instances $X = (B, S)$ such that: (1) $\frac{\frac{1}{3}\|B\|}{\|\mathcal{S}\|} = \frac{2}{3}$. (2) If n is a nonnegative integer such that $\frac{1}{3}\|B\| = 2n$ and $\|\mathcal{S}\| = 3n$ then there is a nonnegative integer t such that $n = 4^t$. To see that the thus restricted #X3C function is #P-parsimonious-complete it is enough to consider Lemma 2.7 and transformation h' from Fact 2.6.

Let φ_s be the standard, parsimonious reduction from #X3C to #SubsetSum (see, e.g., [12, Theorem 9.10]). φ_s has the property that given an instance $(B, \mathcal{S})$, where $\|B\| = 3k$ and $\|S\| = m$, $\varphi_s(B, \mathcal{S})$ is an instance $(s_1, \ldots, s_m; q)$ of SubsetSum such that every subset of $\{s_1, \ldots, s_m\}$ that sums up to q has exactly k elements. Given such an instance $(s_1, \ldots, s_m; q)$, Deng and Papadimitriou [5, Theorem 9] observe that the raw Shapley–Shubik power index of the first player in game $(1, s_1, \ldots, s_m; q+1)$ is exactly $(m-k)!k! \cdot \#\mathrm{SubsetSum}(s_1, \ldots, s_n; q)$. Since φ_s is parsimonious, this value is equal to $(n-m)!m! \cdot \#\mathrm{X3C}(B, \mathcal{S})$.

We now provide functions φ and ψ that constitute a many–one reduction from #X3C′ to SS*. We need to ensure that for each #X3C′ instance X[2] it holds that $\#\mathrm{X3C}'(X) = \psi(\mathrm{SS}^*(\varphi(X)))$. We first describe how to compute φ and ψ and then explain why they have this property.

[2] We assume that the inputs to φ satisfy the requirements of being #X3C′ instances. We implicitly replace any instance that does not fulfill this requirement with a fixed instance that does satisfy it and that has no solutions.

Given #X3C′ instance X, we compute $\varphi(X)$ as follows: We compute SubsetSum instance $\varphi_s(X) = (s_1, \ldots, s_n; q)$ and output game $(1, s_1, \ldots, s_n; q+1)$. Function ψ is a little more involved. Define $r_1(n) = n!(2n)!$ and $r_2(n) = n!(2n)!2^{3n}$. Given a nonnegative integer x, we compute $\psi(x)$ using the following algorithm. If $x = 0$ then return 0. Otherwise, find the smallest nonnegative integer t such that $r_1(4^t) \leq x \leq r_2(4^t)$ and output $\lfloor \frac{x}{r_1(4^t)} \rfloor$. If there is no such t then return 0. Function $\psi(x)$ can be computed in polynomial time via computing $r_1(4^t)$ and $r_2(4^t)$ for successive values of t. It is easy to see that we only need to try $O(\log x)$ many t's and thus ψ is computable in polynomial time with respect to the binary representation of x.

Let us now show that indeed for any #X3C′ instance X it holds that $\#\text{X3C}'(X) = \psi(\text{SS}^*(\varphi(X)))$. Let $X = (B, S)$ be an arbitrary #X3C′ instance and let n be a nonnegative integer such that $\frac{1}{3}\|B\| = 2n$ and $\|\mathcal{S}\| = 3n$. (The existence of such an n is guaranteed via the fact that in any #X3C′ instance $\frac{\frac{1}{3}\|B\|}{\|\mathcal{S}\|} = \frac{2}{3}$.) Via the properties of φ_s and φ we see that $\text{SS}^*(\varphi(X)) = n!(2n)!\#\text{X3C}'(X) = r_1(n)\#\text{X3C}'(X)$. It is easy to see that $\#\text{X3C}'(X) \leq 2^{3n}$ and thus, assuming that $\#\text{X3C}'(X) \geq 1$, we have that $r_1(n) \leq \text{SS}^*(\varphi(X)) \leq r_2(n)$. Via routine calculation we see that for any positive integer n it holds that $r_1(4n) > r_2(n)$. Thus the intervals $[r_1(4^t), r_2(4^t)]$ are disjoint and given $\text{SS}^*(\varphi(X))$ as input, the function ψ correctly identifies the $r_1(n)$ factor and outputs the answer $\#\text{X3C}'(X)$. Clearly, ψ also works correctly when $\text{SS}^*(\varphi(X)) = 0$. □

Lemma 2.9. *There is a polynomial-time algorithm that given two* X3C *instances* $X = (B_x, \mathcal{S}_x)$ *and* $Y = (B_y, \mathcal{S}_y)$ *outputs two* X3C *instances* $X'' = (B''_x, \mathcal{S}''_x)$ *and* $Y'' = (B''_y, \mathcal{S}''_y)$ *such that* $\|B''_x\| = \|B''_y\|$, $\|\mathcal{S}''_x\| = \|\mathcal{S}''_y\|$, $\#\text{X3C}(X) = \#\text{X3C}(X'')$, *and* $\#\text{X3C}(Y) = \#\text{X3C}(Y'')$.

Proof. We first use the algorithm from Lemma 2.7 to derive instances $X' = (B'_x, \mathcal{S}'_x)$ and $Y' = (B'_y, \mathcal{S}'_y)$ such that $\#\text{X3C}(X) = \#\text{X3C}(X')$, $\#\text{X3C}(Y) = \#\text{X3C}(Y')$, $\frac{\frac{1}{3}\|B'_x\|}{\|\mathcal{S}'_x\|} = \frac{2}{3}$, and $\frac{\frac{1}{3}\|B'_y\|}{\|\mathcal{S}'_y\|} = \frac{2}{3}$. Without the loss of generality we can assume that $\|B'_x\| \leq \|B'_y\|$. We set $Y'' = Y'$ and derive X'' via repeatedly applying transformation h' from Fact 2.6 to X', until the condition of the theorem is met. □

In the next lemma and theorem we prove the PP-completeness of $\text{PowerCompare}_{\text{SS}}$.

Lemma 2.10. *Let* f *and* g *be two arbitrary* #P *functions. There exists a polynomial-time computable function* $\text{cmp}_{f,g}(x, y)$ *such that* $(\forall x, y \in \Sigma^*)[f(x) > g(y) \iff \text{cmp}_{f,g}(x, y) \in \text{PowerCompare}_{\text{SS}}]$.

Proof. Let f and g be as in the lemma and let x and y be two arbitrary strings. Since both f and g are in #P and #X3C is #P-parsimonious-complete, there exist functions φ_f and φ_g that compute parsimonious reductions from f to #X3C and from g to #X3C, respectively.[3]

Let $(B_x, \mathcal{S}_x) = \varphi_f(x)$ and $(B_y, \mathcal{S}_y) = \varphi_g(y)$. Via Lemma 2.9 (and through a slight abuse of notation) we ensure that $\|B_x\| = \|B_y\| = 3k$ and that $\|\mathcal{S}_x\| = \|\mathcal{S}_y\| = r$, where r and k are two nonnegative integers. Let φ be the reduction function from the proof of Theorem 2.8. (Note that in the proof of Theorem 2.8 we restricted φ to work only on instances of X3C that fulfill a special requirement. For the purpose of this proof we disregard this requirement.)

We now describe our function $\text{cmp}_{f,g}$. Given the instances $X = (B_x, \mathcal{S}_x)$ and $Y = (B_y, \mathcal{S}_y)$ we compute $G_x = \varphi(X)$ and $G_y = \varphi(Y)$. We define $\text{cmp}_{f,g}(x, y)$ to output $(G_x, G_y, 1)$. Via the properties of φ discussed in the proof of Theorem 2.8, it holds that $\text{SS}^*(G_x, 1) = (r-k)!k! \cdot \#\text{X3C}(B_x, \mathcal{S}_x) = (r-k)!k!f(x)$ and $\text{SS}^*(G_y, 1) = (r-k)!k! \cdot \#\text{X3C}(B_y, \mathcal{S}_y) = (r-k)!k!g(y)$. Thus $f(x) > f(y)$ if and only if $\text{SS}(G_x, 1) > \text{SS}(G_y, 1)$, and so it is clear that the function $\text{cmp}_{f,g}$ does what the theorem claims. Naturally, $\text{cmp}_{f,g}$ can be computed in polynomial time. □

Theorem 2.11. $\text{PowerCompare}_{\text{SS}}$ *is* PP*-complete.*

Proof. Via Lemma 2.2 it is easy to see that $\text{PowerCompare}_{\text{SS}}$ is in PP. Let h be some #P-parsimonious-complete function. PP-hardness of $\text{PowerCompare}_{\text{SS}}$ follows via a reduction from the PP-complete problem Compare_h (see Lemma 2.3). As a reduction we can use, e.g., the function $\text{cmp}_{h,h}$ from Lemma 2.10. This completes the proof. □

3. Conclusions and open problems

We have shown that the problem of deciding in which of the two given voting games our designated player has a higher power index value is PP-complete for both the Banzhaf and the Shapley–Shubik power indices. For the case of Banzhaf, we have used the fact that the raw Banzhaf power index is #P-parsimonious-complete. For the case of Shapley–Shubik, we have shown that the raw Shapley–Shubik power index is #P-many–one-complete but not #P-parsimonious-complete. Nonetheless, using the index's properties we were able to show the PP-completeness of $\text{PowerCompare}_{\text{SS}}$. We believe that these results are interesting and practically important. Below we mention one particular application. In the context of multiagent systems, the Shapley–Shubik power index is often used to distribute players' payoffs, i.e., each player's payoff is proportional to his or her power index value. Recently Bachrach and Elkind [3] asked about the exact complexity of the following problem: Given a weighted voting game $G = (w_1, w_2, \ldots, w_n; q)$, is it profitable for players 1 and 2 to join? That is, if $G' = (w_1 + w_2, w_3, \ldots, w_n; q)$, is it the case that $\text{SS}(G', 1) > \text{SS}(G, 1) + \text{SS}(G, 2)$. Using Lemma 2.10 and the fact that

[3] We assume that neither φ_f nor φ_g ever outputs a malformed instance of X3C. This property is easy to enforce via the following modification: Whenever either φ_f or φ_g is about to output a malformed instance, replace it with a fixed, correct one that has no solutions.

#P is closed under addition we can easily show that this problem reduces to PowerCompare_{SS} and thus is in PP. We believe that Bachrach and Elkind's problem is, in fact, PP-complete and that the techniques presented in this paper will lead to the proof of this fact. However, at this point the exact complexity of the problem remains open.

Acknowledgments

We are grateful to Edith Elkind and Jörg Rothe for helpful discussions on the topic of weighted voting games, to Jörg Rothe and Osamu Watanabe for hosting visits to Heinrich-Heine-Universität Düsseldorf and the Tokyo Institute of Technology during which this work was done in part, and to the anonymous referees for helpful comments. The first author was supported in part by grant NSF-CCF-0426761. The second author was supported in part by grant NSF-CCF-0426761, the Alexander von Humboldt Foundation's TransCoop program, and a Friedrich Wilhelm Bessel Research Award.

References

[1] J. Banzhaf, Weighted voting doesn't work: A mathematical analysis, Rutgers Law Review 19 (1965) 317–343.
[2] J. Balcázar, R. Book, U. Schöning, The polynomial-time hierarchy and sparse oracles, Journal of the ACM 33 (3) (1986) 603–617.
[3] Y. Bachrach, E. Elkind, Divide and conquer: False-name manipulations in weighted voting games, in: Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems, May 2008, pp. 975–982.
[4] C. Dwork, R. Kumar, M. Naor, D. Sivakumar, Rank aggregation methods for the web, in: Proceedings of the 10th International World Wide Web Conference, ACM Press, March 2001, pp. 613–622.
[5] X. Deng, C. Papadimitriou, On the complexity of comparative solution concepts, Mathematics of Operations Research 19 (2) (1994) 257–266.
[6] P. Dubey, L. Shapley, Mathematical properties of the Banzhaf power index, Mathematics of Operations Research 4 (2) (1979) 99–131.
[7] J. Gill, Computational complexity of probabilistic Turing machines, SIAM Journal on Computing 6 (4) (1977) 675–695.
[8] H. Hunt, M. Marathe, V. Radhakrishnan, R. Stearns, The complexity of planar counting problems, SIAM Journal on Computing 27 (4) (1998) 1142–1167.
[9] L. Hemaspaandra, K. Rajasethupathy, P. Sethupathy, M. Zimand, Power balance and apportionment algorithms for the United States Congress, ACM Journal of Experimental Algorithmics 3 (1) (1998) URL: http://www.jea.acm.org/1998/HemaspaandraPower.
[10] M. Krentel, The complexity of optimization problems, Journal of Computer and System Sciences 36 (3) (1988) 490–509.
[11] Y. Matsui, T. Matsui, NP-completeness for calculating power indices of weighted majority games, Theoretical Computer Science 263 (1–2) (2001) 305–310.
[12] C. Papadimitriou, Computational Complexity, Addison-Wesley, 1994.
[13] K. Prasad, J. Kelly, NP-completeness of some problems concerning voting games, International Journal of Game Theory 19 (1) (1990) 1–9.
[14] J. Simon, On some central problems in computational complexity, Ph.D. Thesis, Cornell University, Ithaca, NY, January 1975. Available as Cornell Department of Computer Science Technical Report TR75-224.
[15] L. Shapley, M. Shubik, A method of evaluating the distribution of power in a committee system, American Political Science Review 48 (1954) 787–792.
[16] S. Toda, PP is as hard as the polynomial-time hierarchy, SIAM Journal on Computing 20 (5) (1991) 865–877.
[17] L. Valiant, The complexity of computing the permanent, Theoretical Computer Science 8 (2) (1979) 189–201.
[18] H. Vollmer, On different reducibility notions for function classes, in: Proceedings of the 11th Annual Symposium on Theoretical Aspects of Computer Science, in: Lecture Notes in Computer Science, vol. 775, Springer-Verlag, February 1994, pp. 449–460.
[19] V. Zankó, #P-completeness via many-one reductions, International Journal of Foundations of Computer Science 2 (1) (1991) 76–82.