False-Name-Proof Recommendations in Social Networks

Markus Brill  
Duke University  
brill@cs.duke.edu  

Vincent Conitzer  
Duke University  
conitzer@cs.duke.edu  

Rupert Freeman  
Duke University  
rupert@cs.duke.edu  

Nisarg Shah  
Carnegie Mellon University  
nkshah@cs.cmu.edu  

ABSTRACT

We study the problem of finding a recommendation for an uninformed user in a social network by weighting and aggregating the opinions offered by the informed users in the network. In social networks, an informed user may try to manipulate the recommendation by performing a false-name manipulation, wherein the user submits multiple opinions through fake accounts. To that end, we impose a no harm axiom: false-name manipulations by a user should not reduce the weight of other users in the network. We show that this axiom has deep connections to false-name-proofness. While it is impossible to design a mechanism that is best for every network subject to this axiom, we propose an intuitive mechanism LEGIT+, and show that it is uniquely optimized for small networks. Using real-world datasets, we show that our mechanism performs very well compared to two baseline mechanisms in a number of metrics, even on large networks.

Keywords

Social choice, Recommendation systems, Social networks, False-name-proofness

1. INTRODUCTION

Consider the following problem. An agent wants to receive a recommendation on a specific item—say, a movie the agent has not previously watched. Others have evaluated this item, perhaps by giving it a “thumbs up” or “thumbs down” (0 or 1), or by rating it on a more detailed scale, say, from 0 to 5. We want to give the agent an aggregate rating, such as “73% positive” or 2.7. Alternatively, perhaps the question is merely whether to recommend an item to the agent at all, in which case the aggregate outcome must be binary. How should we arrive at this aggregate outcome?

For simplicity, let us assume that we do not have any information about which agents have preferences most similar to the agent in question. In this case, a natural approach is to simply take the average of all the ratings so far. One problem is that if ratings are not binary, this is not strategy-proof: when the current average is 2.7, an agent who feels the item is a 4 may prefer to report 5 to pull the average closer to his evaluation. As is well known in social choice theory, a good alternative is to choose the median rating instead: this is in fact group-strategy-proof (when preferences are single-peaked, as is likely to be the case here) [21, 4, 6]. Note that for binary ratings, the median is simply the majority choice.

The median, however, remains vulnerable to another type of manipulation, commonly known as false-name manipulation [36]: an agent can rate the same item many times by opening fake accounts, and move the median closer to his evaluation. Thus, the median is not false-name-proof. In fact, without imposing further structure on the problem, no reasonable rule is false-name-proof [9, 30, 31]. On the other hand, if we assume that agents are organized as the nodes of a (say, undirected) social network, possibilities open up [10]. For example, rather than reporting to the agent the median of all the ratings, we can simply report the median of his friends’ ratings. Assuming that the agent will not be duped into befriending fake accounts, this will in fact be false-name-proof.

The downside of this methodology is that, with the exception of very popular items, none or very few of the agent’s friends may have rated the item. Consequently, the median-of-friends rule conveys too little information. Could we include the friends of the agent’s friends as well? Done naively, this may give the friends an incentive to create fake friends of their own. But more subtly, when a friend does not rate the item, we can pretend that his rating was the median of his friends’ ratings. This does not give the friends an incentive to create fake accounts: all this would do for them is change their own hallucinated ratings, but they can more easily just specify those ratings directly. This median-of-medians approach closely resembles the majority-majorities rule from Andersen et al. [2]. Note, however, that this rule ends up double-counting the rating of an agent who is a friend of two of the agent’s friends. Can we circumvent this issue? Also, can we retrieve ratings from deeper in the social network?

Our results. In this paper, we focus on a two-step approach. In the first step, we use a weight-selecting mechanism to assign weights to the agents offering an opinion/rating, called voters, without looking at their opinions (thus, looking only at the network structure). In the second step, we perform a weighted aggregation of the opinions to output a recommendation by only looking at the weights assigned to voters and their opinions. To make the weight-selecting mechanism robust to false-name manipulations, we impose a no harm axiom: false-name manipulations by an agent should not reduce the weight of other agents in the network.

We show that with weighted median aggregation, the no harm axiom implies false-name-proofness (Theorem 1) and, under some conditions, is actually equivalent to false-name-proofness (Theorem 2). We thus focus on designing weight-selecting mechanisms subject to this axiom.

We focus on the case where, ideally, we would like to weight the voters uniformly. As explained in detail in Section 2, this is for multiple reasons. While this does not utilize the network structure for inferring the closeness in opinions of two nodes, it clearly outlines
how to use the network structure for a distinct purpose — achieving the no harm axiom. Section 6 discusses how our results can be extended to take into account correlation among opinions. Second, weighting the voters equally can indeed be ideal, e.g., when aggregating independent noisy estimates of an underlying objective ground truth (see Section 5), or when the goal is not to find a recommendation but to conduct a fair vote.

Unfortunately, weighting all the voters uniformly violates the no harm axiom. What is the “most uniform” weight vector we can return subject to this axiom? In order to formalize what “more uniform” means, we use the classic leximin criterion that compares weight vectors by their smallest weights (preferring the vector with greater smallest weight), and then breaks ties using the second smallest weights, and so on. We show that a weight-selecting mechanism cannot always return the leximin-optimal weight vector subject to the no harm axiom (Theorem 4). We then present an intuitive mechanism and show that it is uniquely optimized for small networks subject to the no harm axiom (Theorem 5), that is, (informally) if a mechanism outputs a more uniform weight vector than our mechanism does on some network, then there is a strict subgraph of the network on which our mechanism outputs a more uniform weight vector.

Using a non-trivial result from graph theory [15], we show that our mechanism can be computed in linear time in the size of the network (Theorem 6). In Section 5, we present experiments with real-world social networks in which our mechanism significantly outperforms two baseline mechanisms in a number of metrics.

Related work. Recommendation systems have been studied extensively in the machine learning literature, see, e.g., [14, 1, 26, 5]. Popular techniques include content-based recommendation [25], where the decision of whether to suggest an item to a target user is made by considering the attributes of the item and the target user’s previously expressed preferences; collaborative filtering [13], where the preferences of other users in the network are given, and their similarity with the target user’s preferences is learned to find a good recommendation; or, both combined [3]. In contrast, we solely focus on the use of the social network structure to design recommendation mechanisms that are robust to false-name manipulations.

Besides the works cited previously, false-name manipulations have also been studied rigorously in a variety of anonymous environments, such as combinatorial auctions [35, 34, 36, 32, 17], matching [29], and voting [33].

2. MODEL

We are given a social network (or simply, a network), which is an undirected simple graph\(^1\) denoted \(G\). The set of nodes and the set of edges of \(G\) are denoted \(V(G)\) and \(E(G)\) (or \(V\) and \(E\), when the graph is clear from the context), respectively. For \(T \subseteq V\), let \(G_T\) denote the subgraph of \(G\) induced by \(T\).

Our task is to find a recommendation for a given node \(v^* \in V\). This task could arise in a number of contexts: we may want to decide whether to recommend a given movie or restaurant to an individual (in which case, we want a binary recommendation), or we may want to show the rating of the movie or restaurant (in which case, we no longer want a binary recommendation). To aid our decision-making, a set of nodes \(S \subseteq V \setminus \{v^*\}\) offer their personal opinion. We call these nodes voters, and denote the opinion offered by voter \(v \in S\) as \(r_v\). Target node \(v^*\) is not a voter itself. As we explain in Section 3, the mechanisms of our interest must discard voters not connected to \(v^*\); thus, for simplicity we assume that \(G\) is connected and has at least one voter.

Weight-Selecting Mechanisms: In this paper, we are interested in finding recommendations through a two-step approach: i) using a weight-selecting mechanism to assign a weight to each voter in the network only as a function of the network structure \(G\), the subset of voters \(S\), and the target node \(v^*\) (thus, independent of the voters’ opinions), and ii) using a weighted aggregation function that takes as input the weights assigned by the weight-selecting mechanism and the voters’ opinions, and outputs the final recommendation. Popular choices for the weighted aggregation function include weighted mean and weighted median; Section 3.1 discusses how this choice impacts the overall recommendation system. For the remaining parts of the paper, we are only interested in studying weight-selecting mechanisms and their properties (the first step).

For a weight-selecting mechanism — ignorant of the voters’ opinions — a problem instance is given by the tuple \((G, S, v^*)\).

Definition 1 (Weight-Selecting Mechanisms). Given an instance \((G, S, v^*)\), a weight-selecting mechanism outputs a weight vector \(w = (w_v)_{v \in S}\) such that \(w_v \geq 0\) for \(v \in S\), and \(\sum_{v \in S} w_v = 1\).

Weight-selecting mechanisms are compelling because they allow harmonious aggregation of opinions of various formats, ranging from binary to real-valued opinions.

False-Name Manipulations: In the absence of additional restrictions, one can simply choose the weight-selecting mechanism that returns the most appropriate weight vector for the setting of interest. In this paper, however, we consider an important restriction that stems from game-theoretic considerations: preventing false-name manipulations.

Online social networks typically lack a proof of authenticity of nodes, thus allowing users to easily create fake accounts. In this case, a weight-selecting mechanism may inadvertently provide an incentive to a malicious user for creating multiple fake accounts and voting through them in order to gain a higher total weight, and thus a greater influence on the final recommendation. Such manipulations are known as false-name manipulations or sybil attacks.

In a false-name manipulation, the malicious node in the network can easily create any desired subset of edges among the identities it controls: its own node, and the fake nodes it creates. Altering edges with other real nodes (e.g., creating new edges or deleting existing edges), on the other hand, is often more costly. Given that (arguably) recommendations are not the primary objective in most social networks, we assume that nodes do not alter their edges with other real nodes as part of a false-name manipulation due to lack of sufficient incentive. That said, alterations to edges with real nodes are a powerful form of manipulation, and preventing such manipulations is an interesting theoretical challenge (see Section 6).

Definition 2 (False-Name Manipulations). In an instance \((G, S, v^*)\), a voter \(v \in S\) can perform a false-name manipulation by creating a set of false nodes \(M\), and edges between a subset of pairs of nodes in \(M' = M \cup \{v\}\), where \(M' = M \cup \{v\}\). Also, \(v\) can choose a subset of nodes in \(M'\) to act as voters, and their recommendations. The resulting instance is given by \((G', S', v^*)\), where \(V(G') = V \cup M, S' \cap (V \setminus \{v\}) = S \setminus \{v\}\), and \(E(G') \cap (V \times V) = E\).

A node that has a personal opinion may choose to abstain from voting as part of a manipulation. Such a node would be a voter in the underlying true instance, but not in the manipulated instance observed by the mechanism. We refer to it as an “opinionated node” to avoid confusing it with “voters” in the observed instance. In this

\(^1\)A simple graph has no self-loops and at most one edge between every pair of vertices.
paper, we only focus on false-name manipulations by individual nodes; Section 6 briefly discusses group false-name manipulations.

**Optimal Weight Vector:** Preventing false-name manipulations may prohibit us from always choosing the most desirable weight vector for the setting at hand. For the purpose of this paper, we assume a setting in which the ideal weight vector has equal weights for all voters, i.e., in the ideal weight vector each voter in $S$ has weight $1/|S|$. This is interesting due to multiple reasons:

- False-name-proof recommendation mechanisms can employ the knowledge of the social network structure in two clearly distinct ways: i) to weight nodes in a way that provides no incentive for false-name manipulations, and ii) to weight nodes to reflect their level of homophily\(^2\) or trust with the target node. Teaching the uniform weight vector as the ideal focuses exclusively on the former purpose. This is also the appropriate choice for networks where no prior information about user opinions is available to conclude homophily.

- Our model also applies to the case where the goal is not to find a recommendation for the target node; instead, the target node conducts a vote on the network, and invites its peers to vote. In this case, treating all voters equally is the de facto fairness consideration in the voting literature, often termed “one person, one vote.”

- Finally, if the opinions offered by the individuals are not subjective preferences, but rather i.i.d. noisy estimates of an objective ground truth, using equal weights for aggregation provably yields the most accurate (e.g., the least squared error) estimation of the underlying ground truth.

That said, aggregating subjective opinions of nodes into a recommendation by weighting the nodes according to their homophily (closeness of opinion) with the target node is an interesting and widely studied topic. As we discuss in Section 6, our results have interesting implications about designing false-name-proof recommendation mechanisms when a model of homophily is given; in this sense, we also view our paper as a stepping stone for studying this more general setting.

Finally, we impose a mild restriction—**symmetry**—on the weight-selecting mechanism. Informally, this requires the mechanism to assign equal weight at least to the nodes that are “symmetrically placed” in the network with respect to the target node.

**Definition 3 (Symmetric Mechanisms).** We call a weight-selecting mechanism symmetric if, given an instance $(G, S, v^*)$, it assigns equal weights to voters $v_1$ and $v_2$ whenever there exists an automorphism of $G$ (i.e., an isomorphism from $G$ to itself) that fixes $v^*$ and maps $v_1$ to $v_2$.

Unless stated otherwise, throughout the paper we will assume a weight-selecting mechanism to be symmetric.

### 3. UNIFORM AGGREGATION

In the context of this paper, the ideal weight-selecting mechanism returns the weight vector that has equal weight for all voters. However, this mechanism suffers from a crucial problem. A node that performs a false-name manipulation by creating an arbitrarily large number of fake nodes and voting through them can accrue a weight arbitrarily close to 1, thus becoming a dictator. Crucially, this manipulation also hurts the other nodes in the network by reducing their weights. To design a robust mechanism, we require that this should not be possible.

\(^2\)Homophily is a commonly observed phenomenon where nodes closer in a network are more likely to agree on opinions.

**Definition 4 (No Harm Axiom).** We say that a weight-selecting mechanism satisfies the no harm axiom if a false-name manipulation by a node does not reduce the weight of any other node in the network. We let $\mathcal{M}^\text{NH}$ denote the family of symmetric weight-selecting mechanisms satisfying the no harm axiom.

#### 3.1 No Harm Versus False-Name-Proofness

The standard desideratum in the literature on false-name manipulations is false-name-proofness, which requires that even with full information an agent should not be able to find a beneficial false-name manipulation. In our setting, this means a voter should not be able to move the recommendation closer to its opinion through a false-name manipulation even if the voter knows the network $G$, the set of voters $S$, their opinions $r$, and the target node $v^*$. The no harm axiom directly implies that a voter cannot gain weight by performing a false-name manipulation. Could the voter, however, increase the weights of other voters with similar opinions, thereby achieving a more favorable recommendation? This of course depends on how the recommendation mechanism uses the weights to aggregate the opinions. We show that for the weighted median aggregation, the no harm axiom implies false-name-proofness.

**Theorem 1.** With real-valued opinions and aggregate recommendation, computing the weighted median of the opinions using the weights returned by a weight-selecting mechanism satisfying the no harm axiom is false-name-proof.

**Proof.** Let $(G, S, v^*)$ be the true instance for which the weight vector is $w$ and the recommendation is $x$. Suppose $v \in S$ performs a false-name manipulation, after which the weight vector becomes $w'$ and the recommendation becomes $x'$. If $r_u = x$, then $v$ has nothing to gain. Without loss of generality, let $r_u > x$. Define $T = \{u \in S \mid r_u \leq x\}$. Let $w(T) = \sum_{u \in T} w_u$ and $w'(T) = \sum_{u \in T} w'_u$. Then, by the no harm axiom and the definition of weighted median, we have $w'(T) \geq w(T) \geq 0.5$, which implies $x' \leq x$. Hence, the manipulation is not beneficial to $v$.

Theorem 1 shows that the no harm axiom easily yields false-name-proofness. But at first glance, it may seem too strong if the ultimate goal is false-name-proofness. The following result shows that in a simple setting with binary (0/1) opinions and reasonable weighted aggregation functions (e.g., the weighted average), the no harm axiom is equivalent to false-name-proofness.

**Theorem 2.** Let the opinions be binary (i.e., in $\{0, 1\}$), and the recommendation be computed using a weighted aggregation function that is strictly monotonically increasing in the total weight of all voters with opinion 1, where the weights are computed using a weight-selecting mechanism $M$. Then, the recommendation system is false-name-proof if and only if $M$ satisfies the no harm axiom.

**Proof.** Suppose $M$ satisfies the no harm axiom. Without loss of generality, consider a voter with opinion 0. If the voter performs a false-name manipulation, none of the real voters with opinion 1 lose weight due to the no harm axiom. Hence, the total weight of all voters with opinion 1 does not decrease after the manipulation. Hence, due to monotonicity of the weighted aggregation function, the recommendation cannot decrease due to the manipulation. That is, no false-name manipulation can be beneficial, implying that the recommendation system is false-name-proof.

Now, suppose that $M$ does not satisfy the no harm axiom. Then, there exists an instance $(G, S, v^*)$, a false-name manipulation by $v \in S$ that results in an instance $(G', S', v^*)$, and a voter $u \in S \setminus \{v\}$ such that under $M$, voter $u$ receives less weight in
(G', S', v') than in (G, S, v'). Suppose in (G, S, v') all voters in S' \( \{u\} \) vote for 1, and only v votes for 0. Since the weights sum to 1 and u loses weight after the manipulation, the total weight of voters with opinion 1 increases after the manipulation. Strict monotonicity of the weighted aggregation function implies that the manipulation would bring the recommendation closer to v's true opinion, 1. Hence, the recommendation system is not false-name-proof in this case.

### 3.2 Search for a Robust Mechanism

Our starting point is a compelling mechanism proposed by Andersen et al. [2] for binary (0/1) recommendations. Imagine doing a random walk on the social network graph starting from the node v', and terminating the walk as soon as a voter is encountered. Then, their mechanism recommends an opinion such that the walk would bring the recommendation closer to v's true opinion, 1. When a voter v' \neq v performs a false-name manipulation, the neighborhoods of nodes on the walk do not change. Hence, the walk still leads the random walk to v with the same probability post-manipulation. As this argument applies to every walk leading to a random walk starting from v' encounters v before any other voter.

Our assumption of G being connected and having at least one voter implies that the weights assigned by RANDOMWALK sum to 1. Also, we assume that the edges of the undirected graph G are essentially bidirectional, that is, a walk can traverse an edge in either direction. Crucially, observe that RANDOMWALK satisfies the no harm axiom: Fix a voter v and a walk that leads the random walk to v. When a voter v' \neq v performs a false-name manipulation, the neighborhoods of nodes on the walk do not change. Hence, the walk still leads the random walk to v with the same probability post-manipulation. As this argument applies to every walk leading the random walk to v in the original graph, the total probability of the random walk terminating on v does not reduce after the manipulation. It is also clear that RANDOMWALK is symmetric.

**Theorem 3.** RANDOMWALK is a symmetric weight-selecting mechanism satisfying the no harm axiom.

**Example 1.** Let G be the network shown on the right. Here, filled nodes represent voters. It is evident that neither v_1 nor v_2 is fake (i.e., they cannot be artificial nodes created by a single node in the network through a false-name manipulation).

Hence, for uniform aggregation we should weight them equally, if possible. Under RANDOMWALK, voters v_1 and v_2 receive (unequal) weights 2/5 and 3/5, respectively. This can be shown by solving systems of linear equations (see Section 4). Note that these probabilities are not 1/4 and 3/4, respectively, due to walks that go from v' to one of its three non-voter neighbors and return to v' a number of times, before finally going to v_1.

Admittedly, Andersen et al. [2] study a slightly different setting than ours. Their ultimate goal, unlike ours, is not to uniformly aggregate the opinions; they want the opinion of a voter to be weighted by the level of "trust" v' can plausibly have for the voter. Hence, in their setting it makes sense to weight the two voters unequally. In other words, our goal is not to evaluate RANDOMWALK in our setting, because RANDOMWALK is not designed to give equal weight to voters in the first place. We use Example 1 simply to demonstrate the need to investigate whether there exists a mechanism satisfying the no harm axiom that can provide more uniform weights.

### 3.3 An Impossibility Result

As the no harm axiom prohibits always selecting the uniform weight vector (with equal weight for all voters), our goal is to find a weight vector that is as uniform as possible. To formalize the notion of "uniformity," we use the classic leximin criterion that compares two weight vectors by their minimum weights (and prefers the one with greater minimum weight), and then breaks ties by comparing their second minimum weights, and so on. For example, according to the leximin criterion, weight vector (0.3, 0.5, 0.2) is better (i.e., more uniform) than weight vector (0.4, 0.5, 0.1), but is no different than weight vector (0.5, 0.3, 0.2). The leximin criterion has been studied extensively in the literature [28, 22, 23], and has been applied successfully in a broad spectrum of domains including constraint programming [7], wireless networks [16], resource allocation [12, 19], cake-cutting [8], and kidney exchange [27].

**Definition 6 (Leximin Comparison).** On an instance (G, S, v*), let weight-selecting mechanisms M and M' return weight vectors w and w', consisting of weights \( w_1, \ldots, w_S \) and \( w'_1, \ldots, w'_S \), respectively, sorted in the non-decreasing order. Then, M is leximin-better than M' on (G, S, v*) if there exists \( t \in \{1, \ldots, S\} \) such that \( w_t = w'_t \) for all \( t \in \{1, \ldots, t-1\} \) and \( w_t > w'_t \).

Comparing mechanisms across instances, we say that M is leximin-better than M' if M is not leximin-better than M on any instance, and M is leximin-better than M' on at least one instance.

**Definition 7 (Leximin-Optimality).** In a family of weight-selecting mechanisms C, mechanism M ∈ C is called leximin-optimal for C if M is leximin-better than every other mechanism in C.

We can now cast our search for a good mechanism as a formal question. Does there exist a mechanism that is leximin-optimal for the family \( M^{\text{NH}} \) of symmetric weight-selecting mechanisms satisfying the no harm axiom? Note that at most one mechanism could satisfy this desideratum. Unfortunately, the next result shows that in our case none meets the bar.

**Theorem 4.** No mechanism is leximin-optimal for \( M^{\text{NH}} \).

**Proof.** Let G_1 and G_2 be the networks shown below.

![Graph G_1](a) Graph G_1

![Graph G_2](b) Graph G_2

Figure 1: Impossibility of leximin-optimality for \( M^{\text{NH}} \).

Suppose for contradiction that there exists a weight-selecting mechanism M ∈ \( M^{\text{NH}} \) that is leximin-optimal for \( M^{\text{NH}} \). It can be shown that there exists a mechanism in \( M^{\text{NH}} \) that weights both voters in G_1 equally. While RANDOMWALK does not satisfy this, the reader may check that the mechanism LEGIT^\text{®} that we later propose in Section 3.4 does. Leximin-optimality of M now implies that M must assign weight 1/2 to both voters in G_1.

Next, note that G_2 is created when voter v_2 in G_1 performs a false-name manipulation. Thus, the no harm axiom implies that M
must still assign a weight of at least 1/2 to \( v \) in \( G_2 \). As the remaining weight is divided equally among the remaining two voters by symmetry, it follows that the minimum weight in \( G_2 \) under \( M \) is at most 1/4. However, it can be checked that the minimum weight in \( G_2 \) under \( \text{RANDOMWALK} \) is 2/7, which is greater than 1/4. Hence, \( \text{RANDOMWALK} \in \mathcal{M}^{\text{NH}} \) is leximin-better than \( M \) on \( G_2 \), which contradicts leximin-optimality of \( M \) for \( \mathcal{M}^{\text{NH}} \).

### 3.4 A Possibility Result

Theorem 4 implies that subject to the no harm axiom, a mechanism cannot be the best on every instance. It faces an inevitable trade-off whereby choosing to be better on one instance requires it to be worse on another. Which instances should get more emphasis? In social networks, often very few users make the effort to vote, and this scarcity of information is further exacerbated in smaller networks. Thus, arguably, achieving a uniform weight vector is more important in smaller networks so that every opinion counts. In larger networks, it is often excusable to discard a few opinions in order to achieve robustness. We translate this informal goal of giving more importance to smaller networks into a formal desideratum, "optimized for small networks", which we view as a novel conceptual contribution of the paper as its formulation may be useful in other settings as well.

**Definition 8 (Dominination).** For weight-selecting mechanisms \( M \) and \( M' \), we say that \( M \) dominates \( M' \) on network \( G \) for target node \( v^* \) if \( M' \) is not leximin-better than \( M \) on \( (G, S, v^*) \) for any \( S \subseteq V(G) \) \( \setminus \{v^*\} \), and \( M \) is leximin-better than \( M' \) on \( (G, S, v^*) \) for some \( S \subseteq V(G) \) \( \setminus \{v^*\} \).

**Definition 9 (Optimized for Small Networks).** For a family of weight-selecting mechanisms \( \mathcal{C} \) and \( M \in \mathcal{C} \), we say that \( M \) is optimized for small networks within \( \mathcal{C} \) if the following holds: If \( M' \in \mathcal{C} \) is leximin-better than \( M \) on an instance \( (G, S, v^*) \), there exists a strict subgraph \( H \) of \( G \) with \( v^* \in V(H) \) such that \( M \) dominates \( M' \) on \( H \) for target node \( v^* \).

While being optimized for small networks is weaker (is implied by) leximin-optimality, the bar is still high, as the next observation shows.

**Proposition 1.** In a family of weight-selecting mechanisms, at most one mechanism is optimized for small networks.

**Proof.** Suppose for contradiction that in a family \( \mathcal{C} \), two distinct mechanisms \( M_1, M_2 \in \mathcal{C} \) are optimized for small networks. Choose a smallest instance \( (G, S, v^*) \) (in terms of \( |V(G)| \)) on which they output different (sorted) weight vectors (break ties arbitrarily). Without loss of generality, let \( M_1 \) be leximin-better than \( M_2 \) on \( (G, S, v^*) \). Then, because \( M_2 \) is optimized for small networks, there must exist an instance \( (H, T, v') \) on which \( M_2 \) is leximin-better than \( M_1 \), where \( H \) is a strict subgraph of \( G \). This is a contradiction as it violates the minimality of \( |V(G)| \).

We now design an intuitive weight-selecting mechanism, and show that it is optimized for small networks within the family \( \mathcal{M}^{\text{NH}} \). A key idea behind the mechanism is due to Conitzer et al. [10], who propose a method of identifying certifiably legitimate nodes in a network in the presence of false-name manipulations. In their more general setting, this is a tricky problem, but in our setting it boils down to a simple observation: \( v \) could possibly be a fake node created by \( u \) if and only if removing \( u \) disconnects \( v \) from \( v^* \).

Let \( F(u) \), called the lobe of \( u \), be the set of all nodes that become disconnected from \( v^* \) by removing \( u \). As a convention, \( u \notin F(u) \), and \( F(v^*) \) is undefined. Now, node \( v \) is certifiably legitimate if \( v \notin F(u) \) for any \( u \in V \setminus \{v^*\} \). In other words, \( v \) should remain connected to \( v^* \) after removing any single node. Equivalently, it should either be a direct neighbor of \( v^* \), or be 2-vertex-connected to \( v^* \). Suppose we can weight all certifiably legitimate voters equally.

The following lemma helps us deal with the remaining voters.

**Lemma 1.** A symmetric weight-selecting mechanism satisfying the no harm axiom cannot assign a positive weight to any node in the lobe of a voter.

**Proof.** Suppose for contradiction that node \( v \in F(u) \) receives weight \( \delta > 0 \), where \( u \) is a voter. Suppose all nodes in the graph (including those in \( F(u) \)) are real. As the nodes in \( F(u) \) are only connected to the remaining network through \( u \), under a false-name manipulation \( u \) can create \( N \) copies of \( F(u) \) that are attached to \( u \) in an identical way to how \( F(u) \) is attached. The no harm axiom implies that \( v \) still receives weight at least \( \delta \), and by symmetry, now so does each of its \( N \) copies. However, this is infeasible when \( N > 1/\delta \) as the weights must sum to 1.

**ALGORITHM 1: Mechanism LEGIT**

**Data:** Social network \( G \), set of voters \( S \), central node \( v^* \)

**Result:** Weight vector \( w = (w_v)_{v \in S} \)

\[
\forall u \in V \setminus \{v^*\}, F(u) \leftarrow \{t \in V \setminus \{u\} | t \text{ is not connected to } v^* \text{ in } G \setminus \{u\}\};
\]

\[
L \leftarrow \{v \in V \setminus \{v^*\} | (\exists u \in V \setminus \{v^*\}, v \in F(u) \land (F(v) \cup \{v\}) \cap S \neq \emptyset)\};
\]

\[
\forall v \in S, w_v \leftarrow 0;
\]

**for** \( v \in L \) **do**

\[
\text{if } v \in S \text{ then }
\]

\[
w_v \leftarrow 1/|L|;
\]

**else**

\[
T \leftarrow F(v) \cup \{v\};
\]

\[
w_{\text{rec}} \leftarrow \text{LEGIT}^+(G_T, S \cap T, v);
\]

\[
\text{for } u \in F(v) \cap S \text{ do } w_u \leftarrow w_{\text{rec}}, 1/|L|;
\]

**end**

**return** \( w = (w_v)_{v \in S} \);

While Lemma 1 requires us to discard all nodes in the lobe of a voter, it does not prevent us from distributing the weight that a certifiably legitimate non-voter would have received (had it been a voter) to the nodes in its lobe. In fact, such distribution is necessary for the weights to sum to 1 when all voters reside in lobes of other nodes. A natural way is to apply this procedure recursively in each such lobe. This leads to our mechanism, which we call \( \text{LEGIT}^+ \) because it recursively passes legitimacy to nodes. It is presented as Algorithm 1. Crucially, we only recursively apply the mechanism to a lobe if it has a voter. Note that our mechanism assigns a positive weight to the maximal set of nodes subject to Lemma 1. This leads us to the next result.

**Lemma 2.** If a node receives zero weight under \( \text{LEGIT}^+ \), it receives zero weight under all mechanisms in \( \mathcal{M}^{\text{NH}} \).

We are now ready for the main result of this paper.

**Theorem 5.** \( \text{LEGIT}^+ \) is optimized for small networks within the family \( \mathcal{M}^{\text{NH}} \).

\footnote{Using Menger’s theorem [20], being 2-vertex-connected to \( v^* \) is equivalent to having two vertex-disjoint paths to \( v^* \).}
Proof. We first show that $\text{LEGIT}^{+} \in \mathcal{M}_{\text{NH}}$. When an opinionated node $v$ (a voter in the true instance) performs a false-name manipulation, $\text{LEGIT}^{+}$ either keeps the weights of all original voters invariant (if $v$ remains a voter after the manipulation), or distributes the weight of $v$ among the voters in its lobe (if $v$ does not vote after the manipulation). As the real nodes in the lobe of $v$ previously had zero weight, in both cases no real voter other than $v$ loses weight due to the manipulation. Also, $\text{LEGIT}^{+}$ is symmetric by definition.

Before proving that $\text{LEGIT}^{+}$ is optimized for small networks within $\mathcal{M}_{\text{NH}}$, we need to prove a special structure of the lobes.

**Lemma 3.** For two distinct nodes $v_1$ and $v_2$, we have i) $v_1 \in F(v_2)$ implies $F(v_1) \subseteq F(v_2)$, ii) $\emptyset \neq F(v_1) \subseteq F(v_2)$ implies $v_1 \in F(v_2)$, and iii) $F(v_1) \cap F(v_2) = \emptyset$, $F(v_1) \neq F(v_2)$.

Proof. For part i), $v_1 \in F(v_2)$ implies that all paths from $v^* \rightarrow v_1$ pass through $v_2$. For any $t \in F(v_1)$, all paths from $v^*$ to $t$ must pass through $v_1$. But since the path segment from $v^*$ to $v_1$ must involve $v_2$, it follows that all paths from $v^*$ to $t$ involve $v_2$ as well. Hence, $t \in F(v_2)$ for all $t \in F(v_1)$, i.e., $F(v_1) \subseteq F(v_2)$.

For part ii), since $F(v_1) \neq \emptyset$, there exists a direct neighbor of $v_1$ in $F(v_1)$. Call it $t$. Then, we have $t \in F(v_2)$. However, after removal of $v_2$, $t$ and $v_1$ should belong to the same connected component, and $t$ and $v^*$ belong to different connected components. Hence, removal of $v_2$ also disconnects $v_1$ from $v^*$. Thus, $v_1 \notin F(v_2)$, as required.

For part iii), if $F(v_1) \cap F(v_2) = \emptyset$, we are done. Else, let $v \in F(v_1) \cap F(v_2)$. Then, every path from $v'$ to $v$ must include both $v_1$ and $v_2$. Choose an arbitrary path, and without loss of generality, let it be composed of path $P_1$ from $v'$ to $v_1$, path $P_2$ from $v_1$ to $v_2$, and path $P_3$ from $v_2$ to $v$. If there exists a path from $v^*$ to $v_2$ that does not include $v_1$, then combining that with $P_3$ would give us a path from $v^*$ to $v$ that does not include $v_1$, which is a contradiction. Hence, there exists no path from $v'$ to $v_2$ that does not include $v_1$, i.e., $v_2 \in F(v_1)$. By part i), it implies $F(v_2) \subseteq F(v_1)$, i.e., $F(v_1) \cap F(v_2) = \emptyset$, as required. (Proof of Lemma 3)

We now prove a seemingly weaker guarantee, but later show that it is sufficient for our purpose.

**Lemma 4.** For a mechanism $M \in \mathcal{M}_{\text{NH}}$, if $M$ is leximin-better than $\text{LEGIT}^{+}$ on an instance $(G, S, v^*)$, then $\text{LEGIT}^{+}$ is leximin-better than $M$ on an instance $(H, T, v^*)$, where $H$ is a strict subgraph of $G$, and $T \subseteq V(H) \setminus \{v^*\}$.

Proof. Consider the instance $(G, S, v^*)$. Since $M$ returns a different weight vector than $\text{LEGIT}^{+}$ on this instance, there must exist a vertex $u_1$ such that its weight under $M$ (denoted $w_{M}^{u_1}$) is less than its weight under $\text{LEGIT}^{+}$ (denoted $w_{\text{LEGIT}^{+}}^{u_1}$). Consider the maximal chain $u_1, u_2, \ldots, u_k$ such that $u_i \in F(u_{i+1})$ for $i \in \{1, \ldots, k-1\}$. From part (i) of Lemma 3, we immediately have $F(u_i) \subseteq F(u_{i+1})$ for $i \in \{1, \ldots, k-1\}$. Let $U = \{u_1, \ldots, u_k\}$. Next, we show a technical condition.

For $v \notin U$, $u_1 \notin F(v)$. (*)

This is intuitively clear: Because lobes are either contained in one another or completely disjoint (by Lemma 3), if $u_1 \in F(v)$, then either $F(v)$ contains even the outermost lobe $F(u_1)$, or there exists an $i \in \{2, \ldots, k\}$ such that $F(u_{i-1}) \subset F(v) \subset F(u_i)$. In that case, we should be able to extend the maximal chain, which is a contradiction. Proving this formally is a bit tricky.

In order to prove (*), suppose for contradiction that $u_1 \in F(v)$. First, since $v$ alone could create a chain of length 2, we have $k \geq 3$.

2. Hence, $u_1 \in F(u_k) \neq \emptyset$. Now, due to part (i) of Lemma 3, $u_1 \in F(v)$ implies $F(u_1) \subseteq F(v)$. Choose $i$ to be the largest integer such that $F(u_i) \subseteq F(v)$. If $i = k$, then $F(u_k) \neq \emptyset$ and $F(u_k) \subseteq F(v)$ implies $u_k \in F(v)$ by part (ii) of Lemma 3. This means we could have extended the chain by adding $u_{k+1} \rightarrow v$, which is a contradiction. Hence, $i < k$. We first show that $u_i \notin F(v)$. If $i = 1$, this is assumed, and if $i \geq 2$, it follows from $F(u_i) \subseteq F(v)$, $F(u_i) \neq \emptyset$, and part (ii) of Lemma 3. Next, we show that $v \in F(u_{i+1})$. Note that $u_i \in F(v) \cap F(u_{i+1})$. Hence, the intersection is not empty. Further, $F(u_{i+1}) \subseteq F(v)$. Hence, by part (iii) of Lemma 3, we have $F(v) \subseteq F(u_{i+1})$. Further, $u_i \notin F(v)$. Hence, $F(v) \neq \emptyset$. Thus, by part (ii) of Lemma 3, we have $v \in F(v)$.

Thus, we have proved that $u_i \in F(v)$ and $v \in F(u_{i+1})$, which is a contradiction because it means we could have extended the chain by inserting $v$ between $u_i$ and $u_{i+1}$. Thus, for $v \notin U$, $u_1 \notin F(v)$.

This completes the proof of (*).

Next, we perform a sequence of operations to transform the instance. In each step, we find an arbitrary node $v \notin U$ such that $F(v) \neq \emptyset$. By (*), $u_1 \notin F(v)$. We remove the vertices in the lobe $F(v)$, and make $v$ a voter (if that was not already the case). Note that this operation is the reverse of a false-name manipulation by $v$. Hence, by the no harm axiom, the weight of $u_1$ under $M$ should not increase during this operation. On the other hand, it is easy to check that the weight of $u_1$ stays invariant under $\text{LEGIT}^{+}$ during this operation. Hence, even in the resulting instance, the weight of $u_1$ under $M$ is less than its weight under $\text{LEGIT}^{+}$.

We continue these operations until we cannot find a node $v \notin U$ such that $F(v) \neq \emptyset$. Let the final instance be denoted $(H, T, v^*)$. In the instance $(H, T, v^*)$, suppose $\text{LEGIT}^{+}$ assigns zero weight to $l$ voters. Then, by Lemma 2, $M$ must also assign zero weight to at least $l$ voters. Also, under $\text{LEGIT}^{+}$ the $(l+1)\text{st}$ smallest weight is the weight of $u_1$, which is $w_{u_1}$, since $u_1$ is contained in every non-empty lobe that remains in $H$. In contrast, under $M$ the $(l+1)\text{st}$ smallest weight is at most the weight of $u_1$, which is at most $w_{u_1}$. Hence, $\text{LEGIT}^{+}$ is leximin-better than $M$ on $(H, T, v^*)$.

By our construction, $H$ is already a subgraph of $G$. To see why it is a strict subgraph of $G$, recall that $M$ was leximin-better than $\text{LEGIT}^{+}$ on the original instance $(G, S, v^*)$. Hence, we must have made at least one “reverse false-name manipulation” operation, which must have resulted in a strictly smaller subgraph. (Proof of Lemma 4)

Finally, to prove that $\text{LEGIT}^{+}$ is optimized for small networks in $\mathcal{M}_{\text{NH}}$, consider a different mechanism $M \in \mathcal{M}_{\text{NH}}$, and suppose $M$ is leximin-better than $\text{LEGIT}^{+}$ on an instance $(G, S, v^*)$. Then, by Lemma 4 there exists an instance $(H, T, v^*)$ on which $\text{LEGIT}^{+}$ is leximin-better than $M$ and where $H$ is a strict subgraph of $G$. Among all such instances, choose one with the smallest number of nodes in $H$. We show that $\text{LEGIT}^{+}$ dominates $M$ on network $H$ for target node $v^*$. To see this, take a subset of voters $Q \subseteq V(H) \setminus \{v^*\}$. If $M$ is leximin-better than $\text{LEGIT}^{+}$ on $(H, Q, v^*)$, then by Lemma 4 there exists another instance $(H', Q', v^*)$ on which $\text{LEGIT}^{+}$ is leximin-better than $M$ and where $H'$ is a strict subgraph of $H$ (and therefore, of $G$). However, this violates the minimality of the number of nodes in $H$ in our choice of the instance $(H, T, v^*)$. Hence, $M$ must not be leximin-better than $\text{LEGIT}^{+}$ on $(H, Q, v^*)$, as required. (Proof of Theorem 5)

**4. COMPUTATIONAL COMPLEXITY**

Let us begin with RANDOMWALK. Andersen et al. [2] show that aggregating binary recommendations under RANDOMWALK amounts to solving a single system of linear equations $Ax = b$, where the LHS matrix $A$ is $n \times n$ ($n = |V|$ is the number
of nodes) and the RHS vector $b$ is $n \times 1$. Solving this system can take, even with recent exact solvers, $O((|V|^{1.5} + (|V| + |E|))$ time [11]. Implementing \textsc{RandomWalk} as a weight-selecting mechanism is computationally even more difficult. We need to solve one system of linear equations for each voter, which can take $O(|V|^{2.5} \cdot (|V| + |E|))$ time [11]. Let us now consider \textsc{Legit$^+$}. Arguably, it is harder to describe than \textsc{RandomWalk}, and Algorithm 1 is more intricate than simply solving a collection of linear systems. More specifically, in the first step of Algorithm 1 simply computing $F(u)$ for every node $u$ would naively take $O(|V| + |E|)$ time. Surprisingly, we show that there exists a more efficient implementation that computes the weights under \textsc{Legit$^+$} in merely $O(|V| + |E|)$ (linear) time. This implementation uses as a subroutine the remarkable linear time algorithm by Hopcroft and Tarjan [15] for finding biconnected components in a graph. A biconnected component (or a block) is a maximal $2$-vertex-connected subgraph. Nodes that belong to multiple blocks (i.e., whose removal disconnects the graph) are called cut vertices or articulation points. A connected graph $G$ decomposes into a block-cut tree $T$ whose vertices are the blocks and the articulation points of $G$, and a block $B$ and an articulation point $u$ are connected if $u \in B$.

Let $A$ denote the set of articulation points of $G$, and $B_u$ denote the set of blocks of $G$ containing $u$. First, $u$ has a non-empty lobe $F(u)$ if and only if $u \in A$. Next, if $u \in A$, the lobe $F(u)$ can be computed as follows. Remove the vertex of $T$ representing $u$, which disconnects $T$ into connected components, one of which contains all blocks containing $u$. The set of nodes in the blocks contained in every other connected component of the tree (except $u$ itself) constitute $F(u)$. This key observation leads us to the linear time implementation of \textsc{Legit$^+$} presented as Algorithm 2. The proof of its correctness and running time analysis are presented in the appendix.

\textbf{Theorem 6.} Weights under mechanism \textsc{Legit$^+$} can be computed in $O(|V| + |E|)$ time.

\section{Experiments}

We compare \textsc{Legit$^+$} with two baseline mechanisms: \textsc{Legit} and \textsc{RandomWalk}. We define weight-selecting mechanism \textsc{Legit} as the simpler version of \textsc{Legit$^+$} that assigns equal weight to all certifiably legitimate voters, but does not apply the procedure recursively within the lobes of certifiably legitimate non-voters. Thus, comparison with \textsc{Legit} indicates the gain from recursively applying \textsc{Legit$^+$} within the lobes of certifiably legitimate non-voters. We note that \textsc{Legit$^+$} is expected to (though theoretically not guaranteed to) outperform \textsc{RandomWalk}, because \textsc{RandomWalk} is not designed to assign uniform weights.

We perform experiments using 16 real-world social networks from the KONECT project [18]. The number of nodes and edges in these networks vary from 23 to 26,475, and from 78 to 146,385, respectively.\footnote{Running experiments on the larger datasets was infeasible due to the prohibitive running time of \textsc{RandomWalk}.} For each network $G$, we sample the target node $v^*$ uniformly at random. For each pair $(G, v^*)$, we determine the set of voters by making each node in the network a voter independently with probability $p_{vote}$. We use both low values (from 0.01 to 0.09 in increments of 0.02) and high values (from 0.1 to 0.9 in increments of 0.2) of $p_{vote}$, representative of varying levels of voter engagement. For each network and each of 10 values of $p_{vote}$, we choose 100 random target nodes, and for each target node, choose 100 random subsets of voters. In the results presented below, we compare \textsc{Legit$^+$} with \textsc{Legit} and \textsc{RandomWalk} across the simulations for each network. To solve the linear system in \textsc{RandomWalk}, we use Matlab’s \texttt{mldivide} operator, and to find the biconnected components in \textsc{Legit$^+$}, we use the MatlabBGL library\footnote{https://www.cs.purdue.edu/homes/dgleich/packages/matlab_bgl/}.

Figure 2(a) shows a log-log plot of the running time of all three mechanisms \textsc{Legit$^+$} as magenta diamonds, \textsc{Legit} as red circles, and \textsc{RandomWalk} as blue stars as a function of the number of nodes in the network. The experiments were performed on a dual-core machine with 3.10 GHz processors and 8 GB RAM. While \textsc{Legit} is trivially faster than \textsc{Legit$^+$} (as it requires a strictly less number of operations), the difference is not significant. On the other hand, while \textsc{RandomWalk} is slightly faster than \textsc{Legit$^+$} on smaller networks, \textsc{Legit$^+$} is significantly faster on networks with more than 200 nodes. This is consistent with our result from Section 4 that the worst-case complexity is significantly lower for \textsc{Legit$^+$} than for \textsc{RandomWalk} (linear versus super-quadratic). Across the entire experiment, \textsc{Legit$^+$} ran 13 times faster than \textsc{RandomWalk}, and only about 3 times slower than \textsc{Legit}.

\begin{algorithm}[H]
\caption{\textsc{Legit$^+$} in linear time}
\begin{algorithmic}
\State \textbf{Data:} Social network $G$, set of voters $S \subseteq V(G) \setminus \{v^*\}$, central node $v^* \in V(G)$\State \textbf{Result:} Weight vector $w = (w_u)_{u \in S}$\State $B \leftarrow$ set of biconnected components of $G$;\State $A \leftarrow$ set of articulation points of $G$;\State $\forall u \in V, B_u \leftarrow \{B \mid u \in B\}$;\State / $\times B, A$, and $\{B_u\}_{u \in V}$ are computed using the linear time algorithm from [15] */\State $\forall u \in A \setminus S, VL_u \leftarrow$ false;\State $\forall u \in S, w_u \leftarrow 0$;\State \textbf{procedure} voting\_lubes($v^*, B^*$)\State \hspace{0.5cm} $b \leftarrow$ false;\State \hspace{0.5cm} \textbf{for} $B \in B_v \setminus B^*$ \textbf{do}\State \hspace{1cm} \textbf{for} $u \in B \setminus \{v^*\}$ \textbf{do}\State \hspace{1.5cm} \textbf{if} $u \in S$ \textbf{then}\State \hspace{2cm} $b \leftarrow$ true;\State \hspace{1.5cm} \textbf{else} if $u \in A$ \textbf{then}\State \hspace{2cm} $VL_u \leftarrow$ voting\_lubes($u, \{B\}$);\State \hspace{2cm} \textbf{if} $VL_u$ then $b \leftarrow$ true;\State \hspace{1cm} \textbf{end}\State \hspace{0.5cm} \textbf{end}\State \textbf{return} $b$;\State \textbf{procedure} weight\_helper($v, T, B^*$)\State \hspace{0.5cm} $L \leftarrow \{u \in V \mid \{u \in B \setminus B^* \wedge (u \in S \lor (u \in A \land VL_u))\}$;\State \hspace{0.5cm} $N \leftarrow |L|$;\State \hspace{0.5cm} \textbf{for} $u \in L$ \textbf{do}\State \hspace{1cm} \textbf{if} $u \in S$ \textbf{then}\State \hspace{2cm} $w_u = T/N$;\State \hspace{1cm} \textbf{else} if $u \in A$ \textbf{then}\State \hspace{2cm} weight\_helper($u, T/N, \{B\})$;\State \hspace{1cm} \textbf{end}\State \hspace{0.5cm} \textbf{end}\State \end{algorithmic}
\end{algorithm}
In the remaining figures, we only plot two lines: one that compares \textsc{Legit} with \textsc{Legit} (with red circles), and one that compares \textsc{Legit} with \textsc{RandomWalk} with blue stars).

Our next goal is to determine which mechanism outputs a more uniform weight vector. Lacking an objective definition of uniformity, we use three metrics: i) leximin comparison as used in our theoretical results in Section 3, ii) the percentage of voters discarded, i.e., assigned zero weight to (the lower, the better), iii) the \((L^2)\)-distance from the uniform weight vector, which is equal to the variance of the weight vector (the lower, the better).

Figure 2(b) shows that \textsc{Legit} is leximin-better than both \textsc{Legit} and \textsc{RandomWalk} in more than 50% simulations in each network. In fact, it is leximin-better than \textsc{Legit} (resp. \textsc{RandomWalk}) in more than 75% (resp. 85%) simulations in all but one (resp. two) networks. Superior empirical performance in such large networks nicely complements our theoretical result (Theorem 5), which indicates that \textsc{Legit} should be superior in small networks in general.

Next, while Lemma 2 ensures that \textsc{Legit} discards the smallest subset of voters subject to the no harm axiom, Figure 2(c) shows that \textsc{Legit} and \textsc{RandomWalk} can discard up to 60% and 20% more voters, respectively, than \textsc{Legit} (about 30% and 10%, respectively, on average across networks).

Finally, comparing variance of the returned weight vector, Figure 2(d) shows that \textsc{Legit} performs better than both \textsc{Legit} and \textsc{RandomWalk} in more than 50% simulations in each network. Further, it outperforms \textsc{Legit} in at least 69% simulations in all but one network, and \textsc{RandomWalk} in at least 89% simulations in all but one network.

So far we have focused on the setting where the opinions of voters are subjective, and the goal is to find a weight vector as close to uniform as possible. We now present empirical results for a slightly different setting in which there exists a binary (0/1) ground truth, and the goal is to pinpoint it by aggregating binary opinions of voters, each of which is “correct” with probability \(p_{ac} > 0.5\). The accuracy of a weight-selecting mechanism on an instance \((G, S, v^*)\) is the probability that the mechanism assigns higher total weight to voters with the correct opinion than to voters with the incorrect opinion. While we do not have theoretical results for this setting, we can evaluate the mechanisms empirically. For \(p_{ac}\), we use both low values (0.51 to 0.59 in increments of 0.02) and high values (0.6 to 0.9 in increments of 0.1).

Figure 3 shows that \textsc{Legit} achieves better accuracy than both \textsc{Legit} and \textsc{RandomWalk} in more than 50% simulations in each network. Also, note that \textsc{Legit} achieves better accuracy than \textsc{RandomWalk} in at least 70% simulations in all but one network.

6. DISCUSSION

Recall the median-of-medians rule from the introduction: the recommendation is the median of the opinions of the target agent’s friends, and for a friend who does not provide an opinion, we construct one by taking the median of his friends’ opinions, and so on. In conjunction with the weighted median aggregation rule (as in Theorem 1), \textsc{Legit} can be seen as a similar rule, “median-of-medians for legitimate nodes”; instead of taking the median of friends’ opinions, take the median of the opinions of (certifiably) legitimate nodes, and for such nodes that do not provide an opinion, construct one recursively from opinions in their lobes.

We uniquely characterize \textsc{Legit} within the family of symmetric weight-selecting mechanisms satisfying the no harm axiom. We show this axiom to be closely related, but in the general setting incomparable, to false-name-proofness. The no harm axiom is only defined for weight-selecting mechanisms. It remains to be seen whether we can pinpoint an overall recommendation mechanism that uses \textsc{Legit} (e.g., median-of-median for legitimate nodes) within the more general family of false-name-proof mechanisms.

Importantly, in this paper we consider the uniform weight vector as idealistic. In the context of aggregating subjective opinions into a personal recommendation for the target node, this only makes sense in the absence of knowledge of correlation among user preferences (e.g., homophily of opinions). However, note that Lemma 1 provides a necessary condition for satisfying the no harm axiom — in the form of having to assign zero weight to specific nodes — even in the presence of homophily. Given a model of homophily, we must start by assigning zero weight to such nodes. Weighting the remaining nodes to maximally align the recommendation with the target node’s preference is still a difficult problem. Fortunately, it can be shown that choosing the remaining weights as a function of the vertex-connectivity of a node to the target node is sufficient to guarantee the no harm axiom. However, this approach is likely to be suboptimal. An immediate next step is to design better ways of incorporating homophily subject to the no harm axiom.

An interesting direction for future research is to study stronger manipulations. For example, \textsc{Legit} does not prevent group false-name manipulations or manipulations where nodes may delete their existing edges with other real nodes. Can we effectively prevent them? While \textsc{RandomWalk} is group false-name-proof, it can be shown that it is not optimized for small networks among symmetric group false-name-proof mechanisms. Does there exist such a mechanism (recall that there can be at most one)?

False-name manipulations are an increasingly serious concern in social networks, especially with the effortless accessibility and
increasing popularity of automated tools for creating fake accounts [24]. Given the difficulty of distinguishing fake accounts from real ones, we believe that the study of false-name-proofness is the key to building the next generation of reliable recommendation systems.

Acknowledgments

We are thankful for support from NSF under awards IIS-1527434, IIS-0953756, CCF-1101659, and CCF-1337215, ARO under grants W911NF-12-1-0550 and W911NF-11-1-0332, ERC under StG 639945 (ACCORD), a Guggenheim Fellowship, and a Feodor Lynen research fellowship of the Alexander von Humboldt Foundation. This work was done in part while Conitzer was visiting the Simons Institute for the Theory of Computing.

REFERENCES

[34] M. Yokoo. Characterization of strategy/false-name proof
combinatorial auction protocols: Price-oriented,
rationing-free protocol. In Proceedings of the 18th
International Joint Conference on Artificial Intelligence
combinatorial auction protocol against false-name bids.
[36] M. Yokoo, Y. Sakurai, and S. Matsubara. The effect of
false-name bids in combinatorial auctions: New fraud in
internet auctions. Games and Economic Behavior,

APPENDIX

A. LINEAR TIME IMPLEMENTATION

Intuitively, Algorithm 2 works as follows. Using the linear time
algorithm of Hopcroft and Tarjan [15], we compute the set of
blocks (biconnected components) $B$, the set of articulation points
$A$, the set of nodes in each block $B \in B$, and for each node $u \in V$,
the set of blocks $B_u$ containing $u$. These are the first three lines of
Algorithm 2.

Next, recall the description of $\text{LEGIT}^\ast$: If there are $N$ certifiably legitimate nodes that are either voting or have a voter in their lobe, we reserve a weight of $1/N$ for each of them. The voters receive their $1/N$ weight, and the weight reserved for a non-voter is distributed recursively within its lobe. How do we identify which certifiably legitimate non-voters have a voter in their lobe? One way is to simply gather that information when we make a recursive call. However, this means we can only know $N$ after all recursive calls are made, and we then need to update the weight of all voters accordingly. It can be shown that such an implementation would not run in linear time because the weight of a voter $u$ will be updated in the recursive call to every node $v$ such that $u \in F(v)$.

To circumvent this issue, we design the helper function $\text{voting\_lobes}$ and create the boolean array $VL$. Formally, we define $VL_u$ for every non-voter articulation point $u$ (i.e., non-voter that has a non-empty lobe), and want it to be true if and only if $F(u)$ contains a voter. By calling $\text{voting\_lobes}$ on $u$, we wish to set the correct value of $VL_u$ for every $u \in A \setminus S$ in linear time.

The remaining task is now very simple. Helper function $\text{weight\_helper}$ is designed to distribute a total weight of $T$ in the lobe of a node $v$. For the purpose of this algorithm, let $F(v^\ast) = V \setminus \{v^\ast\}$. The (empty or singleton) set $B^\ast$ used as an argument to both helper functions can be described intuitively as follows. Remove the vertex representing $v$ from the block cut tree so that exactly one block containing $v$ is in the same connected component of the (now) disconnected tree as all blocks containing $v^\ast$. $B^\ast$ is the singleton set containing that unique block. The function first identifies the list $L$ of certifiably legitimate nodes within the lobe of $v$ that are either voters or have a non-empty lobe containing a voter. If $N \equiv |L|$, it assigns a weight $T/N$ to each voter in $L$, and recursively distributes a total weight of $T/N$ in the lobe of each non-voter in $L$.

We are now ready to prove the correctness of this implementation.

Correctness: First, recall from Section 4 that the lobe of a node $v$, i.e., $F(v)$ is constructed as follows. If $v \notin A$, $v$ does not have a lobe. If $v \in A$, remove the vertex of the block-cut tree representing $v$, and from the resulting components, remove the one that contains (all) blocks containing $v^\ast$. All nodes belonging to blocks contained in the remaining tree (except $v$) constitute $F(v)$. Among these nodes, the ones that are certifiably legitimate (according to $v$) within the lobe of $v$ are the ones that share a block with $v$.

Note that when the helper functions $\text{voting\_lobes}$ or $\text{weight\_helper}$ are called recursively on a node $v$, the block $B$ through which we reached $v$ is the only block containing $v$ that will lie in the connected component of the block-cut tree containing all blocks with $v^\ast$ after $v$ is removed. Hence, to identify certifiably legitimate nodes within the lobe of $v$, we need to look at the remaining blocks containing $v$. That is, in the construction of $L$ in $\text{weight\_helper}$,

$$\{u \in V | u \in B \in B_v \setminus B^\ast\}$$

is precisely the set of certifiably legitimate nodes within the lobe of $v$. We further require that a node $u$ should either be a voter ($u \in S$), or it must have a non-empty lobe ($u \in A$) and that lobe should have a voter (which is true if $VL_u$ is true). Thus, $L$ is precisely the set of certifiably legitimate nodes within the lobe of $v$ that are either voting or have a voter in their lobe. The correctness of $\text{weight\_helper}$ now follows trivially.

Similarly, in $\text{voting\_lobes}$, when we iterate over all $u \in B \setminus \{v\}$ where $B \in B_v \setminus B^\ast$, we are precisely iterating over the set of certifiably legitimate nodes within the lobe of $v$. We set the answer $b$ to be true if and only if there exists such a node that is a voter or has a voter in its own lobe (identified by calling the algorithm recursively).

This completes the correctness analysis.

Running time analysis: The linear running time of the first three steps of Algorithm 2 follows from the result by Hopcroft and Tarjan [15]. We now prove that both $\text{voting\_lobes}$ and $\text{weight\_helper}$ run in linear time when called from the main algorithm. To do that, we attribute the cost of each step within the two helper functions to a node in the network in a way that every node in the network is attributed at most a constant number of steps.

We begin with $\text{voting\_lobes}$. Attribute the cost of running the inner for loop (except for the time spent within the recursive call to $\text{voting\_lobes}$) to the node $u$. Attribute the remaining cost of the function (e.g., the first and the last step) to the node $v$ that the function is called on. To show that each node $u$ is attributed at most a constant cost, we need to prove that $u$ will only appear in the inner for loop of the procedure at most once. That is, we need to prove that $u$ is certifiably legitimate only within the lobe of at most one node $v$. This follows from our structural result Lemma 3. Suppose for contradiction that $u$ is certifiably legitimate within the lobes of $v_1$ and $v_2$, then $u \in F(v_1)$ and $u \in F(v_2)$. However, then we either have $v_1 \in F(v_2)$ or $v_2 \in F(v_1)$. In the former case, $u$ will not be certifiably legitimate within the lobe of $v_2$, and in the latter, not within the lobe of $v_1$.

We now prove that helper function $\text{weight\_helper}$ runs in linear time. Since $u$ is certifiably legitimate within the lobe of at most one node $v$, iterating over $\{u \in V | u \in B \in B_v \setminus B^\ast\}$ (which we have already shown to be the set of certifiably legitimate nodes within the lobe of $v$) for all nodes $v$ will only iterate over every node $u$ at most once. Hence, the aggregate cost of constructing $L$ over all calls to $\text{weight\_helper}$ is linear. Finally, attributing the for loop iteration on the node $u$ to itself and using the previous result, we can show that every $u$ is attributed at most a constant cost. Hence, $\text{weight\_helper}$ runs in linear time overall.

This completes our running time analysis.
Here, we present detailed results for two of the 18 real-world networks. The first network, ego-facebook, is where LEGIT\(^+\) performs the worst compared to RANDOMWALK (Figure 4). This is the network for which the performance of LEGIT\(^+\) over RANDOMWALK suddenly drops in Figure 3. Ego-facebook is a subgraph of the Facebook social network, and has 2,888 nodes and 2,981 edges. The second network, opsahl-powergrid, is where LEGIT\(^+\) performs the worst compared to RANDOMWALK (Figure 4). This network is the one after ego-facebook in Figures 2 and 3. Opsahl-powergrid is a power grid network in the United States, and has 4,941 nodes and 6,594 edges. In contrast to the figures presented in Section 5, in the figures presented in this section we have \(p_{vote}\) on the x-axis. Each data point shows the aggregated result over all simulations for the specific network and specific value of \(p_{vote}\).

In each figure, subfigures (a), (b), (c), (d), and (f) respectively show the running time of the two mechanisms (LEGIT\(^+\) as blue stars and RANDOMWALK as red circles), the frequency of LEGIT\(^+\) being leximin-better than RANDOMWALK, the additional percentage of voters discarded by RANDOMWALK as compared to LEGIT\(^+\), the frequency of the weight vector returned by LEGIT\(^+\) having less variance than the one returned by RANDOMWALK, and the frequency of LEGIT\(^+\) achieving better accuracy than RANDOMWALK in the probabilistic setting with a binary ground truth and binary opinions.

We present two additional graphs for each network: subgraph (e) shows the actual variances of the weight vectors returned by the two mechanisms (LEGIT\(^+\) as blue stars and RANDOMWALK as red circles), and subgraph (g) shows the accuracy of the two mechanisms along with the accuracy of the optimal weight-selecting mechanism given by the uniform weight vector (LEGIT\(^+\) as blue stars, RANDOMWALK as red circles, and the uniform weight vector as black circles).

Note that in both cases, LEGIT\(^+\) runs substantially faster than RANDOMWALK. In the worst (ego-facebook) network, it performs moderately better than RANDOMWALK in terms of leximin comparison (which is the metric we focus on in our theoretical results), but nearly identical to RANDOMWALK in terms of the remaining metrics. Interestingly, while the variance of the weight vector returned by LEGIT\(^+\) is nearly identical to the variance of the weight vector returned by RANDOMWALK, the former is less than the latter in significantly more than 50\% of the simulations. On the other hand, in the best (opsahl-powergrid) network LEGIT\(^+\) shows substantial improvement over RANDOMWALK in all metrics we consider.

It is therefore our general conclusion that in our setting, LEGIT\(^+\) is never worse than RANDOMWALK, but can be significantly better in a number of metrics.
Figure 4: The worst network: ego-facebook
Figure 5: The best network: opsahl-powergrid