An Anytime Approximation Method for the Inverse Shapley Value Problem

Shaheen Fatima
Department of Computer Science, Loughborough University
Loughborough LE11 3TU, UK.
s.s.fatima@lboro.ac.uk

Michael Wooldridge
Department of Computer Science, University of Liverpool
Liverpool L69 3BX, UK.
mjw@csc.liv.ac.uk

Nicholas R. Jennings
School of Electronics and Computer Science
University of Southampton
Southampton SO17 1BJ, UK.
nrj@ecs.soton.ac.uk

ABSTRACT

Coalition formation is the process of bringing together two or more agents so as to achieve goals that individuals on their own cannot, or to achieve them more efficiently. Typically, in such situations, the agents have conflicting preferences over the set of possible joint goals. Thus, before the agents realize the benefits of cooperation, they must find a way of resolving these conflicts and reaching a consensus. In this context, cooperative game theory offers the *voting game* as a mechanism for agents to reach a consensus. It also offers the Shapley value as a way of measuring the *influence* or *power* a player has in determining the outcome of a voting game. Given this, the designer of a voting game wants to construct a game such that a player's Shapley value is equal to some desired value. This is called the *inverse* Shapley value problem. Solving this problem is necessary, for instance, to ensure fairness in the players' voting powers. However, from a computational perspective, finding a player's Shapley value for a given game is #P-complete. Consequently, the problem of verifying that a voting game does indeed yield the required powers to the agents is also #P-complete. Therefore, in order to overcome this problem we present a computationally efficient approximation algorithm for solving the inverse problem. This method is based on the technique of 'successive approximations'; it starts with some initial approximate solution and iteratively updates it such that after each iteration, the approximate gets closer to the required solution. This is an *anytime* algorithm and has time complexity polynomial in the number of players. We also analyze the performance of this method in terms of its approximation error and the rate of convergence of an initial solution to the required one. Specifically, we show that the former decreases after each iteration, and that the latter increases with the number of players and also with the initial approximation error.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent Systems

General Terms

Anytime algorithms, Experimentation

Keywords

Game-theory, Mechanism Design, Coalitions, Shapley Value.

Cite as: An Anytime Approximation Method for the Inverse Shapley Value Problem, Shaheen Fatima, Michael Wooldridge, and Nicholas R. Jennings, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May,12-16.,2008,Estoril,Portugal,pp.935-942.

1. INTRODUCTION

Coalition formation is the process of bringing together two or more agents so as to achieve goals that individuals on their own cannot, or to achieve them more efficiently [12]. However, before the agents can realize the benefits of such cooperation, they must solve two key problems. The first is to find a way of reaching consensus between the agents. The second is to decide how to split the gains from cooperation between the members of the coalition.

In more detail, the problem of reaching consensus arises in situations where there is more than one joint goal that a group of agents may achieve together. In such situations, the agents must decide which of these goals they will actually achieve. Now, this decision-making becomes difficult especially when different agents have different preferences over the joint goals. Moreover, the agents must also decide how to share the gains from cooperation between themselves. To tackle both these problems, we turn to cooperative game theory [12]. Specifically, it provides the *voting game* as a mechanism for modeling situations that require a group of players to reach a consensus and the *Shapley value* as a way of dividing joint gains between the players.

In yet more detail, the voting game gives different amounts of *influence* (or *power*) over decision making to different players [12]. For example consider a joint stock company where each shareholder is given a number of votes (or shares) in proportion to his ownership of stock. Here, each share has equal influence, but different shareholders have different numbers of shares (in game theoretic terms, the number of shares an agent has is referred to as his *weight*) and hence different powers. Another example is federal political bodies such as the European Union council of ministers and the US Presidential Electoral College where the weights reflect populations, rather than contributions, and the individual votes are cast as blocks of different sizes. Although the voting mechanism has so far mostly been used in human contexts, it is now being increasingly studied in the context of multi-agent systems where the agents must reach consensus on what joint goals they will accomplish [15, 18, 4]. In what follows, we will analyze voting games in the abstract without reference to their different contexts.

A voting game can be analyzed from two different perspectives: from that of the individual players, and from that of the designer of a voting game. From the former perspective, the analysis deals with finding how much power a player has in changing the outcome of a voting game. Here an agent's power can be viewed as being analogous to his gains from cooperation; the more his power, the greater are his gains. To this end, cooperative game theory offers a number of ways of measuring a player's voting power including the Shapley value [17]. Thus, analyzing a game from a player's perspective involves taking the parameters of the game as input and finding

these power indices. In contrast, a game can also be analyzed from the *designer*'s perspective. This involves taking the power indices of the individual players as input and then finding a voting game that gives the required power to the players[1]. This paper is in this vein. Specifically, we focus on the Shapley value[2]. Our objective is to solve the *inverse* Shapley value problem (i.e., find a voting game that yields some required Shapley values). This problem is important because the players' voting powers are not immediately obvious from a casual inspection of a voting game (Section 2.2 illustrates this point with an example).

Our approach to solving the inverse Shapley value problem is as follows. We know that, for a given voting game, finding a player's Shapley value is a #P-complete problem [2]. Consequently, the problem of verifying that a voting game does indeed yield the required powers to the agents is also #P-complete. Therefore, in order to overcome this problem we present a computationally efficient approximation algorithm for solving the inverse problem. This method is based on the technique of 'successive approximations'; it starts with some initial approximate solution and iteratively updates it such that after each iteration, the approximate gets closer to the required solution. This is an *anytime* algorithm; the iterative process can be stopped after any number of iterations, but the more the number of iterations, the better the approximation. For a game of n players, the time complexity of a single iteration is $\mathcal{O}(n^2)$. We also analyze the performance of this method in terms of its approximation error and the rate of convergence of an initial solution to the required one. Specifically, we show that the former decreases after each iteration, and that the latter increases with the number of players and also with the initial approximation error.

In so doing, this paper presents the first such method for solving the inverse Shapley value problem. The inverse problem was investigated in the past, but for the weighted Shapley values of the form given in [3], not for the Shapley values as defined in [17] (see Section 5 for details). Moreover, this solution has time complexity exponential in the number of players. In contrast, this paper presents an algorithm for solving the inverse problem for the values defined by Shapley in [17]. Furthermore, the proposed method has two key advantages: it is an *anytime* algorithm, and it has time complexity polynomial in the number of players.

The rest of the paper is organised as follows. Section 2 provides the background to the Shapley value and voting games, and introduces the inverse Shapely value problem. In Section 3, we present our approximation method for solving the inverse problem. In Section 4, we provide an experimental analysis of our method in terms of its error of approximation and its rate of convergence. Section 5 discusses related literature, and Section 6 concludes.

2. BACKGROUND

We begin by introducing coalitional games and the Shapley value and then define the voting game. A a coalition game, $\langle N, v\rangle$, consists of:

1. a finite set ($N = \{1, 2, \ldots, n\}$) of players and
2. a function (v) that associates with every non-empty subset S of N (i.e., a *coalition*) a real number $v(S)$ (the worth of S).

For each coalition S, the number $v(S)$ is the total payoff that is available for division among the members of S (i.e., the set of joint actions that coalition S can take consists of all possible divisions of $v(S)$ among the members of S). Note that, viewed in this abstract way, a coalitional game gives no indication of how a coalition's value might or should be divided amongst coalition members.

In a voting game, the players will only join a coalition if they expect to gain from it. Here, the players are allowed to form binding agreements, and so there is often an incentive to work together to receive the largest total payoff. The problem then is how to split the total payoff between the players. In this context, Shapley [17] constructed a solution using an axiomatic approach. In particular, he defined a *value* for games to be a function that assigns to a game $\langle N, v\rangle$, a number $\varphi_i(N, v)$ for each i in N. This function satisfies three axioms [16]:

1. *Symmetry*: The names of the players play no role in determining the value. That is, two players who are identical with respect to what they contribute to a coalition should have the same Shapley value.
2. *Carrier*: The sum of $\varphi_i(N, v)$ for all players i in any carrier C equals $v(C)$. A carrier C is a subset of N such that $v(S) = v(S \cap C)$ for any subset of players $S \subset N$.
3. *Additivity*: This specifies how the values of different games must be related to one another. It requires that for any games $\langle N, v\rangle$ and $\langle N, v'\rangle$, $\varphi_i(N, v) + \varphi_i(N, v') = \varphi_i(N, v + v')$ for all i in N.

Shapley showed that there is a *unique* function that satisfies these three axioms. He also viewed this value as an index for measuring the power of players in a game. Like a price index or other market indices, the value uses averages (or weighted averages in some of its generalizations) to aggregate the power of players in their various cooperation opportunities. Alternatively, one can think of the Shapley value as a measure of the utility of risk neutral players in a game [16].

Having given these intuitions, we now turn to their formalization. Specifically, we first introduce some notation and then define the Shapley value. Let $\mathcal{S}$ denote the set $N - \{i\}$ and $f_i : \mathcal{S} \rightarrow 2^{N-\{i\}}$ be a random variable that takes its values in the set of all subsets of $N - \{i\}$, and has the probability distribution function (g) defined as:

$$g(f_i(\mathcal{S}) = S) \quad = \quad \frac{|S|!(n - |S| - 1)!}{n!}$$

The random variable f_i is interpreted as the random choice of a coalition that player i joins. Then, a player's Shapley value is defined in terms of its *marginal contribution*. The marginal contribution of player i to coalition S with $i \notin S$ is a function $\Delta_i v$ that is defined as follows:

$$\Delta_i v(S) = v(S \cup \{i\}) - v(S)$$

Thus a player's marginal contribution to a coalition S is the increase in the value of S as a result of i joining it. Given this, the Shapley value is defined as follows [17]:

DEFINITION 1. *The Shapley value (φ_i) of the game $\langle N, v\rangle$ for player i is the expectation (E) of its marginal contribution to a coalition that is chosen randomly:*

$$\varphi_i(N, v) = E[\Delta_i v \circ f_i] \tag{1}$$

The Shapley value may be interpreted as follows. Suppose that all the players are arranged in some order, all orderings being equally likely. Then $\varphi_i(N, v)$ is the *expected marginal contribution*, over all orderings, of player i to the set of players who precede him.

[1]This analysis may be used to ensure, for example, fairness in the players' voting powers.

[2]Future work will deal with the other indices.

Now, the method for finding a player's Shapley value depends on the definition of the value function (v). This function is different for different games, but here we focus specifically on the voting game because it is an important way of modeling situations where there are multiple agents that have different preferences and they want to reach a consensus.

2.1 The Voting Game

For a set N of players, the weighted voting game [12] is a game $G = \langle N, v \rangle$ in which:

$$v(S) = \begin{cases} 1 & \text{if } w(S) \geq q \\ 0 & \text{otherwise} \end{cases}$$

for some $q \in \mathbb{R}_+$ and $w_i \in \mathbb{R}_+$, where:

$$w(S) \quad = \quad \sum_{i \in S} w_i$$

for any coalition S. Thus w_i is the number of votes that player i has and q is the number of votes needed to win the game (i.e., the *quota*). A weighted voting game with quota q is denoted as $\langle q; w_1, \ldots, w_n \rangle$ and it is said to be a k-majority game if the quota for the game is $q = k \times \mu n$ where μ denotes the mean weight of the n players and $0 < k < 1$.

For these games, a player's marginal contribution is either zero or one because the value of any coalition is either zero or one. A coalition with value zero is called a "losing coalition" and with value one a "winning coalition". If a player's entry to a coalition changes it from losing to winning, then the player's marginal contribution for that coalition is one; otherwise it is zero. A coalition S is said to be a *swing* for player i if S is losing but $S \cup \{i\}$ is winning.

2.2 The Inverse Problem

A player's weight in a weighted voting game cannot be interpreted as a measure of his power (i.e., his ability to turn a losing coalition into a winning one) [13]. The following example illustrates this point.

EXAMPLE 1. *A company has three shareholders. Two of them have 49 percent of the shares and one of them has 2 percent. Then these figures (that represent the shareholders weights) do not describe the share of the power a shareholder has in running the company. This is because if the voting game requires 51 percent majority, (i.e., the game $\{51; 49, 49, 2\}$), then any two shareholders are sufficient to form a winning coalition. Thus each player has a Shapley value of 1/3 despite the disparity in their weights. Since any shareholder can win by joining together with any other, the one with 2 percent shares has exactly the same power as the ones with 49 percent shares.*

The above example illustrates that the relation between the players' weights and their values is not obvious. Given this, the designer of a voting game wants to know the weights that need to be given to the individual players such that they yield some desired Shapley values. This problem, of finding the players' weights given their Shapley values, is called the *inverse* problem.

As motivated in Section 1, we will now present a new method for solving the inverse problem. In more detail, for a game with n players and quota q, let $\Phi \in \mathbb{R}^n$ denote the required Shapley values. Given this, our method takes Φ and q as input and generates a vector $w \in \mathbb{R}^n$ such that the Shapley values ($\varphi \in \mathbb{R}^n$) for the game $\langle q; w_1, \ldots, w_n \rangle$ are approximately equal to Φ. Here, μ is the mean of the n weights in w.

The method we propose works as follows. We start with an initial allocation of weights (w). For w, we compute the Shapley values. Then, using a suitable rule (described in Section 3) to update the weights, we iteratively update the players' weights in w such that their Shapley values converge to those required. Now, as mentioned earlier, finding the Shapley values for a voting game is a #P-complete problem. Therefore, in order to overcome this problem, we use the approximation algorithm proposed in [5] to compute the Shapley values. This algorithm has time complexity linear in the number of players. For this method, we first prove a key property that shows how the players' Shapley values change when we update the weights in w. Then we use this property to formulate a rule for updating the weights in w such that the Shapley values for w converge to Φ. In the following section, we briefly describe the approximation method proposed in [5] to find the approximate Shapley values. Then, in Section 3, we present our method for solving the inverse problem.

2.3 Finding an approximate Shapley value

We use the method proposed in [5] to find a player's approximate Shapley value. In this section we give a brief overview of this method. The intuition behind it is as follows As per Definition 1, in order to find a player's Shapley value, we first need to find his marginal contribution to all possible coalitions. For n players, there are 2^{n-1} possible coalitions. Finding a player's marginal contribution to each one of these possible coalitions is computationally infeasible. So, instead of finding the marginal contribution to each possible coalition, the method samples random coalitions from the set of players. There are n sample coalitions that are drawn from N. The first sample coalition is of size one, the second sample is of size two, and so on. Let player i's approximate marginal contribution to sample $Z \subseteq N$ of z players be denoted $E\Delta_i^z$. Then, i's marginal contribution to each of the n samples (i.e., for $1 \leq z \leq n$) is found and the average of all these marginal contributions gives the player's approximate Shapley value.

Now, the approximate marginal contribution of player i to a random sample of size z is one if the total weight of the z players in the sample is greater than or equal to $a = q - w_i$ but less than $b = q - \epsilon$ (where ϵ is an infinitesimally small quantity). Otherwise, his marginal contribution is zero. This approximate marginal contribution is determined using the following rule from sampling theory.

Let the players' weights in N be defined by a probability distribution function. Irrespective of the actual probability distribution function, let μ be the mean weight for the set of players and ν be the variance in the players' weights. From this set (N) if we randomly draw a sample, then the sum of the players' weights in the sample is given by the following rule [6]:

> **Rule A**. If $w_1, w_2, \ldots, w_z$ is a random sample of size z drawn from *any* distribution with mean μ and variance ν, then the sample sum has an *approximate Normal distribution*, $\mathcal{N}$, with mean $z\mu$ and variance $\frac{\nu}{z}$ (the larger the z the better the approximation[3]).

Given this rule, player i's approximate marginal contribution to a random coalition is the area under the curve defined by $\mathcal{N}(z\mu, \frac{\nu}{z})$ in the interval $[a, b]$. This area is shown as region B in Figure 1 (the dotted line in the figure is the mean $z\mu$). Hence i's approximate

[3]Also, for large z, any measurement done on a sample drawn with replacement is the same as that for a sample drawn without replacement [6].

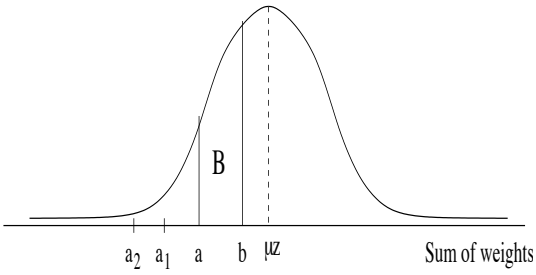

Figure 1: A normal distribution for the sum of players' weights in a coalition of size z.

marginal contribution to Z is:

$$E\Delta_i^z = \frac{1}{\sqrt{(2\pi\nu/z)}} \int_a^b e^{-z\frac{(x-z\mu)^2}{2\nu}} dx. \quad (2)$$

Player i's approximate Shapley value (φ_i) is the average of its aproximate marginal contribution to all possible coalitions:

$$\varphi_i = \frac{1}{n} \sum_{z=1}^{n} E\Delta_i^z \quad (3)$$

This method is described in Algorithm 1. Note that this method only requires the number of players, the quota, the mean weights, and the variance in the weights – the weights of the individual players are not needed. Also, note that Algorithm 1 is based on the normal approximation given in Rule A; the larger the n, the better the approximation. The time complexity of Algorithm 1 is $\mathcal{O}(n)$.

Algorithm 1 ShapleyValue(n, q, μ, ν, w_i)

1: $T_i \Leftarrow 0$; $a \Leftarrow q - w_i$; $b \Leftarrow q - \epsilon$
2: **for** z=1 to n **do**
3: $\quad E\Delta_i^z \Leftarrow \frac{1}{\sqrt{2\Pi\nu/z}} \int_a^b e^{-z\frac{(x-z\mu)^2}{2\nu}} dx$
4: $\quad T_i \Leftarrow T_i + E\Delta_i^z$
5: **end for**
6: $\varphi_i \Leftarrow T_i/n$
7: **return** φ_i

3. SOLVING THE INVERSE PROBLEM

We first analyze Algorithm1, and prove a key property of this method. This is done in Theorem 1. We then use this property to solve the inverse Shapley value problem.

THEOREM 1. *For a set N of n players, let $G_1 = \{q; w_1, \ldots, w_n\}$ and $G_2 = \{q; \bar{w}_1, \bar{w}_2, \ldots, \bar{w}_n\}$ be two voting games such that if the mean and variance of the weights for G_1 are μ and ν respectively, then the mean and variance for G_2 are μ and $\bar{\nu}$ where $\nu \approx \bar{\nu}$. Also, for $1 \leq i \leq n$, let φ_i^1 (φ_i^2) denote player i's Shapley value for G_1 (G_2). Then, if $\bar{w}_i > w_i$, then $\varphi_i^2 \geq \varphi_i^1$. Also, if $\bar{w}_i < w_i$, then $\varphi_i^2 \leq \varphi_i^1$. Lastly, if $\bar{w}_i = w_i$, then $\varphi_i^2 = \varphi_i^1$ – the Shapley values being computed using Algorithm 1.*

PROOF. *We need to find the relation between φ_i^1 and φ_i^2. Both φ_i^1 and φ_i^2 are computed using Equation 3. This equation, in turn, depends on Equation 2 which is used to find i's approximate marginal contribution to a random coalition. Recall that Z is a random coalition of z players drawn from N. Let i's approximate marginal contribution to a random coalition (Z) of $1 \leq z \leq n$ players be $E_1\Delta_i^z$ for G_1 and $E_2\Delta_i^z$ that for G_2. Also let μ_z^1 and ν_z^1 denote the mean and variance of the sum of the weights of the coalition Z for G_1. Likewise, let μ_z^2 and ν_z^2 denote the mean and variance for G_2. Then, from Rule A, we have:*

$$\mu_z^1 = z\mu \quad \text{and} \quad \mu_z^2 = z\mu \quad (4)$$

$$\nu_z^1 = \nu/z \quad \text{and} \quad \nu_z^2 = \bar{\nu}/z \quad (5)$$

Since we are given that $\nu \approx \bar{\nu}$, Equation 4 can be re-written as:

$$\nu_z^1 = \nu/u \quad \text{and} \quad \nu_z^2 \approx \nu/z \quad (6)$$

Let $\bar{w}_i = w_i + \Delta_w$. Also, let a_1, b_1 denote the limits of integration in Equation 2 for G_1 and a_2, b_2 those for G_2. Then, from Step 1 of Algorithm 1, we get the following:

$$a_1 = q - w_i \quad \text{and} \quad b_1 = q - \epsilon \quad (7)$$

$$a_2 = q - w_i - \Delta_w \quad \text{and} \quad b_2 = q - \epsilon \quad (8)$$

We know that $q > 0$ and $w_i > 0$. Hence, the relation between a_1 and a_2 depends on whether Δ_w is positive (Case 1), negative (Case 2), or zero (Case 3). We analyze each case in turn.

Consider Case 1. For this case, we have $q > 0$, $w_i > 0$, and $\Delta_w > 0$. Therefore we have:

$$a_1 > a_2 \quad \text{and} \quad b_1 = b_2 \quad (9)$$

Note that the relation $a_1 > a_2$ is true irrespective of the actual position of a_1 on the x-axis of Figure 1. Also, note that the relations between a_1, a_2 and b_1, b_2 do not depend on X, the sample size.

Now, as per Equation 2, $E_1\Delta_i^z$ is the area under the normal distribution $\mathcal{N}(z\mu_z^1, \frac{\nu_z^1}{z})$ between the limits a_1 and $b = b_1 = b_2$ (see Figure 1), i.e., :

$$E_1\Delta_i^z = \frac{1}{\sqrt{(2\pi\nu_z^1/z)}} \int_{a_1}^{b_1} e^{-z\frac{(x-z\mu_z^1)^2}{2\nu_z^1}} dx. \quad (10)$$

Likewise, we get:

$$E_2\Delta_i^z = \frac{1}{\sqrt{(2\pi\nu_z^2/z)}} \int_{a_2}^{b_2} e^{-z\frac{(x-z\mu_z^2)^2}{2\nu_z^2}} dx. \quad (11)$$

From Equation 4 we have $\mu_z^1 = \mu_z^2$, from Equation 6 we have $\nu_z^1 \approx \nu_z^2$, and form Equation 9 we have $a_1 > a_2$ and $b_1 = b_2$. Thus substituting μ_z^2 with μ_z^1, ν_z^2 with ν_z^1, and b_2 with b_1 in Equation 11 and comparing it with Equation 10 we get $E_2\Delta_i^z \geq E_1\Delta_i^z$. Given this and Equation 3, we get $\varphi_i^2 \geq \varphi_i^1$. Thus, i's Shapley value for G_2 is no less than his value for G_1.

In the same way, for Case 2 (i.e., $\Delta_w < 0$) and Case 3 (i.e., $\Delta_w = 0$) we get $\varphi_i^2 \leq \varphi_i^1$ and $\varphi_i^2 = \varphi_i^1$ respectively. □

As mentioned previously, we use Theorem 1 to solve the inverse Shapley value problem as follows. The method starts with an initial allocation of weights in w. For w, we compute the approximate Shapley values using Algorithm 1. Then, using a suitable updating rule (described below), we iteratively update the players' weights such that the Shapley values for w converge to those required. In more detail, this is done as follows.

Initial weights. Initially, we assign equal weights to all the players. Let ω denote this weight. This initial weight is chosen such that the weights of all the players remain positive even after they are updated. We first compute the Shapley values for w. We then update the weights in w. For the updated weights, we recompute the Shapley values (φ). This process is repeated until φ gets sufficiently close to Φ (i.e., φ converges to Φ).

Convergence. Let Δ_Φ denote the average percentage difference between Φ and φ for all the n players:

$$\Delta_\Phi \quad = \quad \sum_{i=1}^{n} 100 \times abs(\Phi_i - \varphi_i)/\Phi_i \tag{12}$$

When $\Delta_\Phi \leq \epsilon$, we say that φ has converged to Φ and stop the iterative processs.

Algorithm 2 InverseShapleyValue(n, q, Φ, δ, ω)

1: $w \Leftarrow \omega$
2: $\mu \Leftarrow$ Average(w); $\nu \Leftarrow$ Variance(w)
3: $\Delta_w \Leftarrow \delta \times \omega$
4: $\Delta_\Phi \Leftarrow 0$
5: **for** i=1 to n **do**
6: $\quad \varphi_i \Leftarrow$ ShapleyValue(n, q, μ, ν, w_i)
7: $\quad DV_i \Leftarrow \Phi_i - \varphi_i$
8: $\quad PE_i \Leftarrow$ abs$\left(\frac{DV_i \times 100}{\Phi_i}\right)$
9: $\quad \Delta_\Phi \Leftarrow \Delta_\Phi + PE_i$
10: **end for**
11: $\Delta_\Phi \Leftarrow \Delta_\Phi / n$
12: **while** $\Delta_\Phi > \epsilon$ **do**
13: $\quad$ UpdateWeights(w, φ, Φ, δ, PE, Rule)
14: $\quad \nu \Leftarrow$ Variance(w)
15: $\quad \Delta_w \Leftarrow \delta \times \omega$
16: $\quad \Delta_\Phi \Leftarrow 0$
17: $\quad$ **for** i=1 to n **do**
18: $\quad\quad \varphi_i \Leftarrow$ ShapleyValue(n, q, μ, ν, w_i)
19: $\quad\quad DV_i \Leftarrow \Phi_i - \varphi_i$
20: $\quad\quad PE_i \Leftarrow$ abs$\left(\frac{DV_i \times 100}{\Phi_i}\right)$
21: $\quad\quad \Delta_\Phi \Leftarrow \Delta_\Phi + PE_i$
22: $\quad$ **end for**
23: $\quad \Delta_\Phi \Leftarrow \Delta_\Phi / n$
24: **end while**
25: **return** w and Δ_Φ

Algorithm 3 UpdateWeights(w, φ, Φ, δ, PE, Rule)

1: $P \Leftarrow 0$; $N \Leftarrow 0$;
2: **for** i=1 to n **do**
3: $\quad$ **if** $(\Phi_i > \varphi_i)$ **then**
4: $\quad\quad P \Leftarrow P + 1; PEP_{P,1} \Leftarrow i; PEP_{P,2} \Leftarrow PE_i$
5: $\quad$ **else**
6: $\quad\quad$ **if** $(\Phi_i < \varphi_i)$ **then**
7: $\quad\quad\quad N \Leftarrow N + 1; PEN_{N,1} \Leftarrow i; PEP_{N,2} \Leftarrow PE_i$
8: $\quad\quad$ **end if**
9: $\quad$ **end if**
10: **end for**
11: Sort(PEP); Sort(PEN);
12: **if** Rule = Rule1 **then**
13: $\quad T \Leftarrow 1$
14: **end if**
15: **if** Rule = Rule2 **then**
16: $\quad T \Leftarrow$ Minimum(P, N)
17: **end if**
18: **for** i=1 to T **do**
19: $\quad j \Leftarrow PEP_{i,1}$; $k \Leftarrow PEN_{i,1}$
20: $\quad w_j \Leftarrow w_j + \Delta_w$; $w_k \Leftarrow w_k - \Delta_w$
21: **end for**
22: **return** w

Updating rule. We define two rules (Rule1 and Rule2) for updating the player's weights. Both rules are formulated in such a way that the conditions given in Theorem 1 are satisfied. That is, if G_1 represents the game before updating and G_2 that after updating, then the updating is done in such a way that the mean weight for both G_1 and G_2 is μ. Also, the variance in the weights for G_1 is approximately equal to the variance for G_2 (i.e., $\nu \approx \bar{\nu}$). Given this, we first introduce some notation and then describe the rules.

Assume that the n players are separated into two categories: those for whom the required Shapley values are more than their values from w, and those for whom the required Shapley values are less than their values from w. Let the details about the players in the former (latter) category be stored in PEP (PEN) where PEP (PEN) is a two dimensional array of P (N) rows and 2 (2) columns. The first column in PEP (PEN) gives a player's index in w, and the second column gives the absolute percentage difference between the player's required Shapley value and its value from w. Also, let the rows in PEP (PEN) be sorted in the decreasing order of their second column entries. Finally, let T denote the minimum of P and N. Given this, the two rules are defined as follows:

Rule1: Increment the weight of the first player in PEP by Δ_w, and decrement the weight of the first player in PEN by the same amount. Here Δ_w is chosen such that the variance in the weights for G_1 (i.e., before the update) is approximately equal to the variance in the weights for G_2 (i.e., after the update).

Rule2: Increment the weight of the first T players in PEP by Δ_w, and decrement the weight of the first T players in PEN by the same amount. Here Δ_w is chosen such that the variance in the weights for G_1 (i.e., before the update) is approximately equal to the variance in the weights for G_2 (i.e., after the update).

Note that both rules satisfy the conditions of Theorem 1; we increment the weights of one set of T players by Δ_w and decrement the weights of another disjoint set of T players by an equal amount. Thus the mean weight remains unchanged after updating. Also, as per the above defined rules, $\nu \approx \bar{\nu}$. Hence, from Theorem 1, we know that the Shapley values of the first T players in PEP are no less than their values before updating Also, the Shapley values of the first T players in PEN are no more than their values before updating. Finally, the Shapley values of those players whose weights are not updated remain unchanged.

The above method is described in Algorithm 2. Specifically, Step 1 of the algorithm initializes w to ω. In Step 2, we find the average (μ) and the variance (ν) for the weights in w. In Step 3, we find Δ_w as $\delta \times \omega$ where $0 < \delta < 1$ is an input parameter[4]. Step 4 initializes the average percentage difference (Δ_Φ) to zero. In the for loop of Step 5, for each player $1 \leq i \leq n$, we first use Algorithm 1 to find the approximate Shapley value (φ_i). Then we find PE_i which is the absolute percentage difference between Φ_i and φ_i. In Step 9, we find the sum of PE_i for all the n players. After exiting the for loop, in Step 11, we find the average absolute percentage difference (Δ_Φ). If ($\Delta_\Phi > \epsilon$), then we enter the while loop of Step 12. In this loop we update the players weights using Algorithm 3. Then, for the updated weights, we recompute the Shapley values and Δ_Φ. This is done in Steps 14 to 23 (which are the same as Steps 2 to 11). The while loop in Step 12 is repeated until Δ_Φ becomes smaller than ϵ. At this stage, Algorithm 2 returns the weights in w and the average percentage difference in Δ_Φ.

Algorithm 3 works as follows. In Step 1, we initialize P and N. Recall that P (N) denotes the number of players whose required Shapley values are more (less) than their values from w. In the for

[4]In Section 4.3, we show the effect of Δ_w on the performance of Algorithm 2

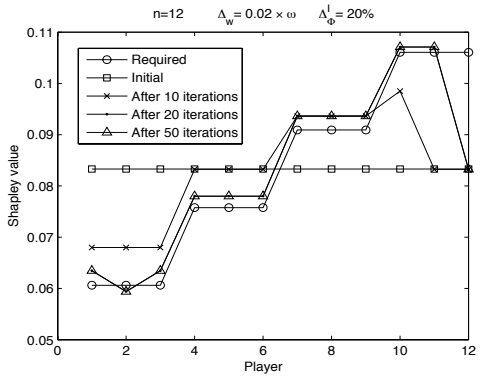

Figure 2: Convergence of the initial Shapley values to the required values.

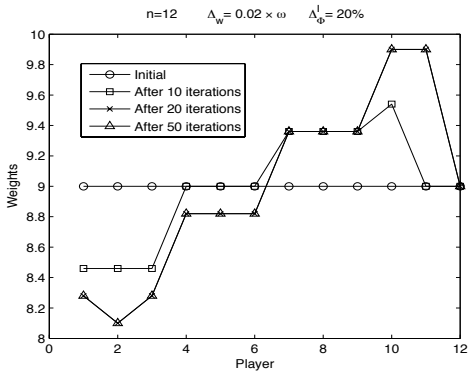

Figure 3: Variation in the players' weights with the number of iterations.

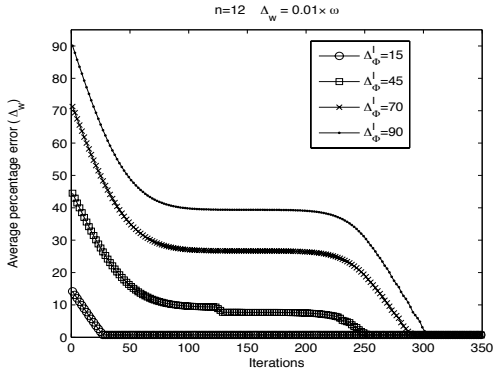

Figure 4: Effect of the initial error (Δ^I_Φ) on Δ_Φ.

loop in Step 2, we compare Φ_i with φ_i ($1 \leq i \leq n$). If ($\Phi_i > \varphi_i$), we store i in the first column of PEP and PE_i in the second column. On the other hand, if ($\Phi_i < \varphi_i$), we store i in the first column of PEN and PE_i in the second column. Step 11 sorts the rows of PEP and PEN in the decreasing order of their second column entries. Then, if the updating rule is Rule1, then in Step 13 we initialize T to 1. However, if the updating rule is Rule2, then, in Step 16, we initialize T to Minimum(P, N). Finally, in the for loop of Step 18, we increment (decrement) the weights of the first T players in PEP (PEN) by Δ_w.

As explained earlier, both Rule 1 and Rule 2 ensure convergence. Hence, Algorithm 2 is an *anytime* algorithm; it can be stopped after any number of iterations but the more the number of iterations, the smaller the error Δ_Φ.

We now analyze the time complexity of Algorithm 3 and then that of Algorithm 2. The time taken by the for loop in Step 2 of Algorithm 3 is $\mathcal{O}(n)$. The sorting in Step 11 takes $\mathcal{O}(nlogn)$ time. The time for the if statement of Step 12 is $\mathcal{O}(1)$, and that for the if statement of Step 15 is $\mathcal{O}(n)$. Finally, the time taken by the for loop of Step 18 is $\mathcal{O}(n)$. Hence the time complexity of Algorithm 3 is $\mathcal{O}(nlogn)$. Now consider Algorithm 2. The time complexity of the for loop of Step 5 is $\mathcal{O}(n^2)$. This is because the time complexity of finding the Shapley value (in Step 6 of Algorithm 2) using Algorithm 1 is $\mathcal{O}(n)$. Since the for loop of Step 5 is repeated n times, its complexity is $\mathcal{O}(n^2)$. Consequently, the time taken to execute Steps 1 to 11 is also $\mathcal{O}(n^2)$. We already know that them time taken by Step 13 (i.e., Algorithm 3) is $\mathcal{O}(nlogn)$. The time taken to execute Steps 14 to 23 is exactly the same as the time taken by Steps 2 to11 (i.e., $\mathcal{O}(n^2)$). Given this, the time taken by a single iteration of the while loop of Step 12 is $\mathcal{O}(n^2)$. Thus the time complexity of a single iteration of Algorithm 2 (by single iteration of Algorithm 2, we mean executing Steps 1 to 11 and then executing the while loop in Step 12 once) is $\mathcal{O}(n^2)$.

Since Algorithm 2 is an approximation, in addition to its time complexity, we also need to analyze its performance in terms of the approximation error Δ_Φ. We do this experimentally[5] in the following section.

4. EXPERIMENTAL ANALYSIS

In order to analyze the performance of Algorithm 2 in terms of its approximation error (Δ_Φ), we conducted experiments in a range of settings. The set up for our experiments is as follows. For a game of n players, we choose the initial weights ω as $\omega = 9$. We do this in order to ensure that for all our experiments, the weights of all the players are positive even after they are updated. Also, all the games we consider are two thirds majority games (i.e., $q = (2/3) \times n \times \omega$). For this set up, let Δ^I_Φ denote Δ_Φ for the initial weights (i.e., $w = \omega$). From Equation 12, we know that Δ_Φ depends on Δ^I_Φ, Δ_w, and n. Hence our objective is to study the effect of each of these three parameters on Δ_Φ. Before doing so, we first consider a sample voting game and show how the players' Shapley values in φ converge to those in Φ and how their weights in w vary with the number of iterations.

4.1 Convergence of φ to Φ for Rule1

For a game of $n = 12$ players, Figure 2 shows how φ converges to Φ for each player, and Figure 3 shows the players' weights at the end of each iteration. As seen in Figure 2, initially all the players have equal values (i.e., $\varphi_i = 1/12$ for $1 \leq i \leq 12$) because they all have equal weights (i.e., $w_i = 9$ for $1 \leq i \leq 12$). But after every iteration, the weights are updated such that the weight of a player whose value from w falls short of his required value is increased by Δ_w. Likewise, the weight of a player whose value from w is in excess of his required value, is decremented by Δ_w. For instance, consider player 8 in Figure 2. The initial Shapley value for this player is 0.0825 but his required value is 0.09. Thus in Figure 3, the weight of this player is increased from 9 to 9.3. On the other

[5]Future work will determine the bound on approximation error.

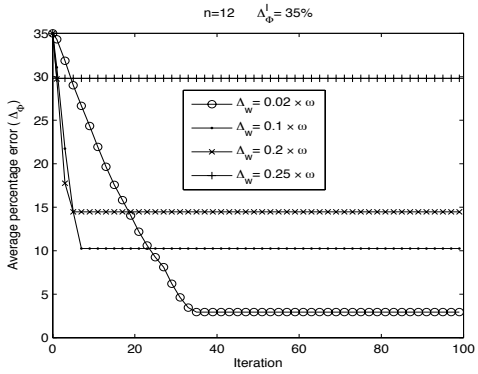

Figure 5: Effect of the weight increment (Δ_w) on Δ_Φ.

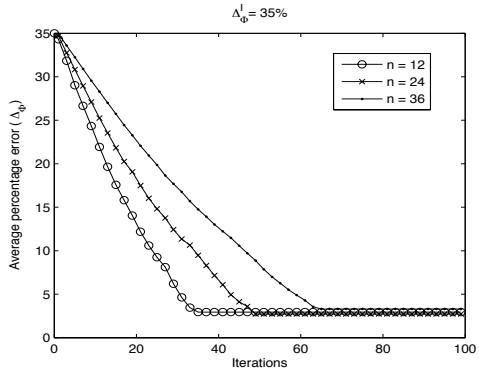

Figure 6: Effect of the number of players (n) on Δ_Φ.

hand, consider player 6. The initial Shapley value of this player is 0.0825 but his required value is 0.075. Thus in Figure 3, his weight is decreased from 9 to 8.8. Likewise for all other players.

4.2 Effect of Δ_Φ^I on Δ_Φ for Rule1

For a game of $n = 12$ players and $\Delta_w = 0.01 \times \omega$, Figure 4 shows how φ converges to Φ for $\Delta_\Phi^I = 15\%$, $\Delta_\Phi^I = 45\%$, $\Delta_\Phi^I = 70\%$, and $\Delta_\Phi^I = 90\%$. As can be seen, the number of iterations it takes for φ to converge to Φ increases with Δ_Φ^I. For an initial error of $\Delta_\Phi^I = 15\%$, the error Δ_Φ^I drops to 2% within 25 iterations. But for an initial error of $\Delta_\Phi^I = 45\%$, it takes 250 iterations for the error to drop to 2%, for $\Delta_\Phi^I = 70\%$ it takes 280 iterations, and for $\Delta_\Phi^I = 90\%$ it takes 300 iterations. Thus, as Δ_Φ^I increases, φ gets further away from Φ and so the number of iterations it takes for φ to converge to Φ also increases.

4.3 Effect of Δ_w on Δ_Φ for Rule1

For a game of $n = 12$ players and an initial error of $\Delta_\Phi^I = 35\%$, Figure 5 shows how Δ_Φ varies with the number of iterations. As shown in the figure, for $\Delta_w = 0.02 \times \omega$ the percentage error decreases from 35% to 2% in 35 iterations. For $\Delta_w = 0.1 \times \omega$, the error goes down to 10% in 7 iterations but remains constant after that. For $\Delta_w = 0.2 \times \omega$, after 6 iterations the error remains constant at 15%. Finally, for $\Delta_w = 0.25 \times \omega$, the error decreases to 30% in two iterations but remains constant after that. Thus, for small Δ_w, the weights are updated in small increments and so it takes longer for φ to converge to Φ but, in the end, φ is closer to Φ. However, as Δ_w increases, the weights are updated in bigger increments so initially, Δ_Φ decreases rapidly but it soon stops decreasing any further. Thus, the error (Δ_Φ), can be reduced by reducing Δ_w.

4.4 Effect of n on Δ_Φ for Rule1

For a game of n players and $\Delta_\Phi^I = 35\%$, Figure 6 shows how Δ_Φ varies with n. We vary n and for each n we find Δ_Φ. Here, for each n, we chose the weight increment (Δ_w) such that Δ_Φ is minimum. As shown in the figure, for $n = 12$, the error Δ_Φ decreases to 2% in 35 iterations. For $n = 24$, it takes 50 iterations, and for $n = 36$, it takes 65 iterations. This shows that, as n increases, it takes longer for φ to converge to Φ. This is because, for small n, incrementing/ decrementing a player's weight by Δ_w has a significant effect on not only his percentage error but also on the average percentage error of all the n players. However, as n increases, the effect of updating a single player's weight by Δ_w on the average percentage error (Δ_Φ) decreases. As a result, convergence takes longer.

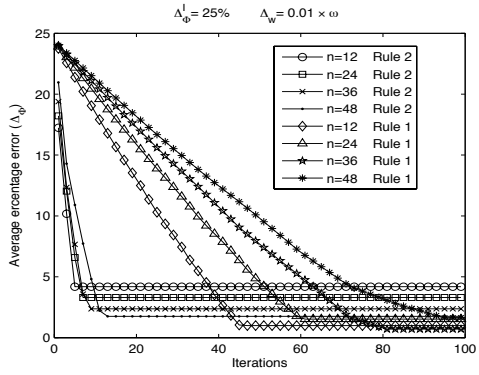

Figure 7: A comparison of convergence for Rule 1 and Rule 2.

4.5 A comparison of Rule1 and Rule2

We then conducted experiments to analyze the effect of Δ_Φ^I, Δ_w, and n on Δ_Φ, using Rule2. The results of these experiments were similar to those described in Sections 4.2, 4.3, and 4.4 for Rule1. However, a key difference in the performance of Rule1 and Rule2 was found to be in terms of the rate of convergence. Figure 7 compares the rate of convergence for the two rules. Here the initial percentage error is $\Delta_\Phi^I = 25\%$ and the weight increment is $\Delta_w = 0.01 \times \omega$. For this set up, we vary the number of players between $n = 12$ and $n = 48$, and for each n, we compare the average percentage errors (Δ_Φ) for Rule 1 and Rule 2 at the end of each iteration. As shown in the figure, for a game of 12 players, it takes 5 iterations for φ to converge to Φ using Rule 2. For the same number of players, it takes 45 iterations using Rule 1. Likewise for other values of n. Since Rule 2 updates the weights of $2T$ players, it reduces the percentage error for the $2T$ players in each iteration. In contrast, Rule1 updates the weights of only 2 players in each iteration. Hence the convergence of φ to Φ is faster for Rule2.

5. RELATED WORK

This section discusses the existing literature on methods for finding the Shapley value and also the literature on methods for solving the inverse Shapley value problem. For the Shapley value, two types of methods have been proposed: *exact* and *approximate*. These methods vary in their approach and computing requirements. Among the exact methods, we have [11, 1, 7, 8]. While they all give the exact Shapley value, they each have disadvantages. These include

requiring exponential time [11], a large memory space [11], or a specific representation[6] for the voting game [1, 7, 8].

In order to overcome the problem of computational complexity, approximation methods were developed. The earliest such method was proposed by Mann and Shapley [10]. This method is based on Monte Carlo simulation and estimates the Shapley value from a random sample of coalitions. The advantage of this method is its linear time complexity. However, its disadvantage is that it does not give details of how the samples are to be taken, which has a significant impact on the method's effectiveness. The multi-linear extension (MLE) approximation method proposed by Owen [14] uses randomization. Its advantage is that it has time complexity linear in the number of players. However, in practice, it only gives a good approximation for games with many small weights and no large weights. A modified MLE approximation method of [9] has a lower approximation error than the original version, but it has exponential time complexity. The method proposed in [5] uses randomization to find the approximate Shapley value. Like Owen's, this method too has linear time complexity. However, as mentioned before, Owen's method, only works well for those game with many small weights and no large weights [14]. In contrast, [5] works for all weights. Hence we use it to find an approximate Shapley value.

We now turn to the literature on solving the inverse problem. For coalition games in general, and the weighted voting game in particular, the Shapley value [17] provides a measure of a player's power. But as explained in Section 2.2, a player's weight in a voting game cannot be interpreted as his power. Thus the work in [3] uses a different weight scheme. For a coalition game $\langle N, v \rangle$, this scheme allows a player's weight to be interpreted as his power. However, for this new scheme, the function v is not the same as that described in Section 2.1. Also, for this new scheme, solving the inverse problem requires finding the function v for a coalition game $\langle N, v \rangle$ such that it generates some required weighted Shapley values. To this end, an algorithm was proposed in [3]. For a coalition game of n players, this algorithm has time complexity $\mathcal{O}(2^n log_2 2^n)$, i.e., exponential in the number of players. There are three key differences between this work and our's. First, the method we propose is for solving the inverse problem for the Shapley value as proposed in [17] and not for the weight scheme presented in [3]. Second, as shown in Section 3, the time complexity of our method is polynomial in the number of players, while that for [3] is exponential. Finally, unlike [3], we present an anytime solution.

6. CONCLUSIONS

This paper presents a computationally efficient approximation method for solving the inverse Shapley value problem. The method is based on the technique of 'successive approximations'. It starts with an initial assignment of weights to the players and then iteratively updates the weights such that the Shapley values after each iteration get closer to the required values. This is an *anytime* method; the iterative process can be stopped after any number of iterations, but the greater the number of iterations, the better the approximation. For a voting game of n players, the time complexity of a single iteration is $\mathcal{O}(n^2)$. The paper also presents an analysis of the performance of this method in terms of its approximation error and the rate of convergence of an initial solution to the required one. Specifically, it shows that the approximation error decreases after each iteration, and the rate of convergence decreases with the number of players and with the initial percentage error.

In future, we would like to generalise our method so that it works not just for the Shapley value but also for other power indices such as the Banzhaf index and the Coleman index. We would also like to generalize our method so that it solves the inverse problem not just for the voting game but also for other coalitional games. Finally, our present analysis focused on an experimental analysis of convergence. In future, we need to determine the bound for the approximation error for our method.

[6]Note that transforming a voting game into these specific forms requires additional computational time.

7. REFERENCES

[1] V. Conitzer and T. Sandholm. Computing Shapley values, manipulating value division schemes, and checking core membership in multi-issue domains. In *Proceedings of the National Conference on Artificial Intelligence*, pages 219–225, San Jose, California, 2004.

[2] X. Deng and C. H. Papadimitriou. On the complexity of cooperative solution concepts. *Mathematics of Operations Research*, 19(2):257–266, 1994.

[3] I. C. Dragan. The potential basis and the weighted Shapley value. *Libertas Mathematica*, XI:139–150, 1991.

[4] E. Elkind, L. A. Goldberg, P. Goldberg, and M. Wooldridge. Computational complexity of weighted threshold games. In *Proceedings of AAAI-2007*, pages 718–723, 2007.

[5] S. S. Fatima, M. Wooldridge, and N. R. Jennings. A randomized method for the Shapley value for the voting game. In *Proc. Sixth Int. Conference on Autonomous Agents and Multi-agent Systems*, pages 955–962, 2007.

[6] A. Francis. *Advanced Level Statistics*. Stanley Thornes Publishers, 1979.

[7] S. Ieong and Y. Shoham. Marginal contribution nets: A compact representation scheme for coalitional games. In *Proceedings of the Sixth ACM Conference on Electronic Commerce*, pages 193–202, Vancouver, Canada, 2005.

[8] S. Ieong and Y. Shoham. Multi-attribute coalition games. In *Proceedings of the Seventh ACM Conference on Electronic Commerce*, pages 170–179, Ann Arbor, Michigan, 2006.

[9] D. Leech. Computing power indices for large voting games. *Management Science*, 49(6):831–837, 2003.

[10] I. Mann and L. S. Shapley. Values for large games iv: Evaluating the electoral college by Monte Carlo techniques. Technical report, The RAND Corporation, Santa Monica, 1960.

[11] I. Mann and L. S. Shapley. Values for large games iv: Evaluating the electoral college exactly. Technical report, The RAND Corporation, Santa Monica, 1962.

[12] M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. The MIT Press, 1994.

[13] G. Owen. A note on the Shapley value. *Management Science*, 14:731–732, 1968.

[14] G. Owen. Multilinear extensions of games. *Management Science*, 18(5):64–79, 1972.

[15] J. S. Rosenschein and G. Zlotkin. *Rules of Encounter*. MIT Press, 1994.

[16] A. E. Roth. Introduction to the Shapley value. In A. E. Roth, editor, *The Shapley value*, pages 1–27. University of Cambridge Press, Cambridge, 1988.

[17] L. S. Shapley. A value for n person games. In A. E. Roth, editor, *The Shapley value*, pages 31–40. University of Cambridge Press, Cambridge, 1988.

[18] O. Shehory and S. Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence Journal*, 101(2):165–200, 1998.