The Inverse Shapley Value Problem

Anindya De\textsuperscript{1,*}, Ilias Diakonikolas\textsuperscript{1,**}, and Rocco Servedio\textsuperscript{2,***}

\textsuperscript{1} UC Berkeley
{anindya, ilias}@cs.berkeley.edu
\textsuperscript{2} Columbia University
rocco@cs.columbia.edu

Abstract. For a weighted voting scheme used by \( n \) voters to choose between two candidates, the \( n \) Shapley-Shubik Indices (or Shapley values) of \( f \) provide a measure of how much control each voter can exert over the overall outcome of the vote. Shapley-Shubik indices were introduced by Lloyd Shapley and Martin Shubik in 1954 [SS54] and are widely studied in social choice theory as a measure of the “influence” of voters. The Inverse Shapley Value Problem is the problem of designing a weighted voting scheme which (approximately) achieves a desired input vector of values for the Shapley-Shubik indices. Despite much interest in this problem no provably correct and efficient algorithm was known prior to our work.

We give the first efficient algorithm with provable performance guarantees for the Inverse Shapley Value Problem. For any constant \( \epsilon > 0 \) our algorithm runs in fixed \( \text{poly}(n) \) time (the degree of the polynomial is independent of \( \epsilon \)) and has the following performance guarantee: given as input a vector of desired Shapley values, if any “reasonable” weighted voting scheme (roughly, one in which the threshold is not too skewed) approximately matches the desired vector of values to within some small error, then our algorithm explicitly outputs a weighted voting scheme that achieves this vector of Shapley values to within error \( \epsilon \). If there is a “reasonable” voting scheme in which all voting weights are integers at most \( \text{poly}(n) \) that approximately achieves the desired Shapley values, then our algorithm runs in time \( \text{poly}(n) \) and outputs a weighted voting scheme that achieves the target vector of Shapley values to within error \( \epsilon = n^{-1/8} \).

1 Introduction

In this paper we consider the common scenario in which each of \( n \) voters must cast a binary vote for or against some proposal. What is the best way to design such a voting scheme? If it is desired that each of the \( n \) voters should have the same “amount of

* Research supported by NSF award CCF-1118083.
** Research supported by a Simons Postdoctoral Fellowship.
*** Research supported in part by NSF awards CCF-0915929 and CCF-1115703.
\textsuperscript{1} Throughout the paper we consider only weighted voting schemes, in which the proposal passes if a weighted sum of yes-votes exceeds a predetermined threshold. Weighted voting schemes are predominant in voting theory and have been extensively studied for many years, see [EGGW07, ZFBE08] and references therein. In computer science language, we are dealing with linear threshold functions (henceforth abbreviated as LTFs) over \( n \) Boolean variables.
power” over the outcome, then a simple majority vote is the obvious solution. However, in many scenarios it may be the case that we would like to assign different levels of voting power to the \( n \) voters – perhaps they are shareholders who own different amounts of stock in a corporation, or representatives of differently sized populations. In such a setting it is much less obvious how to design the right voting scheme; indeed, it is far from obvious how to correctly quantify the notion of the “amount of power” that a voter has under a given fixed voting scheme. As a simple example, consider an election with three voters who have voting weights 49, 49 and 2, in which a total of 51 votes are required for the proposition to pass. While the disparity between voting weights may at first suggest that the two voters with 49 votes each have most of the “power,” any coalition of two voters is sufficient to pass the proposition and any single voter is insufficient, so the voting power of all three voters is in fact equal.

Many different power indices (methods of measuring the voting power of individuals under a given weighted voting scheme) have been proposed over the course of decades. These include the Banzhaf index \([\text{Ban65}]\), the Deegan-Packel index \([\text{DP78}]\), the Holler index \([\text{Hol82}]\), and others (see the extensive survey of de Keijzer \([\text{dK08}]\)). Perhaps the best known, and certainly the oldest, of these indices is the Shapley-Shubik index \([\text{SS54}]\), which is also known as the index of Shapley values (we shall henceforth refer to it as such). Informally, the Shapley value of a voter \( i \) among the \( n \) voters is the fraction of all \( n! \) orderings of the voters in which she “casts the pivotal vote” (see \([\text{Rot88}]\) for much more on Shapley values). We shall work with the Shapley values throughout this paper.

Given a particular weighted voting scheme (i.e. an \( n \)-variable linear threshold function), standard sampling-based approaches can be used to efficiently obtain highly accurate estimates of the \( n \) Shapley values (see also the works of \([\text{Lee03, BMR+10}]\)). However, the inverse problem is much more challenging: given a vector of \( n \) desired values for the Shapley values, how can one design a weighted voting scheme that (approximately) achieves these Shapley values? This problem, which we refer to as the Inverse Shapley Value Problem, is quite natural and has received considerable attention; various heuristics and exponential-time algorithms have been proposed, e.g. \([\text{APL07, FWJ08, dKKZ10, Kur11}]\), but prior to our work no provably correct and efficient algorithms were known.

**Our Results.** We give the first efficient algorithm with provable performance guarantees for the Inverse Shapley Value Problem. Our results apply to “reasonable” voting schemes; roughly, we say that a weighted voting scheme is “reasonable” if fixing a tiny fraction of the voting weight does not already determine the outcome, i.e. if the threshold of the linear threshold function is not too extreme. This seems to be a plausible property for natural voting schemes. Roughly speaking, we show that if there is any reasonable weighted voting scheme that approximately achieves the desired input vector of Shapley values, then our algorithm finds such a weighted voting scheme. Our algorithm runs in fixed polynomial time in \( n \), the number of voters, for any constant error parameter \( \epsilon > 0 \). In a bit more detail, our first main theorem, stated informally, is as follows (see Section 5 for Theorem 3 which gives a precise theorem statement):

**Main Theorem (Arbitrary Weights, Informal Statement).** There is a poly\((n)\)-time algorithm with the following properties: The algorithm is given any constant accuracy