1 Introduction

Today, machine learning models are used for decision making in various high-stake contexts such as criminal risk assessment, admission, credit scoring, etc. Since decisions made by these models greatly impact people’s lives, there has been a major push towards interpretability and explainability. Many notable works in model explanation and interpretability literature revolves around a notion of Shapley value. Despite its frequent usage in this area; quite a few recent works have shown how the concept of shapley value, a game-theoretic construct useful in analyzing individual contribution in a cooperative game may not always align with the requirements of a tool to analyze feature importance in a model.

In particular, have shown that how two models; a linear and one nonlinear model may have the same contribution score for all the features using a very popular algorithm called KernelSHAP and therefore conveys very little useful insight into how intervening on features would affect the model outcome, which is often one of the primary goals of explainability.

Kumar et al. explains this issue by introducing Shapley residual; a novel measure that captures the degree to which a game deviates from being inessential. One of the computational limitation of shapley residual is that it requires solving the full game vector prior to solving a linear system, which is a major bottleneck for models with large number of features since the game vector is exponential in the number of features.

In this work we make the following contributions.

• Provide an approximate test for checking inessentiality of a game avoiding residual computation.
• Give an estimation algorithm for computing a single entry of a residual vector for a game.
• Using the above provide a scheme for estimating the norm of a residual vector.

We introduce the required concepts in the next section and present the algorithms in section and

2 Background

Here we will introduce some preliminary concepts and definitions that will be necessary for illustration throughout the rest of the report. We begin with the definition of a game and its shapley value.

A cooperative game is defined by a set of players and a value function \( v : 2^d \rightarrow \mathbb{R} \) where \( 2^d \) denotes the power set of players. For any coalition (subset) of players \( v(S) \) represents the value of the game for \( S \in 2^d \). Traditionally, in literature it is assumed that \( v(\emptyset) = 0 \).
The Shapley value\[4\] of a cooperative game is denoted by \( \phi_i(v) \), \( \forall i \in [d] \). It is a unique value that satisfy the following properties.

- **Efficiency:** \( \sum_{i=1}^{d} \phi_i(v) = v([d]) \), i.e. the sum of the individual shapley value for each player is equal to the game value of the grand coalition.

- **Symmetry:** \( \phi_i(v) = \phi_j(v) \) for players \( i, j \) iff \( v(S \cup \{i\}) = v(S \cup \{j\}) \forall S \subseteq [d] \).

- **Linearity:** If \( u, v \) are the value function of two games of \( d \) players, then \( \phi_i(\alpha u + \beta v) = \alpha \phi_i(u) + \beta \phi_i(v) \) where \( \alpha, \beta \) are constants.

- **Dummy:** If \( v(S \cup \{i\}) = v(S) \forall S \subseteq [d] \setminus \{i\} \), then \( \phi_i(v) = 0 \)

Explicitly the shapley value of a player \( i \) can be computed by the formula,

\[
\phi_i(v) = \sum_{S \subseteq [d] \setminus \{i\}} \frac{|S|!(d - 1 - |S|)!}{d!} (v(S \cup \{i\}) - v(S))
\]

**Hypercube graph:** A cooperative game can also be looked at in terms of functions over a hypercube graph. The above game is precisely a \( d \)-dimensional hypergraph with vertex set \( V = 2^{[d]} \), and edges set \( E = \{(S, S \cup \{i\}) \in V \times V : S \subseteq [d] \setminus \{i\}, i \in [d]\} \). Here each edge corresponds to addition of a single player oriented in the direction of inclusion.

The value function of a game is a function over vertices of such a graph. Let’s denote the set of all such functions as \( \mathbb{R}^V \), similarly we can denote the set of all functions over edges of a hypergraph as \( \mathbb{R}^E \). A differential operator \( \nabla : \mathbb{R}^V \rightarrow \mathbb{R}^E \) can be defined as \( \nabla v(S, S \cup \{i\}) = v(S \cup \{i\}) - v(S) \). And one can also define a partial differential operator as follows,

\[
\nabla_i v(S, S \cup \{j\}) = \begin{cases} 
\nabla v(S, S \cup \{i\}) & i = j \\
0 & \text{otherwise}
\end{cases}
\]

We can think of \( \nabla_i v \) encoding the marginal contribution of player \( i \) to the game.

**Inessential game:** A game \( v \) of \( d \) players is called inessential iff \( v(S) = \sum_{i \in S} v(i) \) for all \( S \subseteq 2^{[d]} \). In other words, inessentiality means that every player contributes a fixed amount towards any coalition \( S \) irrespective of its composition. Therefore, an inessential game of \( d \) players is a linear function of \( v(\{i\}) \)s for \( i \in [d] \). Later we will use this property for testing for inessentiality efficiently. It is useful to know whether a game is inessential, because then the shapley value of \( i^{th} \) player is simply \( v(\{i\}) \).

The relation between inessentiality of a game in terms of the differential operator of a hypercube graph can be expressed by:

**Proposition 1** [8 Prop 3.3]. A game \( v \) is inessential iff for each \( i \in [d] \) there exists \( v_i \in \mathbb{R}^V \) such that \( \nabla_i v = \nabla v_i \).

These \( v_i \)s are known as component games for the original game \( v \) and Stern and Tettenhorst proved in Theorem 3.4[8] that these \( v_i \)s satisfy certain notions of efficiency, symmetry, linearity and dummy player. Their theorem implies that for any game \( v_i([d]) = \phi_i(v) \).
2.1 Shapley residuals

Shapley values make most intuitive sense when games are inessential, but in general games are often not inessential. Kumar et al.\[4\] introduces the notion of residuals. Shapley residuals essentially characterize the degree to which a game deviates from being inessential. Using fundamental theorem of linear algebra on Stern and Tettenhorst’s formulation we can write,

\[ \nabla_i v = \nabla v_i + r_i \]

**Definition 1 (Shapley Residuals).** We call \( r_i = \nabla_i v - \nabla v_i \) the Shapley Residual of player \( i \).

These \( r_i \)'s capture the deviation from inessentiality for each edge in the hypercube representation of the game. Therefore, when a game is inessential all entries in every residual vector is zero. Since the residuals are of very large dimension, one can use \( \sum_{i \in [d]} ||r_i||^2 \) as a single value representing the total deviation of a game function \( v \) from inessentiality.

And when the game is inessential, we can still get a \( v_i \) that minimizes the deviation by solving the following least square problem,

\[
\min_{x \in \mathbb{R}^V} ||\nabla x - \nabla v_i||
\]

\( x(\emptyset) = 0 \)

2.2 Exact residual computation

Algorithm 1 describes the naive approach to compute the residual for a single player

**Algorithm 1** Exact calculation of \( i \)-th Shapley residual of \( v \)

- Compute \( \nabla_i v \)
- Solve \( v_i = \arg\min_{x \in \mathbb{R}^V} ||\nabla_i v - \nabla x||^2 \)
- Compute \( \nabla v_i \)
- Return Shapley residual \( r_i = \nabla_i v - \nabla v_i \)

A naive way to check whether a game is inessential or not would be to compute all the residuals for individual players and can check whether they are all 0. But that is of \( O(d \times 2^d) \) complexity for a game with \( d \) players. In the next two sections we will propose an approximate test for checking inessentiality that is complete and will derive its sample complexity as well as a scheme to approximate both individual \( r_i \) and its norm.

3 Approximate inessentiality detection strategy

Before we provide a test for inessentiality, lets establish some basics of property testing. A boolean function \( f : \{0,1\}^n \rightarrow \{0,1\} \) is called linear if for all \( x, y \in \{0,1\}^n \), \( f(x) + f(y) = f(x+y) \), where addition is over GF(2). And we call a function \( f \) is \( \epsilon \)-far from linear when its value must be changed at a maximum \( \epsilon \)-fraction of points over its domain to make it linear. In order to determine whether \( f \) is linear, Blum, Luby, and Rubinfeld [2] proposed a randomized algorithm. Given \( f \), choose \( x, y \) uniformly at random and reject if \( f(x) + f(y) \neq f(x+y) \). This is known as BLR test. The probability of accepting linear functions is known as completeness and the probability of rejecting non-linear functions is called soundness of the test. Among other things, Soundness depends on the parameter \( \epsilon \).

Since, the characteristic function for games can be in general real valued and BLR test is valid over any arbitrary field, we present a generalization of BLR for determining linearity/inessentiality of a game.
3.1 BLR test

Given query access to $f : \{0,1\}^d \rightarrow \mathbb{R}$
- Choose $x \sim \{0,1\}^d$, we sample $x$ uniformly from the vertices of the d-dimensional hypercube.
- Choose $y \sim \{0,1\}^d$ such that there is no player overlap between two coalitions $x, y$. We do this because $f$ is only defined at the corners of the hypercube.
- Query $f$ at $x, y$, and $x + y$.
- "Accept" if $f(x) + f(y) = f(x + y)

Note that, the above test can also be used to determine whether $f$ is inessential relative to some coalition $S \subseteq 2^{[d]}$. In order to do this we simply set $x$ to coalition $S$ and follow the rest of the algorithm. It is clear that the modified BLR test has the completeness factor 1 because the test can not fail if $f$ is linear or in other words $f$ represents an inessential game.

3.2 Analysis

In our version of BLR we are interested in testing whether for a specific set of pairs of inputs, $S = \{(x,y) \mid x \land y = 0\}$, the equality $f(x) + f(y) = f(x + y)$ holds or not.

We note that the size of the set $S$ is, $\sum_{i=0}^{d} \binom{d}{i}2^{d-i}$, and the $f$ is of size $2^d$, therefore there are more than one pairs that test a particular input in the domain of $f$. More precisely if input $x$ has $k$ bits set i.e. it has $k$ active features, then the number of pairs in set $S$ that can test $f(x)$ is $2^k$. Then the probability of picking such a pair would be $\frac{2^k}{\sum_{i=0}^{d} \binom{d}{i}2^{d-i}}$. Let’s define a random variable $K_x$ which denotes the number of active feature in $x$. We know $E_{x\sim\{0,1\}^d}[K_x] = d/2$. Therefore probability of catching deviation of a single entry in $f$ is given by $\frac{\epsilon \cdot 2^{d/2}}{\sum_{i=0}^{d} \binom{d}{i}2^{d-i}}$. Now if $f$ is $\epsilon$-far from satisfying linearity over $S$, then the probability of detecting that is lower-bounded by $\frac{\epsilon \cdot 2^{d/2}}{\sum_{i=0}^{d} \binom{d}{i}2^{d-i}}$ since that is when every deviation is independent. Certainly this is not a tight lower-bound.

4 Approximate shapley residual computation

Residual of a player for a given game can be exactly computed using the Algorithm 1. The algorithm needs to compute $\nabla_i v$ before solving a least square instance, but since $v$ is a function over the vertices of the hypercube i.e. it’s complexity is exponential to the number of players (features in a model). Another challenge is that $r_i$ is a even higher dimensional object; it’s defined over the edges of the hypercube i.e. $O(2^{2d})$. Therefore, it is almost moot to exactly compute residual for any model over 30 features.

Here we demonstrate an approach to approximate the residual vector by estimating the residual. Then we use the approach to estimate the norm of the residual for a player (feature) using the sampled entries of the residual.

Stern and Tettenhorst [8] showed that the component games $v_i$ are also unique solutions to the equation $Lv_i = L_i v$ where $L$ is the graph laplacian, $L = \nabla^* \nabla$, and $L_i$ is the laplacian where weights for edges $(S, S \cup \{j\})$ for $i \neq j$ is 0, $L_i = \nabla^* \nabla_i$. 

4
4.1 Single residual entry estimation

Theorem 1.1 of [1] shows a way to approximate any single entry of $x^*$ given an linear equation of the form $Lx = b$ where $L$ is the Laplacian of a regular graph $G$ using sublinear number of probes into $b$. The main idea behind the theorem is to use a number of random walks to estimate a matrix inverse.

In our case we are dealing with hypercube graphs, which are regular with degree $d$ (number of features), $b = L_iv$ and solving for $v_i : \{0, 1\}^d \rightarrow \mathbb{R}$.

Algorithm 2

**How to approximate $v_i(u)$ for any $u \in \{0, 1\}^d$**

1: Set $s = \frac{1.5 + \log_2 d + \log_2 \sqrt{\|b\|_0 - \log_2 \epsilon}}{\log_2 d - \log_2 (d-1)}$
2: Set $\ell = 0 \left(\left(\frac{1}{4}\right)^{-2 \log_2 s}\right)$
3: for $t=0, \ldots, s-1$ do
4: Perform $l$ independent random walks of length $t$ starting at $u$, and let $u_1, \ldots, u_l$ be the end points of the walks.
5: $u_t = \frac{1}{l} \sum_{j \in [l]} b(u_j)$
6: end for
7: Return $\frac{1}{d} \sum_{t=0}^{s-1} u_t$

For any $\epsilon \in [0, 1]$ the following algorithm estimates $\hat{v}_i(u)$ with the following guarantee,

$$Pr[|\hat{v}_i(u) - v_i(u)| \leq \epsilon ||v_i||_\infty] \geq 1 - \frac{1}{s}$$

(1)

The time complexity of calculating a single estimate of one entry of $v_i$ is $O(\ell \cdot s)$. We can avoid computing the zero norm of $L_iv$ simply by using the upper bound $||L_iv||_0 \leq 2^d$. In the following algorithm we can compute $L_i v(u_k) = v(u_k) - v(u_k \oplus (1 << (i - 1)))$ which allows us to avoid computing the exponential $b$ altogether.

Now since residuals live in space $\{0, 1\}^d \times \{0, 1\}^d$, we can approximate a single entry $r_i(s, e)$ as the following,

$$r_i(s, e) = \begin{cases} v(e) - v(s) - \hat{v}_i(e) + \hat{v}_i(s) & \text{if } s \oplus e + 1 = i \\ \hat{v}_i(s) - \hat{v}_i(e) & \text{otherwise} \end{cases}$$

(2)

The space complexity of calculating a $e$ entries of residual for a single player is $O(e)$, if we sample $p$ percentage of edges for a game with $d$ players, the total space complexity of the residual calculation is $O(p \cdot d \cdot 2^{d-1})$. Accordingly the time complexity is $O(p \cdot d \cdot 2^{d-1} \cdot \ell \cdot s)$.

4.2 Norm of a residual estimation

We can estimate the euclidean norm of the residual using the following algorithm

Algorithm 3

**How to estimate $||r_i||^2$**

1: Set $\hat{r} = 0$
2: for $t = 0, \ldots, \tau$ do
3: Choose a random edge $(s, e) \in E$
4: $\hat{r} = \hat{r} + r_i(s, e)^2$, using Equation 2
5: end for
6: Return $\frac{d \cdot 2^{(d-1) \cdot \tau}}{\tau}$
If we only require estimating the approximate norm of residual for a single player, the space complexity is $O(1)$, and the time complexity is the same as above i.e. $O(p \cdot d \cdot 2^{d+1} \cdot \ell \cdot s)$.

The exact computation of residual or its norm requires $O(2^{2d})$ space and $O(2^{2d} \cdot 37)$ time using any generic linear system solver.

5 Experiment

In terms of choice of games; for the inessentiality test we generate random games of $d$ players for various values of $d$. We cite an example for a game where $d = 3$. For testing various metrics related to accuracy, running time of residual and norm of residual estimation we choose two common mathematical games Parity and Surprise which have very different characteristics. In the case of Parity, the game vector is non-zero for every even sized coalition and zero for odd sized ones, whereas Surprise only has a single non-zero entry for the grand coalition.

The following are the three primary scenarios we test our algorithms for;

- Success probability of BLR test in detecting a non-inessential game as controlled non-linearity in introduced to an inessential game.
- Accuracy numbers for norm of residual estimation for games such as Surprise, Parity for varying number of dimensions, values of $\epsilon$ (error tolerance) and $\tau$ (percentage of edges sampled).
- Comparison between time complexity of exact vs approximate residual norm computation.

5.1 BLR test success probability

One thing to note is that a fourier coefficient vector of a game of $d$ player is composed on $2^d$ elements.

For demonstration we here show an example of an inessential game with 3 players with the following value function for singleton player sets,

\[
\begin{align*}
v(\{1\}) &= 0.5430, \\
v(\{2\}) &= 0.8151, \\
v(\{3\}) &= 0.2746
\end{align*}
\]

This values are chosen from a normal distribution with 0 mean and 1 variance.

The fourier coefficients are the following,

\[
(0.8163, -0.2715, -0.4075, 0, -0.1372, 0, 0, 0)
\]

Note that for an inessential game only fourier coefficients relating to null set and single players are non-zero. For $\epsilon = 0.1$ the above game is modified to $v(\{2\}) = 0.5511$. This change leads to the following changes in the fourier coefficients,

\[
(0.8173, -0.2725, -0.4065, -0.001, -0.1362, -0.001, 0.001, -0.001)
\]

Now we observe that all the coefficient has ended up becoming non-zero.

We ran the test 3.1 for $\frac{1}{\epsilon} = 10$ pairs of $(x, y)$ inputs and detected inessentiality successfully with probability 0.2 over 100 independent runs. This is in sync with the lower bound analysis shown in section 3.2 which gives a bound of 0.0838. We also conducted the experiment for several values of $d$ ranging from 3 to 20, in every case the results were consistent.
5.2 Accuracy in residual norm estimation

We assess the accuracy of our residual norm estimation algorithm on several random games with 8 players \((d = 8)\). These were produced by introducing corruption to a game vector which represents an inessential game, the corruption is measured by probability \((\omega)\) that the game function is different at a particular index from its closest inessential counterpart. We vary both the tolerance parameter \((\epsilon)\) and percentage \((\tau)\) of edges of residual sampled in each run while running the estimation algorithm. Table 1 shows the errors for different \((\omega, \epsilon, \tau)\) triples. And the error is calculated as the average of absolute difference in norm estimate of the residual from the exact norm of residual over \(d\) players. All the errors are averaged over 10 independent sample of edges from the game hypercube.

<table>
<thead>
<tr>
<th>(\omega)</th>
<th>(.15,.1)</th>
<th>(.15,.25)</th>
<th>(.15,.5)</th>
<th>(.1,.1)</th>
<th>(.1,.25)</th>
<th>(.1,.5)</th>
<th>(.05,.1)</th>
<th>(.05,.25)</th>
<th>(.05,.5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>.01</td>
<td>105.71</td>
<td>105.68</td>
<td>106.25</td>
<td>65.88</td>
<td>63.6</td>
<td>62.98</td>
<td>39.28</td>
<td>40.53</td>
<td>27.15</td>
</tr>
<tr>
<td>.05</td>
<td>70.95</td>
<td>68.44</td>
<td>66.31</td>
<td>42.44</td>
<td>39.82</td>
<td>33.97</td>
<td>28.72</td>
<td>17.47</td>
<td>14.83</td>
</tr>
<tr>
<td>0.1</td>
<td>42.73</td>
<td>38.92</td>
<td>39.63</td>
<td>26.57</td>
<td>22.94</td>
<td>20.83</td>
<td>14.79</td>
<td>9.93</td>
<td></td>
</tr>
<tr>
<td>0.25</td>
<td>25.46</td>
<td>28.67</td>
<td>27.34</td>
<td>17.32</td>
<td>14.93</td>
<td>13.25</td>
<td>10.78</td>
<td>8.29</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>23.32</td>
<td>27.89</td>
<td>23.86</td>
<td>14.76</td>
<td>12.59</td>
<td>11.93</td>
<td>8.56</td>
<td>6.88</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Error estimates for random 8 player games with varying distance \((\omega)\) from an inessential game

We can see as \(\omega\) increases meaning deviation of a game from inessentiality increases, the Error decreases. Similarly we also observe that as \(\epsilon\) value is gets smaller both individual estimates for residual entries improve and therefore improving the overall norm estimation.

We can see as \(\omega\) increases meaning deviation of a game from inessentiality increases, the Error decreases. Similarly we also observe that as \(\epsilon\) value is gets smaller both individual estimates for residual entries improve and therefore improving the overall norm estimation.

<table>
<thead>
<tr>
<th>(d)</th>
<th>((\epsilon, \tau))</th>
<th>(.15,.1)</th>
<th>(.15,.25)</th>
<th>(.15,.5)</th>
<th>(.1,.1)</th>
<th>(.1,.25)</th>
<th>(.1,.5)</th>
<th>(.05,.1)</th>
<th>(.05,.25)</th>
<th>(.05,.5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>(.15,.1)</td>
<td>33.08</td>
<td>32.42</td>
<td>30.99</td>
<td>21.19</td>
<td>15.41</td>
<td>12.85</td>
<td>16.32</td>
<td>8.8</td>
<td>7.44</td>
</tr>
<tr>
<td>7</td>
<td>(.15,.25)</td>
<td>17.22</td>
<td>19.35</td>
<td>18.84</td>
<td>13.17</td>
<td>10.94</td>
<td>9.01</td>
<td>12.65</td>
<td>8.32</td>
<td>5.27</td>
</tr>
<tr>
<td>8</td>
<td>(.15,.5)</td>
<td>14.32</td>
<td>14.61</td>
<td>13.42</td>
<td>10.6</td>
<td>7.15</td>
<td>6.64</td>
<td>9.58</td>
<td>5.02</td>
<td>3.58</td>
</tr>
<tr>
<td>9</td>
<td>(.1,.1)</td>
<td>10.02</td>
<td>9.71</td>
<td>9.55</td>
<td>8</td>
<td>5.34</td>
<td>4.64</td>
<td>6.56</td>
<td>4.67</td>
<td>2.81</td>
</tr>
<tr>
<td>10</td>
<td>(.1,.25)</td>
<td>7.86</td>
<td>7.31</td>
<td>7.26</td>
<td>5.51</td>
<td>3.91</td>
<td>3.38</td>
<td>4.33</td>
<td>3.15</td>
<td>2.1</td>
</tr>
</tbody>
</table>

Table 2: Error estimates for the parity game with varying player count \(d \in [6\ldots10]\)

<table>
<thead>
<tr>
<th>(d)</th>
<th>((\epsilon, \tau))</th>
<th>(.15,.1)</th>
<th>(.15,.25)</th>
<th>(.15,.5)</th>
<th>(.1,.1)</th>
<th>(.1,.25)</th>
<th>(.1,.5)</th>
<th>(.05,.1)</th>
<th>(.05,.25)</th>
<th>(.05,.5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>(.15,.1)</td>
<td>35.9</td>
<td>36.24</td>
<td>27.38</td>
<td>42.06</td>
<td>34.12</td>
<td>32.86</td>
<td>50.57</td>
<td>47.68</td>
<td>38.53</td>
</tr>
<tr>
<td>7</td>
<td>(.15,.25)</td>
<td>29.27</td>
<td>24.05</td>
<td>35.28</td>
<td>40.34</td>
<td>41.6</td>
<td>34.21</td>
<td>60.35</td>
<td>50.41</td>
<td>38.51</td>
</tr>
<tr>
<td>8</td>
<td>(.15,.5)</td>
<td>33.86</td>
<td>40.9</td>
<td>31.86</td>
<td>42.45</td>
<td>45.24</td>
<td>39.14</td>
<td>61.6</td>
<td>54.81</td>
<td>47.22</td>
</tr>
<tr>
<td>9</td>
<td>(.1,.1)</td>
<td>42.32</td>
<td>39.42</td>
<td>37.57</td>
<td>55.19</td>
<td>50.99</td>
<td>43.35</td>
<td>63.37</td>
<td>58.69</td>
<td>48.79</td>
</tr>
<tr>
<td>10</td>
<td>(.1,.25)</td>
<td>54.09</td>
<td>47.95</td>
<td>38.15</td>
<td>60.23</td>
<td>55.94</td>
<td>46.73</td>
<td>67.46</td>
<td>62.84</td>
<td>49.24</td>
</tr>
</tbody>
</table>

Table 3: Error estimates for the surprise game with varying player count \(d \in [6\ldots10]\)

The above two tables show errors in residual norm estimation for two common games Parity (table 2) and Surprise (table 3). Like the previous experiment the errors here are also averaged over 10
independent runs of edge samples.

The following table shows the error in estimation for Parity and Surprise games respectively for players $d \in [12 \ldots 14]$. $t_{\text{exact}}$ is the time for exact and $t_{\text{approx}}$ for estimation.

<table>
<thead>
<tr>
<th>Game</th>
<th>d</th>
<th>$\epsilon$</th>
<th>edge pct</th>
<th>mean_error</th>
<th>$t_{\text{exact}}$</th>
<th>$t_{\text{approx}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parity</td>
<td>12</td>
<td>0.1</td>
<td>0.01</td>
<td>9.02</td>
<td>6.25</td>
<td>3.23</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>0.1</td>
<td>0.01</td>
<td>8.12</td>
<td>25.84</td>
<td>8.42</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>0.1</td>
<td>0.01</td>
<td>5.57</td>
<td>131.65</td>
<td>31.82</td>
</tr>
<tr>
<td>Surprise</td>
<td>12</td>
<td>0.1</td>
<td>0.01</td>
<td>37.04</td>
<td>5.8</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>0.1</td>
<td>0.01</td>
<td>40.27</td>
<td>25.21</td>
<td>1.42</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>0.1</td>
<td>0.01</td>
<td>49.08</td>
<td>134.58</td>
<td>5.86</td>
</tr>
</tbody>
</table>

Table 4: Time taken to compute exact and approximate residual norm

6 Scope for further research

It is evident from the experiments, the simple uniform random sampling for edges of the residual vector works well when the game vector is not sparse such as Parity. But error in estimation greatly increases as sparsity increases. The error bound; as shown in the inequality 1, is dependent on the infinity norm of the component game vector. Therefore when entries have high variance the estimation may tend to stray off for some entries. Both of these issues can be observed in the table 3. An immediate direction for further research can be improving the sampling procedure when the vector is sparse and have high variance. Also further research is required to understand full properties of residual vectors and their interpretations.

References


