Better than PageRank: Hitting Time as a Reputation Mechanism

A thesis presented
by

Brandon Liu

To
Computer Science
in partial fulfillment of the honors requirement
for the degree of
Bachelor of Arts
Harvard College
Cambridge, Massachusetts

April 1, 2014

Abstract

In online multi-agent systems, reputation systems are needed to distinguish between trustworthy agents and potentially malicious or unreliable agents. A good reputation system should be accurate, resistant to strategic manipulations, and computationally tractable. I experimentally analyze the accuracy and manipulation-resistance of a reputation mechanism called personalized hitting time, and present efficient algorithms for its calculation. I present an alternate definition to hitting time that is amenable to Monte Carlo estimation, and show that it is linearly equivalent to the standard definition for hitting time. I present exact and approximation algorithms for computing personalized hitting time, and I show that the approximation algorithms can obtain a highly accurate estimate of hitting time on large graphs more quickly than an exact algorithm can find an exact solution. An experimental comparison of the accuracy of six reputation systems — global and personalized PageRank, global and personalized hitting time, maximum flow, and shortest path — under strategic manipulation shows that personalized hitting time is the most accurate reputation mechanism in the presence of a moderate number of strategic agents.

Acknowledgements

I would like to express my greatest appreciation to Professor David Parkes for his support as my teacher, mentor, and advisor. I am deeply indebted to him for critiquing my research, sharing his insights, reading over drafts of my thesis, and contributing to my growth as a young researcher. I absolutely could not have done this without him.

I thank Professor Sven Seuken for his research insights and encouragement. I thank Michael Crouse for his generosity with his time and computing resources. I thank Professors Michael Mitzenmacher and Yaron Singer for generously offering to be my thesis readers.

Lastly, I express my warmest gratitude to my mother, for her unconditional support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Transactions in marketplaces suffer from the problem of information asymmetry [1], because sellers have more information about the product for sale than the buyers. This can lead to adverse outcomes, as in the case when a buyer pays more for a product than its true value. It is precisely this economic phenomenon that underlies the common stereotype for used car salesmen: they are often suspected of misleading buyers into purchasing used cars that are not what they immediately seem.

When these transactions are conducted in person, there are some ways to mitigate this problem of information asymmetry. In the case of the used car salesman, one could verify that the salesman is employed by a reputable dealer, or bring an expert friend to inspect the used car. There are some social cues that one can rely on to establish trust and rapport. However, in the case of online marketplaces, this problem is exacerbated: A buyer often has no idea who the seller is, and has no way of verifying the product that they are purchasing.

1.1 Overview

In online multi-agent systems, *reputation mechanisms* are used to help agents determine which other agents they should trust, which facilitates the successful completion of transactions. Informally, reputation mechanisms are algorithms that aggregate available information in order to estimate and assign reputation scores to all the agents in a network. There is a variety information that can be incorporated into reputation algorithms. Past successful and unsuccessful transactions outcomes can be used to predict future transactions. Existing social relationships and similarity scores based on individual characteristics can also inform the likelihood of a successful transaction. In this thesis, I model this information with a single measure that I will call *trust*, which represents an agent's belief that another agent will successfully complete a transaction.

A good reputation mechanism should satisfy three criteria: accuracy, resistance to manipulation, and computational tractability. An accurate reputation mechanism allows agents to meaningfully

distinguish between reputable and disreputable agents in the marketplace. A reputation mechanism that is resistant to manipulation is able to prevent or minimize the degree to which agents can misreport information to the mechanism in order to exploit the mechanism to their own advantage, whether by improving their own rank or by diminishing the ranking of others. A computationally tractable reputation mechanism is feasibly computable on a data set of realistic size. Over the course of the thesis, I will formalize these notions, and examine six reputation mechanisms by these criteria.

1.2 Motivation

The need for good reputation mechanisms is motivated by a variety of online applications. Personalized search engines must process billions of web pages and determine which are the most relevant or trustworthy. Some of these web pages may attempt to strategically place content or links in order to obtain a higher ranking on search engine listings. Search engines like Google and Bing must simultaneously limit the impact of strategic behavior and return accurate search results in a computationally feasible manner. Efficient reputation mechanisms like PageRank are used for these purposes.

Online marketplaces and auction sites such as eBay and Amazon reduce fraud by third-party sellers through rating systems. Buyers and sellers can rate each other after each transaction, providing an incentive to transact honestly in order to obtain higher ratings. Under simple rating systems, agents may boost their own rating by creating fake accounts that report positive ratings. Under more sophisticated reputation mechanisms, this behavior does not improve one's own rating, thus incentivizing truthful behavior. One randomized controlled experiment on eBay showed a 8.1% increase in a buyer's willingness-to-pay when transacting with established sellers compared with new sellers [2].

Peer-to-peer file sharing, such as Napster or BitTorrent, can enhance distribution and transfer speeds of files by leveraging a distributed, decentralized network of computer systems. On such a network, there can exist free-rider problem, in which some strategic agents contribute less than they receive. A reputation mechanism in which nodes report positive and negative interactions with their peers can identify altruistic and selfish nodes. Resources or other benefits can be allocated according to the reputation mechanism, thus encouraging cooperative behavior.

1.3 Related Work

A variety of algorithms similar or related to reputation mechanism have been studies for various applications, including the efficient identification of web spam [3], searching for relevant web pages on the Internet [4], and reputation mechanisms for peer to peer networks [5, 6]. There has been a fair amount of research on reputation mechanisms that specifically limit the effect of the sybil

attack, in which strategic agents create false identities that improve their own reputation. Various reputation mechanisms, such as Sybilguard and Sybillimit, have been proposed [7, 8, 9] and numerous theoretical results on sybilproof strategies have been proven [10, 11, 12]. Other researchers, notably Altman and Tennenholtz, have taken an axiomatic approach to understanding and analyzing reputation mechanisms [13, 14, 15, 16]. For a survey of manipulation-resistant reputation mechanisms, see Josang et al. [17] or Friedman et al. [18].

1.4 Contribution

The overarching contribution of this thesis is to demonstrate that *hitting time* is a good reputation mechanism, by the above three criteria. The main contributions of this thesis are as follows.

- I present two definitions for hitting time: the standard definition that is based on the step length of a random walk, and an alternative definition based on probability. I prove that the values produced by these two definitions are linearly dependent, i.e., there exists a linear function from one set of values to the other.
- I present exact algorithms for computing hitting time quickly using matrix algebra for each of the two definitions. I characterize and compare the running times of the exact algorithms.
- I present three Monte Carlo approximation algorithms for estimating hitting time. I analyze and compare the Monte Carlo algorithms by accuracy and running time, and I identify the algorithm that yields the most accurate estimate of hitting time in a given amount of time. I demonstrate that for large graphs, the Monte Carlo approximation algorithms can obtain a very accurate estimate of hitting time in less time than an exact algorithm can obtain an exact solution.
- I compare six reputation mechanisms in a model of a multi-agent system and analyze their accuracy and resistance to manipulation. I examine the accuracy of the reputation mechanisms with and without strategic manipulations, and I conclude that hitting time finds the best compromise between accuracy and resistance to manipulation.

The thesis will be structured as follows. Chapter 2 introduces the notation and basic theory used throughout the thesis. Chapter 3 presents the alternative definition of hitting time, proves the theorem demonstrating its equivalence to the standard definition, and demonstrates that both definitions yield exact algorithms based on matrix algebra. Chapter 4 presents and characterizes three Monte Carlo algorithms for computing hitting time. Chapter 5 compares the running time and accuracy of the exact and approximation algorithms. Chapter 6 presents the experimental setup for the comparison of reputation mechanism, and discusses the results of the experiments. Chapter 7 concludes.

Chapter 2

Basic Theory

In this chapter, I present the notation and basic theory that will be used throughout the thesis.

2.1 Introductory Theory

Here I define some of the basic mathematical objects that we will deal with throughout this thesis. The core of this thesis depends on a weighted digraph, which is used to model our network of agents.

Definition 2.1 (Weighted Digraph)**.** *A weighted digraph $G = (V, E, w)$ is given by a set of nodes V that are connected by edges $(u, v) \in E$ where $u, v \in V$. Each edge has a weight associated with it, which is defined by $w(u, v) \in \mathbb{R}^+$.*

Many of the algorithms I use will be defined in terms of the *transition matrix*.

Definition 2.2 (Transition Matrix)**.** *Given a weighted digraph $G = (V, E, w)$, the transition matrix is a $|V| \times |V|$ square matrix P. The entries of the matrix are given by $P_{ij} = w(i, j)$, for $(i, j) \in E$ and $P_{ij} = 0$ otherwise.*

In other cases, algorithms will make use of a *standard random walk* on a trust graph.

Definition 2.3 (Standard Random Walk)**.** *Given a trust graph $G = (V, E, w)$, a standard random walk is a sequence of random variables $(X_t)_{t\geq 0}$ that follow the transition probabilities*

$$\mathbb{P}(X_{t+1} = j | X_t = i) = \begin{cases} \dfrac{w(i,j)}{\sum_{(i,j')\in E} w(i,j')} & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases} . \tag{2.1}$$

The standard random walk will normally be conditioned on an initial state $X_0 = i$ it is used. A standard random walk is also a Markovian process, satisfying the Markov property

$$\mathbb{P}(X_{t+1} | X_t, \ldots, X_0) = \mathbb{P}(X_{t+1} | X_t). \tag{2.2}$$

This is because the probability distribution of the future state X_{t+1} depends only on the present state X_t, and does not depend on any past states. I sometimes omit the subscript $t \geq 0$ for convenience when it is implied.

2.2 Graph Algorithms

Here, I present the algorithms that will later be interpreted in the context of reputation mechanisms. The algorithms are classified as global or personalized: A global algorithm will return a single vector of scores, whereas a personalized algorithm will return a vector of scores for each node in the graph. We will later see how these will be interpreted as global and personalized reputation scores.

2.2.1 PageRank

PageRank [4] is one of the most well-known graph algorithms, with a wide range of applications.

Definition 2.4 (Global PageRank)**.** *Given a weighted digraph $G = (V, E, w)$, let P be the transition matrix of G. Let the restart distribution R be the uniform distribution over all nodes such that $R = \frac{1}{|V|}\vec{1}$. Then the PageRank vector π is the stationary distribution of the Markov chain given by the transition probabilities in the matrix*

$$(1 - \alpha)P + \alpha(R \times \vec{1}), \tag{2.3}$$

where the second term is the outer product of R and the vector of ones. I use $\alpha = 0.15$ by convention [4].

This is a weighted form of the PageRank algorithm: whereas standard PageRank uses a transition matrix in which the entries $A_{ij} = 1/outdegree(i)$ for $(i, j) \in E$, here I take into account the weights of the out-edges.

Page et al. [4] also show that PageRank can be personalized, simply by modifying the restart distribution to always return to a specific page or set of pages.

Definition 2.5 (Personalized PageRank)**.** *Given a weighted digraph G, for each i, compute a PageRank vector $\pi^{(i)}$ for each node i in the same manner as the Global PageRank algorithm, but with a different restart distribution R_i for each i. For each i, I use $R_i = e_i$, where e_i is the vector of all zeros except for a one at the ith entry. The vectors $(\pi^i)_{i \in V}$ give the personalized PageRank score.*

The computational complexity for PageRank is dominated by the complexity of finding the stationary distribution of a finite Markov chain with $|V|$ states. This problem is equivalent to finding the left principal eigenvector of a $|V| \times |V|$ matrix, which takes $\mathcal{O}(|V|^3)$ time. As personalized PageRank must repeat this process for each $i \in V$, it takes $\mathcal{O}(|V|^4)$.

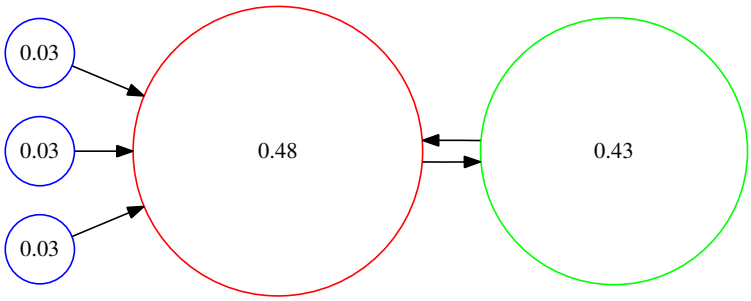


Figure 2.1: Example of PageRank on an unweighted digraph. The PageRank scores for the nodes are written inside each node.

As a simple numerical example, consider the directed graph in Figure 2.1. Intuitively, we can think of the PageRank score of a node as the proportion of time spent by a random walk that jumps to a random node at each step with probability α. The blue nodes have very low PageRank scores because they have no in-edges, whereas the red and green nodes have very high PageRank scores because they point to each other.

2.2.2 Hitting Time

Hitting time has been a long-standing area of study within the area of stochastic processes and Markov chains. Aldous and Fill [19] provide a rigorous treatment of hitting times. Refer to the classic text by Kemeny and Snell [20] for a more general treatment on finite Markov chains, including hitting time and other quantities of interest. Hitting time is sometimes referred to as the *first passage time* in the literature.

Hitting time can be defined in two ways: one based on the step-length of a random walk, and one based on probability. In this chapter, I will present the step-length definition to provide some initial intuition for hitting time. Chapter 3 will discuss the two definitions in greater detail and prove that they are equivalent.

The hitting time h_{ij} between two nodes i and j is the expected number of steps that a standard random walk starting at node i will take to hit node j for the first time. Note that hitting time is not a symmetric quantity, i.e., in general, $h_{ij} \neq h_{ji}$.

Definition 2.6 (Personalized Hitting Time)**.** *Given a weighted digraph G, let $(X_t)_{t\geq 0}$ be a standard random walk on G. Define the random variable τ_j by*

$$\tau_j = \inf\{t : X_t = j\}. \tag{2.4}$$

The hitting time between two nodes i and j is

$$h_{ij} = \mathbb{E}(\tau_j | X_0 = i). \tag{2.5}$$

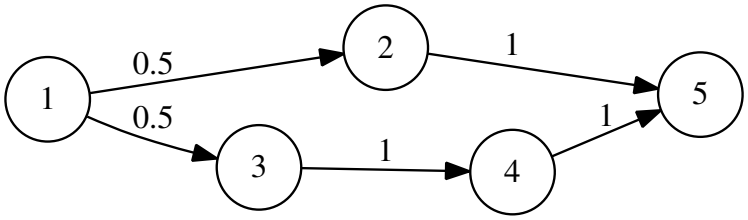


Figure 2.2: Example of hitting time. The personalized hitting time between nodes 1 and 5 is 2.5.

The global hitting time is simply a linear combination of the personalized hitting times:

Definition 2.7 (Global Hitting Time)**.** *Given a weighted digraph G, and a starting distribution q over the nodes of G, let τ_j be as defined in (2.4). Then the global hitting time for a node j is defined as*

$$h_j = \mathbb{E}(\tau_j | X_0 \sim q). \tag{2.6}$$

I will generally choose q to be the uniform distribution over all nodes in V.

We will see in Chapter 3 that the hitting time can be computed exactly in $\mathcal{O}(|V|^4)$ time, and in Chapter 4 I will present approximation algorithms with better asymptotic complexity.

As a simple numerical example, consider the weighted digraph in Figure 2.2. The personalized hitting time between nodes 1 and 5 is 2.5, because a random walk starting at node 1 will take either a 2-step path or a 3-step path to node 5 with equal probability.

2.2.3 Maximum Flow

Maximum flow problems have been long studied in computer science, and have a wide range of applications. The maximum flow problem involves a flow network represented as a weighted directed graph and finding the maximum flow that may be pushed from a source node to a sink node.

Definition 2.8 (Maximum Flow)**.** *Given a weighted digraph $G = (V, E, w)$, and sink and source nodes $s, t \in V$, a* flow *is defined as a mapping $f : E \to \mathbb{R}^+$ subject to the following constraints.*

1. *Capacity constraint: The flow on each edge cannot exceed its capacity. $f(u, v) \leq w(u, v)$ for each $(u, v) \in E$*

2. *Conservation of flows: The amount of flow entering a node must match the amount of flow leaving that node, except for the sink and source nodes. $\sum_{u:(u,v)\in E} f(u, v) = \sum_{u:(v,u)\in E} f(v, u)$ for $v \in V \setminus \{s, t\}$.*

Define the value $|f|$ of a flow by sum of the flows leaving the source node $\sum_{v:(s,v)\in E} f(s, v)$. The maximum flow of a graph is the maximum $|f|$ over all possible flows.

The simplest algorithm for the maximum flow problem is the Edmonds-Karp algorithm, which guarantees to find the optimal maximum flow in a graph in $\mathcal{O}(|V||E|^2)$ time. The push-relabel algorithm improves this to $\mathcal{O}(|V|^2|E|)$ time. Recent research has shown that the maximum flow problem can be solved in as fast as $\mathcal{O}(|V||E|)$ time [21]. In this thesis, I use the Edmonds-Karp algorithm for its simplicity.

2.2.4 Shortest Path

The shortest path between two nodes i and j in a weighted digraph is the sequence of edges connecting i and j whose sum of weights is smallest.

Definition 2.9 (Shortest Path)**.** *Given a weighted digraph $G = (V, E, w)$ and two nodes i and j, define a path as a sequence of k vertices $(v_1, v_2, \ldots, v_k)$ such that $(v_i, v_{i+1}) \in E$ for $i = 1, \ldots, k-1$. The length of such a path is given by the sum of edge weights $\sum_{i=1}^{k-1} w(v_i, v_{i+1})$. The shortest path between i and j is the path connecting i and j with the smallest length.*

The shortest path is often solved by Dijkstra's algorithm, which finds the shortest paths from one source node to all other nodes. Its time complexity depends on the data structure used. The commonly used binary heap results in a time complexity of $\Theta(|E| \log |V|)$, although a more sophisticated Fibonacci heap can improve this time to $\mathcal{O}(|E| + |V| \log |V|)$. For our purposes, we will often want to compute the shortest path between all pairs of nodes, and in the case of dense graphs, it will be more efficient to use the Floyd-Warshall algorithm, which runs in $\Theta(|V|^3)$.

2.3 Reputation Mechanisms

In this thesis, I will model a multi-agent system by a weighted digraph, in which each vertex corresponds to an agent. Each agent is modeled as having a *true type* θ_i, which denotes the trustworthiness of agent i. In practice, I interpret θ_i as the probability that a transaction with agent i will complete with a good outcome.

In a multi-agent system, such as an online marketplace, agents engage with each other in transactions, which can result in good or bad outcomes. Agents then report the outcome of their transactions to the reputation mechanism, which enables the identification of the trustworthy agents. For example, on the online marketplace eBay, buyers and sellers can rate their experience with other users after each transaction occurs. In our setting, I formalize this as an *agent report*. Each agent has a truthful belief about other agents, but a report may or may not reflect those truthful beliefs.

Definition 2.10 (Agent Report)**.** *Given a set of agents V, an agent i has beliefs about agents $V_i \subseteq V$ represented by $w_i : V_i \to \mathbb{R}^+$. The agent reports $(\hat{V}_i, \hat{w}_i)$ as its beliefs. Agent i is truthful if and only if $V_i = \hat{V}_i$ and $w_i = \hat{w}_i$.*

The beliefs encapsulated in these agent reports are modeled by weighted directed edges. The weight of an edge between vertices i and j denotes agent i's belief about agent j's trustworthiness. This edge weight can be thought of as derived from samples taken according to agent j's true type, although it may be determined in other ways.

The agent reports are combined to form a *trust graph*, which aggregates the agent reports.

Definition 2.11 (Trust Graph)**.** *Given a set of agent reports $(\hat{V}_i, \hat{w}_i)$, the trust graph is a weighted digraph $G = (V, E, w)$, in which $V = \bigcup_i \hat{V}_i$, $E = \bigcup_i \bigcup_{v \in \hat{V}_i} (i, v)$ and $w(i, j) = \hat{w}_i(j)$.*

Given a trust graph, I define a reputation mechanism as an algorithm that takes a trust graph to compute the reputation scores of all the agents in the trust graph. In this thesis, I will present reputation mechanisms that produce *rankings* of agents by reputation, as this will allow a more natural comparison between different reputation mechanisms. Furthermore, many practical applications only make use of rankings. The search results of a search engine provide only the rankings of query results, and not each individual result's relevance score. An buyer in an online marketplace may only care about finding the most reputable seller available.

Definition 2.12 (Reputation Mechanism)**.** *A reputation mechanism RM is a mapping between any trust graph G to a complete ordering $\prec_i$ for each agent $i \in V$. $j \prec_i k$ denotes that from the perspective of i, j has a lower reputation ranking than k.*

A reputation mechanism can potentially compute a different ordering for each agent, and thus provide a "personalized" reputation ranking. Such a reputation mechanism is called a personalized reputation mechanism. When a reputation mechanism returns the exact same ranking for each agent, it is called a global reputation mechanism.

Each of the graph algorithms presented in Section 2.2 can be adapted as a global or personalized reputation mechanism, simply by defining a rank order according to the trust scores. Higher PageRank scores and maximum flows correspond to higher ranks, while lower hitting times correspond to higher rank. The shortest path mechanism finds the shortest paths on the inverted weight graph $G' = (V, E, w')$ for which $w'(i, j) = 1/w(i, j)$. Shorter paths on G' correspond to higher ranks.

2.4 Desirable Properties of Reputation Mechanisms

Before I can meaningfully compare these reputation mechanisms, we must first discuss the desirable properties of a reputation mechanism. I posit that there are three criteria for evaluating the desirability of a reputation mechanism: accuracy, resistance to manipulation, and computational tractability.

2.4.1 Accuracy

Naturally, a reputation mechanism should satisfy some notion of accuracy, allowing agents to meaningfully distinguish between reputable and disreputable agents. In this thesis, I will use two metrics called informativeness and efficiency, which were used and discussed in Tang et al.[22].

The informativeness metric attempts to capture the extent to which the estimated reputation scores match ground truth reputation scores. This metric depends on a choice of rank correlation score, or a statistic that measures the relationship between two sets of rankings. Two of the more popular rank correlation scores are Spearman's ρ and Kendall's τ. I choose to use Spearman's ρ for the informativeness metric because it is less sensitive to small perturbations and more sensitive to large misrankings than Kendall's τ.

Definition 2.13 (Spearman's ρ)**.** *Given two sets of n raw scores X_i and Y_i and their ranks x_i and y_i, let $\overline{x}$ and $\overline{y}$ be the means of x_i and y_i, respectively. Then Spearman's ρ is defined by*

$$\rho = \frac{\sum_k (x_k - \bar{x})(y_k - \bar{y})}{\sqrt{\sum_k (x_k - \bar{x})^2 \sum_k (y_k - \bar{y})^2}}. \tag{2.7}$$

The informativeness metric takes the average Spearman correlation between the agent true types and the computed ranking for each agent. The correlation for each agent excludes the agent itself and is calculated over all $n - 1$ other agents. Not only is this a more useful measure of informativeness, but also most reputation mechanisms will return rankings such that each agent ranks themselves the highest.

Definition 2.14 (Informativeness)**.** *Given a trust graph G and true agent types $\vec{\theta}$, let $\prec_i$ be the rankings given by a reputation mechanism RM. Let ρ_i be the Spearman correlation of $\vec{\theta}$ and $\prec_i$ with agent i excluded. The informativeness ϕ is the average Spearman correlation*

$$\phi = \frac{1}{|V|} \sum_i \rho_i. \tag{2.8}$$

The efficiency metric captures the extent to which a reputation mechanism enables agents to successfully complete transactions with other agents when choosing agents based on estimated reputation scores. I model an agent as choosing to transact with the agent estimated to have the highest ranking in a random subset of κ agents. The transaction completes with a good outcome with probability equal to the true type of the chosen agent. Thus, the efficiency is defined as the expected proportion of transactions that are successful when every agent behaves in this manner, according to a fixed set of reputation rankings (i.e., the rankings are not updated as these transactions are completed).

More procedurally, a round of transactions is defined by the following steps.

1. Each agent i selects one other agent to transact with, by picking the agent ranked highest

according to $\prec_i$ from a random subset of κ agents.

2. The transaction with agent j succeeds with probability θ_j.

Definition 2.15 (Efficiency)**.** *Given a trust graph G and agent types $\vec{\theta}$, the efficiency η of a reputation mechanism RM is given by the expected proportion of transactions that are successful in one round of transactions.*

It turns out that I can give a closed form expression for η.

Theorem 2.1. *Given a trust graph G, agent types $\vec{\theta}$, and a reputation mechanism RM that computes rankings $\prec_i$, let r_i be the rank vector corresponding to $\prec_i$, such that $r_i(j)$ gives the ranking of j from the perspective of i, i.e., $r_i(j) \in \{1, \dots |V|\}$ and $j \prec_i k \Rightarrow r_i(j) < r_i(k)$. The probability that an agent i would select agent j for a transaction is given by*

$$p_{ij} = \frac{\kappa}{N} \cdot \frac{\binom{r_i(j)-1}{\kappa-1}}{\binom{N-1}{\kappa-1}} \tag{2.9}$$

where N is the number of agents and κ is the size of the subset used in our agent model. Then the efficiency η is given by

$$\eta = \sum_i \sum_j p_{ij}\theta_j. \tag{2.10}$$

Proof. I must prove that p_{ij} is actually the probability that agent i selects agent j given the above selection process. Consider the more general problem of finding the probability that j is the highest number when taking a κ-subset of the set $\{1, \dots N\}$. This probability is given by

$$\frac{\kappa}{N} \cdot \frac{\binom{j-1}{\kappa-1}}{\binom{N-1}{\kappa-1}}. \tag{2.11}$$

The first fraction is the probability that j is in the κ-subset. The second term gives the probability that the remaining $\kappa - 1$ elements are all less than j. To obtain our expression for p_{ij} I must replace j with $r_i(j)$ to reflect the different rankings $\prec_i$ from each agent's perspective.

The final expression for η is simply the expected proportion of transactions that are successful, as desired. □

In Chapter 6, I will compare the accuracy of the reputation mechanisms by computing the informativeness and efficiency of the reputation rankings over many trust graphs while varying the graph structure and the strategic behavior of the agents.

2.4.2 Resistance to Manipulation

A good reputation mechanism should be able to maintain its accuracy even when agents attempt to strategically manipulate the reputation mechanism to their advantage. By providing untruthful reports, an agent can make itself look better or make others look worse. A good reputation mechanism should limit or eliminate the impact of such manipulations. All other things equal, an ideal reputation would have the property that no agent can do better than be completely truthful. However, there is generally a trade-off between accuracy and resistance to manipulation. Reputation mechanisms that are very accurate in the absence of manipulation tend to be more inaccurate in the midst of manipulation, while reputation mechanisms that are resistant to manipulation are not as accurate in the absence of manipulation. The intuition is that a reputation mechanism is better able to resist manipulation by discarding data, but doing so in the absence of manipulation also lowers the accuracy by discarding accurate data.

I categorize the manipulations we care about into two kinds: *sybil attacks* and *cutting outlinks*. All the manipulations that I will present in this thesis will be either one of or a composition of these two kinds.

The first strategy involves creating false identities, known as sybils, which report a high level of trust for the strategic agent. Sybil attacks can be effective in improving a strategic agent's reputation score by dominating truthful reports from real agents.

Definition 2.16 (Sybil Attack)**.** *Given a trust graph $G = (V, E)$, an agent i can apply a sybil attack by creating N sybils. Let $V_s = (s_1, s_2, \ldots, s_N)$ denote the sybils, and let $E_s = \{(i, s_k, w), (s_k, i, w) | k = 1, \ldots, N\}$ denote the new edges used for the sybil attack. The edge weight w can be chosen in conjunction with the number of sybils N to achieve the desired effect. The sybil attack produces a new graph $G' = (V \cup V_s, E \cup E_s)$.*

Sybil-sybil edges are not applied, as they can actually diminish the effect of the sybil attack. As we will see below, a key feature of the sybil is that the only out-edge of the sybil is back to the strategic agent.

The other kind of strategy involves misreporting other agents as less trustworthy than they actually are in order to diminish their ranking. A malicious agent may be able to surpass another more trustworthy agent in the ranking by misreporting this agent as untrustworthy. I will examine the extreme case of this manipulation, in which a strategic agent marks all other agents as completely untrustworthy by giving all agents a trust report of 0. It turns out that for all our reputation mechanisms, this is equivalent to cutting all the outgoing edges for the strategic agent.

Definition 2.17 (Cutting Outlinks Manipulation)**.** *A strategic agent i applying the* cutting outlinks manipulation *cuts all its outgoing edges from the graph. Given a trust graph $G = (V, E, w)$, this manipulation produces a graph $G' = (V, E_{-i}, w)$, where $E_{-i} = \{(u, v) | (u, v) \in E, u \neq i\}$.*

	Sybil attack	Cutting outlinks
Global PageRank	Not SP	Not SP
Personalized PageRank	Not SP	Not SP
Global Hitting Time	Not SP	Not SP
Personalized Hitting Time	Not SP	Not SP
Personalized Maximum Flow	SP	Not SP
Personalized Shortest Path	SP	SP

Table 2.1: Strategyproofness (SP) of reputation mechanisms to sybil attacks and cutting outlinks.

	Restart SP?	2-loop SP?	Path Lengthening SP?
Global PageRank	No	No	Yes
Personalized PageRank	Yes	No	Yes
Global Hitting Time	No	Yes	No
Personalized Hitting Time	Yes	Yes	No
Personalized Maximum Flow	Yes	Yes	Yes
Personalized Shortest Path	Yes	Yes	Yes

Table 2.2: Sybil attack strategyproofness (SP).

Another kind of manipulation that I will not consider here is *whitewashing*. An agent that has a low reputation score could choose to assume a new identity, thus discarding its old poor reputation. I will not study this manipulation here, as it requires a dynamic analysis that is beyond the scope of this thesis.

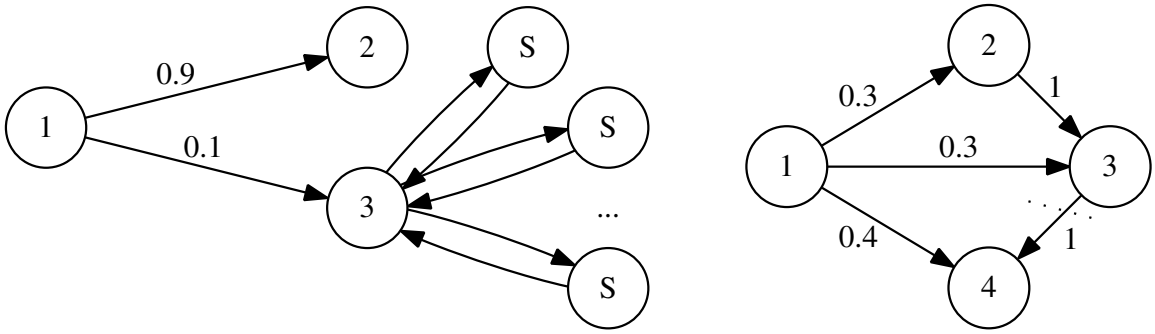
I can show that some reputation mechanisms are entirely immune to certain kinds of strategic manipulation. I call such reputation mechanisms *strategyproof*.

Definition 2.18 (Strategyproofness)**.** *I call a reputation mechanism RM* strategyproof *to a kind of manipulation σ if for any trust graph G, any manipulation of type σ by an agent i resulting in a graph G' does not improve i's ranking from the perspective of any other agent.*

I briefly discuss the strategyproofness properties of each reputation mechanism. The full list of strategyproofness results are given in Table 2.1.

Sybil attacks are effective in three ways. First, sybils create many loops of length 2, which I call 2-loops. A reputation mechanism like PageRank is vulnerable to 2-loops because it increases the stationary distribution mass for any node that employs 2-loops. Hitting time is strategyproof to 2-loops because it only looks at the *first* time a random walk hits a node.

Second, sybils can also "steal" restart probability away from other real nodes, and direct that probability mass to a strategic agent. Global reputation mechanisms are vulnerable to this, as they make use of a restart distribution spread across all agents, including sybils. Personalized reputation mechanisms are strategyproof to this restart attack because the restart distribution is restricted to a single node.



(a) Sybil attack by 2-loops.

(b) Cutting outlinks.

Figure 2.3: Example of graphs under manipulation.

Hopcroft and Sheldon [23] address the issue of the restart probability capture problem by defining a "pretrusted set" of nodes over which the restart probability is distributed. However, this leads to the problem of choosing a good pretrusted set, which is non-trivial.

Third, sybils increase the length of random walks that pass through strategic agents with sybils, by adding extraneous steps to a random walk. Although this is structurally similar to the 2-loop effect, it is conceptually different because the effect is in diminishing other agents' reputation ranking rather than improving one's own. Hitting time is vulnerable to this aspect of the sybil attack. To see this, consider a strategic agent v, and the computation of the hitting time between agents i and j. If there exists a path from i to j that passes through v, then agent v is able to increase h_{ij} by employing a sybil attack. If $v \prec_i j$, then v is potentially able to surpass j from i's perspective with this method. I call this the path lengthening effect of the sybil attack.

Maximum flow and shortest path are both completely strategyproof to sybil attacks. They are both personalized mechanisms, which make them strategyproof to restart probability capture. The maximum flow problem requires the flows out of a node to be equal to the flows into the node, and so the existence of sybils does not change the maximum flow. Similarly, the shortest path is not changed by the existence of sybils because a path through a sybil will always be longer.

The analysis with regard to sybil attacks is shown in Table 2.2. Personalized mechanisms are strategyproof to restart probability capture. PageRank mechanisms are strategyproof to the path lengthening aspect of the sybil attack. Hitting time mechanisms are strategyproof to 2-loops. Maximum flow and shortest path are entirely strategyproof to sybil attacks.

As an example, consider Figure 2.3a. From agent 1's perspective, agent 2 should be more reputable than agent 3. Under global PageRank, a mechanism vulnerable to sybil attacks, agent 3 will appear more reputable, due to its sybils. Under personalized hitting time, a mechanism strategyproof to 2-loops and restart probability capture, agent 3's sybils have no effect, and agent 2 will be ranked higher than agent 3 from agent 1's perspective.

Intuitively, the cutting outlinks manipulation will affect any reputation mechanism that inte-

	Complexity	Notes
Global PageRank	$\mathcal{O}(V^3)$	
Personalized PageRank	$\mathcal{O}(V^4)$	
Global Hitting Time	$\mathcal{O}(V^4)$	Proved in Chapter 3
Personalized Hitting Time	$\mathcal{O}(V^4)$	Proved in Chapter 3
Maximum Flow	$\mathcal{O}(V^3E^2)$	Edmonds-Karp
Shortest Path	$\mathcal{O}(V^3)$	Floyd-Warshall

Table 2.3: Algorithmic complexity of reputation mechanisms.

grates information from the majority or all the edges in the trust graph. Any such reputation mechanism would be affected by cutting outlinks, as it removes information available to the reputation mechanism. All the mechanisms with the exception of shortest path make use of the majority of edges in the graph. The shortest path mechanism only looks at the edges on the shortest path between two nodes, so the disappearance (or diminished weight) of edges not on the shortest path does not affect the mechanism. Indeed, shortest path is the only mechanism strategyproof to the cutting outlinks mechanism.

Let's see why no agent can improve its own ranking by cutting outlinks in the shortest path mechanism. Suppose $i \prec_k j$ for agents i, j, k, and agent i wishes to surpass agent j from k's perspective. If j is ranked higher than i, then the shortest path from k to j does not pass through i. There is nothing that i can do to either make the path from k to j longer than it is, and nothing to make the path from k to i shorter. Hence, i cannot improve his ranking.

As an example, consider Figure 2.3b. From agent 1's perspective, agent 4 should be the most reputable. Under the maximum flow mechanism, which is vulnerable to cutting oulinks, agent 3 can cut its outlink to agent 4 and decrease the flow from agent 1 to agent 4. By doing so, agent 3 can surpass agent 4 from agent 1's perspective. However, under the shortest path mechanism, which is strategyproof to cutting outlinks, agent 3 cutting its outlink does not affect the shortest path, and does not change its ranking.

2.4.3 Computational Tractability

Finally, it is important that the output of these reputation mechanisms can be feasibly computed on realistically sized datasets. Table 2.3 summarizes the runtime complexity for each of the algorithms. However, the runtime complexity leaves much of the detail and nuance that go into the analysis of a particular mechanism's computational tractability. Numerous additional factors must be weighed, along with actual business requirements, when considering computational tractability.

For example, some of these algorithms are more easily parallelized than others, which makes them more tractable to horizontal scaling. Some or all of these algorithms admit approximation algorithms, which trade off accuracy for a speed up. Some of these mechanisms may be more

amenable to incremental computation than others, able to update reputation rankings quickly when the trust graph is modified only slightly.

The next three chapters of this thesis will be focused on examining the computational tractability of the hitting time mechanisms. Chapter 3 presents exact algorithms for hitting time. Chapter 4 presents Monte Carlo algorithms for estimating hitting time. Chapter 5 provides an empirical comparison of these algorithms according to accuracy and running time.

Chapter 3

Defining Hitting Time

In this chapter I provide an alternative definition of hitting time as a probability, which is useful because it makes hitting time more amenable to Monte Carlo algorithms. In Section 3.4, I will prove that the two definitions are linearly equivalent, i.e., there is a linear mapping between the values produced by each definition. In Section 3.5 I provide exact algorithms, based on matrix algebra, for each definition of hitting time.

3.1 Useful Theory

In this section, I first provide a series of definitions and lemmas which will be useful for the rest of the chapter.

Definition 3.1 (Convergent Matrix)**.** *I call an $n \times n$ matrix M a* convergent matrix *if*

$$\lim_{k \to \infty} (M^k) = 0_{n,n}, \tag{3.1}$$

where $0_{n,n}$ is the zero map, or the mapping that takes its entire domain to zero.

Definition 3.2 (j-Absorbing Stochastic Matrix)**.** *I call a $n \times n$ matrix M a j-absorbing stochastic matrix if it fulfills the following properties:*

1. *With the exception of the jth row, M is a positive matrix, i.e., $M_{ik} > 0 \; \forall i, k$ where $i \neq j$.*
2. *With the exception of the jth row, M is a* right stochastic matrix. *That is, every row except the jth row sums to 1, i.e., $\sum_k M_{ik} = 1 \; \forall i \neq j$.*
3. *The jth row of M consists of all zeros, i.e., $M_{jk} = 0 \; \forall k$, indicating that j is an absorbing state of the corresponding Markov chain.*

The j-absorbing stochastic matrix of a weighted digraph $G = (V, E, w)$ and a state $j \in V$ is the transition matrix M with its jth row set to all zeros.

The j-absorbing stochastic matrix is how I will represent the transition probabilities of a random walk on a weighted digraph that terminates at a node j. M_{ik} is interpreted as the probability that the random walk will take a step from i to k. By the first condition, the random walk can step to any other node, including itself, and by the second condition, the walk will always take a step to another node, unless it has reached j. The third condition simply states that the walk will end if it ever hits the absorbing state j. The expression $(M^p)_{ik}$ denotes the probability that a random walk starting at i will end up at k in exactly p steps (without ever hitting j).

I note that much of the analysis that uses a j-absorbing stochastic matrix can also be done with a stochastic matrix in which j has a self-loop, which allows every row to sum to 1. However, for our purposes, the theory will be a bit cleaner and more intuitive with a j-absorbing stochastic matrix.

Lemma 3.1. *Any j-absorbing stochastic matrix is a convergent matrix.*

Proof. Let M be an $n \times n$ j-absorbing stochastic matrix, and let $\mathbf{x}$ be some vector in $\mathbb{R}^n$. Consider the sequence of vectors $(M\mathbf{x}, M^2\mathbf{x}, \ldots)$, and let $v^{(k)} = M^k\mathbf{x}$. If $\lim_{k\to\infty} \mathbf{v}^{(k)} = \mathbf{0}$, then it must also be that $\lim_{k\to\infty} M^k = 0_{n,n}$, because $\mathbf{x}$ can be any vector in $\mathbb{R}^n$.

I use the infinity norm over $\mathbb{R}^n$ such that

$$\|\mathbf{v}\| = \max_i |\mathbf{v}_i| \tag{3.2}$$

for any $\mathbf{v} \in \mathbb{R}$.

Consider some k in $2, 3, \ldots$. Clearly $\mathbf{v}_j^{(k-1)} = \mathbf{v}_j^{(k)} = 0$ because the jth row of M consists of all zeros. For each index $i \neq j$ in the vector $\mathbf{v}^{(k)}$, the absolute value of the ith element is bounded by

$$\begin{aligned}
\left|\mathbf{v}_i^{(k)}\right| &= \left|\sum_\ell M_{i\ell}\mathbf{v}_\ell^{(k-1)}\right| && (3.3)\\
&\leq \sum_\ell \left|M_{i\ell}\mathbf{v}_\ell^{(k-1)}\right| && (3.4)\\
&= \sum_\ell M_{i\ell}\left|\mathbf{v}_\ell^{(k-1)}\right| && (3.5)\\
&= \sum_{\ell\neq j} M_{i\ell}\left|\mathbf{v}_\ell^{(k-1)}\right| && (3.6)\\
&\leq \sum_{\ell\neq j} M_{i\ell}\|\mathbf{v}^{(k-1)}\| && (3.7)\\
&< \|\mathbf{v}^{(k-1)}\|. && (3.8)
\end{aligned}$$

The final inequality follows from the fact that the non-absorbing rows of M sum to 1, but all elements of the non-absorbing rows are also assumed to be strictly positive. Thus the sum of any strict subset of elements must be strictly less than 1.

If this inequality holds for all indices $i \neq j$, then we must have that $\|\mathbf{v}^{(k)}\| < \|\mathbf{v}^{(k-1)}\|$, which

holds for all $k > 1$. Thus the norm of the vectors in the sequence is strictly decreasing, and must eventually converge to $\mathbf{0}$. This implies that $\lim_{k\to\infty} M^k$ must converge to the zero map, as desired. □

The following lemma provides an identity that allows certain matrix inversions to be expressed as infinite summations, which will allow us to express certain expression in a more intuitive form. The following proof is adapted from Iosifescu [24], where it is presented as Theorem 1.7 (pg. 45).

Lemma 3.2. *Given a convergent matrix M, the matrix inverse $(I - M)^{-1}$ exists and is given by*

$$(I - M)^{-1} = \sum_{k=0}^{\infty} M^k. \tag{3.9}$$

Proof. Consider the sum

$$(I - M)\sum_{k=0}^{n} M^k = (I - M) + (M - M^2) + \cdots + (M^n - M^{n+1}) = I - M^{n+1}. \tag{3.10}$$

As M is a convergent matrix, there exists an n sufficiently large such that $\det(I - M^{n+1}) \neq 0$. Thus the determinant of the left hand expression must also be nonzero. As the determinant of a product of matrices is equal to the product of the determinants, we must have that $\det(I - M) \neq 0$, which implies that $(I - M)^{-1}$ exists. Now when we the limit of both sides as $n \to \infty$,

$$\lim_{n\to\infty}(I - M)\sum_{k=0}^{n} M^k = I - \lim_{n\to\infty} M^{n+1} = I. \tag{3.11}$$

Multiplying $(I - M)^{-1}$ on both sides of the equation yields the desired expression. □

The following lemma formalizes the intuition that for a random walk of length p, the probability of "survival" after p steps plus the probability of having been absorbed at a step before p is equal to 1.

Lemma 3.3. *Given a j-absorbing stochastic matrix M, for $p = 1, \ldots$ and any $i \neq j$,*

$$\sum_{\ell}(M^p)_{i\ell} + \sum_{k=0}^{p-1}(M^k)_{ij} = 1. \tag{3.12}$$

Proof. I first prove the following intermediate result holds for $k = 1, 2, \ldots$ and $i \neq j$:

$$\sum_{\ell}(M^k)_{i\ell} = \sum_{\ell\neq j}(M^{k-1})_{i\ell}. \tag{3.13}$$

Intuitively, this expresses that the probability that a random walk "survives" (i.e., is at any state) in k steps is equal to the probability that the random walk survives to any of the non-absorbing states (i.e., any state other than j) in $k-1$ steps.

Consider each element in the summation as the inner product of the matrices M^{k-1} and M:

$$\sum_{\ell}(M^k)_{i\ell} = \sum_{\ell}\sum_{p\neq j}(M^{k-1})_{ip}M_{p\ell} \tag{3.14}$$

$$= \sum_{p\neq j}\left[(M^{k-1})_{ip}\sum_{\ell}M_{p\ell}\right] \tag{3.15}$$

$$= \sum_{p\neq j}(M^{k-1})_{ip} \tag{3.16}$$

where the last equality comes from the fact that all rows except the jth row of M sum to 1.

Now I can prove the lemma by induction on p. The base case of $p=1$ holds trivially. Suppose the lemma holds for some $p=n-1$. I wish to show the lemma holds for $p=n$:

$$\sum_{\ell}(M^n)_{i\ell} + \sum_{k=0}^{n-1}(M^k)_{ij} = \sum_{\ell\neq j}(M^{n-1})_{i\ell} + \sum_{k=0}^{n-1}(M^k)_{ij} \tag{3.17}$$

$$= \sum_{\ell}(M^{n-1})_{i\ell} + \sum_{k=0}^{n-2}(M^k)_{ij} \tag{3.18}$$

$$= 1. \tag{3.19}$$

The first equality follows from the above intermediate result, and the last equality follows from the inductive hypothesis. □

This next lemma extends our intuition from Lemma 3.3 to the case when $p\to\infty$: it says that the probability that a random walk is absorbed by an absorbing state converges to 1 as the path length p goes to infinity.

Lemma 3.4. *Given a j-absorbing stochastic matrix M,*

$$\sum_{k=0}^{\infty}(M^k)_{ij} = 1 \tag{3.20}$$

for $i\neq j$.

Proof. Consider Lemma 3.3 as p goes to infinity:

$$\lim_{p\to\infty}\left(\sum_{\ell}(M^p)_{i\ell} + \sum_{k=0}^{p-1}(M^k)_{ij}\right) = \lim_{p\to\infty}1 \tag{3.21}$$

The first term goes to zero, because M is a convergent matrix, and the second term becomes our desired expression, which is shown to be equal to unity. □

3.2 Step Length Definition of Hitting Time

In Chapter 2, I gave a definition of hitting time that was based on the number of steps in a random walk. In this section, I will manipulate this definition into a number of different forms.

Recall that I defined the hitting time h_{ij} between two nodes i and j as the expected number of steps for a standard random walk starting at i to reach j for the first time. If I suppose that h_{ij} is infinite if j is unreachable from i, then I can express h_{ij} recursively as

$$h_{ij} = 1 + \sum_k P_{ik} h_{kj}, \tag{3.22}$$

where P is the transition matrix for the weighted digraph G.

However, I would like h_{ij} to always be finite, so that the nodes can be completely ordered by hitting time. To solve this issue, I discount longer path lengths so that they cannot become infinitely long.

Definition 3.3 (α-Discounted Length of a Standard Random Walk)**.** *The α-discounted length of a standard random walk of length τ is defined as*

$$\sum_{t=1}^{\tau} (1-\alpha)^t. \tag{3.23}$$

In this manner, even if the number of steps a random walk extends to infinity because it can never reach its destination node, the discounted length stays finite. Applying this discounting changes the hitting time values, but it does not change the ranking because the discount is a monotonic function of the path length.

I now adapt the recursive expression to reflect the discounted path length.

Definition 3.4 (Step Length Hitting Time)**.** *Given a weighted digraph G with transition matrix P, the hitting time h_{ij} between every pair of nodes i and j is given by the recursive equation*

$$h_{ij} = (1-\alpha) + (1-\alpha) \sum_k P_{ik} h_{kj} \tag{3.24}$$

for $i \neq j$. I let $h_{jj} = 0$ for all j.

This recursive expression implicitly encapsulates our discounted path length because at each step, the recursive definition adds $1-\alpha$ and multiplies the remainder of the path by $1-\alpha$. Repeating

these operations yields the geometric sum as defined. The following theorem explicitly shows that the recursive expression α-discounts the length of the random walk.

Theorem 3.1. *Let G be a weighted digraph. The step length definition of hitting time can be expressed as*

$$h_{ij} = \sum_{k=0}^{\infty} \left[\left(\sum_{t=1}^{k} (1-\alpha)^t \right) (M^k)_{ij} \right] \tag{3.25}$$

where M is the j-absorbing stochastic matrix for G and state j.

Proof. I begin by vectorizing the recursive expression, taking care to handle the case for h_{jj} correctly. Define $\vec{f_j} \triangleq \vec{1} - \vec{e}_j$ as the vector of all ones except for a zero at the jth index. If I let $\vec{h}(j)$ be the vector of hitting times to j, where $\vec{h}_i(j) = h_{ij}$, I can now write

$$\vec{h}(j) = (1-\alpha)\vec{f_j} + (1-\alpha)M\vec{h}(j) \tag{3.26}$$

$$\vec{h}(j) = (I - (1-\alpha)M)^{-1} \cdot (1-\alpha)\vec{f_j} \tag{3.27}$$

$$= \left[\sum_{k=0}^{\infty} (1-\alpha)^k M^k \right] \cdot (1-\alpha)\vec{f_j} \tag{3.28}$$

where M is the j-absorbing stochastic matrix for G and state j The last equality follows from Lemma 3.2.

The above expression for each individual element of the vector can be written as

$$h_{ij} = \sum_{k=0}^{\infty} (1-\alpha)^{k+1} \sum_{\ell \neq j} (M^k)_{i\ell}. \tag{3.29}$$

Applying Lemma 3.3 yields

$$= \sum_{k=0}^{\infty} (1-\alpha)^{k+1} \left(1 - \sum_{t=0}^{k} (M^t)_{ij} \right) \tag{3.30}$$

$$= \sum_{k=0}^{\infty} (1-\alpha)^{k+1} - \sum_{k=0}^{\infty} \left[(1-\alpha)^{k+1} \sum_{t=0}^{k} (M^t)_{ij} \right] \tag{3.31}$$

$$= \frac{1-\alpha}{\alpha} - \sum_{k=0}^{\infty} \left[(1-\alpha)^{k+1} \sum_{t=0}^{k} (M^t)_{ij} \right] \tag{3.32}$$

Careful rearrangement of the terms in the summation yields

$$= \frac{1-\alpha}{\alpha} - \sum_{k=0}^{\infty} \left[(M^k)_{ij} \sum_{t=k+1}^{\infty} (1-\alpha)^t \right] \tag{3.33}$$

$$= \frac{1-\alpha}{\alpha} - \sum_{k=0}^{\infty} \frac{(1-\alpha)^{k+1}}{\alpha} (M^k)_{ij} \tag{3.34}$$

Finally, applying Lemma 3.4 to the first term allows us to combine terms:

$$= \sum_{k=0}^{\infty} \frac{1-\alpha}{\alpha} (M^k)_{ij} - \sum_{k=0}^{\infty} \frac{(1-\alpha)^{k+1}}{\alpha} (M^k)_{ij} \tag{3.35}$$

$$= \sum_{k=0}^{\infty} \frac{(1-\alpha) - (1-\alpha)^{k+1}}{\alpha} (M^k)_{ij} \tag{3.36}$$

$$= \sum_{k=0}^{\infty} \left[\left(\sum_{t=1}^{k} (1-\alpha)^t \right) (M^k)_{ij} \right] \tag{3.37}$$

□

This closed-form expression has an intuitive interpretation. It represents the expected value of the discounted path length over all possible paths. It sums for each path length k the product of the discounted path length and the probability of such a path occurring. Thus, we can see that the recursive definition given in Definition 3.4 correctly captures the α-discounted path length.

3.3 Probability Definition of Hitting Time

Hopcroft and Sheldon [23] provide a probability-based approach to compute reputation scores for nodes in a graph, which I first present here.

Definition 3.5 (α-Random Walk)**.** *Given a trust graph $G = (V, E, w)$ and a restart distribution q over the nodes V of G, an α-random walk is a sequence of random variables $(X_t)_{t \geq 0}$ for which $X_0 \sim q$ and the transition probabilities are given by*

$$\mathbb{P}(X_{t+1} = j | X_t = i) = \begin{cases} \alpha q(j) + (1-\alpha) \dfrac{w(i,j)}{\sum_{(i,j') \in E} w(i,j')} & \text{if } (i,j) \in E \\ \alpha q(j) & \text{otherwise} \end{cases} . \tag{3.38}$$

The α-random walk is simply a standard random walk, modified so that at each step the random walk jumps back to the restart distribution q with probability α. Hopcroft and Sheldon define the hitting time of a node j as the random variable representing the number of steps such an α-random walk hits j for the first time. More formally, for an α-random walk $(X_t)_{t \geq 0}$, the hitting time of j is

$H_\alpha(j) = \min\{t : X_t = j\}$. While this looks similar to our step length hitting time, it is not quite the same, as the α-random walk allows for restarts while continuing to count the number of steps.

Hopcroft and Sheldon do not use $\mathbb{E}(H_\alpha)$ for their reputation scores. Rather, they use $\mathbb{P}(H < J)$ for their reputation mechanism, where H is the hitting time on a standard random walk, and J is a random variable representing the "jump," given by a geometric random variable with parameter α. This captures the probability that the random walk reaches the destination node before the first jump.

I take inspiration from this approach, and adapt it to a personalized form that does not depend on a restart distribution q. In the process, I also give an alternate definition in terms of an α-terminating random walk that yields a cleaner notation for my purposes.

Definition 3.6 (α-Terminating Random Walk)**.** *An α-terminating random walk is identical to the standard random walk except that at each step, the walk will end with probability α. More precisely, suppose we have a weighted digraph $G = (V, E, w)$. Then an α-terminating random walk on G is a finite sequence of random variables $(X_t)_{t=0}^{\tau}$ that satisfy the transition probabilities*

$$\mathbb{P}(X_{t+1} = j | X_t = i) = \begin{cases} (1-\alpha)\dfrac{w(i,j)}{\sum_{(i,j')\in E} w(i,j')} & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}. \tag{3.39}$$

Note that τ follows a geometric distribution with parameter α, and that $(X_t)_{t=0}^{\tau}$ satisfies the Markov property.

I note that the α-terminating random walk can also be expressed as a random walk that steps with probability α to a special state representing termination. However, the α-terminating random walk will be conceptually and notationally simpler for our purposes.

I can now define the hitting time between two nodes as the probability that an α-terminating random walk starting from one node will hit the other node before terminating. This is identical to Hopcroft and Sheldon's notion of hitting time.

Definition 3.7 (Probability Hitting Time)**.** *Given a weighted digraph G, let $(X_t)_{t=0}^{\tau}$ be an α-terminating random walk on G. For any two nodes i and j, define the hitting time h_{ij} to be*

$$h_{ij} = \mathbb{P}(j \in \{X_t\}_{t=0}^{\tau} | X_0 = i), \tag{3.40}$$

where $\{X_t\}_{t=0}^{\tau}$ is the set of states that the random walk reached before terminating. Note that $h_{jj} = 1$ for all j.

As I did with step length hitting time, I will manipulate this expression into a more intuitive form, that I will use to compare with step length hitting time.

Theorem 3.2. *Let G be a weighted digraph. The probability hitting time h_{ij} can be expressed as*

$$h_{ij} = \sum_{k=0}^{\infty} (1-\alpha)^k (M^k)_{ij} \tag{3.41}$$

where M is the j-absorbing transition matrix for G and state j.

Proof. I begin by conditioning the definition on X_1 and expanding by the Law of Total Probability:

$$h_{ij} = \sum_{k} \mathbb{P}(j \in \{X_t\}_{t=0}^{\tau} | X_0 = i, X_1 = k) \cdot \mathbb{P}(X_1 = k | X_0 = i). \tag{3.42}$$

When $i \neq j$, I can simplify the first probability term using the Markov property of the random walk, and I note that it is equivalent to h_{kj}. The second term is equal to the product of the survival probability and the transition probability, or $(1-\alpha)P_{ik}$, where P is the transition matrix of G:

$$h_{ij} = (1-\alpha) \sum_{k} P_{ik} h_{kj} \tag{3.43}$$

Now I can vectorize this expression, taking care to ensure that $h_{jj} = 1$. I can take the j-absorbing stochastic matrix M of G and the standard basis vector $\vec{e}_j$ to produce the vectorization

$$\vec{h}(j) = (1-\alpha) M \vec{h}(j) + \vec{e}_j, \tag{3.44}$$

where $\vec{h}(j)$ is the vector of hitting times from all nodes to j, i.e., $\vec{h}_i(j) = h_{ij}$.

Solving for $\vec{h}(j)$ yields

$$\vec{h}(j) = (I - (1-\alpha)M)^{-1} \vec{e}_j. \tag{3.45}$$

Applying Lemma 3.2, we have

$$\vec{h}(j) = \left[\sum_{k=0}^{\infty} (1-\alpha)^k M^k \right] \vec{e}_j. \tag{3.46}$$

The expression for each element of $\vec{h}(j)$ can be written as

$$h_{ij} = \sum_{k=0}^{\infty} (1-\alpha)^k (M^k)_{ij}, \tag{3.47}$$

as desired. □

This expression admits an intuitive interpretation: the probability hitting time is given by the

sum over all path lengths k that an α-terminating random walk will hit j for the first time in exactly k steps. I note that our use of the j-absorbing stochastic matrix ensures that we only count the first hit to j, because the random walk immediately terminates after arriving at j for the first time.

3.4 Linear Equivalence of Hitting Times

The probability definition of hitting time will allow us to define Monte Carlo approximation algorithms in the next chapter. The values resulting from each definition are clearly different: the probability definition is always less than or equal to one, but the step length definition will often exceed one. I demonstrate that the definitions are linearly equivalent: the value resulting from each definition can be expressed as a linear function of the other. This is sufficient for our purposes because we only care about the ranking induced by the hitting times.

Let $h_{ij}^{(1)}$ be the probability based hitting time and $h_{ij}^{(2)}$ be the step length based hitting time.

Theorem 3.3. *$h_{ij}^{(1)}$ and $h_{ij}^{(2)}$ are linearly dependent. In particular,*

$$h_{ij}^{(2)} = \frac{1-\alpha}{\alpha}\left(1 - h_{ij}^{(1)}\right). \tag{3.48}$$

Proof. Consider equation (3.34) from the derivation of Theorem 3.1:

$$h_{ij}^{(2)} = \frac{1-\alpha}{\alpha} - \sum_{k=0}^{\infty} \frac{(1-\alpha)^{k+1}}{\alpha}(M^k)_{ij}. \tag{3.49}$$

Substituing in the expression for $h_{ij}^{(1)}$ from Theorem 3.2 yields the desired expression. □

3.5 Exact Algorithms

Hopcroft and Sheldon [23] only present a Monte Carlo algorithm in their work. In this section, for each definition of hitting time, I present exact algorithms based on matrix algebra.

3.5.1 Step Length Hitting Time

Theorem 3.4 (LS Step Length Algorithm)**.** *Given a graph G, the step length hitting times h_{ij} can be computed in $\mathcal{O}(|V|^4)$ time by solving $|V|$ systems of $|V|$ linear equations in $|V|$ variables.*

Proof. Equation (3.27) can be written in the form

$$(I - (1-\alpha)M)\vec{h}(j) = (1-a)\vec{f_j}. \tag{3.50}$$

This is represents a system of linear equations with $|V|$ equations and $|V|$ variables (one for each element of $\vec{h}(j)$). Solving this system for each $j \in V$ involves solving $|V|$ systems. As finding the

solution to a system of linear equations generally takes $\mathcal{O}(|V|^3)$ time, solving $|V|$ systems takes $\mathcal{O}(|V|^4)$ time. □

3.5.2 Probability Hitting Time

Theorem 3.5 (LS Probability Algorithm)**.** *Given a graph G, the probability hitting times h_{ij} can be computed in $\mathcal{O}(|V|^4)$ time by solving $|V|$ systems of $|V|$ linear equations in $|V|$ variables.*

Proof. Equation (3.45) can be written as

$$(I - (1-\alpha)M)\vec{h}(j) = \vec{e}_j. \tag{3.51}$$

As with the algorithm for step length hitting time, we can find all the h_{ij} in $\mathcal{O}(|V|^4)$ time by solving $|V|$ of these linear systems. □

While the probability hitting time can be found by solving linear systems, it turns out that it can also be found by finding the stationary distribution of particular Markov chains. I depend on theorem proved by Hopcroft and Sheldon relating $\mathbb{E}(H_\alpha)$ and $\mathbb{P}(H < J)$:

Theorem 3.6 (Hopcroft and Sheldon [23] Theorem 1 Part (ii))**.** *Let H and J be random variables defined as above with parameter α. Then*

$$\mathbb{E}[H_\alpha] = \frac{1}{\alpha} \cdot \frac{\mathbb{P}(H \geq J)}{\mathbb{P}(H < J)}. \tag{3.52}$$

My algorithm will also rely on this classic result of Markov chains.

Definition 3.8 (Expected Return Time)**.** *Given a Markov chain, the return time for a state v is*

$$R(v) = \min\{t \geq 1 : X_t = v, X_0 = v\}. \tag{3.53}$$

Theorem 3.7. *For an ergodic Markov chain, the expected return time $R(v)$ for a state v satisfies*

$$\mathbb{E}[R(v)] = 1/\pi(v), \tag{3.54}$$

where π is the stationary distribution of the Markov chain.

Now I am ready to present the algorithm.

Theorem 3.8 (Eigen Probability Algorithm)**.** *The probability hitting times can be computed in $\mathcal{O}(|V|^5)$ time by finding stationary distributions of Markov chains.*

Proof. Given a weighted digraph $G = (V, E, w)$, define $G_{ij} = (V, E_{ij}, w)$ where $E_{ij} = (E \setminus \{(j, i') : i' \in V\}) \cup \{(j, i)\}$, i.e., G_{ij} is a modification of G such that j only has one out-edge, directed to i. Each G_{ij} will be used to compute each h_{ij}.

Now let π_{ij} be the stationary distribution of the Markov chain defined by the α-random walk on G_{ij} with restart distribution $q(v) = \mathbf{1}\{v = i\}$, which can be found in $\mathcal{O}(|V|^3)$ time. Crucially, I claim that

$$\mathbb{E}[R_{ij}(j)] = \mathbb{E}[H_{\alpha,ij}(j)] + 1. \tag{3.55}$$

This is because the random walk that starts at j and returns to j must always take its first step to i, at which point the remainder of the walk is identical to a random walk for hitting time.

Now I can combine Theorem 3.6 and Theorem 3.7 to yield

$$\mathbb{P}(H_{ij}(j) < J) = \frac{1}{1 - \alpha + \alpha/\pi_{ij}(j)}. \tag{3.56}$$

As our α-random walk placed all of the restart distribution on i, we have that

$$h_{ij} = \mathbb{P}(H_{ij}(j) < J), \tag{3.57}$$

as desired. Computing this for each pair of nodes results in a total time complexity of $\mathcal{O}(|V|^5)$. $\square$

Chapter 4

Monte Carlo Algorithms

The exact algorithms we saw in the previous chapter have fairly high computational complexities. They are not be feasible to compute on large, real-world graphs. Some applications may not even need an exact solution, but can make do with approximate solutions. Here, I present three Monte Carlo algorithms for computing hitting time, which trade off accuracy for running time, allowing us to obtain an estimate for hitting time more quickly than finding an exact solution.

Throughout this chapter, I make use of the probability hitting time definition, which we will recall is defined as

$$h_{ij} = \mathbb{P}(j \in \{X_t\}_{t=0}^{\tau} | X_0 = i) \tag{4.1}$$

for an α-terminating random walk $(X_t)_{t=0}^{\tau}$.

4.1 Related Work

Hopcroft and Sheldon [23] present a Monte Carlo algorithm in their paper. However, the first step of their algorithm is a standard PageRank computation, which is normally $\mathcal{O}(|V|^3)$ time, unless it is computed in a Monte Carlo fashion. The core of their algorithm involves estimating the probability of a random walk returning to its start before jumping, i.e., $\mathbb{P}(R(v) < J)$ in order to compute $\mathbb{P}(H(v) < J)$. We will see in Section 4.5 that the expected number of walks needed to obtain an accurate estimator is greater than the Monte Carlo algorithms presented here below.

Furthermore, Hopcroft and Sheldon's algorithm appears to omit the restart distribution q in its computation, which is a crucial part of their approach to hitting time. Regardless, their algorithm is for computing the global hitting time; applying it to compute the personalized hitting time requires running the algorithm $|V|$ times total to compute the hitting time from each node's perspective. The Monte Carlo algorithms presented below simultaneously compute the personalized hitting time from every agent's perspective.

This work parallels many contributions [25, 26, 27, 28] on fast Monte Carlo algorithms for estimating PageRank, and further work could adapt results from PageRank computation to hitting time.

4.2 Naive Monte Carlo Hitting Time

I begin by presenting a simple naive approach to estimating hitting time The naive algorithm for estimating hitting time simply makes repeated attempts to hit node j from node i.

Algorithm 4.1 (Naive Algorithm)**.** *Given a trust graph $G = (V, E, w)$, simulate $N = mn^2$ runs of α-terminating random walks by taking m walks for each pair of nodes i, j, starting the random walks at node i. Let ζ_{ij} be the number of random walks for the pair i, j that hit node j. Estimate hitting time by $\hat{h}_{ij}^{naive} = \frac{\zeta_{ij}}{m}$.*

Theorem 4.1. *The naive estimator $\hat{h}_{ij}^{naive}$ is unbiased and consistent.*

Proof. The count ζ_{ij} is the sum of m Bernoulli random variables with parameter h_{ij}. Thus, I can interpret $\hat{h}_{ij}^{naive}$ as a random variable

$$\hat{h}_{ij}^{naive} \sim \frac{1}{m}\text{Binomial}(m, h_{ij}). \tag{4.2}$$

We can see that $\mathbb{E}(\hat{h}_{ij}^{naive}) = h_{ij}$ and $Var(\hat{h}_{ij}^{naive}) = \frac{h_{ij}(1-h_{ij})}{m}$, which goes to zero as $m \to \infty$. Thus, $\hat{h}_{ij}^{naive}$ is both an unbiased an consistent estimator. □

4.3 Multihit Monte Carlo Hitting Time

This naive algorithm can be improved considerably. The key insight is that the random walk used in the naive algorithm for estimating h_{ij} and h_{ik} are identical. One random walk could be used to estimate both of these values simultaneously. More generally, a single random walk starting at any node i can be used to estimate the hitting times to all other nodes in the graph.

Algorithm 4.2 (Multihit Algorithm)**.** *Given a trust graph $G = (V, E, w)$, simulate $N = mn^2$ runs of α-terminating random walks initiated at each node exactly mn times. Let ζ_{ij} be the number of random walks that started at node i and passed through j at some point before terminating. Estimate the hitting time by $\hat{h}_{ij}^{mhit} = \frac{\zeta_{ij}}{mn}$.*

Theorem 4.2. *The multihit estimator $\hat{h}_{ij}^{mhit}$ is unbiased and consistent. It also has a smaller variance than the naive estimator $\hat{h}_{ij}^{naive}$.*

Proof. If I write out the count ζ_{ij} from the multihit algorithm as a sum of indicator variables over all random walks that start at i and hit j, i.e.,

$$\zeta_{ij} = \sum_{X:X_0=i} \mathbf{1}\{j \in \{X_t\}_{t=0}^{\tau}\}, \tag{4.3}$$

then we can see that

$$\hat{h}_{ij}^{mhit} \sim \frac{1}{mn}\text{Binomial}(mn, h_{ij}). \tag{4.4}$$

It follows that

$$\mathbb{E}(\hat{h}_{ij}^{mhit}) = h_{ij}, \qquad Var(\hat{h}_{ij}^{mhit}) = \frac{h_{ij}(1-h_{ij})}{mn}, \tag{4.5}$$

which demonstrates that $\hat{h}_{ij}$ is both unbiased and consistent, and has smaller variance than the naive estimator. □

4.4 Multiwalk Monte Carlo Hitting Time

An additional insight improves the multihit algorithm even further by making use of the memoryless property of the Markov process. The idea is that when a random walk passes through a node j, all subsequent states can also be interpreted as a random walk initiated from node j. Thus, a random walk $(X_t)_{t=0}^{\tau}$ that terminates at time τ can be used to simulate $\tau + 1$ subwalks, which initiate from each of the $\tau + 1$ states that the random walk passes through.

Definition 4.1 (Subwalk)**.** *A subwalk of a random walk $(X_t)_{t\geq 0}$ is a sequence of random variables $(Y_t)_{t\geq 0}$ such that $Y_t = X_{t+k}$ for some $k \geq 0$, for all t.*

I also define $\mathcal{S}((X_t)_{t\geq 0})$ as the set of all subwalks of $(X_t)_{t\geq 0}$.

The set of all subwalks can also be defined by considering the subwalks for $k = 0, \ldots, \tau$. For convenience and clarity, I sometimes omit the subscript and simply write a random walk as (X_t) when the condition $t \geq 0$ is implied. I also write $\{X_t\}$ as the set of states that a random walk (X_t) passes through.

Algorithm 4.3 (Multiwalk algorithm)**.** *Given a trust graph $G = (V, E, w)$, simulate $N = mn^2$ runs of α-terminating random walks initiated at each node exactly mn times. Let π_i be the number of subwalks that start at node i, and ζ_{ij} be the number of subwalks that start at node i and hit j before terminating. Then estimate the hitting time by $\hat{h}_{ij}^{mwalk} = \frac{\zeta_{ij}}{\pi_i}$.*

More formally, let χ be the set of all N random walks, and let $\mathcal{S}(\chi)$ be the set of all subwalks of the random walks in χ. I also define χ_i as the subset of random walks that start at i and $\mathcal{S}_i(\chi)$ as

the subset of subwalks that start at i, i.e.,

$$\chi_i = \{(X_t) : X_0 = i, (X_t) \in \chi\}, \tag{4.6}$$

$$\mathcal{S}_i(\chi) = \{(X_t) : X_0 = i, (X_t) \in \mathcal{S}(\chi)\}. \tag{4.7}$$

Then I define the multiwalk estimator as

$$\hat{h}_{ij}^{mwalk} = \frac{\zeta_{ij}}{\pi_i} = \frac{|\{(X_t) : j \in \{X_t\}, (X_t) \in \mathcal{S}_i(\chi)\}|}{|\mathcal{S}_i(\chi)|}. \tag{4.8}$$

Theorem 4.3. *The multiwalk estimator is unbiased and consistent. Its variance is at most the variance of the multihit estimator.*

Proof. I can rewrite our estimator as

$$\hat{h}_{ij}^{mwalk} = \frac{1}{|\mathcal{S}_i(\chi)|} \sum_{(X_t) \in \mathcal{S}_i(\chi)} \mathbb{P}(j \in \{X_t\}), \tag{4.9}$$

which suggests that it follows a binomial distribution:

$$\hat{h}_{ij}^{mwalk} \sim \frac{1}{|\mathcal{S}_i(\chi)|} \text{Binomial}(|\mathcal{S}_i(\chi)|, h_{ij}) \tag{4.10}$$

Similar to above, this shows that the multiwalk estimator is unbiased and consistent.

To show that the multiwalk estimator has variance at at most the variance of the multihit estimator, all I need to do is show that

$$|\mathcal{S}_i(\chi)| \geq mn \tag{4.11}$$

for all possible realizations of the random variable $|\mathcal{S}_i(\chi)|$. We can see this by noting that $|\chi_i| = mn$, and $\chi_i \subseteq \mathcal{S}_i(\chi)$, as desired. □

4.5 Complexity Analysis

The expected number of steps of a single α-terminating random walk is $\Theta(1/\alpha)$ because the walk terminates at each step with probability α. Thus, a Monte Carlo simulations with N total walks takes $\Theta(N/\alpha)$ steps. Both the naive and the multihit estimators simulate N random walks and so should take $\Theta(N/\alpha)$ steps. The multiwalk estimator also considers every subwalk of each walk, which in total takes $\Theta(1/\alpha^2)$ steps for each random walk. Thus in total, the multiwalk estimator takes $\Theta(N/\alpha^2)$ steps.

Now we can see that there is a rough trade-off between the variance and expected number of steps for the Monte Carlo estimators. The multiwalk estimator has an improved variance, but only

by considering subwalks, which also increases its asymptotic complexity. The naive and multihit estimators have a better asymptotic complexity, but they also have a larger variance.

I can theoretically bound the number of random walks needed to obtain an accurate estimate with the Chernoff bound.

Definition 4.2 (Two-Sided Chernoff Bound)**.** *Given N independent random variables $X_1, \ldots, X_N$ with $\mathbb{E}[X_i] = \mu$ and $0 \leq X_i \leq 1$ for all i, let $\overline{X}$ be their mean. Then for any ϵ,*

$$\mathbb{P}(|\overline{X} - \mu| \geq \epsilon\mu) \leq 2\exp\left(-\frac{\epsilon^2}{3}N\mu\right). \tag{4.12}$$

The Chernoff bound allows us to obtain a lower bound for the number of independent samples needed to guarantee that an estimate is within ϵ of the true value. Given ϵ and $0 < \delta < 1$, I call $\overline{X}$ an (ϵ, δ)-approximation if $\mathbb{P}(|\overline{X} - \mu| \geq \epsilon\mu) \leq \delta$. The Chernoff bound tells us we must have $N \geq \frac{3\ln(2/\delta)}{\epsilon^2\mu}$.

Now for the naive algorithm, N total random walks results in $N/|V|^2$ walks for each pair of nodes, so for $\hat{h}_{ij}^{naive}$ to be an (ϵ, δ)-approximation, it requires $N \geq \frac{3|V|^2\ln(2/\delta)}{\epsilon^2 h_{ij}}$. The multihit algorithm takes $N/|V|$ samples to estimate $\hat{h}_{ij}^{mhit}$, and so for it to be an (ϵ, δ)-approximation requires $N \geq \frac{3|V|\ln(2/\delta)}{\epsilon^2 h_{ij}}$. The multiwalk algorithm takes $|\mathcal{S}_i(\chi)|$ samples for each estimator, so for $\hat{h}_{ij}^{mwalk}$ to be an (ϵ, δ)-approximation requires $|\mathcal{S}_i(\chi)| \geq \frac{3\ln(2/\delta)}{\epsilon^2 h_{ij}}$.

In comparison, Hopcroft and Sheldon's algorithm requires $k \geq \frac{3|V|\ln(2/\delta)}{\epsilon^2\alpha}$ for k random walks from a single node to obtain an (ϵ, δ)-approximation for a single h_{ij}. Thus, to approximate all the h_{ij}, the total number of random walks N must be $\mathcal{O}(|V|^3)$, which is higher than any of our Monte Carlo estimators.

4.6 Discussion

This chapter presented three Monte Carlo algorithms for estimating hitting time. There is ample opportunity for research to further improve these algorithms.

Monte Carlo algorithms are embarrassingly parallelizable: The same algorithm can run simultaneously on multiple core or machines, and the estimators from each algorithm only need to be averaged to produce an improved estimator.

Random walk-based simulations also lend themselves to distributed algorithms. Each machine could be responsible for a different subset of the graph, and random walk simulations can be transferred between machines when it passes from one machine's subset to another's.

There is also interesting research to be done in further relaxations of the problem that focus on top-k estimation, i.e., finding the k largest h_{ij} for each i. In the context of reputation mechanisms, there are situations when one only wants to estimate the k most reputable agents, and doesn't need to estimate the reputation score of every single agent in the network. In these cases, an

adaptive algorithm that reallocates random walk simulations to achieve the highest precision for high reputation nodes could be more valuable. Furthermore, given that hitting time and other centrality measures are characterized by a power law distribution under certain network conditions [29], such an adaptive algorithm may be able to find the top-k ranking without too much difficulty.

There exist other promising directions for estimating hitting time by random walk methods, such as truncated random walks [30], which use random walks that are truncated at a maximum length.

Chapter 5

Empirical Comparison of Algorithms

In this chapter, I compare the empirical performance of the exact and Monte Carlo approximation algorithms. I investigate the running time performance, and empirical measures of accuracy for the Monte Carlo estimators.

5.1 Exact Algorithms

I begin by comparing the running times of the exact algorithms. I presented three exact algorithms in Chapter 3:

1. **LS Step Length Algorithm** (Theorem 3.4): Computes step length hitting time in $\mathcal{O}(|V|^4)$ time by solving linear systems of equations.
2. **LS Probability Algorithm** (Theorem 3.5): Computes probability hitting time in $\mathcal{O}(|V|^4)$ time by solving linear systems of equations.
3. **Eigen Probability Algorithm** (Theorem 3.8): Computes probability hitting time in $\mathcal{O}(|V|^5)$ time by solving the eigensystems of matrices.

My theoretical analysis predicts that the Eigen probability algorithm will have the longest running time. However, it is not obvious how the constant factors will play out in empirical tests. For each algorithm, I produce completely connected graphs, as the algorithms do not depend on graph structure. For each graph size, I apply each algorithm to 10 graphs and took the average running time. For consistency, the algorithms are applied to the same set of 10 graphs for each graph size.

The underlying algorithms depend on linear algebra routines from the LAPACK software package, which are written in Fortran 90. In particular, the linear system algorithms use the `_gesv` routine, which gives exact solutions of a linear independent system of equations. The eigensystem algorithm uses the `_geev` routine, which computes the eigensystem of square matrices.

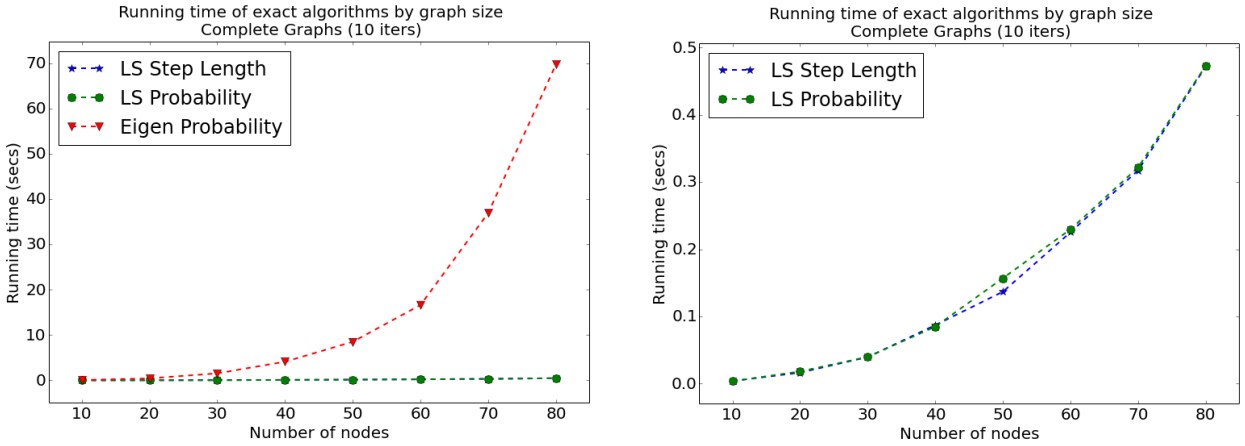


Figure 5.1: Running times of exact algorithms. Full view (left) and detailed view (right).

We see in Figure 5.1 that the two linear system algorithms outperform the eigensystem algorithm even on very small graphs. As the graphs get larger, the running time for the Eigen algorithm quickly explodes, while the LS algorithms continue to compute quickly. It's clear that the linear system algorithms are superior in speed.

5.2 Monte Carlo Simulations

I now provide simulations to demonstrate how each of the Monte Carlo algorithms perform in reality. Not only do I empirically corroborate the theoretical results demonstrated in Chapter 4, but also I examine the running times for each algorithm. I compare the following three Monte Carlo approximation algorithms. Each of the algorithms are implemented in Python.

1. Naive Monte Carlo (Algorithm 4.1).
2. Multihit Monte Carlo (Algorithm 4.2).
3. Multiwalk Monte Carlo (Algorithm 4.3).

Whereas the exact algorithms did not depend on the graph structure, the Monte Carlo algorithms are based on random walk simulations that relate closely to the graph structure. Thus, in the simulations below, I use random graphs generated by the Barabási-Albert (BA) algorithm [31], which produces scale-free graphs by a preferential attachment mechanism. The resultant graphs exhibit a power law degree distribution, which is widely seen in real-world networks [32]. I use N to denote the number of nodes in a graph, K to denote the number of edges per node, and $G(N, K)$ to denote a BA random graph with N nodes and K edges per node. I assign each edge a random weight sampled from a uniform distribution with support $[0, 1]$.

5.2.1 Error

For each of the three Monte Carlo algorithms, I simulate 5,000, 10,000, 15,000, 20,000, 25,000, 50,000, 75,000, and 100,000 random walks to compute an estimate of hitting time. I generate sets of 10 BA graphs $G(50, 5)$ for each simulation and apply all three algorithms to the same set of graphs for comparable results. For each random graph, I compute a gold standard estimate of the hitting times by the LS Probability algorithm, which I use to compute a measure of error for the Monte Carlo algorithms.

Definition 5.1 (Monte Carlo Error)**.** *The error ε of a Monte Carlo algorithm is given by the average difference of each pair of node's hitting time with its estimate:*

$$\varepsilon = \frac{1}{|V|^2} \sum_{i,j \in V} |h_{ij} - \hat{h}_{ij}|. \tag{5.1}$$

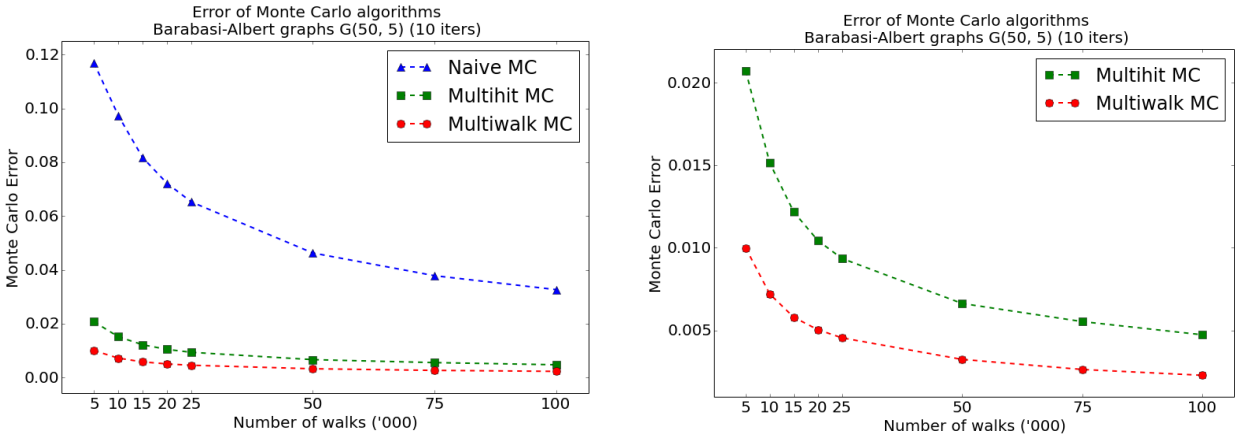


Figure 5.2: Error of Monte Carlo algorithms. Full view (left) and detailed view (right).

We see in Figure 5.2 that the error of each of the three estimators converge toward zero as the number of random walks increases, indicating that all the estimators are consistent. We also see that the naive estimator converges the slowest, while the multiwalk estimator converges the quickest, which corroborates my theoretical results.

5.2.2 Informativeness

While the error of these estimators appear to be relatively small, it isn't clear how they would perform in the context of a reputation mechanism, which depends on the rankings of the estimators. I also measure the informativeness of these estimators, which is computed by taking the average Spearman correlation of the hitting times from each agent's perspective. This is identical to Definition 2.14, except there is a ground truth vector for each agent here, instead of one global ground truth vector.

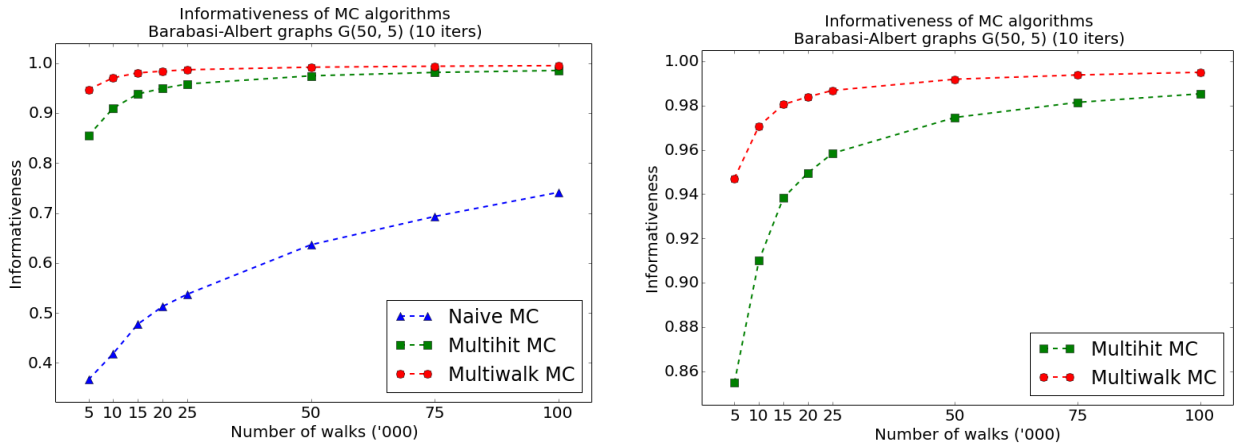


Figure 5.3: Informativeness of Monte Carlo algorithms. Full view (left) and detailed view (right).

Figure 5.3 shows that the multihit algorithm easily obtains informativeness levels above 0.9, whereas the multiwalk algorithm obtains informativeness levels above 0.98. We can also see that the naive algorithm has very poor informativeness in comparison to the other algorithms, to the point that it is likely unusable in practice.

5.2.3 Running Time

As mentioned in Chapter 4, the increased accuracy of the more sophisticated Monte Carlo algorithms comes at the cost of greater time complexity. I measure the running time for each of the algorithms when applied in the experiments above, and took the average over all the trials.

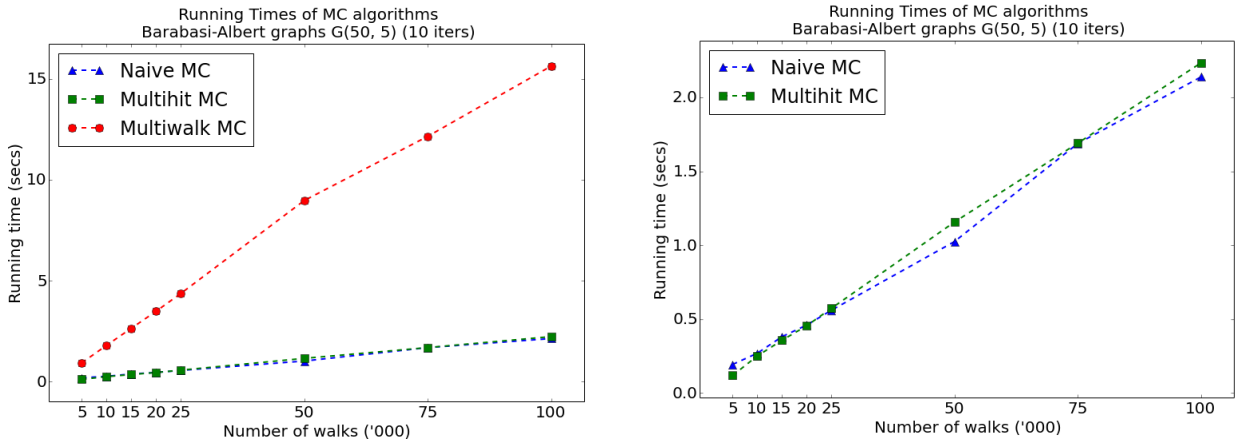


Figure 5.4: Running time of Monte Carlo algorithms. Full view (left) and detailed view (right).

As expected, Figure 5.4 shows that the running time for each algorithm is linear in the number of random walks, and that each algorithm is successively slower. As predicted by our asymptotic

complexity analysis, we can see that the multihit algorithm takes essentially the same amount of time as the naive algorithm, whereas the multiwalk algorithm takes substantially more time.

5.2.4 Graph Size

I examine the effect of increasing graph size on informativeness. I apply the Monte Carlo algorithms with the same number of random walks on BA graphs of increasing size. This gives us a sense of how the performance of the estimators declines as the graph becomes larger and the random walks become more spread out.

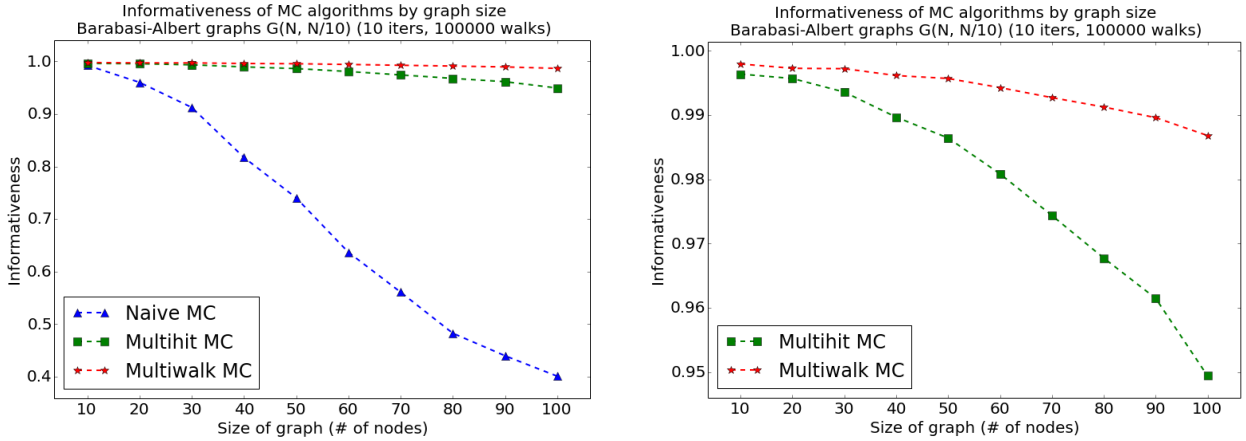


Figure 5.5: Accuracy of Monte Carlo algorithms when varying graph size. Full view (left) and detailed view (right).

Figure 5.5 shows that while the naive esimator quickly deteriorates as the graph becomes larger, the multihit and multiwalk esimators do not decline as rapidly.

5.2.5 Speed and Accuracy

Which Monte Carlo algorithm is best to use? The multiwalk algorithm clearly produces the better estimator given the same number of walks, but it also takes a much longer time to do so. Figure 5.6 plots informativeness against the running time over several trials. The data suggest that overall, the multihit estimator is the best one to use, when considering both informativeness and running time. However, the multihit estimator does not appear to be significantly better than the multiwalk estimator. Which of the two estimators is better depends on variations in the particular implementation.

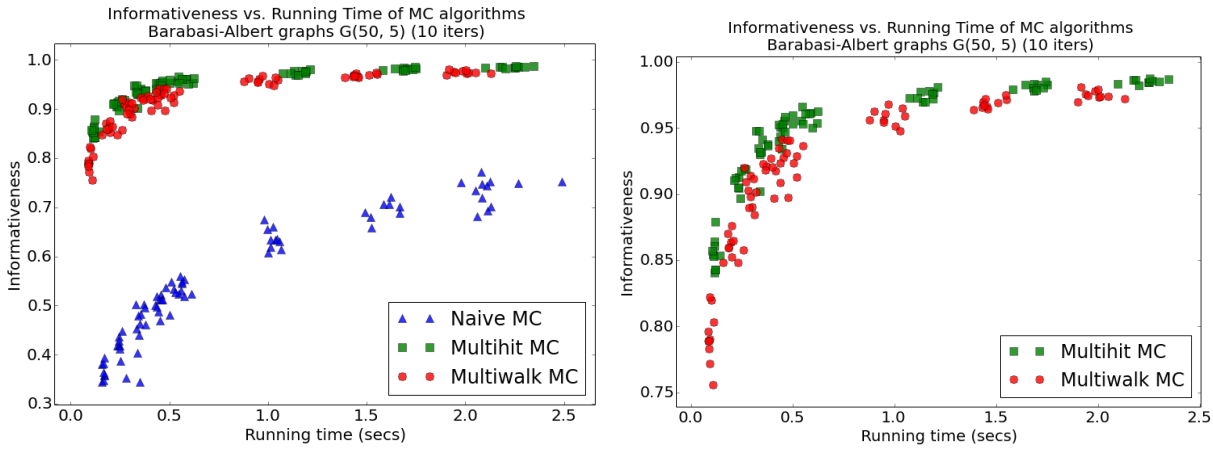


Figure 5.6: Accuracy of Monte Carlo algorithms when varying running time. Full view (left) and detailed view (right).

5.3 Exact vs. Approximation Algorithms

Now I examine how the exact and approximation algorithms would compare in practice. I use the linear system probability algorithm and the multihit algorithm, which appear to be the fastest of the exact and approximation algorithms, respectively. I apply both of these to graphs of increasing size to pinpoint the crossover point at which an approximation algorithm becomes more feasible to compute than an exact algorithm. For comparable results, I use the number of random walks needed for the Monte Carlo algorithm to produce estimates with an informativeness between 0.985 and 0.99 at each graph size.

For a fair comparison between the two algorithms, the linear system algorithm is implemented as Gauss-Jordan elimination in Python. The Fortran implementation is heavily optimized and runs many orders of magnitude faster than a Python implementation. The graphs would have to be extremely large before the Python algorithm would run faster than the Fortran algorithm.

Figure 5.7 shows that even when the approximation algorithm is tuned to return very accurate estimates, it surpasses the exact algorithm in running time at graphs of about 80 nodes in size. This demonstrates that in situations where a close approximation is acceptable, these Monte Carlo algorithms can easily outperform exact algorithms, even for graphs of modest size. However, it is worth noting that this exact crossover point will differ for a different implementation of these algorithms, although the order of magnitude is likely to be similar.

We also see that the number of random walks needed to maintain a stable level of informativeness is roughly quadratic to the graph size. In Chapter 4, we saw that the multihit algorithm only needs a number of random walks linear to the graph size for a given level of error. Indeed, we can see that while the informativeness stays stable, the error continues to drop. This is likely because as the number of nodes increases, the average difference in the hitting times between nodes also decreases,

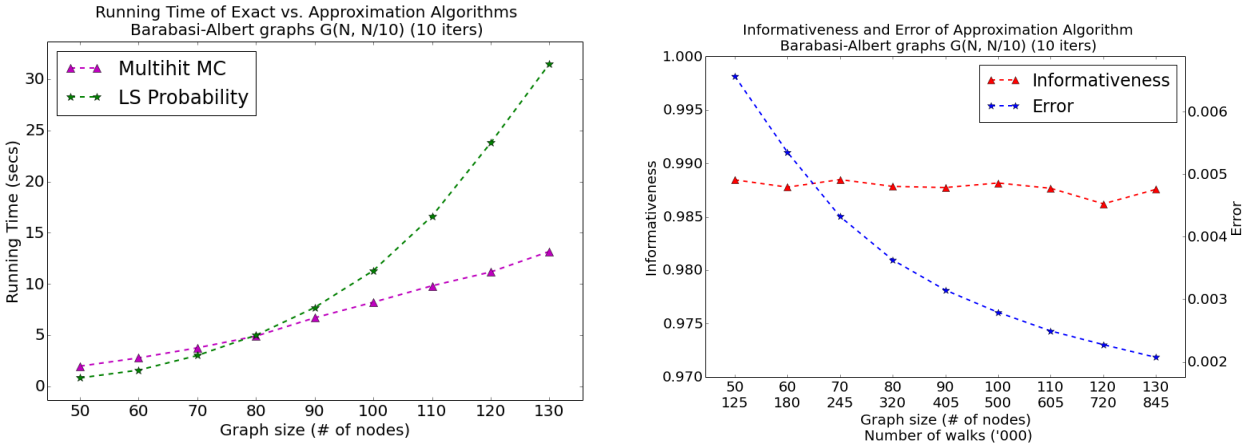


Figure 5.7: Comparison of exact and approximation algorithms for hitting time. Running time (left) and accuracy (right) with increasing graph size.

so a smaller error is needed to rank correctly all the different nodes.

Chapter 6

Empirical Comparison of Reputation Mechanisms

Chapter 2 discussed accuracy and resistance to manipulation as two of the three desirable characteristics of reputation mechanisms. In this chapter, I experimentally compare the six reputation mechanisms defined in Chapter 2 by these two criteria. The first set of experiments measure the informativeness of the reputation mechanisms when there are no strategic manipulations. The second set of experiments examine how the efficiency of the reputation mechanisms fares under varying degrees of manipulation.

6.1 Informativeness Without Manipulation

I first compare the informativeness of the reputation mechanisms in the absence of strategic manipulations. In Chapter 2, I discussed the trade-off between accuracy and resistance to manipulation. Thus, here we should see that the mechanisms with the least resistance to manipulation are the most accurate when there are no manipulations.

6.1.1 Experimental Setup

At a high level, the experimental setup is summarized in five steps: (1) initialize agent true types; (2) select directed edges between agents; (3) determine edge weights; (4) apply reputation mechanisms; (5) measure informativeness.

Initializing Agent Types

I considered three different prior distributions for initializing the agent types $\vec{\theta} \in [0,1]^N$:

Uniform : $\theta_i \sim \text{Unif}(0,1)$

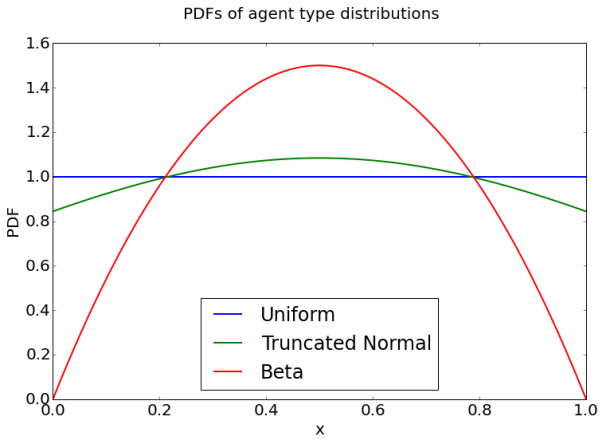


Figure 6.1: PDFs of agent type distributions.

Truncated Normal : $\theta_i \sim \text{Normal}(0.5, 0.5), \theta_i \in [0, 1]$

Beta : $\theta_i \sim \text{Beta}(2, 2)$

Figure 6.1 plots the probability density functions of these three distributions. The beta distribution has the highest probability density at the mean, while the normal is a bit more spread out. The uniform distribution is perfectly distributed across all agent types.

Selecting Edges

In this experiment, I assume that all agents in the graph have the same number of K outgoing edges. This number K can be interpreted as the size of the *memory set*, or the number of counterparties that each agent remembers. I consider two ways of selecting neighbors for each agent.

First, I consider a **uniform** method, in which each agent selects a set of K neighbors uniformly from all K-subsets of the other $N - 1$ agents. In this method, the graph structure itself contains no information about the agents or their reputation.

However, it is fairly common that the structure of real world graphs embed information about the nodes involved. For instance, PageRank is an effective algorithm for search engines because it is based on the premise that the importance of web pages can be determined by the web link structure. Thus, I consider a **cluster** method, which is based on the intuition that high type agents are more likely to associate with other high type agents in a homophilic manner. This kind of graph is similar to a preferential attachment model, like the Barabási-Albert graph model, except it explicitly takes into account the true types of the nodes involved.

In the cluster method, each agent i selects another neighbor j by the following process. With probability θ_i, agent i selects agent j according to the softmax probability $p_j = \frac{e^{\theta_j/z}}{\sum_{j'} e^{\theta_{j'}/z}}$ for non-neighbors j', and with probability $1 - \theta_i$, selects j uniformly from the set of non-neighbors. The

softmax distribution is more likely to sample high type agents than low type agents. Thus, high type agents are more likely to select other high type agents as neighbors, whereas low type agents are more likely to have a random subset of agents as neighbors. The parameter z can be tuned for the degree of homophily. Here, I use $z = 0.05$, such that there is a noticeable difference between the cluster method and the uniform method.

I vary the value of K, which increases the amount of information contained in the trust graph. We expect the performance of the reputation mechanisms to improve with K.

Determining Edge Weights

I use a parameter t for the accuracy of information on each edge. This denotes the *number of interactions* each agent has with its neighbors. In this manner, w_{ij} represents agent i's estimate of agent j's trustworthiness, based on these interactions. I vary the value of t and also consider the limiting case of $t = \infty$. I consider three methods for determining edge weights.

In the **sample** method, I consider a simple model in which all agents are better able to estimate the true type of other agents as the number of interactions increase. I suppose that each interaction on edge (i, j) is good with probability θ_j, and that each agent simply takes the average success rate of its interactions with other agents. Thus, $w(i, j) \sim \frac{1}{t}\text{Binom}(t, \theta_j)$, and $w(i, j) = \theta_j$ when $t \to \infty$.

I also consider a **noisy** method, in which I model high type agents as having better information than low type agents. For each interaction on edge (i, j), with probability θ_i the outcome is good with probability θ_j (i.e., the true type is used), and with probability $1 - \theta_i$ the outcome is good with probability 0.5 (i.e., the mean of the distribution on agent types is used). When $t \to \infty$, we have $w(i, j) = \theta_i\theta_j + (1 - \theta_i)(0.5)$. In this manner, low type agents are more likely to get a "noisy" view of other agents through their interactions.

One potential issue with the noisy method is that low type agents are always be biased in their view of other agents, even as t becomes very large. To ameliorate this, the **prior** method improves upon the noisy method by reducing the bias of $w(i, j)$ for low type agents i. For each interaction on edge (i, j), with probability θ_i the outcome is good with probability θ_j, and with probability $1 - \theta_i$, the interaction is good with probability θ'_j, which is drawn from a distribution centered at θ_j.

In the case that the prior distribution is uniform, θ'_j is sampled from a uniform distribution with mean θ_j and a support of width $1 - \theta_i$. If the prior distribution is truncated normal, then θ'_j is sampled from a truncated normal with mean θ_j, variance $\frac{1}{2}(1-\theta_i)^2$, and a support of width $1-\theta_i$. If the prior distribution is beta, then I sample from a Beta(2, 2) distribution that is shifted and scaled such that the support is centered at θ_j with width $1 - \theta_i$. In all cases, any probability mass outside of $[0, 1]$ is placed as dirac-delta mass at 0 or 1. When $t \to \infty$, we have $w(i, j) = \theta_i\theta_j + (1 - \theta_i)E[\theta']$. Whenever the distribution for θ'_j must be "squished" to fit inside the interval $[0, 1]$, $\mathbb{E}[\theta'_j] \neq \theta_j$. Thus, $w(i, j)$ is more likely to biased for low type agents, although overall the bias is less than in the noisy method. In this manner, high type agents still have better information than low type agents, but

all agents still generally correct information.

Applying Reputation Mechanisms

I consider the six reputation mechanisms defined in Chapter 6: Global PageRank (PR), personalized PR, global hitting time (HT), personalized HT, personalized maximum flow (MF), and personalized shortest path (SP).

Measuring Informativeness

I compute informativeness as defined in Definition 2.14, and I take the average over multiple iterations to obtain an accurate estimate.

6.1.2 Experimental Results

My first set of experiments vary the size K of the memory set for each agent. Increasing K increases the density of the trust graph, and informally speaking, increases the amount of information contained in the trust graph. Intuitively, we would expect informativeness to improve as K increases. Given the trade-off between accuracy and resistance to manipulation, we should also expect those mechanisms with the least resistance to manipulation to perform the best. Based on the results from Chapter 2, we should expect global PR to perform the best, followed by global HT, personalized PR, personalized HT, personalized MF, and personalized SP.

I produce trust graphs with 50 nodes and vary the number of edges per node. I use 32 samples for each edge when determining the edge weights. I produce a set of 20 graphs for each K, and run all six reputation mechanisms on each graph, measuring their performance by computing the average informativeness of their rankings over all 20 graphs in each set.

Figure 6.2 shows three representative plots from the different combinations of experimental parameters discussed above. Global PR and global HT perform better than the rest, and perform identically to each other. They are closely followed by personalized PR and personalized HT, which are both slightly lower than their global counterparts. Personalized MF and SP perform worse than the rest. This corroborates our theoretical predictions based on strategyproofness results.

Because I found no qualitative difference between different agent type distributions, I do not show plots from the Truncated Normal or Beta agent type distributions. The cluster method of edge selection does not differ significantly from the uniform method, except that the performance of every mechanism improves and the gap between the relative performance closes. Interestingly, MF outperforms SP in the noisy edge weight method, but SP outperforms MF in the sample and prior methods. It appears that the SP mechanism's performance improves as the graph density increases, but MF does not improve substantially. The Appendix confirms that the performance of the MF mechanism does not improve as the graph density increases (Appendix A).

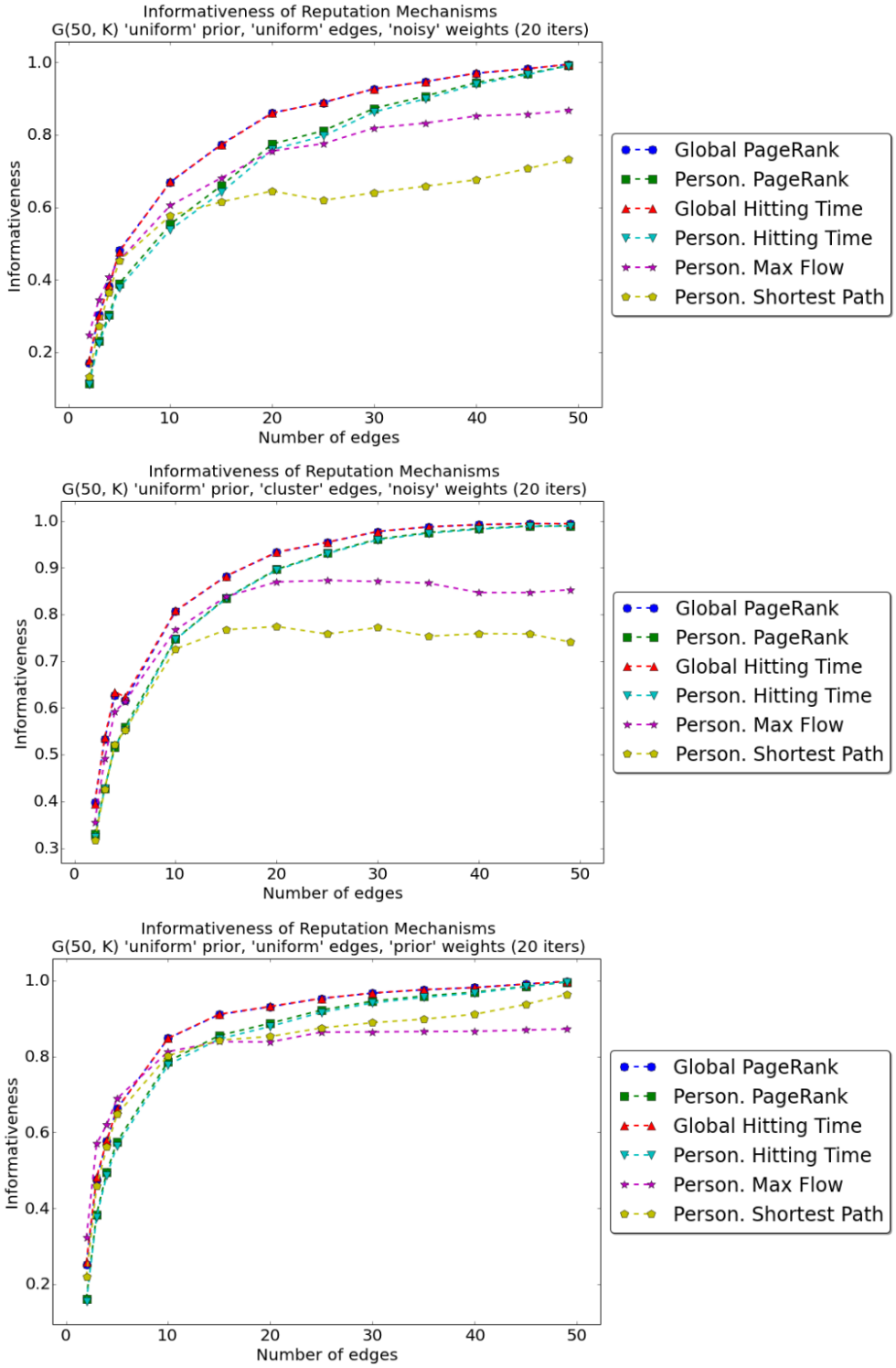


Figure 6.2: Informativeness without manipulations, varying the number of edges per node.

The second set of experiments vary the number of interactions t between each pair of agents. Increasing t improves the quality of the agent trust reports, because each agent has more samples to judge other agents. Intuitively, the informativeness should increase with t.

I produce trust graphs with 50 edges and 30 edges per node, and vary the number of samples t used to determine the edge weights. I produce a set of 20 graphs for each t, and compare each mechanism's average informativeness over the 20 graphs.

Figure 6.3 shows three representative plots. Overall, all the mechanisms with the exception of SP improve very little with increasing t. The two global mechanisms outperform the other mechanisms, and are closely followed by personalized PR and HT. The performance of MF is good for low levels of t, but does not improve by much as t increases. The performance of SP is very poor for low levels of t, but increases dramatically with t and sometimes surpasses the performance of MF.

Again, there are no significant qualitative differences when varying the agent type distribution. It is fascinating to see that the PR and HT mechanisms perform extremely well on trust graphs produced by the cluster edge selection method even when t is very low. The cluster method embeds additional information about the agent true types in the graph structure, which the PR and HT algorithms are able to leverage.

In summary, the global mechanisms have the best performance in the absence of manipulation. When both K and t are not too small, the personalized PR and HT perform quite well as well. For a given K, the PR and HT mechanisms generally outperform MF and SP when t is small. However, as t increases, MF and SP may sometimes outperform the personalized PR and MT mechanisms, but never their global counterparts.

6.2 Efficiency Under Manipulation

Now we shall see how these mechanisms hold up under strategic manipulation. I apply the manipulations described in Chapter 2, and assess the performance of each reputation mechanisms under modified trust graphs.

I adopt the efficiency metric as the measure for a reputation mechanism's performance. While the informativeness metric gives a sense of how a reputation mechanism's ranking matches up to the true ranking of agent types, it does not capture the extent to which a reputation mechanism's rankings are able to improve the number of successful transactions, which is ultimately the desired result of a reputation mechanism. Now that we are considering strategic behavior, it is more suitable to adopt a metric that models an economic setting.

6.2.1 Experimental Setup

At a high level, the experimental procedure can be summarized in four steps: (1) Generate a trust graph, according to the methods described in Section 6.1; (2) apply strategic manipulations; (3)

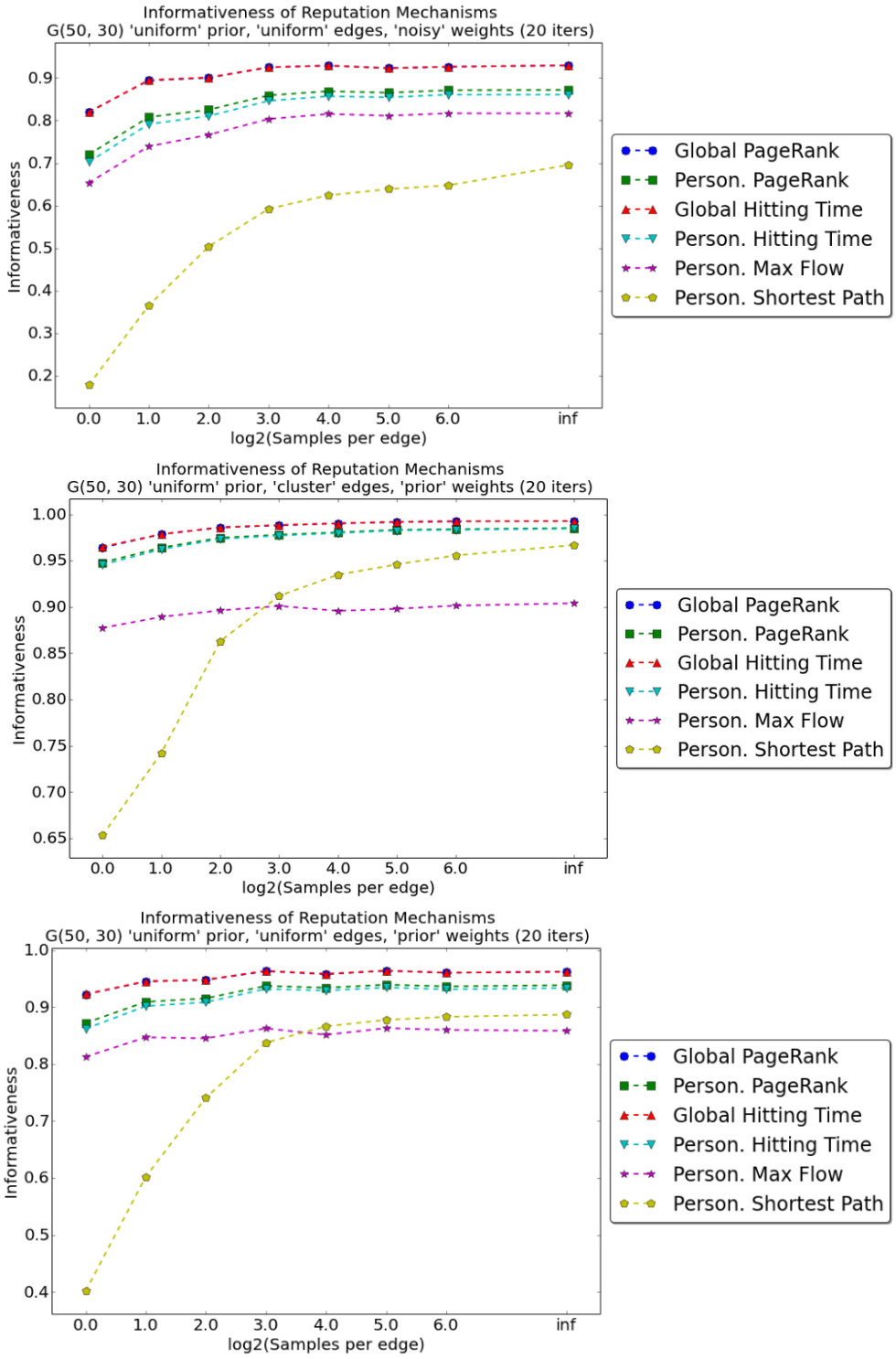


Figure 6.3: Informativeness without manipulations, varying the number of samples per edge.

apply reputation mechanisms; (4) measure efficiency.

The procedures for steps (1) and (3) are identical to the previous set of experiments. The procedure for measuring efficiency is given by Definition 2.15 in Chapter 2.

Strategic Manipulations

I select a subset of $N_m = p_m N$ agents that apply strategic manipulations. These N_m agents are selected uniformly from all N agents. For each agent, I consider two kinds of manipulations, as discussed in Chapter 2: **cutting outlinks** and **sybil attacks**.

An agent applying the cutting outlinks manipulation reports no edges in its trust report to the reputation mechanism. There are a few details in the experimental procedure of applying this manipulation, which are described in Appendix B.

An agent applying a sybil attack manipulation creates N_s sybils, and reports edges with very high weight between each sybil and itself. The number of sybils created by each strategic agent is given by a proportion p_s of non-sybil agents, i.e., $N_s = p_s N$. However, one must be careful to randomize the number of sybils created by each agent – otherwise we can actually observe a rebound effect in which the efficiency decreases and then increases as the number of strategic agents increases (Appendix C).

With the exception of the first set of experiments, I only apply manipulations for vulnerable mechanisms, i.e., only to mechanisms for which the manipulations are actually beneficial to the strategic agent. In particular, I never apply the sybil attack manipulation to MF and SP, and I never apply the cutting outlinks manipulation to SP.

I vary the proportion of strategic agents p_m as well as the proportion of sybils per agent p_s, thus evaluating the effect of the concentration of strategic agents and the number of sybils created per agent.

Measuring Efficiency

After the rankings are generated by the reputation mechanisms, I compute efficiency according to the expression given in Theorem 2.1 and take the average over multiple iterations to obtain an accurate estimate. I use $\kappa = 5$ when computing efficiency.

6.2.2 Experimental Results

The first set of experiments varies the proportion of sybils per agent p_s. In these experiments, I apply only the sybil attack manipulation. For each value of p_s, I create a set of 20 graphs, apply each manipulation to all 20 graphs, and take the average efficiency over all 20 graphs. In this experiment, I created sybils for all reputation mechanisms, including MF and SP, to empirically demonstrate that they are strategyproof to sybil attacks.

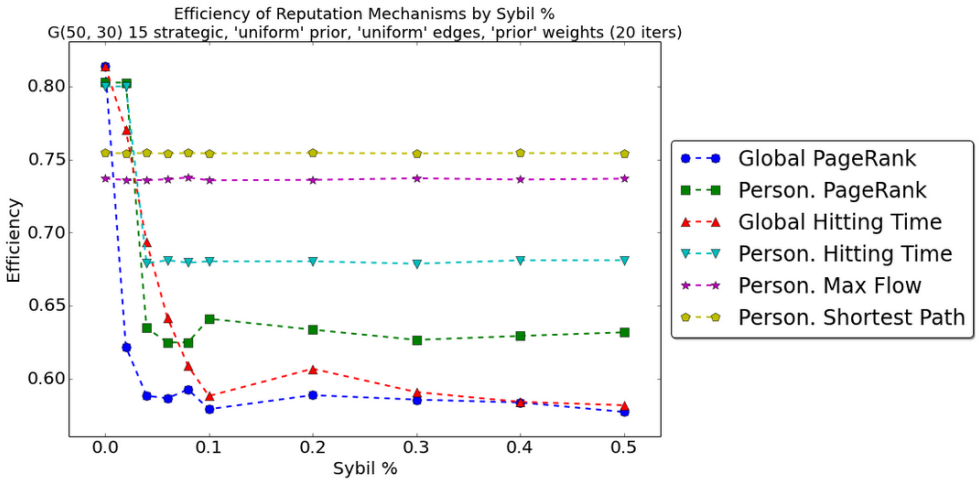


Figure 6.4: Efficiency of reputation mechanisms with varying sybil proportion.

The plots are not qualitatively different for different methods of generating the trust graph, so I show only one representative plot here. We see in Figure 6.4 that MF and SP are completely unaffected by sybils. The remaining reputation mechanisms quickly decline in efficiency until $p_s = 0.1$ and then flattens out. Notably, the two global mechanisms perform the worst, followed by personalized PR, and with personalized HT performing the best out of the non-strategyproof mechanisms. This suggests that the ability for sybils to capture restart probability is the most potent part of the sybil attack, and that the 2-loop aspect of the sybil attack is slightly less effective. The effect of a sybil attack on lengthening random walks, as in the case of personalized HT, seems to be the least effective of all.

For the remainder of the experiments, I use $p_s = 0.4$ with applying sybil attacks to ensure the full effect of the manipulation, although a lower value would likely suffice as well.

I vary the proportion p_m of agents that behave strategically and apply manipulations. I take trust graphs with 50 nodes, 30 edges per node, and 8 samples taken per edge. For each mechanism, I only apply the manipulations to which the mechanism is vulnerable. I consider the cases when only the cutting outlinks or sybil attack manipulations are used, respectively, and the case when both are used together.

Figure 6.5 shows a representative plot of the effect of increasing p_m when we only consider the cutting outlinks manipulation. The plots are not qualitatively different for different methods of generating the trust graph except for the change in order between MF and SP with the edge weight method, as seen above. The efficiency of all the mechanisms except for SP decreases only slightly as p_m increases. The ordering of the mechanisms is generally the same as we saw in the informativeness experiments without manipulations: the global mechanisms, followed by personalized PR and HT, with MF and SP trailing. Even though SP is strategyproof to this manipulation, cutting outlinks has such a minor effect on the other mechanisms that SP still ultimately performs the worst of all.

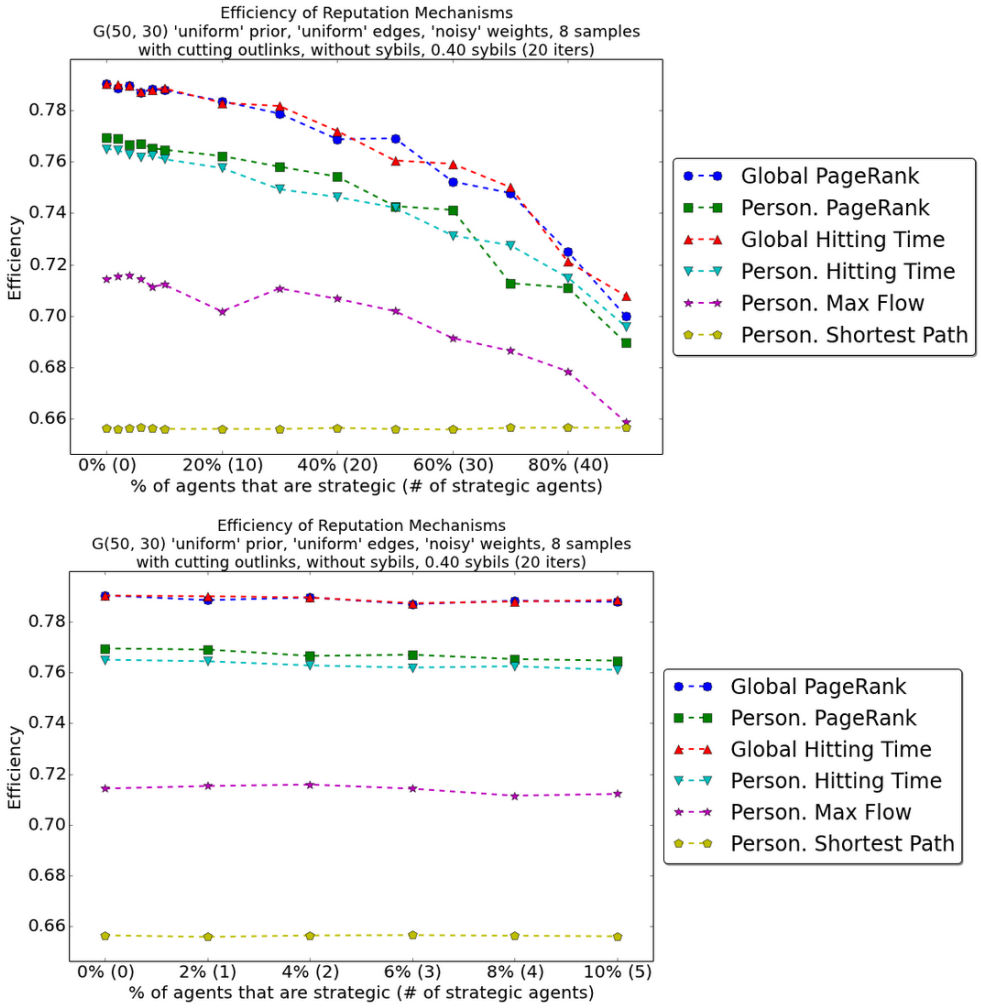


Figure 6.5: Efficiency of reputation mechanisms when varying the proportion of strategic agents, and restricting to the cutting outlinks manipulation. Full view (top) and detailed view (bottom).

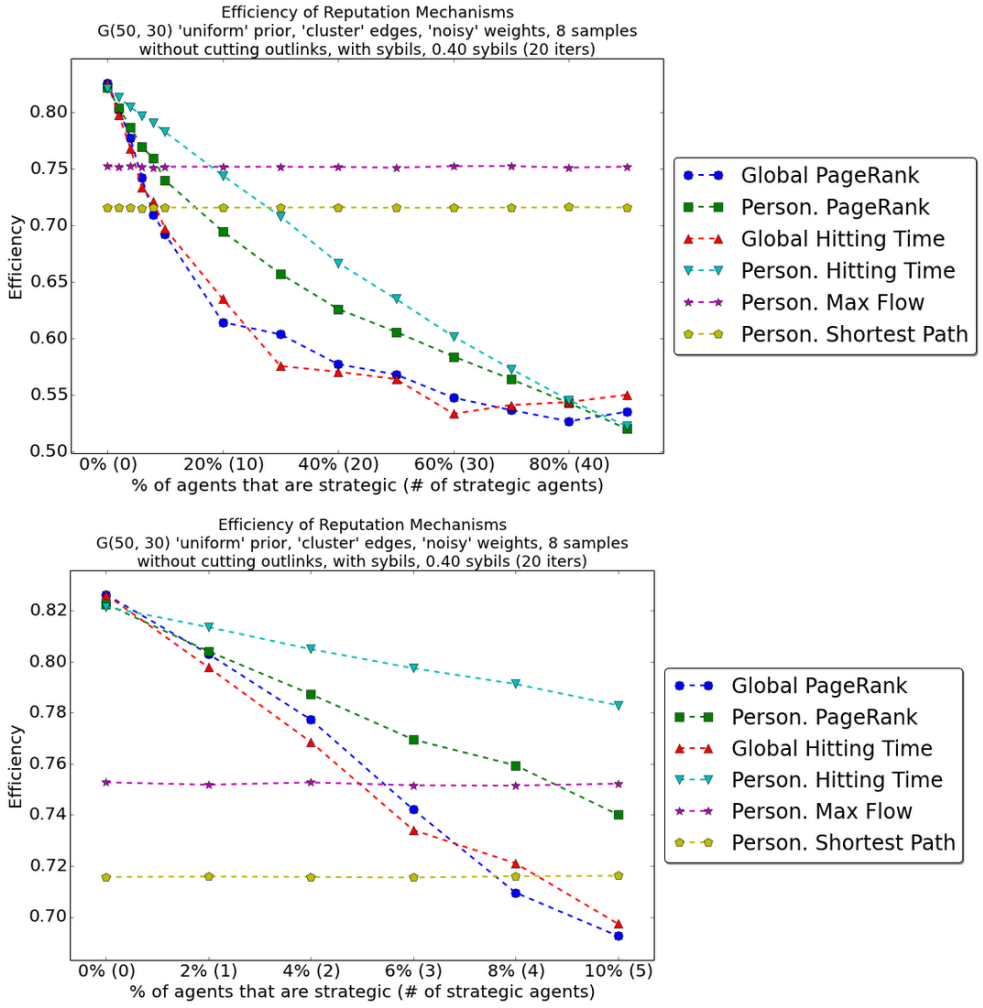


Figure 6.6: Efficiency of reputation mechanisms when varying the proportion of strategic agents, and restricting to the sybil attack manipulation. Full view (top) and detailed view (bottom).

Figure 6.6 shows a representative plot of the effect of increasing p_m when we only consider the sybil attack manipulation. The plots with other methods of generating the trust graph are not qualitatively different. This starkly shows the trade-off between accuracy and resistance to manipulation. The MF and SP mechanisms performed the poorest when there was no manipulation, but when the majority of the agents behave strategically, MF and SP outperform all the other mechanisms. We also see that the global mechanisms, which had the highest accuracy when there was no manipulation, performs the worst with only a small proportion of strategic agents. Personalized PR and HT reside in a middle ground: they are more accurate than MF and SP while being more resistant to manipulations than global PR and HT, and they excel when there is only a moderate proportion of strategic agents. In particular, personalized HT is the most efficient mechanism when the proportion of strategic agents is between 1% and 20% of the total number of agents, for these parameters.

Figure 6.7 shows the effect of increasing p_m when considering both kinds of manipulations. Given that the effect of the cutting outlinks manipulation is relatively minor, this plot is quite similar to Figure 6.6. Similar to above, we see that personalized HT performs the best when the proportion of strategic agents is less than 20%. When the proportion of strategic agents is greater than 20%, MF and SP outperform the other mechanisms by a large margin. However, the MF mechanism is very poor at ranking high type agents (Appendix A). In practice, a good ranking of high type agents is more important than a good ranking of low type agents. Thus, the SP mechanism may be preferable in situations with many strategic agents.

6.3 Discussion

In this chapter, I provided an empirical comparison of six reputation mechanisms by accuracy and resistance to manipulation. In accordance with our theoretical results, the global reputation mechanisms, which had the least resistance to manipulation, have the best accuracy when there are no manipulation, but perform the worst in the presence of strategic agents. Maximum flow and shortest path, which are the only mechanisms strategyproof to the sybil attack manipulation, perform worse when there are no manipulations, but outperform all other mechanisms when a significant proportion of the agents behave strategically.

This chapter shows that the most efficient reputation mechanism depends on the properties of the trust graph and the behavior of the agents in the graph. If none of the agents behave strategically, then the global mechanisms are the most efficient. If the majority of the agents behave strategically, then a mechanism with strong resistance to manipulation, like shortest path, is preferable. If only a modest proportion of agents behave strategically, then personalized hitting time is best.

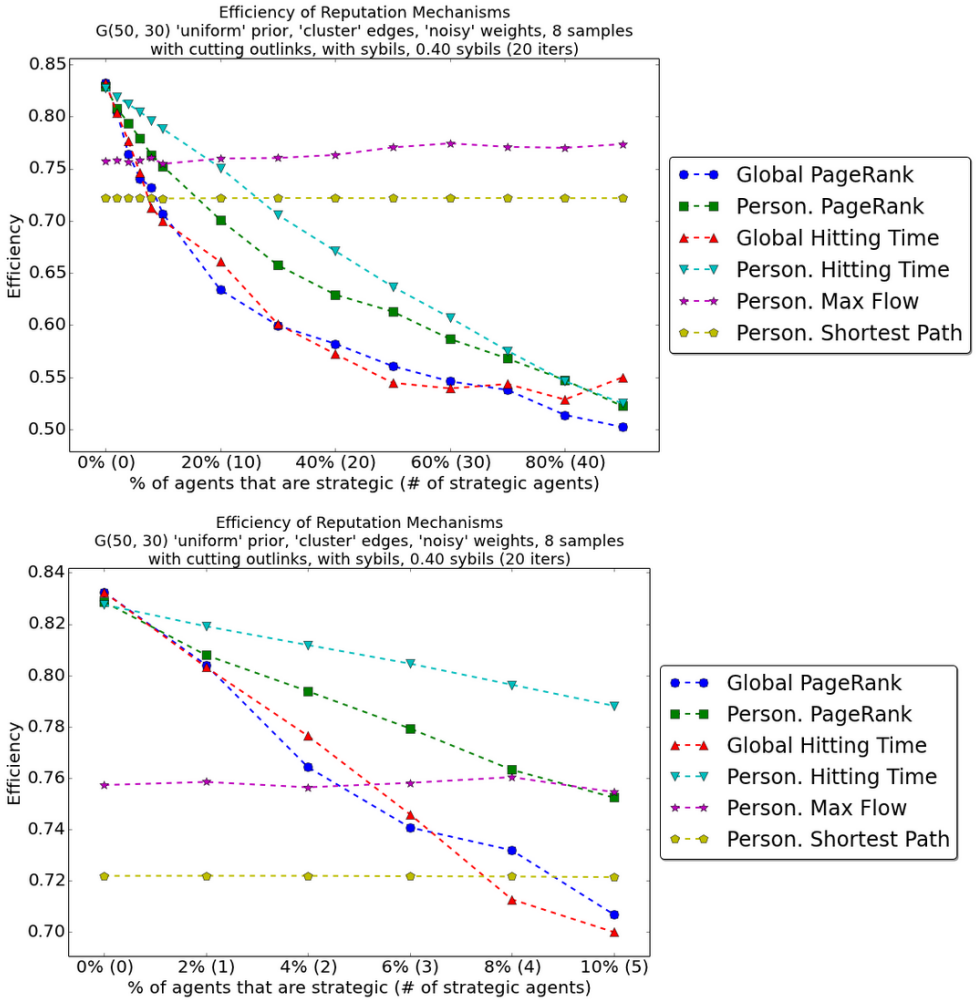


Figure 6.7: Efficiency of reputation mechanisms when varying the number of strategic agents, and applying both cutting outlinks and sybil attack manipulations. Full view (top) and detailed view (bottom).

Chapter 7

Conclusion

In this thesis, I showed that personalized hitting time is a good reputation system according to the three criteria of accuracy, resistance to manipulation, and computational tractability. I presented two definitions for hitting time, one of which is amenable to Monte Carlo algorithms. I presented exact algorithms based on matrix algebra and approximation algorithms based on Monte Carlo random walk simulations, and proved their correctness. I showed that the approximation algorithms can obtain highly accurate estimates of hitting time on large graphs in less time than an exact algorithm can obtain an exact calculation. I experimentally compared the performance of six reputation mechanisms in both the absence and presence of strategic manipulations, and showed that personalized hitting time has the best performance in the presence of a moderate number of strategically behaving agents.

Previous work has theoretically shown hitting time to be a manipulation-resistant reputation mechanism [23], but this is the first experimental validation of hitting time's performance compared against other reputation mechanisms. I also present efficient algorithms for computing hitting time, which makes the personalized hitting time reputation mechanism a practical possibility for real-world implementations.

The excellent experimental performance of personalized hitting time warrants further research into more efficient algorithms for practical usage. The end of Chapter 4 described many exciting directions to improve the Monte Carlo algorithms for computing hitting time, including parallelization, top-k adaptive algorithms, and truncated random walks. A hitting time implementation that can scale to massive real-world datasets would open the potential for a highly accurate, manipulation-resistant reputation mechanism to be applied in numerous online contexts, thus enabling more cooperative and socially beneficial behavior.

Appendix A

Maximum Flow Mechanism

A.1 Sensitivity of Informativeness to Number of Edge Samples

For reasons that are not yet completely understood, the performance of the maximum flow mechanism appears to be very sensitive to the particular value of t used in the edge weight determination method. Figure A.1 shows that when $t = 30$ (left figure) is used, the informativeness of MF can actually decrease with the graph size. In contrast, a value of $t = 32$ (right figure) shows a roughly monotonically increasing curve for the informativeness of maximum flow.

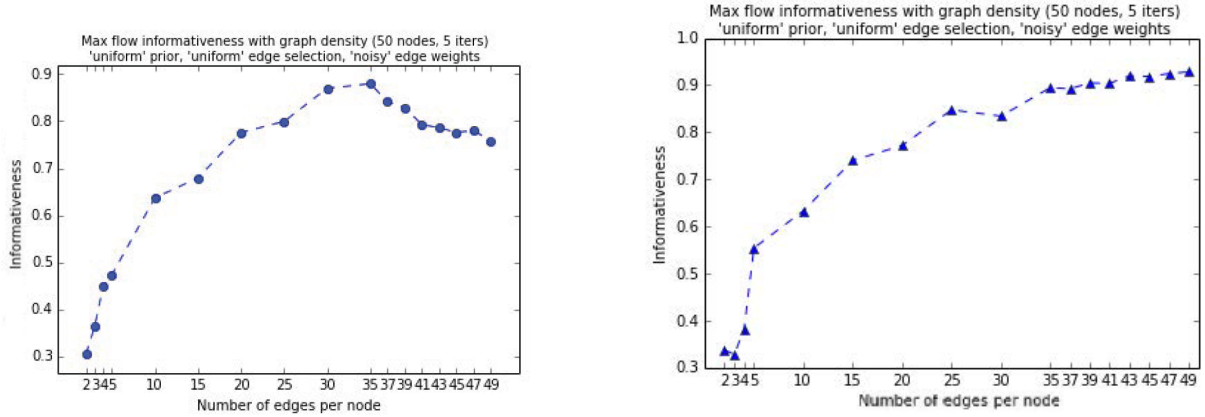


Figure A.1: Informativeness of maximum flow when varying edge density on trust graphs generated with 30 (left) and 32 (right) samples per edge.

Further investigation shows that $t = 32$ is the only value out of $t = [30, 33]$ that results in a monotonically increasing curve (Figure A.2, left figure). The right figure in Figure A.2 shows that the informativeness of maximum flow on complete graphs varies dramatically with the value of t. The plots shown here were generated using the noisy edge selection method, but the same phenomenon is seen for all edge selection methods.

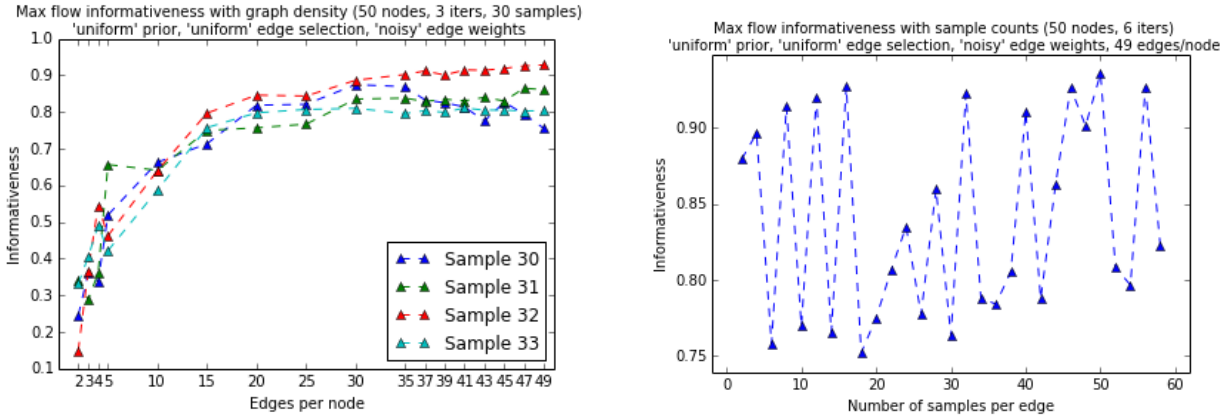


Figure A.2: Informativeness of maximum flow (MF) when varying the number of samples per edge. Informativeness of MF when varying the graph size (left), and when varying the number of samples per edge on complete graphs (right).

I was not able to determine the underlying cause for this phenomenon. However, I found that values of t that were a power of 2 produced informativeness curves that were monotonically increasing. Thus, in this thesis, I only used values of $t = 2^k$ for varying k.

A.2 Limitations of the Maximum Flow Mechanism

A major limitation of the maximum flow mechanism is that agents poorer at ranking high type agents than at ranking low type agents. This phenomenon especially holds true for graphs with higher density. In Figure A.3, I plot the ability of agents to correctly rank lower and higher type agents. I partition the nodes into lower ($\theta_i < 0.5$) and higher ($\theta_i > 0.5$) types, and compute the Spearman correlation of each partition against the true types. The ranking of lower type agents is much closer to the true ranking than the ranking of higher type agents.

This phenomenon is due to the fact that the amount of flow any node can push out of its out-edges is likely to stay constant, but the amount flow an node can receive through its in-edges is correlated to its true type. If we suppose the uniform edge selection and the sample edge weight determination method, then each node has K out-edges each with an expected weight of 0.5 (the mean of the agent type distribution). However, an node j with true type θ_j will have an expected K in-edges each with an expected with of θ_j. In a dense graph where most of the outflow from the source will make it to the sink node, the maximum flow to a sink j with $\theta_j > 0.5$ will likely be close to $0.5K$, which makes the higher type nodes difficult to distinguish. The maximum flow to a sink j with $\theta_j < 0.5$ is more likely to be close to $\theta_j K$, which makes lower type nodes easier to distinguish because the maximum flow to a lower type node correlates with its true type.

This phenomenon is unfortunate, because an accurate ranking of higher type agents is more

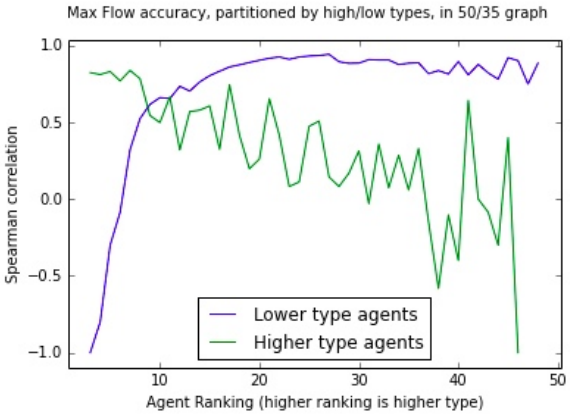


Figure A.3: Correlation of maximum flows to agent true types in lower and higher type agents.

desirable than an accurate ranking of lower type agents. In practice, it is often much more important to determine the best agent to transact with, or the top k agents to transact with. Any reputation mechanism that depends on maximum flow will likely have to overcome this issue.

Appendix B

Cutting Outlinks Manipulation

B.1 Restoring Outlinks of Strategic Agents

When a personalized mechanism computes the personalized ranking $\prec_i$ for a strategic agent i, it needs the true out-edges of agent i. Without the out-edges of agent i, the agent is not connected to any of the other nodes and the reputation mechanism cannot meaningfully compute $\prec_i$. In reality, this reflects a kind of distributed mechanism, in which strategic agents know their own true trust report, which they can use when evaluating other agents, but which may be withheld from other agents' uses. In the context of a centralized mechanism in which an agent can only submit one trust report, this model of cutting outlinks captures the maximal possible effect that this manipulation can have on diminishing the efficiency of the reputation mechanism.

When a large number of agents are strategically cutting their out-edges, portions of the graph may become disconnected. The personalized mechanisms are unable to evaluate pairs of nodes that are disconnected from each other. To address this issue, I apply a "thin" edge of weight ε where ε is very small from any agent that cut its outlinks to all other agents. This ensures that the graphs stays connected, while minimizing the perturbation to the reputation mechanism. I select ε sufficiently small such that any real edge outweighs the presence of other thin edges.

Appendix C

Sybil Attack Manipulation

C.1 Randomizing the Number of Sybils

When the sybil attack manipulation is applied in an experimental context, we must take care to randomize the number of sybils that the strategic agents create. In the case that all the strategic agents create the exact same number of sybils, we can actually observe a "rebound" phenomenon in which the informativeness and efficiency of a mechanism can appear higher in the presence of a high proportion of strategic agents than a moderate proportion of strategic agents. Figure C.1 shows how this rebound effect manifests for global PR, global HT, and personalized PR.

When a strategic agent applies a sybil attack manipulation in the context of the three reputation mechanisms mentioned above, it is able to increase its reputation ranking significantly, surpassing agents without sybils. However, when multiple agents are applying the sybil attack, how do they compare to one another? As the proportion of agents applying the sybil attack nears unity, the sybil attack becomes less meaningful, because most of the agents are also doing it. The differences between the agents' reputation scores become very minute. The truthful edges in the graph, which capture the true types of all the agents, allow the reputation mechanism to meaningfully distinguish between strategic agents.

For example, in one instance, the Spearman correlation of the global PageRank scores with the true types was only 0.154, which reflects the successful manipulation by strategic agents. However, the Spearman correlation of the global PR scores and true types of the strategic agents and non-strategic agents were 0.71 and 0.83, respectively. This indicates that the ranking within each of those two partitions is still relatively correct, and thus the global informativeness also improves as the proportion of strategic agents becomes very high.

To address this issue, I randomize the number of sybils that each strategic agent applies, according to a uniform distribution centered at $N_s = p_s N$. The ranking of strategic agents essentially becomes a random ranking. This gives us our desired effect of the sybil attack manipulation.

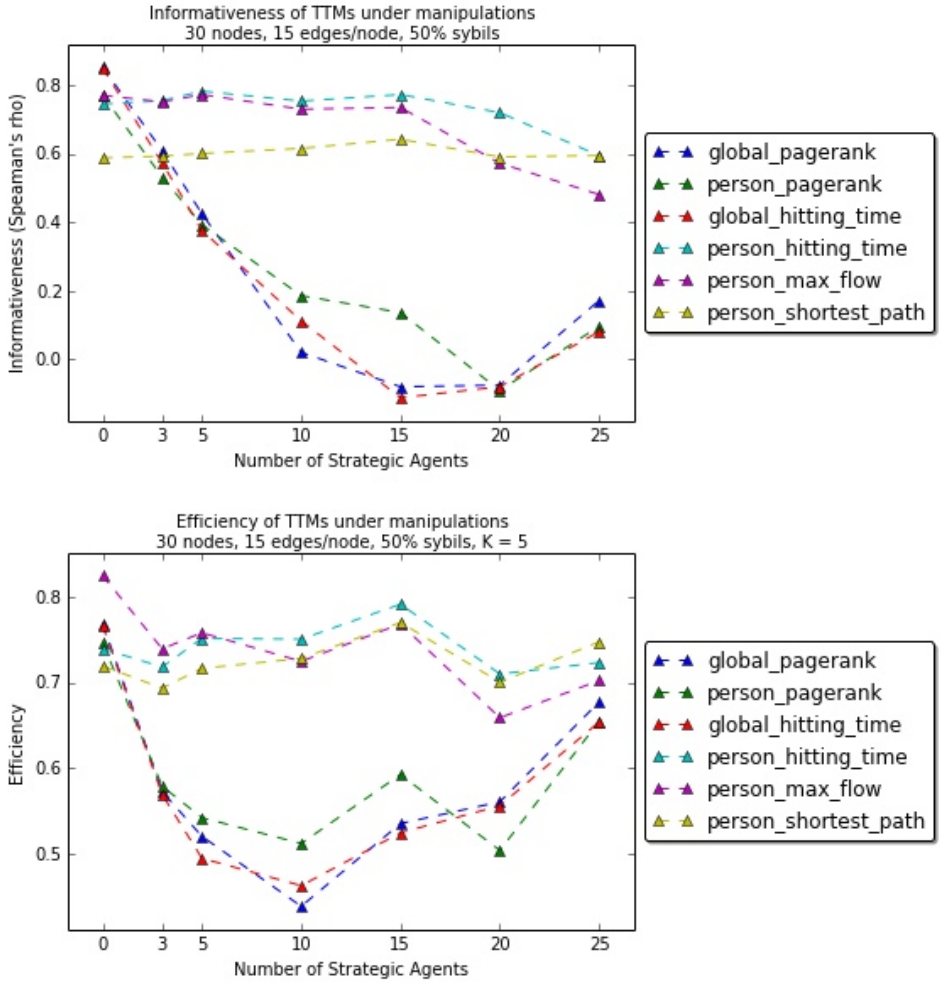


Figure C.1: Informativeness and Efficiency increases with a higher proportion of stategic agents when not randomizing the number of sybils per agent.

Bibliography

[1] G. Akerlof, "The market for" lemons": Quality uncertainty and the market mechanism," *The Quarterly Journal of Economics*, vol. 84, no. 3, pp. 488–500, 1970.

[2] P. Resnick, R. Zeckhauser, J. Swanson, and K. Lockwood, "The value of reputation on eBay: A controlled experiment," *Experimental Economics*, vol. 9, pp. 79–101, June 2006.

[3] Z. Gyongyi, H. Garcia-Molina, and J. Pedersen, "Combating Web Spam with TrustRank ," in *Proceedings of the 30th International Conference on Very Large Databases*, pp. 576–587, Mar. 2004.

[4] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking: bringing order to the web.," 1999.

[5] R. Landa, D. Griffin, R. G. Clegg, E. Mykoniati, and M. Rio, "A sybilproof indirect reciprocity mechanism for peer-to-peer networks," in *INFOCOM 2009*, pp. 343–351, IEEE, 2009.

[6] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in p2p networks," in *Proceedings of the 12th International Conference on World Wide Web*, pp. 640–651, ACM, 2003.

[7] D. N. Tran, B. Min, J. Li, and L. Subramanian, "Sybil-Resilient Online Content Voting.," in *NSDI*, pp. 15–28, 2009.

[8] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman, "Sybilguard: defending against sybil attacks via social networks," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4, pp. 267–278, 2006.

[9] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao, "Sybillimit: A near-optimal social network defense against sybil attacks," *Security and Privacy*, pp. 3–17, 2008.

[10] P. Resnick and R. Sami, "Sybilproof transitive trust protocols," in *Proceedings of the 10th ACM Conference on Electronic Commerce*, pp. 345–354, ACM, 2009.

[11] A. Cheng and E. Friedman, "Manipulability of PageRank under sybil strategies," in *First Workshop on the Economics of Networked Systems*, NetEcon, 2006.

[12] A. Cheng and E. Friedman, "Sybilproof reputation mechanisms," in *Proceedings of the 2005 ACM SIGCOMM Workshop on Economics of peer-to-peer systems*, pp. 128–132, ACM, 2005.

[13] A. Altman and M. Tennenholtz, "Ranking systems: The PageRank Axioms," in *Proceedings of the 6th ACM Conference on Electronic Commerce*, pp. 1–8, ACM, 2005.

[14] R. Andersen, C. Borgs, J. Chayes, U. Feige, A. Flaxman, A. Kalai, V. Mirrokni, and M. Tennenholtz, "Trust-based recommendation systems: an axiomatic approach," in *Proceedings of the 17th International Conference on World Wide Web*, pp. 199–208, ACM, 2008.

[15] A. Altman and M. Tennenholtz, "Axiomatic Foundations for Ranking Systems.," *J. Artif. Intell. Res.(JAIR)*, vol. 31, pp. 473–495, 2008.

[16] A. Altman and M. Tennenholtz, "An axiomatic approach to personalized ranking systems," *Journal of the ACM (JACM)*, vol. 57, no. 4, p. 26, 2010.

[17] A. Jøsang, R. Ismail, and C. Boyd, "A survey of trust and reputation systems for online service provision," *Decision Support Systems*, vol. 43, no. 2, pp. 618–644, 2007.

[18] E. Friedman, P. Resnick, and R. Sami, "Manipulation-resistant reputation systems," in *Algorithmic Game Theory*, pp. 677–697, Cambridge University Press, 2007.

[19] D. Aldous and J. Fill, "Hitting and Convergence Time," in *Reversible Markov Chains*, 2001.

[20] J. G. Kemeny and J. L. Snell, *Finite Markov Chains*, vol. 28. Springer New York, 1976.

[21] J. B. Orlin, "Max flows in O(nm) time, or better," in *Proceedings of the 45th annual ACM Symposium on Theory of Computing*, pp. 765–774, ACM, 2013.

[22] J. Tang, S. Seuken, and D. C. Parkes, "Hybrid transitive trust mechanisms," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, pp. 233–240, International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[23] J. Hopcroft and D. Sheldon, "Manipulation-resistant reputations using hitting time," in *Algorithms and Models for the Web-Graph*, pp. 68–81, Springer, 2007.

[24] M. Iosifescu, *Finite Markov Processes and Their Applications*. Chichester: Wiley, 1980.

[25] D. Fogaras, B. Rácz, K. Csalogány, and T. Sarlós, "Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments," *Internet Mathematics*, vol. 2, no. 3, pp. 333–358, 2005.

[26] K. Avrachenkov, N. Litvak, D. Nemirovsky, and N. Osipova, "Monte Carlo methods in PageRank computation: When one iteration is sufficient," *SIAM Journal on Numerical Analysis*, vol. 45, no. 2, pp. 890–904, 2007.

[27] B. Bahmani, A. Chowdhury, and A. Goel, "Fast incremental and personalized PageRank," *Proceedings of the VLDB Endowment*, vol. 4, no. 3, pp. 173–184, 2010.

[28] A. D. Sarma, A. R. Molla, G. Pandurangan, and E. Upfal, "Fast Distributed PageRank Computation," pp. 11–26, 2013.

[29] J. D. Noh and H. Rieger, "Random walks on complex networks," *Physical Review Letters*, vol. 92, no. 11, p. 118701, 2004.

[30] P. Sarkar and A. Moore, "A tractable approach to finding closest truncated-commute-time neighbors in large graphs," *arXiv preprint arXiv:1206.5259*, 2007.

[31] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Reviews of Modern Physics*, vol. 74, no. 1, p. 47, 2002.

[32] A. Clauset, C. R. Shalizi, and M. E. J. Newman, "Power-Law Distributions in Empirical Data," *SIAM Review*, vol. 51, pp. 661–703, Nov. 2009.