1-1-2013

The Power Of Locality In Network Algorithms

Michael Brautbar
University of Pennsylvania, mbrautbar@gmail.com

The Power Of Locality In Network Algorithms

Abstract
Over the last decade we have witnessed the rapid proliferation of large-scale complex networks, spanning many social, information and technological domains. While many of the tasks which users of such networks face are essentially global and involve the network as a whole, the size of these networks is huge and the information available to users is only local. In this dissertation we show that even when faced with stringent locality constraints, one can still effectively solve prominent algorithmic problems on such networks.

In the first part of the dissertation we present a natural algorithmic framework designed to model the behaviour of an external agent trying to solve a network optimization problem with limited access to the network data. Our study focuses on local information algorithms --- sequential algorithms where the network topology is initially unknown and is revealed only within a local neighborhood of vertices that have been irrevocably added to the output set. We address both network coverage problems as well as network search problems.

Our results include local information algorithms for coverage problems whose performance closely match the best possible even when information about network structure is unrestricted. We also demonstrate a sharp threshold on the level of visibility required: at a certain visibility level it is possible to design algorithms that nearly match the best approximation possible even with full access to the network structure, but with any less information it is impossible to achieve a reasonable approximation.

For preferential attachment networks, we obtain polylogarithmic approximations to the problem of finding the smallest subgraph that connects a subset of nodes and the problem of finding the highest-degree nodes. This is achieved by addressing a decade-old open question of BollobÃ¡s and Riordan on locally finding the root in a preferential attachment process.

In the second part of the dissertation we focus on designing highly time efficient local algorithms for central mining problems on complex networks that have been in the focus of the research community over a decade: finding a small set of influential nodes in the network, and fast ranking of nodes. Among our results is an essentially runtime-optimal local algorithm for the influence maximization problem in the standard independent cascades model of information diffusion and an essentially runtime-optimal local algorithm for the problem of returning all nodes with PageRank bigger than a given threshold.

Our work demonstrates that locality is powerful enough to allow efficient solutions to many central algorithmic problems on complex networks.

Degree Type
Dissertation

Degree Name
Doctor of Philosophy (PhD)

Graduate Group
Computer and Information Science

First Advisor
Michael Kearns

Second Advisor
Sanjeev Khanna

Keywords
Algorithms, Locality, Networks

Subject Categories
Computer Sciences

THE POWER OF LOCALITY IN NETWORK ALGORITHMS

Michael Brautbar

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfilment of the Requirements for the

Degree of Doctor of Philosophy

2013

ACKNOWLEDGMENT

I would like to deeply thank my advisor, Michael Kearns, for his valuable guidance and support throughout my Ph.D. studies. Thank you for introducing me into the fascinating field of Social and Economic Networks. I would like to deeply thank Jennifer Chayes and Christian Borgs for their valuable mentorship and support. What started for me as an internship with you had grown into a close and fruitful collaboration which I am so grateful for.

I would like to thank my dissertation committee, Jennifer Chayes, Ali Jadbabaie, Sanjeev Khanna and Aaron Roth for their insightful feedback and suggestions. I would like to thank my other collaborators, Shang-Hua Teng, Sanjeev Khanna, Brendan Lucier and Umar Syed. It was a true pleasure working with you and learning from you.

Last, but certainly not least, I would like to gratefully thank my wife Yael for her love and support in every step of the way.

ABSTRACT

THE POWER OF LOCALITY IN NETWORK ALGORITHMS

Michael Brautbar

Michael Kearns

Over the last decade we have witnessed the rapid proliferation of large-scale complex networks, spanning many social, information and technological domains. While many of the tasks which users of such networks face are essentially global and involve the network as a whole, the size of these networks is huge and the information available to users is only local. In this dissertation we show that even when faced with stringent locality constraints, one can still effectively solve prominent algorithmic problems on such networks.

In the first part of the dissertation we present a natural algorithmic framework designed to model the behaviour of an external agent trying to solve a network optimization problem with limited access to the network data. Our study focuses on local information algorithms — sequential algorithms where the network topology is initially unknown and is revealed only within a local neighborhood of vertices that have been irrevocably added to the output set. We address both network coverage problems as well as network search problems. Our results include local information algorithms for coverage problems whose performance closely match the best possible even when information about network structure is unrestricted. We also demonstrate a sharp threshold on the level of visibility required: at a certain visibility level it is possible to design algorithms that nearly match the best approximation possible even with full access to the network structure, but with any less information it is impossible to achieve a reasonable approximation.

For preferential attachment networks, we obtain polylogarithmic approximations to the problem of finding the smallest subgraph that connects a subset of nodes and the problem of finding the highest-degree nodes. This is achieved by addressing a decade-old open question of Bollobás and Riordan on locally finding the root in a preferential attachment process.

In the second part of the dissertation we focus on designing highly time efficient local algorithms for central mining problems on complex networks that have been in the focus of the research community over a decade: finding a small set of influential nodes in the network, and fast ranking of nodes. Among our results is an essentially runtime-optimal local algorithm for the influence maximization problem in the standard independent cascades model of information diffusion and an essentially runtime-optimal local algorithm for the problem of returning all nodes with PageRank bigger than a given threshold.

Our work demonstrates that locality is powerful enough to allow efficient solutions to many central algorithmic problems on complex networks.

“*When we try to pick out anything by itself, we find it hitched to everything else in the universe.*”
—John Muir, “My First Summer in the Sierra” (1911)

Contents

List of Illustrations

Chapter 1

Introduction

Over the last decade we have witnessed the rapid growth of large-scale complex networks, spanning many social, information and technological realms [40, 63, 98]. Examples include social networks (such as Facebook, LinkedIn), information linkage networks (such as the Web, Wikipedia), communication networks (such as the Internet, telephone networks), and many others.

One of the main characteristics of these modern networks is their huge size. Facebook recently announced they now have more than a billion active users [116]. As of early May 2013, the indexed World Wide Web is estimated to have more than fourteen billion pages [117]. Typically, tasks which users of such complex networks face are global and involve the network as a whole. Some tasks include network coverage, such as used for advertising on networks, others include centrality computation of nodes, such as PageRank computation. As the sizes of complex networks become bigger these tasks become harder to solve. An inherent difficulty in solving such tasks is the natural locality of connections where one only learns about the structure of the network in close vicinity to nodes previously accessed. In social networks like Facebook, an external user can learn of the local structure around a node only up to a small radius. On the Web, one can learn about the out-links (but not the in-links) of a page by examining it. Can one solve such problems with only local information? This is the research focus of this dissertation.

There are two flavors of problems one needs to address. In many settings, commonly in social networks and routing networks among others, there is a cost of connecting to a node but such a connection holds new information about the local ball around the new node in the network. The goal is then to minimize the number of connections initiated by an external user of the system in his endeavour to solve some global problem on the network. Coverage problems are a common example in this framework. Can one do so competitively without knowing the structure of the whole network in advance? This is the topic of the first part of the dissertation.

In other settings, we are mainly interested in runtime efficiency — the massive growth and volatility of information networks necessitate the need to mine them in the quickest time possible. Prominent examples include fast computation of PageRank for the purpose of ranking of Web pages by search engines [15, 100] and finding a small set of influential node in social networks for the purpose of viral marketing [31, 71]. In these problems, a typical instance of the network is represented in an incidence list form and one needs to pay for every piece of information gained, including the traversal of the nodes in the incidence list of a previously accessed node. The importance of being able to solve these problems quickly is paramount since methods that do not scale with network size are doomed to be intractable in practice. Over the past decade a surge of techniques have been created to mine such large data without scanning through its whole, colloquially referred to as "sublinear time algorithms" [106]. In the context of networks and the lack of full knowledge of the network, one is particularly interested in local methods. Such methods are based on local advancements from a node to its neighbors together with some basic mechanism that allows some access to other areas in the network. Can we design local methods, with provable guarantees, that are highly time efficient to solve these problems? The second part of the dissertation addresses and answers this question.

Before describing the dissertation contributions we proceed to discuss the network access mechanism which lies at the heart of our algorithms and results.

1.1 The Jump and Crawl Paradigm

What is a natural way to access very large networks? Crawling is a common feature as well as some type of search mechanism that permits the discovery of nodes from other areas in the network. In social networks, the Web, routing networks and many others, one has access to a neighbor of a previously acceded node which can then be itself accessed (crawled). Modern networks also provide some sort of global search mechanism that permits the discovery of new nodes that may be quite distant from all those previously known. Web search lets us enter text phrases and see relevant pages; in information networks one can use random indexing to get a uniform node; and in several social networks, like Facebook, one can type in a name and get a profile of someone with that name. One natural modelling decision is to allow random sampling, what we call a *Jump*, to access other parts of the network. Random sampling is also pivotal in several related paradigms — sublinear-time algorithms and property testing, local computations, and graph limits, discussed below. In fact the Jump and Crawl paradigm can be viewed as distilling the type of queries used in these models, for both goals of non-costly algorithmic exploration and efficient algorithmic mining of very large networks.

In our framework a node can choose to query the network with either a Jump or a Crawl operation at any time step. Some problems can be solved well by using only Crawls; for other problems, Jumps are necessary in order to get good solution quality and their usage brings to rich theoretical contributions. Throughout the dissertation we have taken a conservative approach: our algorithms use Jumps only where it is clear that without the use of Jumps local algorithms to the problem at hand would perform very poorly.

We are now ready to survey the main contributions in this dissertation.

1.2 Dissertation Structure and Results

The dissertation is divided into two parts. In the first part we deal with local information algorithms for search problems (Chapter 3) and local information algorithms for coverage problems (Chapter 4). In the second part we move to design highly runtime efficient local algorithms for networks, under the adjacency list representation of sparse networks, for the target set selection problem (Chapter 6) and for PageRank and personalized PageRank computations (Chapter 7).

We note that our dissertation can be thought of as divided into chapters based on the type of problems solved, but also as divided into chapters according to solution techniques. Indeed, each chapter deploy a novel, and very different solution technique for the set of problems it deals with. In Chapter 3 we develop a novel coupling of the traversal path of a greedy algorithm on preferential attachment graphs to a super-critical branching process. In Chapter 4 we use an amortized analysis technique to show that several greedy coverage algorithms yield good approximations. In Chapter 6 we develop a novel technique to summarize large-scale diffusion information in a concise hypergraph representation. And in Chapter 7, we develop a novel PageRank computation method that is based on a multi-scale node sampling that repeatedly invokes a novel random-walk based personalized PageRank approximation.

We proceed with a concise exposition of our results, followed by a survey of related prior work.

1.2.1 Local Information Algorithms

In the first part of the dissertation, we study the power of *local information algorithms* for optimization problems on networks. We focus on sequential algorithms where the network topology is initially unknown and is revealed only within a local neighborhood of vertices that have been irrevocably added to the output set. This framework models the behavior of an external agent that does not have direct access to the network data, such as a user interacting with an online social network.

Local Information Algorithms for Network Search Problems

In Chapter 3, we focus on the ability of local algorithms to solve central search problems — searching for a connecting path between two source nodes, and searching for a node with the highest degree. We show that in general, such problems cannot be solved efficiently with local information. We then focus on a natural model of social networks — the "preferential attachment" process.

When the underlying graph is a preferential attachment network, we show that one can find the root (i.e. initial node) in a polylogarithmic number of steps, using a greedy, local algorithm that repeatedly queries the visible node of maximum degree. This addresses a decade-old open question of Bollobás and Riordan. Our solution technique is based on a novel coupling technique where we couple the growth of the set of nodes accessed by the local algorithm with a super-critical branching process. This result is motivated by its implications: we obtain polylogarithmic approximations to the problems of finding a connecting path between two source nodes and that of finding a highest-degree node.

Local Information Algorithms for Network Coverage Problems

In Chapter 4 we study several network coverage problems.

We first address the min dominating set coverage problem where the goal is to find a small set of nodes that dominate the whole network (i.e. all nodes are at a distance of at most one from that set). We then consider two natural variants of this problem. The first is inspired by prize-collecting problems where the goal is to get a reward for each node dominated and get penalized for those node left undominated. We call this problem the "neighbor-collecting" problem. In the second variant, the partial dominating set problem, the goal is only to cover a certain proportion of the population.

For both the minimum dominating set problem and the neighbor-collecting problem we show how to design local information algorithms for arbitrary networks whose performances approximately match the best possible even when information about network structure is unrestricted. We also demonstrate a sharp threshold on the level of visibility required: at a certain visibility level it is possible to design algorithms that nearly match the best approximation possible even with full access to the network structure, but with any less information it is impossible to achieve a reasonable approximation. These results demonstrate that a network provider's decision of how much structure to make visible to its users can have a significant effect on a user's ability to interact strategically with the network.

For the partial dominating set problem (where the goal is to cover a given constant fraction of the network with as few nodes as possible) we give an impossibility result which rules out the existence of good local algorithms for the problem. We then show how to devise good local algorithms for a relaxed setting, a bicriteria one, where the algorithm is compared against an adversary who must cover an additional small fraction of the network.

Our results are based on amortized analysis techniques, which allow us to show that the behaviours of several greedy coverage algorithms are not too far from optimal.

1.2.2 Local Algorithms for Stringent Time Constraints

In the second part of the dissertation, we focus on designing highly time efficient local algorithms for two prominent mining goals in large networks: influence maximization and node ranking.

Quasilinear and Sublinear Time Local Algorithms for Influence Maximization

In Chapter 6, we address the algorithmic problem of finding a set of k initial seed nodes in a network so that the expected size of the resulting cascade is maximized, under the standard *independent cascade* model of network diffusion. The promise of such an algorithm lies in applications to viral marketing. However, runtime is of critical importance in this endeavor due to the massive size and volatility of the relevant networks.

Our first result is a novel local algorithm for the influence maximization problem that obtains the near-optimal approximation factor of $(1 - \frac{1}{e} - \epsilon)$, for any $\epsilon > 0$, in time $O((m+n)\epsilon^{-3}\log n)$ where n and m are the number of vertices and edges in the network. The runtime of our algorithm is independent of the number of seeds k and improves upon the previously best-known algorithms which run in time $\Omega(mnk \cdot \text{POLY}(\epsilon^{-1}))$. Importantly, our algorithm is essentially runtime-optimal (up to a logarithmic factor) as we establish a lower bound of $\Omega(m+n)$ on the runtime required to obtain a constant approximation.

Our second result is a novel local algorithm that allows a provable tradeoff between solution quality and runtime. Our algorithm is the first to provide such a tradeoff for the influence maximization problem, giving an $\Theta(\frac{1}{\beta})$-approximation in time $O(n \cdot a(\mathcal{G}) \log^3(n)/\beta)$ for any $\beta > 1$, where $a(\mathcal{G})$ denotes the arboricity of the diffusion network $\mathcal{G}$. In particular, for graphs with bounded arboricity (as is the case for many models of network formation and empirically observed social networks) our algorithm is nearly runtime-optimal (up to logarithmic factors) for any fixed seed size k.

Our solution is based on a novel preprocessing scheme that generates a sparse hypergraph representation of the underlying network via sampling. We show that this representation makes it possible to efficiently estimate marginal influence in the original diffusion process with very few samples, and that the quality of this estimation degrades gracefully with reduced preprocessing time.

Sublinear Time Local Algorithms for PageRank Computations

Finally, in Chapter 7 we study the fundamental algorithmic problem of outputting all nodes in a network whose PageRank is more than a given threshold value Δ, and no node with PageRank less than Δ/c, given a fixed $c > 1$. We call the problem SIGNIFICANTPAGERANKS.

We develop a nearly optimal, local algorithm for the problem with time complexity $\tilde{O}(n/\Delta)$ on networks with n nodes, where the tilde hides a polylogarithmic factor. We show that any algorithm for solving this problem must have runtime of $\Omega(n/\Delta)$, rendering our algorithm optimal up to logarithmic factors. Our algorithm has sublinear time complexity for applications including Web crawling and Web search that require efficient identification of nodes whose PageRanks are above a threshold $\Delta = n^{\delta}$, for some constant $0 < \delta < 1$.

Our algorithm comes with two main technical contributions. The first is a multi-scale sampling scheme for a basic matrix problem that could be of interest on its own. In the abstract matrix problem it is assumed that one can access an unknown matrix, with entries from $[0,1]$, by querying its rows, where the cost of a query and the accuracy of the answers depend on a precision parameter ϵ. At a cost propositional to $1/\epsilon$, the query will return a list of $O(1/\epsilon)$ entries and their indices that provide an ϵ-precision approximation of the row. Our task is to find a set that contains all columns whose sum is at least Δ, and omits any column whose sum is less than Δ/c. Our multi-scale sampling scheme solves this problem with cost $\tilde{O}(n/\Delta)$, while traditional sampling algorithms would take time $\Theta((n/\Delta)^2)$.

Our second main technical contribution is a new local algorithm for approximating personalized PageRank, which is more robust than the earlier ones developed in [4, 64] and is highly efficient particularly for networks with large in-degrees or out-degrees. Together with our multiscale sampling scheme we are able to optimally solve the SIGNIFICANT-PAGERANKS problem.

1.3 Related Work

In this section we survey related prior work and compare it with our work on local information algorithms and the design of highly time efficient Jump and Crawl algorithms.

1.3.1 Local Distributed Algorithms

Distributed computation aims to understand how computation can be carried out by a set of connected computers, each knowing only a small piece of the input and responsible to generate a small piece of the output. The set of computers is fixed and so is the connecting topology describing which computers can directly communicate with each other. We will particularly be interested in the synchronous model, where communication between neighboring computers is simultaneous and on each round a node can send messages to any of its neighbors [88]. It is not surprising that for most tasks the number of communication rounds is at least the diameter of the network, as information needs to spread between all nodes. However, some problems can be computed in a number of rounds which is independent of the network size. The Local Distributed Model aims to understand what problems can be computed in constant number of rounds.

The work on local distributed algorithms was initiated by Linial [86] who showed that at least $\Omega(\log^* n)$ rounds of communications are necessarily in order to compute a maximal independent set on a ring. Later, Naor and Stockmeyer presented the first positive result, showing that there are several problems of node labeling, such as weak 2-colorings of nodes, that can be computed locally in a constant number of rounds on interesting families of graphs [96]. Much work has focused on graphs with bounded degree Δ, where the number of communication rounds depends only on Δ; see [115] for a recent survey on local distributed algorithms. In recent years, a weaker notion of locality called pseudo-locality was suggested in order to address a larger class of problems. Pseudo-local algorithms allow for mid-range information to be available by permitting the number of rounds to depend polylogarithmically on the network size [78–80]. Pseudo-local distributed algorithms have been developed for several combinatorial optimization problems that do not admit local distributed algorithms with reasonable approximation guarantees. Among the problems addressed are the problems of maximum matching, minimum vertex cover and minimum dominating set [80].

Recently, Parnas and Ron [102] provided a neat reduction that allows one to transform any local distributed approximation algorithm (as well as pseudo-local approximation algorithm) for an optimization problem to a sublinear-time centralized algorithm with a weak approximation guarantee. The main idea behind the reduction is that any processor v in a local distributed algorithm gets to see only information from a fixed small radius r around it before making its final output decision. Thus v's output decision can be figured out by simulating the communication in the r-ball around v. Let $\epsilon > 0$. If D is a distributed algorithm that computes an α-multiplicative approximation to some optimization problem, then by sampling a fixed number of nodes $O(1/\epsilon^2)$ and simulating D's behavior on these nodes one can get an approximation value in the interval $[|OPT|, \alpha|OPT| + \epsilon n]$, where $|OPT|$ is the size of the optimal solution and n is the number of nodes. Thus, one can devise such an algorithm in time $O(\Delta^r/\epsilon^2)$.

We note that such a reduction can also be seen as generating a local information algorithm that has radius r of information. The reduction generates a weak approximation guarantee on the size of the optimal solution with an ϵn additive error. By taking $\epsilon = \frac{(\alpha-1)|OPT|}{n}$, this can be used to get an α-multiplicative approximation to the size of OPT, namely an approximation to $|OPT|$ that lies in the interval $[|OPT|, \alpha|OPT|]$, in time $\tilde{O}\left(\left(\frac{n}{(\alpha-1)|OPT|}\right)^2\right)$, which in general can be at least linear in the size of OPT. In contrast, the goal of local information algorithms is to generate a *set* that yields a good multiplicative approximation, rather than just estimating the size of OPT. Importantly, the goal is to do so in number of operations that is very close to $|OPT|$. Moreover, our main interest would be in local information algorithms with radius r that is a small fixed number, e.g. 2, rather than a function of Δ; for many networks, such as social networks, the radius of local information is only a small fixed constant, even though the largest degree Δ scales with the network size.

It is interesting to compare the models of local distributed algorithms and local information algorithms with respect to a main source of computation. In local distributed algorithms it is the number of rounds of distributed computation while in local information algorithms it is the amount (radius) of local information. Consider the minimum dominating set problem. In this dissertation we devise a local information algorithm with radius less than 2 to achieve a $O(\Delta)$-multiplicative approximation, while it is known that any distributed algorithm must have at least $\Omega\left(\max\{\sqrt{\log n}, \log\Delta\}\right)$ rounds of computation in order to achieve a polylogarithmic approximation ratio [80].

On the other hand, consider the problem of finding a connecting path between two source nodes. If r is the distance between these nodes, even with local information of radius

$\Theta(r)$, any local information algorithm (possibly randomized) needs in general $\Omega(n/r^2)$ Jump and Crawl queries. However, one can device a distributed algorithm (essentially distributed-BFS), that finds such a path in $O(r)$ rounds (which is essentially optimal).

Thus the two models are not directly comparable and some problems that are "easy" in one model are not so in the other model.

1.3.2 Local Computations

Over the past decade there has been much interest in local computations, especially for the problems of graph partitioning [4, 113] and personalized PageRank computations [3, 4]. Local computations are based on procedures that run in time independent of the network size and return a solution near the vertex they are initialized with. Usually these procedures only use crawl operations and are then incorporated with random sampling to create the final algorithm. For example, for graph partitioning, Spielman and Teng [113] developed a procedure, *Nibble*, that given target conductance ϕ finds a local cluster around the node it is initialized with conductance close to ϕ, given that such cluster exists near that node of interest. Nibble is then used repeatedly to get good graph partitioning by repeatedly sampling nodes (from the degree distribution) in the network and running Nibble on them, resulting in an almost linear time algorithm for graph partitioning.

Our algorithms in the second part of the dissertation can be thought of as extending the local computation paradigm by allowing Crawls and Jumps to be used in the same phase in order to create an algorithm with a highly efficient runtime. However, our solution techniques are very different. Local clustering and local graph partitioning are based on deterministic simulations of random walks from chosen source nodes [4, 113]. Similarly, so does the work on approximating personalized PageRank [3, 4]. In contrast, our algorithm for target set selection is based a careful simulation of realized adoption trees in the network. These trees are realized from a diffusive process that spreads to all neighbors rather than a walk-based one than spreads to only one neighbor at a time, making the two types of processes very different. In the context of PageRank estimation, instead of simulating the

PageRank induced random walk, our algorithm is based on a multi-scale framework where we sample rows according to a precision parameter of the values appearing in that row. We estimate that row's values using a carefully designed Monte-Carlo random walk simulation rather than a deterministic one, that runs fast for the required precision.

1.3.3 Local Computation Algorithms (LCAs)

Local Computation Algorithms (LCAs) were introduced by Rubinfeld *et al.* [107]. Imagine computing some function defined over the input locations; for example, consider a boolean function describing for each node whether it is in a particular maximal matching of the graph. The goal of an LCA is to do so locally — answer for a given set of input locations *consistently* while only exploring the local neighborhoods of those locations. In other words, the answer for each input location must be consistent with some (possibly many) functions (a maximal matching in the example given). LCAs require that with high probability the answer is correct for any such input set.

Beck's algorithmic approach to the Lovász Local Lemma [10], as well as a technique of Onak and Nguyen [99] for simulating several classical greedy algorithms locally, have been successfully adapted to give LCAs for the problems of maximal independent set, maximal matching and hypergraph-coloring [2, 89, 107]. These LCAs have runtime that is polylogarithmic (in the total input length) times the number of input locations specified.

First, we note that one could use a similar approach to that of Parnas and Ron [102] and essentially use any LCA to an optimization problem to design a local information algorithm that simulates the size of an optimal solution. However, as for the case of local distributed algorithms, this would translate to a large additive error of the form ϵn and seems to suffer from the same drawbacks of Parans and Ron's approach where one needs a pure multiplicative approximation (and the approximation set itself).

Second, while the LCAs we surveyed above fall under the Jump and Crawl paradigm as they make use of Crawls only (which use no local information), their goal is very different. LCAs only give consistent local answers to a predefined set of input locations, and in runtime complexity that scales with the number of those input locations. In contrast, the goal of local information algorithms is to compute a valid solution to an optimization problem with a cost that scales only with the size of an optimal solution, even if the optimal solution has a very small size compared to the whole input. In particular, the radius of local information plays a central role in algorithm design. Thus in order to develop good local information algorithms in the Jump and Crawl setting one seems to need a different set of techniques than those that are successful for LCAs.

1.3.4 Property Testing and Sublinear-Time Algorithms

In property testing one is interested in deciding in sublinear time in the size of the input if the input has the given property P or is far from satisfying P. Over the past two decades there has been much interest in understanding the testability of graph properties, both for dense and sparse graphs [51, 52, 106]. It is not hard to see that the way the graph is represented matters to the measurement of its distance to having a property and the research has been divided into dealing with dense graphs (represented with adjacency matrices) and bounded-degree graphs (represented with adjacency lists). We will focus on testability of bounded-degree graphs as social, information and technological networks are typically sparse and as such can be compactly represented in the linked-list network representation.

The seminal work of Benjamini, Schramm and Shapira [14] shows that every minor-closed graph property is testable in time that depends only on the parameter measuring how far the input is to having the property and does not depend on the size of the graph. The authors show that one can always use a canonical tester that samples a number of nodes k, and explore radius r balls around each of the sampled nodes. As they beautifully show, both k and r can depend only on the approximation parameter ϵ and nothing else. (a bounded-degree graph of maximum degree d is ϵ-far to having P if one needs to change at least ϵdn edges to make the graph have the property P). As such, the Benjamini, Schramm and Shapira tester can be thought of as a Jump and Crawl one.

Except for testing, there has been some work on constructing sublinear time algorithms for estimating graph properties under the adjacency-list representation of graphs. Prominent examples include estimating the number of connected components in a graph and estimating the weight of a minimum spanning tree in a weighted sparse graph [29]. Our algorithms for target set-selection and PageRank operates under a stringent version of the graph-access model used by property-testers and sublinear-time algorithms and as such can also be thought of as sublinear-time algorithms in that model. We note that the techniques we present in this dissertation to analyze these problems have not been proposed or used before in the literature of testing and sublinear-time algorithms.

1.3.5 Low-Rank Matrix Approximations

One approach to simplify computational costs is to use a simpler structure that is similar to the original one but on which performing computations is substantially less costly. This approach has been quite fruitful in the context of matrices; see for example [58, 68] for recent surveys. Given a matrix A and parameter k (smaller than the dimension of A), one typically looks for a matrix $\tilde{A}$ with dimension k such that $\tilde{A}$ is close to A in the Frobenius norm. For some applications finding an $\tilde{A}$ that is close to A under the ℓ_2 operator norm is also of high interest. This approach could turn relevant to achieve highly efficient algorithm on graphs, as one can represent the graph by its adjacency or Laplacian matrices. In particular, one can ask whether such approach can yield good approximations for the target set selection and PageRank approximation problems considered in this dissertation. We note that our approaches to solve these problems are orthogonal to low-rank matrix approximations. Our algorithms work well for any graph, even if that graph does not have good low rank approximation. In addition, we note that all methods for low-rank approximation take time at least linear in the number of rows and columns of the matrix to build good low-rank approximations [58, 68]. In contrast, our methods for target set selection and PageRank computations run in sublinear-time and are runtime optimal for a large set of the parameter space in the case of target set selection, and runtime optimal for PageRank computations. As such, low-rank matrix approximations do not seem to be a good avenue to tackle these problems.

1.3.6 Graph Sparsifications

Here the goal is to keep only a small number of the edges of the graph but to maintain some of the properties of the original graph. The seminal example is the work of Karger and Benczúr that designs a randomized sparsifier so the resulting graph keeps the value of all cuts approximately [12]. This idea was later extended to deal with good spectral approximations [112] as well as to approximately keeping flow values through the graph [41]. However, it is not clear how such methods can be used for creating low-cost local information algorithms or highly efficient local algorithms. Such methods seem inherently linear in the size of the network — one needs to first sparsify the whole graph and then perform the necessary computation in the less complex graph. In contrast, all our methods are highly local and essentially avoid the need to rely on such preprocessing of the graph.

1.3.7 Graph Limits

Over the past few years Lovász and colleagues have developed an elegant theory of graph limits. A book describing their many contributions and ongoing research on the topic has recently been published [87]. The idea is to use the limit of a sequence of graphs to learn about the properties of the sequence itself. Defining the limit for a sequence of bounded-degree graphs is more involved (and perhaps less natural) than for a sequence of dense graphs due to a lack of a known pseudo-regular partition (namely, a Szemerédi partition) of such graphs. The idea of *Graphing* was introduced to represent such a limit and can be thought of as a measure on an infinite graph that is measure persevering: counting edges from A to B from A under this measure gives the same outcome as counting the edges from B to A from B, for any two measurable subsets A and B.

At the heart of the limiting behaviour of a sequence of bounded-degree graphs lies a local view developed by Benjamini and Schramm [13]: sample a node uniformly at random and look at the rooted r-ball around this node. For a given rooted graph F, denote by

$\rho_{H,r}(F)$ the probability that this sampling method returns F on the graph H. We then say that the sequence of graphs (G_n) with $|V(G_n)| \to \infty$ is locally convergent to a graphing G if the r-local densities $\rho_{G_n,r}(F)$ converge to $\rho_{G,r}(F)$, for every r and finite rooted graph F.

The topic of relating graph limits of bounded degree graphs to the analysis of algorithms on graphs (such as propriety testing algorithms) is a recent area of research and little is currently known. Moreover, it is not clear whether the graph limit perspective would bring to the design of more efficient graph algorithms. The current effort is to translate the known results on algorithmics on graphs to the language of graph limits.

1.3.8 Streaming and Sketching Algorithms

The main goal of a streaming algorithm is to process elements as they come, and by using only a small space, to be able to still produce good approximations to quantities of interest [95]. A canonical example is the frequency moments problem where one wants to compute the first few moments of the empirical distribution of a data stream [1, 62]. Some work has been devoted to graph streams, where the edges forming a graph arrive one by one in a streaming fashion. Problems considered include deciding if the graph is connected and counting small network motifs (such as triangles) [91].

One very successful approach to solving streaming problems is of sketching: a homomorphism projecting the stream into a low-dimensional space. The hope is then that many operations can be performed, using limited space, on the space-efficient sketch rather than on the original stream. In particular, for some problems such as estimating the ℓ_2 frequency moment of a stream, one can deal with the concatenation of streams by simply adding their respective sketches [91, 95].

We would like to compare these data analysis paradigms to the Jump and Crawl one. The Jump and Crawl paradigm is quite different from streaming and our solution techniques benefit much from multiple runs on small portions of the input. In particular, there is no emphasize at all on small space and our methods are not space-efficient in general. Instead we focus on minimizing cost of friending nodes (in local information algorithms) or on minimizing runtime (in highly runtime-efficient local algorithms). Also, Jump and Crawl's main concern is with locality, and as such, methods like sketching that depend on the input as a whole and access all nodes in the network do not seem applicable to our purposes.

1.4 Bibliographical Notes

The work on local information algorithms for network search problems, presented in Chapter 3, is based on joint work with Christian Borgs, Jennifer Chayes, Sanjeev Khanna, and Brendan Lucier [20]. The local information algorithm for finding a high degree node discussed in Section 3.2.2 is based on joint work with Michael Kearns [23]. The work on local information algorithms for coverage problems in networks, presented in Chapter 4, is based on joint work with Christian Borgs, Jennifer Chayes, Sanjeev Khanna, and Brendan Lucier [20]. The almost-linear and sublinear-time local algorithms for influence maximization and their analysis, presented in Chapter 6, are based on joint work with Christian Borgs, Jennifer Chayes, and Brendan Lucier [21]. Finally, the work on sublinear-time local algorithms for PageRank and personalized PageRank computations, presented in Chapter 7, is based on joint work with Christian Borgs, Jennifer Chayes, and Shang-Hua Teng [22].

Part I

Local Information Algorithms

Chapter 2

Introduction

In the past decade there has been a surge of interest in the nature of complex networks that arise in social, information and technological contexts. In the computer science community, this attention has been directed largely towards algorithmic issues, such as the extent to which network structure can be leveraged into efficient methods for solving complex tasks. Common problems include finding influential individuals, detecting communities, constructing subgraphs with desirable connectivity properties, and so on.

The standard paradigm in these settings is that an algorithm has full access to the network graph structure. More recently there has been growing interest in *local* algorithms, in which decisions are based upon local rather than global network structure. This locality of computation has been motivated by applications to distributed algorithms [46, 96], improved runtime efficiency [43, 113], and property testing [59, 106]. In this work we consider a different motivation: in some circumstances, an optimization is performed by an external user who has inherently restricted visibility of the network topology.

For such a user, the graph structure is revealed incrementally within a local neighborhood of nodes for which a connection cost has been paid. The use of local algorithms in this setting is necessitated by constraints on network visibility, rather than being a means toward an end goal of efficiency or parallelizability.

We consider graph algorithms in a setting of restricted network visibility. We focus on optimization problems for which the goal is to return a subset of the nodes in the network; this includes coverage and search problems. An algorithm in our framework proceeds by incrementally and adaptively building an output set of nodes, corresponding to those vertices of the graph that have been queried (or connected to) so far. When the algorithm has queried a set S of nodes, the structure of the graph within a small radius of S is revealed, guiding future queries. The principle challenge in designing such an algorithm is that decisions must be based solely on local information, whereas the problem to be solved may depend on the global structure of the graph. In addition to these restrictions, we ask for algorithms that run in polynomial time.

Before delving into algorithm design, we would like to demonstrate the broad applicability of local information algorithms.

A leading example is of advertising in social networks. An advertiser may wish to maximize the exposure of users to his product by giving it to a set of users that together cover a large number of neighbors [109]. Except for being an important goal by itself, it is a natural alternative to influence maximization under complex diffusion models; for many such models it is often hard to analyse and track the diffusion effects, namely, who influenced who, when, dealing with missing data, etc. [73, 108].

More generally, an advertiser may want to find a small set such that all network nodes are exposed, namely, a small set of nodes that dominate the network and can post information about the product in their online profile. This corresponds to the minimum dominating set problem. Other natural goals would be to choose a small set of users such that the amount of exposure (namely, nodes at distance at most one from that set) is at least a fixed fraction of the network's nodes. This corresponds to the partial dominating set problem. Yet another, quite different goal for advertising, is given k to find a set of k nodes with the highest degree. This has been shown to be the set with the highest influence for the well-known voter model of information diffusion, under its long run behaviour [42].

Finding a high-degree node is also beneficial socially [45]. Consider an agent in a social network who wishes to find (and link to) a highly connected individual. For instance, this agent may be a newcomer to a community (such as an online gaming or niche-based community) wanting to interact with influential or popular individuals. Finding a high-degree node is a straightforward algorithmic problem without information constraints, but many online and real-world social networks reveal graph structure only within one or two hops from a user's existing connections.

More generally, one might want to be able to find a path to some other agent in the network, without knowing the whole network in advance.

The common feature in all these examples is that befriending a node is a costly operation — both socially and time-wise and in some networks (such as LinkedIn) also monetary-wise (as one can pay the system in order to get introduced to new nodes). Importantly, once a node is befriended, one gets to learn new information — the structure of the local neighborhood around that node up to a small radius. This is a standard feature in all online social networks.

We note that the applicability of the local information model is not confined to the social networks' domain. For example, finding a small set of nodes dominating large parts of the network is important for monitoring communication and power delivery networks [53, 114]. In such networks information about the small neighborhood around a node can be compactly store at a node but there is a cost of installing and maintaining monitoring software at the nodes.

In chapter 3 we focus on local information algorithms for two fundamental search problems: s-t connectivity and finding a high-degree node. In chapter 4, we focus on design of efficient algorithms for coverage problems. We will use very different techniques to tackle search problems than those we use to tackle coverage problems. While analysis of greedy-walks on preferential attachment networks would be our main technical contribution to solve search problems, in coverage problems an amortized-analysis technique would be rendered successful and would allow us to show that the solution produced locally is not too far from optimal.

We now turn to formally describe the model of local information algorithms.

2.1 The Model

We shall study graph optimization problems in which the goal is to return a minimal-cost set of vertices S satisfying a feasibility constraint. In most of the problems we consider, the cost of set S will simply be $|S|$. We will consider a class of algorithms that build S incrementally under local information constraints. We begin with a definition of distance to sets and a definition local neighborhoods.

Definition 1. *The distance of v from a set S of nodes in a graph G is the minimum, over all nodes $u \in S$, of the shortest path length from v to u in G.*

Definition 2 (Local Neighborhood)**.** *Given a set of nodes S in the graph G, the r-*closed *neighborhood around S is the induced subgraph of G containing all nodes at distance less than or equal to r from S, plus the degree of each node at distance r from S. The r-*open *neighborhood around S is the r-closed neighborhood around S, after the removal of all edges between nodes at distance exactly r from S.*

Definition 3 (**Local Information Algorithm**)**.** *Let G be an undirected graph unknown to the algorithm, where each vertex is assigned a unique identifier. For integer $r \geq 1$, a (possibly randomized) algorithm is an r-local algorithm if:*

1. *The algorithms gets as input the size of the network n and a set S, initialized to either $\emptyset$ or to a given set of seed nodes.*

2. *The algorithm proceeds sequentially, growing step-by-step, adding each grown node into S.*

3. *Given that the algorithm has queried a set S of nodes so far, it can only observe the r-open neighborhood around S.*

4. *On each step, the algorithm can add a node to S either by selecting a specified vertex from the r-open neighborhood around S (a* crawl*) or by selecting a vertex chosen uniformly at random from all graph nodes (a* jump*).*

5. *In its last step the algorithm returns S as its output.*

Similarly, for $r \geq 1$, we call an algorithm a r^+*-local algorithm* if its local information (i.e. in item 2) is made from the r-closed neighborhood around S.

We focus on computationally efficient (i.e. polytime) local algorithms. Our framework naturally applies to coverage and search problems, where the family of valid solutions is upward-closed.

More generally, it is suitable for measuring the complexity, using only local information, for finding a subset of nodes having a desirable property. In this case the size of S measures the number of queries made by the algorithm; we think of the graph structure revealed to the algorithm as having been paid for by the cost of S.

For our lower bound results, we will sometimes compare the performance of an r-local algorithm with that of a (possibly randomized) algorithm that is also limited to using Jump and Crawl queries, but may use full knowledge of the network topology to guide its query decisions. The purpose of such comparisons is to emphasize instances where it is the lack of information about the network structure, rather than the necessity of building the output in a local manner, that impedes an algorithm's ability to perform an optimization task.

The approximation achieved by an algorithm is the ratio between the number of Jump and Crawl queries made by the algorithm to find a feasible solution (e.g. connecting subgraph) to the minimum number of Jump and Crawl queries made by an algorithm with full knowledge of the graph.

Chapter 3

Local Information Algorithms for Network Search Problems

3.1 Introduction

In this chapter we will develop and analyze local information algorithms for two prominent network search problems: s-t connectivity and finding high-degree. The goal in the s-t connectivity problem is to locally find a connecting subgraphs containing s and t and the goal in the high degree problem is to used a small number of queries to find a node of close to high degree.

For both these problems we derive strong lower bounds on the performance of local algorithms for general graphs. We therefore turn to the class of preferential attachment (PA) graphs, which model properties of many real-world social and technological networks. For PA networks, we prove that local information algorithms do surprisingly well at these two search problems.

Detailed Results and Techniques

The main focus of this chapter concerns local information algorithms for search problems in preferential attachment (PA) networks. Such networks are defined by a process by which nodes are added sequentially and form random connections to existing nodes, where the probability of connecting to a node is proportional to its degree.

We first consider the problem of finding the root (first) node in a PA network. A random walk would encounter the root in $\tilde{O}(\sqrt{n})$ steps (where n is the number of nodes in the network). The question of whether a better local information algorithm exists for this problem was posed by Bollobás and Riordan [18], who state that "an interesting question is whether a short path between two given vertices can be constructed quickly using only 'local' information" [18]. They conjecture that such short paths can be found locally in $\Theta(\log n)$ steps. We make the first progress towards this conjecture by showing that polylogarithmic time is sufficient: there is an algorithm that finds the root of a PA network in $O(\log^4(n))$ time, with high probability. We then show how to use this algorithm to obtain polylogarithmic approximations for finding the smallest subgraph that connects a subset of nodes (including shortest path) and finding the highest-degree node.

The local information algorithm we propose uses a natural greedy approach: at each step, it queries the visible node with highest degree. Demonstrating that such an algorithm reaches the root in $O(\log^4(n))$ steps requires a probabilistic analysis of the PA process. A natural intuition is that the greedy algorithm will find nodes of ever higher degrees over time. However, such progress is impeded by the presence of high-degree nodes with only low-degree neighbors. We show that these bottlenecks are infrequent enough that they do not significantly hamper the algorithm's progress. To this end, we derive a connection between node degree correlations and supercritical branching processes to prove that a path of high-degree vertices leading to the root is always available to the algorithm.

Related Work Over the last decade there has been a substantial body of work on understanding the power of sublinear-time approximations. In the context of graphs, the goal is to understand how well one can approximate graph properties in a sublinear number of queries. See [106] and [51] for recent surveys. In the context of social networks a recent work has suggested the Jump and Crawl model, where algorithms have no direct access to the network but can either sample a node uniformly (Jump) or access a neighbor of a previously discovered node (a Crawl) [23]. Local information algorithms can be thought of as a generalization the Jump and Crawl query framework that includes a dimension of information: a Crawl query can now return any node in the local neighborhood of nodes seen so far.

Motivated by distributed computation, a notion of local computation was formalized by [107] and further developed in [2]. The goal of a local computation algorithm is to compute only certain specified bits of a global solution. In contrast, our notion of locality is motivated by informational rather than computational constraints imposed upon a sequential algorithm. As a result, the informational dimension of visibility tends not to play a role in the analysis of local computation algorithms. In contrast, the main issue in designing a local information algorithm is to find a good tradeoff between the amount of local information the algorithm is allowed to see to the number of queries it needs to make in order to solve a network optimization problem.

Local algorithms motivated by efficient computation, rather than informational constraints, were explored by [4, 113]. These works explore local approximation of graph partitions to efficiently find a global solution. In particular, they explore the ability to find a cluster containing a given vertex by querying only close-by nodes.

Preferential attachment (PA) networks were suggested by [9] as a model for large social networks. There has been much work studying the properties of such networks, such as degree distribution [19] and diameter [18]; see [17] for a short survey. The problem of finding high degree nodes, using only uniform sampling and local neighbor queries, is explored by Brautbar and Kearns in [23]. The results of that work, as applied to our local infromatioh setting, are dicussed later in this chapter.

The low diameter of PA graphs can be used to implement distributed algorithms in which nodes repeatedly broadcast information to their neighbors [37, 46]. A recent work [37] showed that such algorithms can be used for fast rumor spreading. Our results on the ability to find short paths in such graphs differs in that our algorithms are sequential, with a small number of queries, rather than applying broadcast techniques.

The ability to quickly find short paths in social networks has been the focus of much study, especially in the context of small-world graphs [47, 75]. It is known that local routing using short paths is possible in such models, given some awareness of global network structure (such as coordinates in an underlying grid).

In contrast, our shortest-path algorithm for PA graphs does not require an individual know the graph structure beyond the degrees of his neighbors. However, our result requires that routing can be done from both endpoints; in other words, both nodes are trying to find each other.

3.2 Search problems in Arbitrary Networks

We start with a formal description of the search problems we analyze in this chapter.

The s-t Connectivity Problem The local information algorithm is given two source nodes s,t in a graph G with a connecting path between s and t. The goal of the algorithm is, given sources s and t, to find a small connected subgraph containing both s and t.

The High-Degree Node Problem The local information algorithm is given a precision parameter α. The goal of the algorithm is to find a node in a graph G with degree at least α time the maximum degree in G.

3.2.1 Searching for a Connecting Subgraph

In this section we will focus on the local information algorithms for the s-t connectivity problem in arbitrary graphs. We present a lower bound demonstrating that in general one cannot hope to get good approximations.

Theorem 3.1. *Let r, $n \geq 12r + 24$ be positive integers. For any r-local algorithm for the s-t connectivity problem with success probability bigger than $\frac{7}{8}$, the expected approximation over successful runs is $\Omega(\frac{n}{r^2})$ on connected graphs with n nodes.*

Corollary 3.2. *For any 1-local algorithm for the s-t connectivity problem with success probability bigger than $\frac{7}{8}$, the expected approximation over successful runs is $\Omega(n)$, the* *worst asymptotically possible, on connected graphs with n nodes.*

Proof of Theorem 3.1. The proof will invoke the application of Yao's Minimax Principle for the performance of Monte Carlo randomized algorithms on a family of inputs [119]. The lemma states that half the least expected cost of deterministic Monte Carlo algorithms with failure probability of $\epsilon \in [0,1]$ on a distribution over any family of inputs is a lower bound on the expected cost of the optimal randomized Monte Carlo algorithm with failure probability of $\frac{\epsilon}{2}$ over that family of inputs.

To use the lemma we will focus on Monte Carlo deterministic algorithms that have failure probability smaller than $\frac{1}{4}$ on the uniform distribution over a carefully chosen set of inputs. We proceed with the details. Each input is a graph on n nodes constructed as follows: we have two distinct nodes s and t. We define a broken path as path on $2r+4$ nodes where the 'middle' edge has been removed. The graph will be made from $\frac{n-2-(2r+4)}{2r+4}$ distinct broken paths from s to t together with one distinct connected path, connecting s with t. The identity of the connected path would be chosen uniformly at random from the set of $\ell = \frac{n-2}{2r+4}$ paths. In total the family of inputs contains $\frac{n-2}{2r+4}$ members.

See Figure 3.1 for an illustration.

As the algorithm is r-local, being at s or t it cannot see the middle node on the broken path so it cannot decide if a path is broken before traversing at least one node in it. A compelling property therefore holds: if the algorithm has not found the connected path after i queries then the algorithm learns nothing about the identity of the broken path except that it is not one of the paths it traversed so far. As the connected path is chosen uniformly at random from all paths, the probability that after $\frac{1}{6}\frac{n-2}{2r+4} = \ell/6$ queries the connected path is not found is

$$\left(1-\frac{1}{\ell}\right)\left(1-\frac{1}{(\ell-1)}\right)\cdots\left(1-\frac{1}{5\ell/6+1}\right) \geq \left(1-\frac{1}{5\ell/6+1}\right)^{\ell/6} \geq$$

$$\exp\left(-2\frac{\ell}{6}\frac{1}{5\ell/6+1}\right) \geq \exp\left(\frac{-2\ell}{5\ell+6}\right) \geq$$

$$\geq \exp(-1/3) \geq 0.71,$$

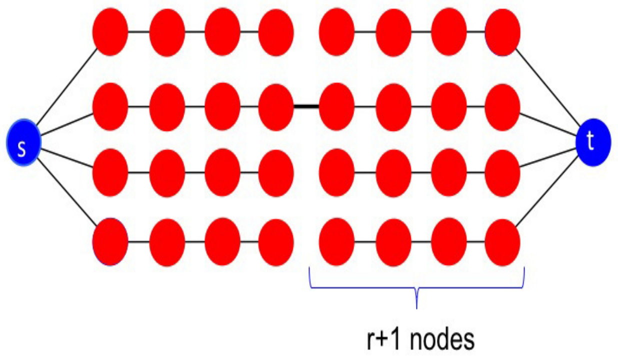

Figure 3.1: An example illustrating the “broken path” lower bound construction for the s-t connectivity problem.

where we used the fact that $1-x \geq \exp(-2x)$ for $0 \leq x \leq 1/3$ (here $x = \frac{1}{5\ell/6+1} < 1/3$ by assumption), and that $\ell \geq 6$.

By assumption, the algorithm is successful with probability at least $\frac{3}{4}$. Thus with constant probability at least $0.75 - 0.29 = 0.46$, the cost of finding a path from s to t must be $\Omega(\frac{n}{r})$, so the expected cost is at least $\Omega(\frac{n}{r})$. Using Yao’s principle applied to Monte Carlo algorithms, the cost of any randomized algorithm with success probability of at least $7/8$ would be at least $\Omega(\frac{n}{r})$ on one of the inputs. However, on any of the input graphs, an algorithm with full knowledge of the graph can query all the nodes in the unique path connecting s and t in $2r+4$ queries. The approximation ratio of the r-local algorithm would therefore be worse than $\Omega(\frac{n}{r^2})$. □

3.2.2 Searching for a High-Degree Node

In this section we will focus on the local information algorithms for the High-Degree Node Problem in arbitrary graphs. We begin with a lower bound demonstrating that in general one cannot hope to get good approximations.

Theorem 3.3. *Let r, $n \geq r$ and $4 \leq d \leq n/10$ be positive integers. For any r-local algorithm for finding a node of degree at least four with success probability at least $\frac{7}{8}$, the expected approximation over successful runs is $\Omega(\frac{n}{dr^2})$ on connected graphs with n nodes and maximum degree d.*

The theorem provides us with the following stringent impossibility result.

Corollary 3.4. *For any r-local algorithm for finding a node of degree at least four in graphs on n nodes with success probability at least $\frac{7}{8}$, the expected approximation over successful runs is $\Omega(\sqrt{n})$ on connected graphs with maximum degree $\sqrt{n}$.*

Proof of Theorem 3.3. We first focus our attention on proven the claim for 1-local algorithms on connected graphs. The proof will follow similar lines to that of theorem 3.1. As done previously, we shall invoke Yao's MinImax Principle on the performance of Monte Carlo randomized algorithms [119].

For that we focus on analyzing the performance of deterministic Monte Carlo algorithms with success probability $3/4$ on a uniformly at random chosen input from a carefully chosen family of inputs. We proceed with the details. Each input is a graph on n nodes constructed as follows: each input is a graph made from a complete binary tree on $n-d$ nodes called T, labeled with a random permutation of the numbers $1, 2, \ldots, n-d$. In addition one leaf node s would be connected to d new nodes, forming a star subgraph on these nodes, where s is its center. We denote this star subgraph on node s and its new neighbors by H. The nodes of H (except s) will be labeled with a random permutation of the numbers $n-d+1, n-d+2, \ldots, n$.

Note that node only nodes in H (and in fact only node s) have degree bigger than three and so any algorithm that want to achieve a good approximation to the problems must at least access a node in H.

See Figure 3.2 for an illustration.

Such an algorithm knows only the degrees of the nodes it already traversed. The input comes with a compelling property: if the algorithm has not found a node in the subgraph

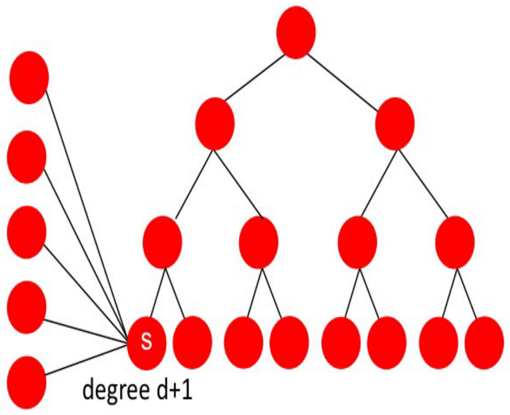

Figure 3.2: An example illustrating the “augmented leaf” lower bound construction for the high-degree problem.

H after i queries then it learned nothing about the identity of s except that it is one of the leaf nodes not queried so far[1]. Without loss of generality we can assume that the algorithm makes all its Jumps before its Crawls as the result of a Jump is independent of the outcome of any Jump and Crawl queries taken so far.

We now focus on the probability of success of the algorithm. As noted before, the algorithm is successful if it returns a node in H. We will now show that the probability the algorithm accesses, after making $\frac{n}{6d}$ queries, any node in H is strictly smaller than $3/4$, proving the result.

The probability of hitting a node in H is the probability of hitting a node in H given that no Jump has hit a node in H plus the probability that a Jump query had hit a node in H.

With taking at most $\frac{n}{6d}$ Crawls, the probability of hitting H is tantamount to either

[1]the deterministic algorithm “knows” the distribution over inputs, i.e. that s is a leaf connected to a star subgraph

starting from H or starting randomly in T and hitting the node s using Crawls (as H is connected to T only through node s). The probability of starting from H is $d/n \leq 1/10$. If we denote the number of leafs in T by ℓ, then the probability of not hitting the node s is at least

$$\left(1-\frac{1}{\ell}\right)\left(1-\frac{1}{(\ell-1)}\right)\dots\left(1-\frac{1}{\ell-n/6d+1}\right) \geq \left(1-\frac{1}{n/6d}\right)^{n/6d} \geq$$

$$\exp\left(-2\cdot\frac{6d}{6d}\right) \geq 0.86,$$

where we used the fact that $1-x \geq \exp(-2x)$ for $0 \leq x \leq 1/3$. So in this case, the total probability of hitting s is at most $0.1+0.14=0.24$. The first inequality follows from the simple fact that the number of leaves in a complete binary tree on $n-d$ nodes is $\ell = \frac{n-d+1}{2}$ and that $d < n/3$.

Next, with taking at most $\frac{n}{6d}$ Jumps, the probability of not hitting a node in H is

$$\left(1-\frac{(d+1)}{n}\right)^{\frac{n}{6d}} \geq \exp\left(-2\frac{d+1}{n}\cdot\frac{n}{6d}\right) \geq$$

$$\exp\left(-\frac{(d+1)}{3d}\right) \geq \exp(-2/5) \geq 0.67\,,$$

where we used the simple fact $1-x \geq \exp(-2x)$ for $0 \leq x \leq 1/3$ and that $\frac{d+1}{d} \leq \frac{6}{5}$.

Thus the total success probability of the algorithm is at most $0.24+0.33=0.57$.

Thus, for any algorithm that is successful with fixed probability of at least $1-\frac{1}{4}$, with constant probability of at least $0.75-0.57=0.18$ the query cost is $\Omega(n/d)$ and so is the expected cost over the inputs.

By Yao's principle the expected cost of any randomized algorithm with success probability at least $7/8$ would be at least $\Omega(n/d)$ on at least one of the inputs. However, an algorithm with full knowledge of the graph can find node s with at most $\Theta(\log(n))$ queries on any of the inputs by reaching the root of T and then taking the right path to connect to reach the leaf s. We conclude that the expected approximation of any 1-local algorithm on at least one of the inputs would be $\Omega(\frac{n}{d\log(n)})$.

To generalize the bound to hold for r-local algorithms we replace each edge (u,v) in the complete binary tree subgraph of the construction above by a distinct path from u to v

of length r. The total number of nodes becomes $C = n + (n-1)(r-2)$. The number of queries one needs to make in order to access a node in H with probability at least $3/4$ is essentially given by the preceding analysis and is $\Omega(\frac{n}{d}) = \Omega(\frac{C}{dr})$.

As an algorithm with full information of the graph can find node s in at most $\log(C)r$ time, the result follows.

□

We next focus on 1-local algorithms and present an almost matching upper bound for the problem for such algorithms. The designed algorithm first guesses the value of the maximum degree d^* using a simple doubling trick. It then takes about $\frac{n}{d^*}$ Jumps and return a node if it has degree at least d^*. The pseudo-code of the algorithm is given on the next page.

Theorem 3.5. *Algorithm FindAlmostHighestDegreeNode makes $O(\frac{n}{d^*}\log^2 n)$ Jump and Crawl queries, where d^* is the maximum degree in the network, and returns with probability $1-1/n$ a node with degree at least half of d^*.*

Proof. Note that $n \leq k < 2n$. The outer loop of the algorithm (line 2) costs at most $\log(2n)$ and the inner loop (line 3) costs at most $\frac{n}{d^*}\log n$. There are no other query calls. The total cost is therefore bounded by $O(n \log n^2)$.

Let d be a guess of d^* that differ from it by two. We focus on the iteration when such a d is picked in line 2 of the algorithm. We therefore assume that the algorithm has not terminated yet. The maximum degree vertex v has d^* neighbors. Therefore, the probability of hitting such a neighbor by making a Jump query is $\frac{d^*}{n}$. In line 3 the algorithm proceeds by making $\frac{n}{d^*}\log n$ Jump queries and would therefore hit a neighbor of v with probability at least $1-(1-\frac{d^*}{n})^{\frac{n}{d^*}\log n} \geq 1 - \exp\left(\frac{d^*}{n}\cdot\frac{n}{d^*}\log n\right) \geq 1-\frac{1}{n}$ and terminate successfully. Here we used the simple fact that $1-x \leq \exp(-x)$.

Last we note that if in a previous iteration to the one d was picked we found a node of degree at least $2d$ then it must be a highest degree node since $d^* \leq 2d$, and the algorithm would successfully terminate. □

Algorithm 1 FindAlmostHighestDegreeNode

```
1: Set $k$ to be the smallest power of 2 bigger or equal to $n$.
2: for $d = k, k/2, k/4, \ldots, 1$ do
3:     for $\frac{n}{d} \log n$ times do
4:         Make a Jump query. Let $v$ be the vertex found.
5:         if $v$ has a neighbor $u$ of degree at least $d$ then
6:             Terminate and return $u$.
7:         end if
8:     end for
9: end for
10: Return Fail.
```

3.3 Preferential Attachment Networks

Given the stringent impossibility results for solving the s-t connectivity problem and solving the high-degree problem on arbitrary graphs we focus on the solvability of these problems in networks generated according to a standard model for social and information networks — the preferential attachment (PA) process. We will show that in contrast to the impossibility results proven on general networks, we can achieve good approximations to search problems in preferential attachment networks by using a greedy procedure that aims to find the oldest node in the network (i.e. “the root”). The main focus of this chapter would be to analyze the performance of this greedy procedure in preferential attachment networks, addressing an open question of Bollobás and Riordan [18].

The preferential attachment (PA) process was conceived by Barabási and Albert [9]. Informally, the process is defined sequentially with nodes added one by one. When a node is added it sends m links backward to previously created nodes, connecting to a node with probability proportional to its current degree.

We will use the following, now standard, formal definition of the process, due to [18]. Given $m \geq 1$, we inductively define random graphs G_m^t, $1 \leq t \leq n$. The vertex set for G_m^t

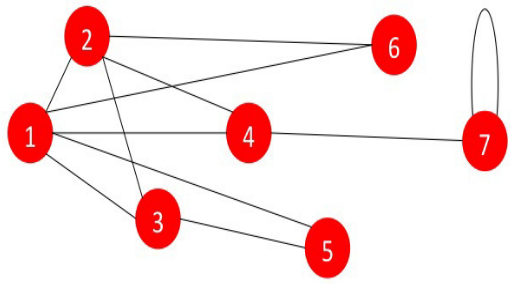

Figure 3.3: An illustration of the Preferential Attachment Process.

is $[t]$ where each node is identified by its creation time (index). G_m^1 is the graph with node 1 and m self-loops. Given $G_m^{(t-1)}$, form G_m^t by adding node t and then forming m edges from t to nodes in $[t]$, say $p_1(t), \ldots, p_m(t)$. The nodes $p_k(t)$ are referred to as the *parents* of t. The edges are formed sequentially. For each k, node s is chosen with probability $\deg(s)/z$ if $s < t$, or $(\deg(s)+1)/z$ if $s = t$, where z is a normalization factor.

Note that $\deg(s)$ denotes degree in G_m^{t-1}, counting previously-placed edges.See Figure 3.3 for an illustration.

We first present a 1-local approximation algorithm for the following natural problem on PA graphs: given an arbitrary node u, return a minimal connected subgraph containing nodes u and 1 (i.e. the root of G_m^n).

Our algorithm, TraverseToTheRoot, is listed as Algorithm 2. The algorithm grows a set S of nodes by starting with $S = \{u\}$ and then repeatedly adding the node in $N(S) \backslash S$ with highest degree. We will show that, with high probability, this algorithm traverses the root node within $O(\log^4(n))$ steps.

Theorem 3.6. *With probability* $1 - o(1)$ *over the preferential attachment process on* n

Algorithm 2 TraverseToTheRoot

```
1: Initialize a list $L$ to contain an arbitrary node $\{u\}$ in the graph.
2: while $L$ does not contain node 1 do
3:     Add a node of maximum degree in $N(L)\backslash L$ to $L$.
4: end while
5: return $L$.
```

nodes, TraverseToTheRoot returns a set of size $O(\log^4(n))$.

Remark: For convenience, we have defined TraverseToTheRoot assuming that the algorithm can determine when it has successfully traversed the root. This is not necessary in general; our algorithm will instead have the guarantee that, after $O(\log^4(n))$ steps, it has traversed node 1 with high probability.

Before proving Theorem 3.6, we discuss its algorithmic implications to the search problem we doscussed before.

3.3.1 Applications of TraverseToTheRoot to Search problems

We now describe how to use TraverseToTheRoot to implement local algorithms fr s-t connectivity and the high-degree problem. For ease of readability, the proofs of all auxiliary lemmas will be only presented at the end of the chapter.

s-t connectivity. As we now show, one may use TraverseToTheRoot to obtain a polylogarithmic approximation to this problem.

Corollary 3.7. *Let G be a PA graph on n nodes. Then, with probability* $1 - o(1)$ *over the PA process, Algorithm 3 (listed above), a* 1*-local algorithm, returns a connected subgraph of size* $O(\log^4(n))$ *containing vertices s and t. Furthermore, for any fixed k, with probability* $1 - o(1)$ *over the PA process, a subset of k nodes can be connected by a local algorithm in* $O(k\log^4(n))$ *steps, using a subset of size* $O(k\log^4(n))$.

Proof. Theorem 3.6 implies that, algorithm TravesetToTheRoot(G, s) returns a path from s to node 1 in time $O(\log^4(n))$, with probability $1 - o(1)$.

Algorithm 3 s-t-Connect

```
1: $P_1 \leftarrow$ TraverseToTheRoot($G, s$)
2: $P_2 \leftarrow$ TraverseToTheRoot($G, t$)
3: Return $P_1 \cup P_2$
```

Similarly, with probability $1 - o(1)$, TraverseToTheRoot(G, t) returns a connected path from s to node 1 in time $O(\log^4(n))$. Concatenating the two paths at node 1 is a path of length $O(\log^4(n))$ from s to t. Given k terminal one can connect all of them to nodes 1. Theorem 3.6 implies that for each terminal, with probability $1 - o(1)$, algorithm TravesetToTheRoot(G, s) returns a path from s to node 1 in time $O(\log^4(n))$. For a fixed k, the claim then follows from the union bound.

□

Finding a high degree node. As we now show, the algorithm TraverseToTheRoot obtains a polylogarithmic approximation to this problem.

Corollary 3.8. *Let G be a preferential attachment graph on n nodes. Then,*

- *Algorithm TraverseToTheRoot will return, with probability $1 - o(1)$, a node of degree at least $\frac{1}{\log^{2.9}(n)}$ of the maximum degree in the graph, in time $O(\log^4(n))$.*
- *For any fixed k independent of n, by running algorithm TraverseToTheRoot for an additional k steps and returning the k nodes with the highest degree found in its entire run, with probability $1 - o(1)$ and in time $O(\log^4(n))$, this set would provide an $O(1/\log^{2.9}(n))$ approximation to the largest sum of k node-degrees, over all k nodes in the network.*

Proof of Corollary 3.8. TraveseToTheRoot ends when node 1 is found. From lemma 3.10 (presented later on), with probability $1 - o(1)$, nodes 1 has degree at least $\frac{m\sqrt{n}}{\log^{1.9}(n)}$. However, from [17] (Theorem 17 therein), with probability $1 - o(1)$, the maximum degree is less than $m\sqrt{n}\log(n)$. As TraveseToTheRoot runs with probability $1 - o(1)$ in $O(\log^4(n))$ steps, and stops when it founds node 1, the first part is proven.

We now prove the second part of the corollary. From Lemma 3.10 (presented later on), with probability $1 - o(1)$, all nodes $j \leq \log(n)$ have degree at least $\frac{m\sqrt{n}}{\log^{1.9}(n)}$. Also, with probability $1 - o(1)$, the network on the first $\log(n)$ nodes is connected (see for example corollary 5.15 in [37], used with $n := \log(n)$). Thus running TraveseToTheRoot for an additional k steps after node 1 is found, and returning the k largest nodes found in its entire run, would guarantee that each of the k nodes has degree at least $\frac{m\sqrt{n}}{\log^{1.9}(n)}$, with probability $1 - o(1)$.

□

3.3.2 Analysis of TraverseToTheRoot Algorithm

We now turn to develop, step by step, a proof to Theorem 3.6. Our proof will make use of an alternative specification of the preferential attachment process, which is now standard in the literature [18], [37]. We will now describe this model briefly. Sample mn pairs $(x_{i,j}, y_{i,j})$ independently and uniformly from $[0,1] \times [0,1]$ with $x_{i,j} < y_{i,j}$ for $i \in [n]$ and $j \in [m]$. We relabel the variables such that $y_{i,j}$ is increasing in lexicographic order of indices. We then set $W_0 = 0$ and $W_i = y_{i,m}$ for $i \in [n]$. We define $w_i = W_i - W_{i-1}$ for all $i \in [n]$. We then generate our random graph by connecting each node i to m nodes $p_1(i), \ldots, p_m(i)$, where each $p_k(i)$ is a node chosen randomly with $\mathbb{P}[p_k(i) = j] = w_j / W_i$ for all $j \leq i$. We refer to the nodes $p_k(i)$ as the *parents* of i.

Bollobás and Riordan showed that the above random graph process is equivalent[2] to the preferential attachment process. They also show the following useful properties of this alternative model. Set $s_0 = 160\log(n)(\log\log(n))^2$ and $s_1 = \frac{n}{2^{25}\log^2 n}$. Let $I_t = [2^t + 1, 2^{t+1}]$. Define constants $\beta = 1/4$ and $\zeta = 30$.

Lemma 3.9 (adapted from [18]). *Let $m \geq 2$ be fixed. Using the definitions above, each of the following events holds with probability* $1 - o(1)$*:*

[2]As has been observed elsewhere [37], this process differs slightly from the preferential attachment process in that it tends to generate more self-loops. However, it is easily verified that all proofs in this section continue to hold if the probability of self-loops is reduced.

- $E_1 = \{|W_i - \sqrt{\frac{i}{n}}| \leq \frac{1}{100}\sqrt{\frac{i}{n}} \textit{ for } s_0 \leq i \leq n\}$.
- $E_2 = \{I_t \textit{ contains at most } \beta|I_t| \textit{ nodes } i \textit{ with } w_i < \frac{1}{\zeta\sqrt{in}} \textit{ for } \log(s_0) \leq t \leq \log(s_1)\}$.
- $E_3 = \{w_1 \geq \frac{4}{\log n\sqrt{n}}\}$.
- $E_4 = \{w_i \geq \frac{1}{\log^{1.9}(n)\sqrt{n}} \textit{ for all } i \leq s_0\}$.
- $E_5 = \{w_i \leq \frac{\log(n)}{\sqrt{in}} \textit{ for } s_0 \leq i \leq n\}$

Note that we modified these events slightly (from [18]) for our purposes: event E_2 uses different constants β and ζ, and in event E_4 we provide a bound on w_i for all $i \leq s_0$ rather than $i \leq n^{1/5}$. Finally, event E_5 is a minor variation on the corresponding event from [18]. The proof of Lemma 3.9 follows closely that of Lemma 7 in [18] and can be found in section 3.4.

Given Lemma 3.9, we can think of the W_i's as arbitrary fixed values that satisfy events $E_1, \dots, E_5$, rather than as random variables. Lemma 3.9 implies that, if we can prove Theorem 3.6 for random graphs corresponding to all such sequences of W_i's, then it will also hold for preferential attachment graphs. We now turn to the proof of Theorem 3.6. Let us provide some intuition. We would like to show that TraverseToTheRoot queries nodes of progressively higher degrees over time. However, if we query a node i of degree d, there is no guarantee that subsequent nodes will have degree greater than d; the algorithm may encounter local maxima. Suppose, however, that there were a path from i to the root consisting entirely of nodes with degree at least d. In this case, the algorithm will only ever traverse nodes of degree at least d from that point onward. One might therefore hope that the algorithm finds nodes that lie on such "good" paths for ever higher values of d, representing progress toward the root.

Motivated by this intuition, we will study the probability that any given node i lies on a path to the root consisting of only high-degree nodes (i.e. not much less than the degree of i). We will argue that many nodes in the network lie on such paths. We prove this in two steps. First, we show that for any given node i and parent $p_k(i)$, $p_k(i)$ will have high degree relative to i with probability greater than $1/2$ (Lemma 3.11). Second, since each node i has at least two parents, we use the theory of supercritical branching processes to argue that, with constant probability for each node i, there exists a path to a node close to the root following links to such "good" parents (Lemma 3.12).

This approach is complicated by the fact that existence of such good paths is highly correlated between nodes; this makes it difficult to argue that such paths occur "often" in the network. To address this issue, we show that good paths are likely to exist even after a large set of nodes (Γ in our argument below) is adversarially removed from the network. We can then argue that each node is likely to have a good path independently of many others nodes, as we can remove all nodes from one good path before testing the presence of another.

We will now proceed with the details of the proof. The proofs of all technical lemmas appear at the end of this section. Set $s_0 = 160\log(n)(\log\log(n))^2$ and $s_1 = \frac{n}{2^{25}\log^2 n}$. We think of vertices in $[1, s_0]$ as close to the root, and vertices in $[s_1, n]$ as very far from the root. Let $I_t = [2^t + 1, 2^{t+1}]$ be a partition of $[n]$ into intervals. Define constants $\beta = 1/4$ and $\zeta = 30$. We now define what we mean by a typical node.

Definition 4 (Typical node)**.** *A node i is* typical *if either $w_i \geq \frac{1}{\zeta\sqrt{in}}$ or $i \leq s_0$.*

Note that event E_2 implies that each interval I_t, $\log(s_0) \leq t \leq \log(s_1)$ contains a large number of typical nodes.

The lemma below encapsulates concentration bounds on the degrees of nodes in the network as well as other useful properties of PA networks.

Lemma 3.10. *The following events hold with probability* $1 - o(1)$*:*

- $E_6 = \{\forall i \geq s_0 : \ \deg(i) \leq 6m\log(n)\sqrt{\frac{n}{i}}\}$.
- $E_7 = \{\forall s_0 \leq i \leq s_1 \text{ that is typical} : \ \deg(i) \geq \frac{m}{2\zeta}\sqrt{\frac{n}{i}}\}$.
- $E_8 = \{\forall i \leq s_0 : \ \deg(i) \geq \frac{m\sqrt{n}}{5\log^{1.9}(n)}\}$.
- $\forall i \geq s_0 : \ \mathbb{P}[i \text{ is connected to } 1] \geq \frac{3.9}{\log(n)\sqrt{i}}$.

- $\forall j \geq i \geq s_0, 1 \leq k \leq m :\ \mathbb{P}[p_k(j) \leq i] \geq \frac{0.9\sqrt{i}}{\sqrt{j}}$.

Our next lemma states that, for any set Γ that contains sufficiently few nodes from each interval I_t, and any given parent of a node i, with probability greater than $1/2$ the parent will be typical, not in Γ, and not in the same interval as i.

Definition 5 (Sparse set)**.** *A subset of nodes* $\Gamma \subseteq [n]$ *is* sparse *if* $|\Gamma \cap I_t| \leq |I_t| / \log\log(n)$ *for all* $\log s_0 \leq t \leq \log s_1$*. That is,* Γ *does not contain more than a* $1/\log\log n$ *fraction of the nodes in any interval* I_t *contained in* $[s_0, s_1]$.

Lemma 3.11. *Fix sparse set* Γ*. Then for each* $i \in [s_0, s_1]$ *and* $k \in [m]$*, the following are true with probability* $\geq 8/15$ *:* $p_k(i) \notin \Gamma$*,* $p_k(i) \leq i/2$*, and* $p_k(i)$ *is typical.*

We now claim that, for any given node i and sparse set Γ, there is likely a short path from i to vertex 1 consisting entirely of typical nodes that do not lie in Γ. Our argument is via a coupling with a supercritical branching process. Consider growing a subtree, starting at node i, by adding to the subtree any parent of i that satisfies the conditions of Lemma 3.11, and then recursively growing the tree in the same way from any parents that were added. Since each node has $m \geq 2$ parents, and each satisfies the conditions of Lemma 3.11 with probability $> 1/2$, this growth process is supercritical and should survive with constant probability (within the range of nodes $[s_0, s_1]$). We should therefore expect that, with constant probability, such a subtree would contain some node $j < s_0$.

To make this intuition precise we must define the subtree structure formally. Fix sparse set Γ and a node $i \in [s_0, s_1]$. Define $H_\Gamma(i)$ to be the union of a sequence of sets $H_0, H_1, \ldots$, as follows. First, $H_0 = \{i\}$. Then, for each $\ell \geq 1$, H_ℓ will be a subset of all the parents of the nodes in $H_{\ell-1}$. For each $j \in H_{\ell-1}$ and $k \in [m]$, we will add $p_k(j)$ to H_ℓ if and only if the following conditions hold:

1. $p_k(j)$ is typical, $p_k(j) \notin \Gamma$, and $p_k(j) \leq j/2$,

2. $p_k(j) \notin H_r$ for all $r \leq \ell$, and

3. For the interval I_t containing $p_k(j)$, $|I_t \cap (H_0 \cup \ldots \cup H_\ell)| < 10\log\log n$.

Item 1 contains the conditions of Lemma 3.11. Item 2 is that $p_k(j)$ has not already been added to the subtree; we add this condition so that the set of parents of any two nodes in the subtree are independent. Item 3 is that the subtree contains at most $10\log\log n$ nodes from each I_t. We will use this condition to argue that Γ remains sparse if we add all the elements of $H_\Gamma(i)$ to Γ.

Our next lemma states that any given node $i \in [s_0, s_1]$ has a short path to the root consisting of only typical nodes with probability at least $3/4$.

Lemma 3.12. *Fix any sparse set* Γ. *Then for each node* $i \in [s_0, s_1]$, *the probability that* $H_\Gamma(i)$ *contains a node* $j \le s_0$ *is at least* $1/5$.

Lemma 3.12 implies the following result, which we will use in our analysis of the algorithm TraverseToTheRoot. First a definition.

Definition 6 (Good path)**.** *A path* $(j_0, j_1, ..., j_k)$ *is* good *if* $j_k \le s_0$, *each* j_ℓ *is typical and, for each* $\ell > 0$, $j_\ell \le j_{\ell-1}/2$. *We say vertex* $i \in [s_0, s_1]$ has a good path *if there is a good path with* $j_0 = i$.

Lemma 3.13. *Choose any set* T *of at most* $16\log n$ *nodes from* $[s_0, s_1]$. *Then each* $i \in T$ *has a good path with probability at least* $1/5$, *independently for each* i.

We will apply Lemma 3.13 to the set of nodes queried by TraverseToTheRoot to argue that progress toward the root is made after every sequence of polylogarithmically many steps.

We can now complete the proof of Theorem 3.6, which we give below.

Proof of Theorem 3.6. Our analysis will consist of three steps, in which we consider three phases of the algorithm. The first phase consists of all steps up until the first time TraverseToTheRoot traverses a node $i < s_1$ with a good path. The second phase then lasts until the first time the algorithm queries a node $i < s_0$. Finally, the third phase ends when the algorithm traverses node 1. We will show that each of these phases lasts at most $O(\log^4(n))$ steps.

We note that we will make use of Lemma 3.13 in our analysis by way of considering whether certain nodes have good paths. We will check at most $16\log n$ nodes in this manner, and hence the conditions of Lemma 3.13 will be satisfied.

Analysis of phase 1 Phase 1 begins with the initial node u, and ends when the algorithm traverses a node $i < s_1$ with a good path. We divide phase 1 into a number of iterations. Iteration zero starts at node u. Define iteration t as the first time, after iteration $t-1$, that the algorithm queries a node $i \leq s_1$.

Each new node i considered in iteration t will have $i < s_1$ with probability at least $W_{s_1}/1 \geq \frac{1}{2^{13}\log n}$, regardless of the previous nodes traversed. By the multiplicative Chernoff bound (A.1), with probability of at least $1 - 1/n^2$, after at most $5\log^2(n)$ steps such a node $i < s_1$ would be found. By Lemma 3.13 we know that node i has a good path with probability at least $1/5$ independent of all nodes traversed so far.

By the multiplicative Chernoff bound (A.1), we conclude that after at most $10\log(n)$ iterations, and total time of $O(\log^3(n))$, the algorithm traverses a node that has both $i < s_1$ and a good path, with probability at least $1 - \frac{10\log(n)}{n^2} - \frac{1}{n}$.

We note that the number of invocations of Lemma 3.13 made during the analysis of this phase is at most $2\log n$ with high probability, and hence the cardinality restriction of Lemma 3.13 is satisfied.

Analysis of phase 2 Phase 2 begins once the algorithm has traversed some node $i < s_1$ with a good path, and ends when the algorithm traverses a node $j < s_0$. We split phase 2 into a number of epochs. For each $\log s_0 < t \leq \log s_1$, we define epoch t to consist of all steps of the algorithm during which some node $i \in I_t$ with a good path has been traversed, but no node in any I_ℓ for $\ell < t$ with a good path has been traversed. Define random variable Y_t to be the length of epoch t. Note phase 2 ends precisely when epoch $\log s_0$ ends. Further, the total number of steps in phase 2 is $\sum_{t=\log s_0}^{\log s_1} Y_t$.

Fix some $\log s_0 \leq t \leq \log s_1$ and consider Y_t. Suppose the algorithm is in epoch t, and let $i \in I_t$ be the node with a good path that has been traversed by the algorithm. Then, from the definition of a good path and event E_7, i has a parent $j \in I_\ell$ for some $\ell < t$ with

$deg(j) \geq \frac{m}{2\zeta}\sqrt{\frac{n}{i}}$. This node j is a valid choice to be traversed by the algorithm, so any node queried before j must have degree at least $\frac{m}{2\zeta}\sqrt{\frac{n}{i}}$. Moreover, traversing node j would end epoch t, so every step in epoch t traverses a node with degree at least $\frac{m}{2\zeta}\sqrt{\frac{n}{i}}$. By event E_6, any such node ℓ satisfies $\ell < zi\log^2(n)$ for constant $z = (4\zeta)^2$. But we now note that, for any node $\ell < zi\log^2(n)$ traversed by the algorithm, the probability that ℓ has a parent[3] $r < i/2$ is at least $\frac{W_{i/\log^2(n)}}{W_\ell} \geq \frac{1}{4\zeta\log^2 n}$. Any such node r has degree greater than any node in I_t, again by Lemma 3.10, so if a queried node had such a parent then the subsequent step must query a node of index at most 2^t. Moreover, Lemma 3.13 implies that this node of index at most 2^t has a good path with probability at least $1/5$. Thus each step of the algorithm results in the end of epoch t with probability at least $\frac{1}{20\zeta\log^2 n}$. We conclude that Y_t is stochastically dominated by a geometric random variable with mean $20\zeta\log^2 n$. Also, the number of invocations of Lemma 3.13 made during epoch t is dominated by a geometric random variable with mean 5.

We conclude that $\sum_{t=\log s_0}^{\log s_1} Y_t$ is dominated by the sum of at most $\log n$ geometric random variables, each with mean $20\zeta\log^2 n = 600\log^2 n$. Concentration bounds for geometric random variables (Lemma A.3) now imply that, with high probability, this sum is at most $2^{10}\log^3 n$. We conclude that phase 2 ends after at most $2^{10}\log^3 n$ steps with high probability. Similarly, the total number of invocations of Lemma 3.13 made during the analysis of this phase is at most $6\log n$ with high probability, again by Lemma A.3.

Analysis of phase 3 We turn to analyze the time it takes from the first time the algorithm encountered a node of $i \leq s_0$ until node 1 is found. We start by noting that the induced graph on the first s_0 nodes is connected with probability $1 - o(1)$ (see for example corollary 5.15 in [37], used with $n := s_0$). We note that by Lemma 3.10 every node $j \leq s_0$ has degree at least $d = \frac{m\sqrt{n}}{5\log^{1.9}(n)}$. As there is a path from i to node 1 where all nodes have degree at least d, the algorithm, as it follows the highest neighbor of its current set S, will reach node 1 before it had traversed any node of degree less than d. We can therefore assume that the

[3]Note that even if the algorithm queried node ℓ via its connection to one of its parents, it will still have at least one other parent that is independent of prior nodes queried by the algorithm since $m \geq 2$.

algorithm only traverses nodes of degree at least d.

By Lemma 3.10, each node $j > s_0$ has $deg(j) \leq 6m\log(j)\sqrt{\frac{n}{j}}$ with high probability, and therefore any node j with degree $\geq d$ must satisfy $j \leq (60\zeta)^2 \log^{5.8}(n)$. For any such node, E_1 implies that $W_j \leq \frac{11}{10}\frac{(60\zeta)\log^{2.9}(n)}{\sqrt{n}}$. Thus, for each such j, the probability that j is connected to the root is $w_1/W_j \geq \frac{1}{2^{11}\log^{3.9}(n)}$, by event E_3. Chernoff bounds (Lemma A.1) then imply that such an event will occur with high probability after at most $O(\log^4(n))$ steps. Thus, with high probability, phase 3 will end after at most $s_0 + O(\log^4(n)) = O(\log^4(n))$ steps. □

3.4 Deferred Proofs from the Analysis of TraveseToTheRoot

3.4.1 Proof of Lemma 3.9

We provide details for the proof of Lemma 3.9. The proof follows closely a similar lemma of Bollobás and Riordan, Lemma 7 in [18].

The proof that events E_1 and E_2 hold with high probability follows without change (as in Lemma 7 in [18]), except for a trivial modification of certain constants. The proof that events E_3 holds with high probability follows entirely without change.

We next show that event E_4 holds with high probability, by showing that $\Pr[E_4^c \cap E_1] = o(1)$. Suppose that $E_4^c \cap E_1$ holds and let $\delta = \frac{1}{\log^{1.9}(n)\sqrt{n}}$. As E_1 holds we have $W_{s_0} \leq \frac{11\log\log(n)\sqrt{\log(n)}}{10\sqrt{n}}$. As E_4 does not hold there exists some interval $[x, x+\delta]$ with $0 \leq x \leq \frac{11\log\log(n)\sqrt{\log(n)}}{10\sqrt{n}}$ that contains two of the W_i and hence two of the $y_{i,j}$. Each such interval is contained in some interval $J_t = [t\delta, (t+2)\delta]$ for $0 \leq t \leq \delta^{-1}\frac{11\log\log(n)\sqrt{\log(n)}}{10\sqrt{n}} < 2\log^{2.5}(n)$. The probability that some $y_{i,j}$ lands in such an interval is $(4t+4)\delta^2$, so the probability that at least two lie in J_t is at most $m^2n^2(4t+4)^2\delta^4/2 < 32m^2/\log^{2.6}(n)$.

Thus

$$\mathbb{P}(E_4^c \cap E_1) \leq \sum_{t=0}^{2\log^{2.5}(n)} 32m^2/\log^{2.6}(n) = o(1)$$

as required.

We will next show that the event E_5 holds with high probability. Recall that event E_5 is $\{w_i \leq \frac{\log(n)}{\sqrt{in}}$ for $s_0 \leq i \leq n\}$. We will show that $\mathbb{P}(E_5^c \cap E_1) = o(1)$, which will imply that $\mathbb{P}(E_5) = 1 - o(1)$ as required.

Suppose that $E_5^c \cap E_1$ holds. Then there is some $i \geq s_0$ is such that $w_i > \frac{\log(n)}{\sqrt{in}}$. Define $\delta = \frac{\log(n)}{\sqrt{in}}$; it must therefore be that the interval $(W_{i-1}, W_{i-1} + \delta]$ does not contain W_i, and hence contains at most $m-1$ of the $y_{i,j}$. Since E_1 holds, we must have $W_{s_0} \geq \frac{9}{10}\sqrt{\frac{s_0}{n}}$. We now define a partition of $[\frac{9}{10}\sqrt{\frac{s_0}{n}}, 1]$ into intervals $J_t = [x_t, x_{t+1})$ for $t \geq 0$, where we define $x_0 = \frac{9}{10}\sqrt{\frac{s_0}{n}}$ and $x_t = x_{t-1} + \frac{\log(n)}{x_{t-1}nm}$ for all $t \geq 1$, until $x_t \geq 1$. We note that there are at no more than mn intervals J_t in total. We also note that, since E_1 holds, each interval $(W_{i-1}, W_{i-1} + \delta]$ contains at least $m-1$ intervals J_t, each satisfying $x_t \geq W_{i-1}$, one of which must contain no $y_{i,j}$ since E_5 does not hold.

For a given t satisfying $x_t \geq W_{i-1}$, the number of $y_{i,j}$ in J_t has a $Bi(mn, p_t)$ distribution with

$$p_t = x_{t+1}^2 - x_t^2 \leq 2x_t \frac{\log(n)}{x_t nm} = \frac{2\log(n)}{nm}.$$

The probability that no $y_{i,j}$ lies in this interval is thus

$$(1-p_t)^{mn} \leq e^{-mnp_t} < e^{-2\log(n)} = o(n^{-1}).$$

Summing over the $O(n)$ values of t shows that $\Pr(E_5^c \cap E_1) = o(1)$, as required.

3.4.2 Proof of Lemma 3.10

We will first prove that the following events hold with probability $1 - o(1)$:

- $E_6 = \{\forall i \geq s_0 : \ \deg(i) \leq 6m\log(n)\sqrt{\frac{n}{i}}\}$.
- $E_7 = \{\forall s_0 \leq i \leq s_1$ that is typical : $\deg(i) \geq \frac{m}{2\zeta}\sqrt{\frac{n}{i}}\}$.

- $E_8 = \{\forall i \le s_0 : \ \deg(i) \ge \frac{m\sqrt{n}}{5\log^{1.9}(n)}\}$.

Note that event E_7 states that typical nodes have typical degree, motivating our choice of terminology.

We start by noting that $\deg(i) = \sum_{j=i+1}^{n} \sum_{k=1}^{m} Y_{k,j}$ where each of the $Y_{i,j}$s is an i.i.d Bernoulli random variable that gets the value of one with success probability of $\frac{w_i}{W_j}$. This follows from the fact the each new node j sends m edges backwards and the probability of each hitting node i is exactly $\frac{w_i}{W_j}$. From E_1 and E_5,

$$\mathbb{E}(\deg(i)) \le \sum_{j=i+1}^{n} \left(m \frac{\log(n)\frac{1}{\sqrt{in}}}{\frac{9}{10}\sqrt{\frac{j}{n}}} \right) = \sum_{j=i+1}^{n} \left(m \log(n) \frac{10}{9} \frac{1}{\sqrt{ij}} \right).$$

By estimating the sum with an integral we get

$$\mathbb{E}(\deg(i)) \le \frac{10m}{9} \log(n) \frac{\sqrt{n}}{\sqrt{i}}.$$

From the multiplicative Chernoff bound (A.1) we conclude that with probability bigger than $1 - 1/n^2$, $\deg(i) \le 3m\log(n)\frac{\sqrt{n}}{\sqrt{i}}$ for a given node i. By using the union bound, event E_6 then holds with probability $1 - 1/n$.

To prove E_7 holds with probability $1 - 1/n$, we first recall that, for a typical node i, $w_i \ge \frac{m}{\zeta\sqrt{in}}$. This implies, similarly to the first part of the proof, that

$$\mathbb{E}(\deg(i)) \ge \frac{10m}{11} \frac{\sqrt{n}}{\zeta\sqrt{i}}.$$

As $\text{Exp}(\deg(i)) \ge 16m\log(n)$ for any $s_0 \le i \le s_1$ (since $s_1 = \frac{n}{2^{25}\log^2 n}$), we can invoke the Chernoff bound (A.1) to get that E_7 holds with probability bigger than $1 - 1/n^2$ for a given node i. This follows by thinking of $\deg(i)$ as a sum of Bernoulli random variables $Y_{i,j}$, where $Y_{i,j}$ succeeds with probability $\frac{\frac{1}{\zeta\sqrt{in}}}{W_j}$. By using the union bound, event E_7 then holds with probability $1 - 1/n$.

The proof that E_8 holds with probability $1 - 1/n$ follows similarly to the proof for such a claim for E_7, by using the property that $w_i \ge \frac{1}{\log^{1.9}(n)\sqrt{n}}$.

To complete the proof of Lemma 3.10, we must show that

- $\forall i \geq s_0 :\ \mathbb{P}[i \text{ is connected to } 1] \geq \frac{3.9}{\log(n)\sqrt{i}}$, and
- $\forall j \geq i \geq s_0, 1 \leq k \leq m :\ \mathbb{P}[p_k(j) \leq i] \geq \frac{0.9\sqrt{i}}{\sqrt{j}}$.

The first item follows from events E_1 and E_3 of Lemma 3.9, plus the fact that $\mathbb{P}[p_k(i) = 1] = \frac{w_1}{W_i}$ for every i and k. The second item follows from event E_1 of Lemma 3.9, plus the fact that $\mathbb{P}[p_k(j) \leq i] = \frac{W_i}{W_j}$.

3.4.3 Proof of Lemma 3.11

We first recall the statement of the lemma. Fix any sparse set Γ. Then for each i, $s_0 \leq i \leq s_1$, and each $k \in [m]$, the following statements are all true with probability at least $8/15$: $p_k(i) \notin \Gamma$, $p_k(i) \leq i/2$, and $p_k(i)$ is typical.

Fix i and k. For each of the three statements in the lemma, we will bound the probability of that statement being false.

First, we will show that $\mathbb{P}[p_k(i) \text{ not typical }] < 1/15$. Note that, given that $p_k(i)$ falls within an interval I_t, this probability is bounded by the total weight of atypical nodes in I_t divided by the total weight of I_t. Since each atypical node j has weight at most $\frac{1}{10\sqrt{jn}}$ and $j > 2^t$ for all $j \in I_t$, E_4 implies that the total weight of atypical nodes in I_t is at most

$$\beta |I_t| \frac{1}{10\sqrt{2^t n}} = \beta \frac{\sqrt{2^t}}{10\sqrt{n}}.$$

Also, E_1 implies that the total weight of I_t is

$$W_{2^{t+1}} - W_{2^t} \leq \sqrt{\frac{2^t}{n}} \left(\frac{99}{100}\sqrt{2} - \frac{101}{100} \right).$$

Since these bounds hold for all t, we conclude that

$$\mathbb{P}[p_k(i) \text{ not typical }] < \frac{10\beta}{99\sqrt{2} - 101}$$

which will be at most $1/15$ for $\beta = 1/4$.

Next, we will show that $\mathbb{P}[p_k(i) > i/2] < \frac{1}{3}$. Event E_1 implies that

$$\mathbb{P}[p_k(i) > i/2] = 1 - W_{i/2}/W_i \leq 1 - \frac{99}{101\sqrt{2}} < \frac{1}{3}.$$

Finally, we will show that $\mathbb{P}[p_k(i) \in \Gamma] < 1/15$. Given that $p_k(i)$ falls within an interval I_t, this probability is bounded by the total weight of $I_t \cap \Gamma$ divided by the total weight of I_t. In this case, due to the assumed sparsity of Γ and E_5, the former quantity is at most $|I_t|\frac{1}{(\log\log n)\sqrt{2^t n}} \leq \sqrt{\frac{2^t}{n}}$. Also, as above, the total weight of I_t is at most $\sqrt{\frac{2^t}{n}}(\frac{99}{10}\sqrt{2} - \frac{101}{10})$. Since these bounds hold for all t, we conclude that $\mathbb{P}[p_k(i) \in \Gamma] < \frac{1}{99\sqrt{2}-101}$ which is at most $1/15$.

Taking the union bound over these three events, we have that the probability none of them occur is at least $8/15$ as required.

3.4.4 Proof of Lemma 3.12

Let us first recall the statement of the lemma. Fix any sparse set Γ. Then for each node $i \in [s_0, s_1]$, the probability that $H_\Gamma(i)$ contains a node $j \leq s_0$ is at least $1/5$.

Fix Γ and i, and write $H = H_\Gamma(i)$. Let $C = [s_0]$, the set of all nodes with index s_0 or less. We will show that the probability that $H \cap C = \emptyset$ is at most $4/5$.

Let ℓ be such that $i \in I_\ell$. We will say that H *saturates* a given interval I_t if $|H \cap I_t| = 10 \log\log n$. (Note that we must have $|H \cap I_t| \leq 10 \log\log n$, from the definition of H). Let us first consider the probability that $H \cap C = \emptyset$ and H does not saturate any intervals. Since H does not saturate any intervals, and since the set $H \cup \Gamma$ is itself a sparse set, then for each node $j \in H$ and $k \in [m]$ the parent $p_k(j)$ will be added to H precisely if the conditions of Lemma 3.11 hold, which occurs with probability at least $8/15$. We can therefore couple the growth of the subtree H within the range $[s_0, i]$ with the growth of a branching process in which each node spawns up to two children, each with probability at least $8/15$. In this coupling, the event $H \cap C = \emptyset$ implies the event that this branching process generates only finitely many nodes. Write p for the probability that the branching process generates infinitely many nodes. Then $p = \frac{8}{15}p + (1 - \frac{8}{15}p)\frac{8}{15}p$, from which we obtain $p = \frac{15}{64}$. We therefore have $\mathbb{P}[H \cap C = \emptyset] \leq 1 - p = \frac{49}{64}$ conditional on H not saturating any intervals. Next consider the probability that $H \cap C = \emptyset$ given that H does saturate some interval. In this case, there is some smallest t such that I_t is saturated by H.

Then, given that H saturates I_t but no interval $I_{t'}$ for $t' < t$, then we can again couple the growth of subtree H from interval I_t onward with $10\log\log n$ instances of the branching process described above, each one starting at a different node in $H \cap I_t$. In this case, the probability that $H \cap C = \emptyset$ is bounded by the probability that each of these $10\log\log n$ copies of the branching process all generate only finitely many children. This probability is at most $(49/64)^{10\log\log n} = o(\frac{1}{\log^2(n)})$. Thus, taking the union bound over all possibilities for the value of t (of which there are at most $\log n$), the probability that $H \cap C = \emptyset$ given that H saturates some interval is at most $o(\log n / \log^2(n)) = o(1)$.

Combining these two cases, we see that $\mathbb{P}[H \cap C = \emptyset] \leq 49/64 + o(1) < 4/5$.

3.4.5 Proof of Lemma 3.13

Write $T = \{t_1, \dots, t_k\}$. We will apply Lemma 3.12 to each node t_i in sequence. First, for node t_1, define $\Gamma_1 = \emptyset$. Lemma 3.12 with $\Gamma = \Gamma_1$ implies that $H_{\Gamma_1}(t_1)$ contains a node $j \leq s_0$ with probability at least 1/5.

For each subsequent node t_i, define $\Gamma_i = \Gamma_{i-1} \cup H_{\Gamma_{i-1}}(i-1)$. We claim that Γ_i is sparse. To see this, recall that each $H_\Gamma(t_{i-1})$ contains at most $10\log\log n$ nodes in each interval I_t, and Γ_i is the union of at most $16\log n$ such sets, therefore $|\Gamma_i \cap I_t| \leq 160\log(n)\log\log(n)$ for each t. Since $|I_t| \geq s_0 \geq 160\log(n)(\log\log(n))^2$, we have $|\Gamma_i \cap I_t| \leq |I_t| / \log\log(n)$ and hence Γ_i is sparse. Lemma 3.12 with $\Gamma = \Gamma_i$ then implies that $H_{\Gamma_i}(t_i)$ contains a node $j \leq s_0$ with probability at least 1/5. Moreover, this probability is independent of the events for nodes $t_1, \dots, t_{i-1}$, since $H_{\Gamma_i}(t_i)$ is constrained to not depend on nodes in Γ_i, which contains all nodes that influenced the outcome for $t_1, \dots, t_{i-1}$.

We conclude that, for each i, $H_{\Gamma_i}(t_i)$ contains a node $j \leq s_0$ with probability at least 1/5, independently for each t_i. For any given i, in the case that this event occurs and by the definition of $H_{\Gamma_i}(t_i)$, $H_{\Gamma_i}(t_i)$ contains a path P from t_i to j consisting entirely of typical nodes, all of which are at most t_i, and each node on the path P has creation time (index) at most half of its immediate predecessor.

3.5 Discussion

We would like to end this chapter with a short discussion on the potential applicability of the correctness analysis of TraverseToTheRoot to other problems. At the core of the analysis (and intuition) behind TraverseToTheRoot lie Lemma 3.11. For convenience of the reader, we restate it here:
"Fix sparse set Γ. Then for each $i \in [s_0, s_1]$ and $k \in [m]$, the following are true with probability $\geq 8/15$: $p_k(i) \notin \Gamma$, $p_k(i) \leq i/2$, and $p_k(i)$ is typical."
At its highest level, Lemma 3.11 brings an appealing mathematical description of the preferential attachment process: with constant probability, all nodes not too far or too close to the root (namely in $[s_0, s_1]$) have typical parents and these parents are far enough back. However, the lemma brings an extra combinatorial ingredient: such a picture of the preferential attachment process is true even if a fraction of $1/\log\log(n)$ nodes from each interval I_t is forbidden to be used. As such, the lemma provides a tool for analysis of traversal processes that have rather large dependencies in the way they work. This dependency would be encoded as the forbidden set (namely Γ) and the lemma can be used as if there is no dependency, as long as Γ does not overlap too much with any interval I_t. Specifically in the context of TraverseToTheRoot, the lemma, combined with natural properties of preferential attachment networks, allows us to show progress towards a closer to the root node although many past paths might have been rendered unsuccessful.

It is an interesting future direction to see whether there are other useful traversal processes on preferential attachment networks where the lemma could help to analyze.

Chapter 4

Local Information Algorithms for Network Coverage Problems

4.1 Introduction

In this chapter we continue our exploration of local information algorithms. Our focus in this chapter is on the expected cost of local information algorithms for network coverage problems. As we saw in the previous chapter, between others, this problem is motivated by network advertising and network monitoring.

We begin by exploring local algorithms for finding small number of individuals that assist in covering all the nodes in the network. We then consider two natural variants of this problem. The first variant, which we call the "neighbor-collecting" problem, is inspired by prize-collecting problems where the goal is to get a reward for each node we choose, and get penalized for those nodes we leave undominated (i.e. getting penalized for nodes that neither have a neighbor which was chosen nor were chosen themselves). In the second variant, the goal is to cover a given proportion of the population.

Detailed Results and Techniques

First, we explore local information algorithms for the min dominating set problem. A dominating set is a set S such that each node in the network is either in S or the neighborhood of S. We design a randomized local information algorithm for the minimum dominating set problem that achieves an approximation ratio that nearly matches the lower bound on polytime algorithms with no information restriction. As has been noted in [57], the greedy algorithm that repeatedly selects the visible node that maximizes the size of the dominated set can achieve a very bad approximation factor. We consider a modification of the greedy algorithm: after each greedy addition of a new node v, the algorithm will also add a random neighbor of v. We show that this randomized algorithm, which uses only Crawl queries, obtains an approximation factor that matches the known lower bound of $\Omega(\log \Delta)$ (where Δ is the maximum degree in the network) up to a constant factor. We note that our algorithm produces a connected dominating set and as such can also be seen as an $O(H(\Delta))$ approximation to the connected dominating set problem [4]. We also show that having enough local information to choose the node that maximizes the incremental benefit to the dominating set size is crucial: any algorithm that can see only the degrees of the neighbors of S would achieve a poor approximation factor of $\Omega(n)$.

We then discuss the "neighbor-collecting" problem, in which the goal is to minimize $c|S|$ plus the number of nodes left undominated by S, for a given parameter c. For this problem we show that the modified greedy algorithm presented above (which is 1^+-local algorithm) can be used to give an $O(c \log \Delta)$ approximation (where Δ is the maximum degree in the network). We also show that the dependence on c is unavoidable, and that the amount of local information is crucial: any 1-local algorithm would achieve a poor approximation factor of $\Omega(\sqrt{n}/c)$.

For the partial dominating set problem (where the goal is to cover a given constant fraction of the network with as few nodes as possible) we give an impossibility result: no local information algorithm can obtain an approximation better than $O(\sqrt{n})$ on networks with n nodes. However, a natural modification to the local information algorithm for minimum dominating set yields a a bicriteria result: given $\epsilon > 0$, we compare the performance of an algorithm that covers ρn nodes with the optimal solution that covers $\rho(1+\epsilon)n$ nodes

[4]this immediately follows from the fact that the size of the minimum connected dominating set is at most twice that of the minimum dominating set

(assuming $\rho(1+\epsilon) \leq 1$). Our modified algorithm achieves a $O((\rho\epsilon)^{-1}H(\Delta))$ approximation (in which we compare performance against an adversary who must cover an additional ϵ fraction of the network).

Related Work

Pseudo-local distributed algorithms for the minimum dominating set problem were developed in [80]. A multiplicative approximation factor of order $n^{O(1/k)}\log(\Delta)$ can be achieved in $O(k)$ rounds, and so a polylogarithmic approximation can be achieved in at least $\Theta(\log n)$ rounds. As discussed in the dissertation's introduction, such an algorithm can be simulated by a local information algorithm with a fixed radius of information (such as 2) but with an exponential number of queries as a function of the radius of information. This is clearly too expensive (resulting in a need to query the whole network). Furthermore, it is known that at least a logarithmic number of rounds are necessary for any local distributed algorithm to achieve a polylogarithmic approximation for many coverage problems, including the minimum dominating set problem [80]. These results indicate that local distributed computation may not be a good approach for the design of competitive local information algorithms.

For the minimum dominating set problem, Guha and Khuller [57] designed an $O(\log\Delta)$-approximation algorithm (where Δ is the maximum degree in the network) that can be viewed as a local information algorithm in our framework. As a local information algorithm, their method requires that the network structure is revealed up to distance two from the current dominating set. By contrast, our local information algorithm requires less information to be revealed on each step. Our focus, and the motivation behind this distinction, is to determine sharp bounds on the amount of local information required to approximate this problem (and others) effectively.

The "neighbor-collecting" problem we consider is inspired by the long work on prize-collecting network design problems [8, 66]. Highly speaking, in such problems there is a cost c associated in satisfying a demand that needs to be "served" and a penalty associated with demands that are left unsatisfied. In our context, the cost c is charged for each node in

S and a penalty is given for each of the nodes left undominated.

Exact, exponential time algorithms for the partial dominating set problem are discussed in [76]. Approximation algorithms for the partial dominating set problem are not discussed explicitly in the literature. However, it is well known that that the min dominating set problem has polynomial-time approximation preserving reduction to the min set cover problem and vise-versa. See [67], Theorem $A.1$. It is easy to see that the same reductions are approximation persevering between the the partial dominating set problem and partial set cover problem (where in the partial set cover problem the goal is to find the smallest set that covers a given number ρn of the elements). Kearns is the first to study the partial dominating set problem [70] (Theorem 5.15). He shows that the standard greedy set-cover algorithm can be adapted to provide an approximation factor of $2H(|U|)+3$ on a universe U. Slavík shows that a minor modification to the greedy algorithm obtains an approximation factor of $H(\min\{\Delta, \rho n\})$, where Δ here means the maximum size of a set in the system [110].

Könemann et al. show that using a specially designed LP program for a generalized partial cover problem (where items have profits and sets has costs) one can achieve an approximation factor of $(4/3+\epsilon)H(\Delta)$, for any fixed $\epsilon > 0$ [77].

4.2 Preliminaries

Notation and definitions We write n_G for the number of nodes in an undirected graph $G = (V, E)$, $d_G(v)$ for the degree of a vertex v in $G = (V, E)$, and $N_G(v)$ for the set of neighbors of v in $G = (V, E)$. Given a subset of vertices $S \subseteq V$, $N_G(S)$ is the set of nodes adjacent to at least one node in S. We also write $D_G(S)$ for the set of nodes *dominated* by S: $D_G(S) = N_G(S) \cup S$. We say S is a *dominating set* if $D_G(S) = V$. Given nodes u and v, the distance between u and v is the number of edges in the shortest path between u and v. The distance between vertex sets S and T is the minimum distance between a node in S and a node in T. Given a subset S of nodes of G, the subgraph induced by S is the subgraph consisting of S and every edge with both endpoints in S. Finally, Δ_G is the maximum degree in G. In all of the above notation we often suppress the dependency on G when clear from context.

4.3 Minimum Dominating Set on Arbitrary Networks

We now consider the problem of finding a dominating set S of minimal size for an arbitrary graph G (with n nodes). Even with full (non-local) access to the network structure, it is known that for any $\epsilon > 0$ it is impossible to approximate the Minimum Dominating Set Problem to within a factor of $(1-\epsilon)\ln n$ in polynomial time, unless $NP \subseteq$ DTIME $\left(n^{O(\log\log n)}\right)$ [44]. Since the n'th harmonic number $H(n)$ converges to $\ln n + \gamma$ where (the Euler-Mascheroni constant) $\gamma \approx 0.577$, and the maximum degree Δ is smaller than n, this yields a $(1-\epsilon)H(\Delta)$ hardness under the same assumptions, for any $\epsilon > 0$.

In this section we explore how much local network structure must be made visible in order for it to be possible to match this lower bound.

Guha and Khuller [57] design an $O(H(\Delta))$-approximate algorithm for the minimum dominating set problem, which can be interpreted in our framework as a 2^+-local algorithm. Their algorithm repeatedly selects a node that greedily maximizes the number of dominated nodes, considering only nodes within distance 2 of a previously selected node. As we show, the ability to observe network structure up to distance 2 is unnecessary if we allow the use of randomness: we will construct a randomized $O(H(\Delta))$ approximation algorithm that is 1^+-local. We then show that this level of local information is crucial: no algorithm with less local information can return a non-trivial approximation.

4.3.1 A 1^+-local Algorithm

We now present a 1^+-local randomized $O(H(\Delta))$-approximation algorithm for the min dominating set problem. Our algorithm obtains this approximation factor both in expectation and with high probability in the optimal solution size. Our algorithm actually generates

Algorithm 4 AlternateRandom

```
1: Select an arbitrary node $u$ from the graph and initialize $S = \{u\}$.
2: while $|D(S)| < |V|$ do
3:     Choose $x \in \arg\max_{v \in N(S)}\{|N(v)\backslash D(S)|\}$ and add $x$ to $S$.
4:     if $N(x)\backslash S \neq \emptyset$ then
5:         Choose $y \in N(x)\backslash S$ uniformly at random and add $y$ to $S$.
6:     end if
7: end while
8: return $S$.
```

a connected dominating set, so it can also be seen as an $O(H(\Delta))$ approximation to the connected dominating set problem.

Roughly speaking, our approach is to greedily grow a subtree of the network, repeatedly adding vertices that maximize the number of dominated nodes. Such a greedy algorithm is 1^+-local, as this is the amount of visibility required to determine how much a given node will add to the number of dominated vertices. Unfortunately, this greedy approach does not yield a good approximation; it is possible for the algorithm to waste significant effort covering a large set of nodes that are all connected to a single vertex just beyond the algorithm's visibility. To address this issue, we introduce randomness into the algorithm: after each greedy addition of a node x, we will also query a random neighbor of x. The algorithm is listed above as Algorithm 4 (AlternateRandom).

We now show that AlternateRandom obtains an $O(H(\Delta))$ approximation, both in expectation and with high probability. In what follows, $\mathcal{OPT}$ will denote an optimal solution to the problem at hand, on the implicit input graph.

Theorem 4.1. *AlternateRandom is* 1^+*-local and returns a dominating set S where* $\mathbb{E}[|S|] \leq (2H(\Delta)+4)|\mathcal{OPT}|+1$. *Also,* $\mathbb{P}[|S| > (2H(\Delta)+6)|\mathcal{OPT}|+1] < e^{-\frac{|\mathcal{OPT}|}{4}}$.

Proof. Correctness follows from line 2 of the algorithm. To show that it is 1^+-local, it is enough to show that line 3 can be implemented by a 1^+-local algorithm. This follows because, for any $v \in N(S)$, $|N(v) \backslash D(S)|$ is precisely equal to the degree of v minus the number of edges between v and other nodes in $D(S)$.

We will bound the expected size of S via the following charging scheme. Whenever a node x is added to S on line 4, we place a charge of $1/|N(x) \backslash D(S)|$ on each node in $N(x) \backslash D(S)$. These charges sum to 1, so sum of all charges increases by 1 on each invocation of line 4. We will show that the total charge placed during the execution of the algorithm is at most $(2 + H(\Delta))\mathcal{OPT}$ in expectation. This will imply that $\mathbb{E}[(|S| - 1)/2] \leq (2 + H(\Delta))\mathcal{OPT}$ as required.

Let $\mathcal{OPT}$ be a minimum dominating set. Partition the nodes of G as follows: for each $i \in \mathcal{OPT}$, choose a set $S_i \subseteq D(\{i\})$ containing i such that the sets S_i form a partition of G. Choose some $i \in \mathcal{OPT}$ and consider the set S_i. We denote by a "step" any execution of line 4 in which charge is placed on a node in S_i. We divide these steps into two phases: phase 1 consists of steps that occur while $S_i \cap S = \emptyset$, and phase 2 is all other steps. Note that since we never remove nodes from S, phase 1 occurs completely before phase 2.

We first bound the total charge placed on nodes in S_i in phase 1. In each step, some number k of nodes from S_i are each given some charge $1/z$. This occurs when $|N(x) \backslash D(S)| = z$ and $(N(x) \backslash D(S)) \cap S_i = k$. In this case, if phase 1 has not ended as a result of this step, there is a k/z probability that a node in S_i is selected on the subsequent line 6 of the algorithm, which would end phase 1. We conclude that if the total charge added to nodes in S_i on some step is $p \in [0,1]$, phase 1 ends for set S_i with probability at least p. The following probabilistic lemma now implies that the expected sum of charges in phase 1 is at most 1.

Lemma 4.2. *For $1 \leq i \leq n$, let X_i be a Bernoulli random variable with expected value $p_i \in [0,1]$. Let T be the random variable denoting the smallest i such that $X_i = 1$ (or n if $X_i = 0$ for all i). Then $\mathbb{E}_T\left[\sum_{i=1}^{T} p_i\right] \leq 1$.*

Proof. We proceed by induction on n. The case $n = 1$ is trivial. For $n > 1$, we note that

$$\mathbb{E}_T\left[\sum_{i=1}^{T} p_i\right] = p_1 + (1 - p_1)\mathbb{E}_T\left[\sum_{i=2}^{T} p_i \mid X_1 = 0\right] \leq p_1 + (1 - p_1) \cdot 1 = 1$$

where the inequality follows from the inductive hypothesis applied to $X_2, \ldots, X_n$.

□

Consider the charges added to nodes in S_i in phase 2. During phase 2, vertex i is eligible to be added to S in step 4. Write $u_j = |S_i \backslash D(S)|$ for the number of nodes of S_i not dominated on step j of phase 2. Then, on each step j, $u_j - u_{j+1}$ nodes in S_i are added to $D(S)$, and at least u_j nodes in G are added to $D(S)$ (since this many would be added if i were chosen, and each choice is made greedily). Thus the total charge added on step j is at most $\frac{u_j - u_{j+1}}{u_j}$. After at most $t \leq \Delta + 1$ iterations all the nodes in S_i are covered, so the total charge over all of phase 2 is at most

$$\begin{aligned}\sum_{j=1}^{t} \frac{u_j - u_{j+1}}{u_j} &= \sum_{j=1}^{t} \sum_{\ell=(u_{j+1}+1)}^{u_j} \frac{1}{u_j} \\ &\leq \sum_{j=1}^{t} \sum_{\ell=(u_{j+1}+1)}^{u_j} \frac{1}{\ell} \quad \text{(becuase } \ell \leq u_j\text{)} \\ &= \sum_{j=1}^{t} \big(H(u_j) - H(u_{j+1})\big) = H(u_1) - H(u_{t+1}) \\ &\leq H(|S_i|) - H(0) \leq H(\Delta+1) \leq H(\Delta) + 1.\end{aligned}$$

We conclude that the expected sum of charges over both phases is at most $2 + H(\Delta)$.

We now turn to show that $\mathbb{P}[|S| > 2(3 + H(\Delta))|\mathcal{OPT}| + 1] < e^{-\frac{|\mathcal{OPT}|}{4}}$. We will use the same charging scheme we defined in the main text; it suffices to show that the total charge placed, over all nodes in G, is at most $(2 + H(\Delta))|\mathcal{OPT}|$ with probability at least $1 - e^{-\frac{|\mathcal{OPT}|}{4}}$. Note that our bound on the charges from phase 2 in the analysis of the expected size of $|S|$ holds with probability 1. It is therefore sufficient to bound the probability that the sum, over all i, of the charges placed in phase 1 of S_i is at most $2|\mathcal{OPT}|$.

For each node x added to S on line 4, consider the total number of nodes in $N(x)\backslash D(S)$ that lie in sets S_i that are in phase 1. Now suppose there are k such nodes, and that $|N(x)\backslash D(S)| = z$. Then the sum of charges attributed to phase 1 increases by k/z on this invocation of line 4. Also, the probability that any of these k nodes is added to S on the next execution of line 6 is at least k/z, and this would end phase 1 for at least one set S_i.

We conclude that, if the sum of charges for phase 1 increases by some $p \in [0,1]$, then with probability p at least one set S_i leaves phase 1. Also, no more charges can be attributed to phase 1 once all sets S_i leave phase 1, and there are $|\mathcal{OPT}|$ such sets. The event that the sum of charges attributed to phase 1 is greater than $2|\mathcal{OPT}|$ is therefore dominated by the event that a sequence of Bernoulli random variables $X_1,\ldots,X_n$, each X_i having mean p_i with $\sum p_i > 2|\mathcal{OPT}|$, has sum less than $|\mathcal{OPT}|$. However, by the multiplicative Chernoff bound (lemma A.1), this probability is at most

$$\mathbb{P}\left[\sum_{i=1}^{n} X_i < |\mathcal{OPT}|\right] \leq \mathbb{P}\left[\sum_{i=1}^{n} X_i < \frac{1}{2}\mathbb{E}[\sum_{i=1}^{n} X_i]\right] < e^{-\frac{|\mathcal{OPT}|}{4}}$$

as required.

□

We end this section by showing that 1^+-locality is necessary for constructing good local approximation algorithms.

Theorem 4.3. *For any randomized* 1*-local algorithm A for the min dominating set problem, there exists an input instance G for which $\mathbb{E}[|S|] = \Omega(n)|T|$, where S denotes the output generated by A on input G, and T denotes the output of the best algorithm with full knowledge of the input graph.*

Proof. We consider a distribution over input graphs $G = (V,E)$ of size n, described by the following construction process. Choose $n-2$ nodes uniformly at random from V and form a clique on these nodes. Choose an edge at random from this clique, say (u,v), and remove that edge from the graph. Finally, let the remaining two nodes be u' and v', and add edges (u,u') and (v,v') to E. See Figure 4.1 for an illustration.

By Yao's Minimax Principle [119], it suffices to consider the least expected performance of deterministic 1-local algorithms on inputs drawn from this distribution.

Note that each such graph has a dominating set of size 2, namely $\{u,v\}$ and that using Crawl queries, $T \leq 3$. Moreover, any dominating set of G must contain at least one node in $C = \{u,v,u',v'\}$, and hence a 1-local algorithm must query a node in C. However, if no

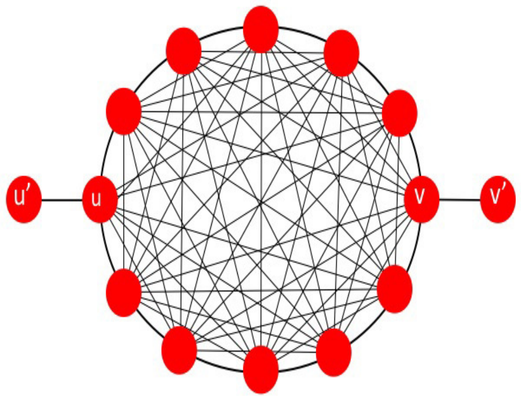

Figure 4.1: An illustration of the lower bound construction for 1-local algorithms for the min dominating set problem.

nodes in C have been queried, then nodes u and v are indistinguishable from other visible unqueried nodes (as they all have degree $n-1$). Thus, until the algorithm queries a node in C, any operation is equivalent to querying an arbitrary unqueried node from $V \backslash \{u', v'\}$. With high probability, $\Omega(n)$ such queries will be executed before a node in C is selected. By Yao's Minimax Principle this gives a lower bound of $\Omega(n)$ on the expected performance of any randomized 1-local algorithm, on at least one of the inputs. □

4.4 Partial Coverage Problems on Arbitrary Networks

We next study problems in which the goal is not necessarily to cover all nodes in the network, but rather dominate only sections of the network that can be covered efficiently. We consider two central problems in this domain: the the neighbor collecting problem and the partial dominating set problem.

4.4.1 The Neighbor Collecting Problem

Inspired by prize-collecting network design problems, we next consider the objective of minimizing the total cost of the selected nodes plus the number of nodes left uncovered: choose a set S of G that minimizes $f(S) = c|S| + |V \backslash D(S)|$ for a given parameter $c > 0$.

Note that when $c < 1$ the problem reduces to the minimum dominating set problem: it is always worthwhile to cover all nodes. Assuming $c \geq 1$, the 1^+-local algorithm for the minimum dominating set problem achieves an $O(cH(\Delta))$ approximation.

Theorem 4.4. *For any $c \geq 1$ and set $\mathcal{OPT}$ minimizing $f(\mathcal{OPT})$, algorithm AlternateRandom returns a set S for which $\mathbb{E}[f(S)] \leq 2c(2 + H(\Delta))f(\mathcal{OPT})$.*

Proof of Theorem 4.4. We have $f(\mathcal{OPT}) = |V - D(\mathcal{OPT})| + c|\mathcal{OPT}|$ and $f(\mathcal{OPT} \cup \{V - D(\mathcal{OPT})\}) = c|\mathcal{OPT} \cup \{V - D(\mathcal{OPT})\}| = c|\mathcal{OPT}| + c|V - D(\mathcal{OPT})| \geq c|T^*|$ where T^* is a minimum dominating set of the graph. Next, we know from Theorem 4.1 that $|T^*| \geq (2(2 + H(\Delta)))^{-1}\mathbb{E}[|S|] = (2(2 + H(\Delta)))^{-1}c^{-1}\mathbb{E}[f(S)]$.

Finally, $f(\mathcal{OPT}) = |V - D(\mathcal{OPT})| + c|\mathcal{OPT}|$ so $c|\mathcal{OPT}| + c|V - D(\mathcal{OPT})| \leq cf(\mathcal{OPT})$. We conclude that $\mathbb{E}[f(S)] \leq 2(2 + H(\Delta))cf(\mathcal{OPT})$. □

Since the neighbor-collecting problem contains the minimum dominating set problem as a special case (i.e. when $c = 1$), we cannot hope to avoid the dependency on $H(\Delta)$ in the approximation factor in Theorem 4.4. As we next show, the dependence on c in the approximation factor we obtain in Theorem 4.4 is also unavoidable.

Theorem 4.5. *Let $c = o(n^{1/4})$. Then for any randomized 1^+-local algorithm A for the neighbor-collecting problem, there exists an input instance G for which*

$$\mathbb{E}[f(S)] = \Omega(\max\{c, H(\Delta)\}) \cdot f(|T|),$$

where S denotes the output generated by A on input G, and T denotes the output of the best algorithm with full knowledge of the input graph.

Proof. As done previously, we shall invoke the application of Yao's Minimax Principle for the performance of randomized algorithms [119].

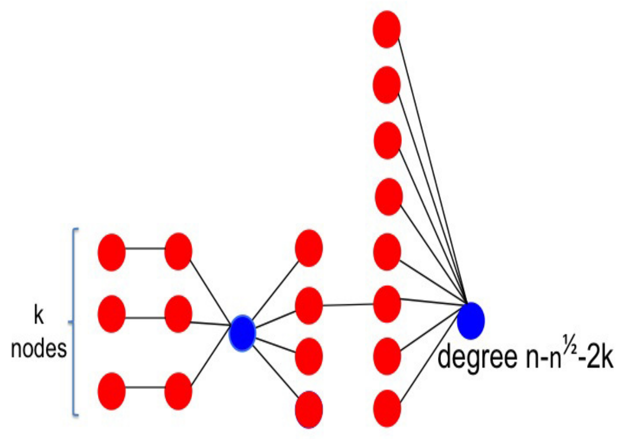

Figure 4.2: An illustration of the lower bound construction for 1^+-local algorithms for the neighbor-collecting problem.

We will build the following family of inputs. Given n we construct a connected graph on n nodes in the following way. Let $k = o(\sqrt{n}/c)$ be a parameter to be set specifically later. Create two star subgraphs one on $n - \sqrt{n} - 2k$ nodes and one on $\sqrt{n}$ nodes. We connect one arbitrary leaf of the big star subgraph to one arbitrary leaf of the smaller star subgraph. Next, choose k spoke nodes from the bigger star subgraph and connect each of them to one new node of degree one. To complete the construction, randomly assign labels to the nodes from $[n]$. This gives us a connected graph on n vertices. Note that T is at most $6c + k$ as an algorithm can always cover the hubs of the two stars in four Crawl steps. See Figure 4.2 for an illustration of the construction.

We show that the expected cost of a deterministic algorithm on the distribution of inputs is at least $(1 + k/2 + 2)c$. This happens when the 1^+ local algorithm starts from a spoke in the bigger star component (with occurs with probability $1 - o(1)$) and needs to traverse half of the k spokes that were assigned one new neighbor. Only after traversing such nodes it moves into a spoke of the small star subgraph and then to the hub of the smaller star subgraph. We note that $k = o(\sqrt{n}/c)$, so taking any Jump queries would result in a worse cost.

Thus the approximation ratio is at least $\frac{(1+k/2+2)c}{6c+k}$. This expression is the biggest (as a function of $k = o(\sqrt{n}/c)$) for $k = \Theta(c)$. In that case the expression is $\Theta(c)$. By Yao's Minimax Principle this gives a lower bound of $\Omega(c)$ on the expected performance of any randomized 1^+-local algorithm, on at least one of the inputs. □

Finally, one cannot move from 1^+-local algorithms to 1-local algorithms without significant loss: every 1-local algorithm has a polynomial approximation factor.

Theorem 4.6. *For any randomized* 1*-local algorithm A for the neighbor-collecting problem, there exists an input instance G for which $\mathbb{E}[f(S)] = \Omega(\sqrt{n}/c) \cdot f(|T|)$, where S denotes the output generated by A on input G, and T denotes the output of the best algorithm with full knowledge of the input graph.*

Proof. As done previously, we shall invoke the application of Yao's Minimax Principle for the performance of randomized algorithms [119].

We will build the following family of inputs. Build a clique on $n - \sqrt{n}$ vertices and remove one edge (u,v). Next build a star with $\sqrt{n} - 1$ leaves, say with root r, and label one of the leaves v'. Finally, add edge (v,v'). To complete the construction, randomly assign labels to the nodes from $[n]$. See Figure 4.3 for an illustration of the construction.

For such graph, the set $\{r,v\}$ has cost $2c$ and can be found in five Crawls, so $f(|T|) \leq 5c$. Consider the set S returned by a 1-local deterministic algorithm; we will show that S will have cost at least $\sqrt{n}$ with high probability. If S does not include r then it must leave $\sqrt{n}$ nodes uncovered (or else contain at least $\sqrt{n}$ vertices), in which case it has cost at least $\sqrt{n}$. So S must contain some node in the star centered at r. A node in the star can be found either via a random query or by querying node v. Since the star contains $\sqrt{n}$ nodes, it would take $\Omega(\sqrt{n})$ random queries to find a node in the star with high probability. On the other hand, node v is indistinguishable from the other nodes but u in the $(n - \sqrt{n})$-clique until after it has been queried; it would therefore take $\Omega(n)$ queries to the nodes in the clique to find v, again with high probability. We conclude that the cost of S is at least $\sqrt{n}$

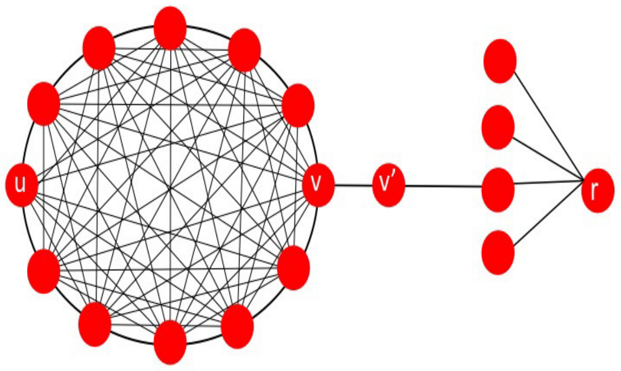

Figure 4.3: An illustration of the lower bound construction for 1-local algorithms for the neighbor-collecting problem.

in expectation. By Yao's Minimax Principle, the expected cost of a randomized 1-local algorithm is at least $\Omega(\sqrt{n})$ and its approximation ratio is therefore $\Omega(\sqrt{n}/c)$. $\square$

4.4.2 The Partial Dominating Set Problem

In the partial dominating set problem we are given a parameter $\rho \in (0,1]$. The goal is to find the smallest set S such that $|D(S)| \geq \rho n$.

We begin with a negative result: for any constant k and any k-local algorithm, there are graphs for which the optimal solution has constant size, but with high probability $\Omega(\sqrt{n})$ queries are required to find any ρ-partial dominating set. Our example will apply to $\rho = 1/2$, but can be extended to any constant $\rho \in (0,1)$.

Theorem 4.7. *Let $k = o(\sqrt{n})$. For any randomized k-local algorithm A (with success probability, say $2/3$) for the partial dominating set problem with $\rho = 1/2$, there exists an input G for which the expected size of the output returned is $\mathbb{E}[|S|] = \Omega(\frac{\sqrt{n}}{k}) \cdot |T|$, where*

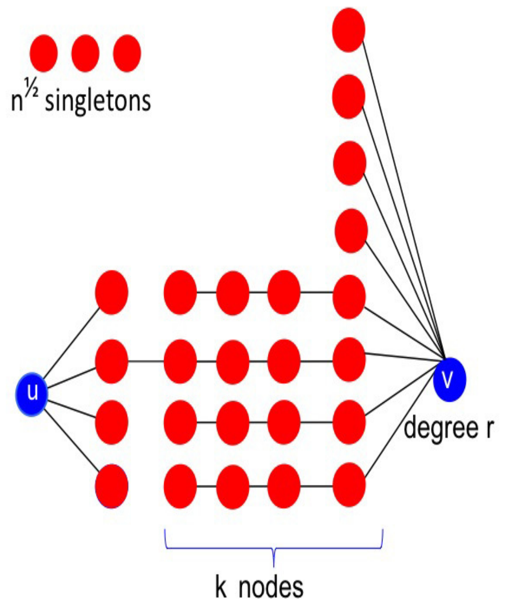

Figure 4.4: An illustration of the lower bound construction for k-local algorithms for the partial dominating set problem.

S denotes the output generated by A on input G, and T denotes the output of the best algorithm (with success probability, say $2/3$) that has full knowledge of the input graph.

Proof. Fix n and write $r = \frac{n/2-\sqrt{n}-1}{k}$. We define a distribution over input graphs on n nodes corresponding to the following construction process. Build two stars, one with $n/2 - \sqrt{n} - 1$ leaves and one with $\sqrt{n} - 1$ leaves, where the nodes in these stars are chosen uniformly at random. Let v and u be the roots of these stars, respectively. Construct r paths, each of length $k + 1$, again with the nodes being chosen uniformly at random. Connect one endpoint of each path to a separate leaf of the larger star, the one rooted at v. Choose one of these r paths and connect its other endpoint to node u. Last, add $\sqrt{n}$ isolated nodes to get the number of nodes equal n in the construction. By Yao's Minimax Principle [119] it suffices to consider the least expected performance of deterministic algorithms on a graph chosen from this distribution. See Figure 4.4 for an illustration of the construction.

With full knowledge of such a graph, one can find a node in the large connected component with probability $2/3$, using $O(1)$ Jumps, and then use at most $k+2$ Crawls to get to the roots of star graphs (node $v$and u). Thus the least expected cost (over successful runs) is $\mathbb{E}[T] \leq O(k)$.

We claim that any deterministic k-local algorithm performs at least $\sqrt{n}$ queries in expectation (over successful runs) on this family of inputs. First, if the algorithm does not return the root of the smaller star as part of its solution, then it must return at least $O(\sqrt{n})$ nodes and hence it must use $\Omega(\sqrt{n})$ queries. On the other hand, suppose that the algorithm does return the root of the smaller star. Then it must have either traversed the root some node along the path connecting the centers of the stars, or else found a node in the smaller star via a random jump query. The latter takes $\Omega(\sqrt{n})$ Jump queries, in expectation (over the randomness in the Jump queries). For the former, note that an algorithm cannot distinguish the path connecting the two stars from any other path connected to node v, until after a vertex on one of the two paths has been queried. It would therefore take $\Omega(r) = \Omega(n/k)$ queries in expectation to traverse one of the nodes on the path between the two stars. By Yao's Minimax Principle, we conclude that any randomized k-local algorithm must perform at least $\Omega(\sqrt{n})$ queries in expectation in order to construct an admissible solution. $\square$

Motivated by this lower bound, we consider a bicriteria result: given $\epsilon > 0$, we compare the performance of an algorithm that covers ρn nodes with the optimal solution that covers $\rho(1+\epsilon)n$ nodes (assuming $\rho(1+\epsilon) \leq 1$). We show that a modification to Algorithm 4, in which jumps to uniformly random nodes are interspersed with greedy selections, yields an 1^+-local algorithm with an $O((\rho\epsilon)^{-1}H(\Delta))$ approximation guarantee.

Theorem 4.8. *Given any $\rho \in (0,1)$, $\epsilon \in (0, \rho^{-1} - 1]$, and set of nodes $\mathcal{OPT}$ with the property $|D(\mathcal{OPT})| \geq \rho(1+\epsilon)n$, Algorithm 5 (AlternateRandomAndJump) returns a set S of nodes with $|D(S)| \geq \rho n$ and $\mathbb{E}[|S|] \leq (\rho\epsilon)^{-1}(H(\Delta)+2)|\mathcal{OPT}|$.*

Proof. We apply a modification of the charging argument used in Theorem 4.1. Let $\mathcal{OPT}$ be a set of nodes as in the statement of the theorem. We will partition the nodes of $D(\mathcal{OPT})$ as follows: for each $i \in \mathcal{OPT}$, choose a set $S_i \subseteq D(\{i\})$ containing i, such that the sets S_i form a partition of $D(\mathcal{OPT})$.

Algorithm 5 AlternateRandomAndJump

```
1: Initialize $S = \emptyset$.
2: while $|D(S)| < |V|$ do
3:     Choose a node $u$ uniformly at random from the graph and add $u$ to $S$.
4:     Choose $x \in \arg\max_{v \in N(S)}\{|N(v) \backslash D(S)|\}$ and add $x$ to $S$.
5:     if $N(x) \backslash S \neq \emptyset$ then
6:         Choose $y \in N(x) \backslash S$ uniformly at random and add $y$ to $S$.
7:     end if
8: end while
9: return $S$.
```

During the execution of algorithm 5, we will think of each node in $D(OPT)$ as being marked either as **Inactive**, **Active**, or **Charged**. At first all nodes in $D(\mathcal{OPT})$ are marked **Inactive**. During the execution of the algorithm, some nodes may have their status changed to **Active** or **Charged**. Once a node becomes **Active** it never subsequently becomes **Inactive**, and once a node is marked **Charged** it remains so for the remainder of the execution. Specifically, all nodes in $D(\mathcal{OPT}) \cap D(S)$ are always marked **Charged**, in addition to any nodes that have been assigned a charge by our charging scheme (described below). Furthermore, for each $i \in \mathcal{OPT}$, the nodes in S_i that are not **Charged** are said to be **Active** if $i \in D(S)$; otherwise they are **Inactive**.

Our charging scheme is as follows. On each iteration of the loop on lines 2-7, we will either generate a total charge of 0 or of 1. Consider one such iteration. Let u be the node that is queried on line 3 for this iteration. If $u \notin D(OPT) \backslash D(S)$ then we will not generate any charge on this iteration. Suppose instead that $u \in D(OPT) \backslash D(S)$. If no nodes are **Active** after u has been queried[5] then we place a unit of charge on u. Otherwise, let x be the node selected on line 4. Let $z = |N(x) \backslash D(S)|$ be the number of new nodes dominated by x, and let z' be the number of **Active** nodes. Let $w = \min\{z, z'\}$. We will then charge $1/w$ to w different vertices, as follows. First, we charge $1/w$ to each vertex

[5]This situation can occur only when u is the only node in $S_i \backslash D(S)$ for some i.

in $D(OPT) \cap (N(x) \backslash D(S))$ (note that there are at most w such nodes). If fewer than w nodes have been charged in this way, then charge $1/w$ to (arbitrary) additional **Active** nodes until a total of w nodes have been charged. We mark all charged nodes as **Charged**.

We claim that the total expected charge placed over the course of the algorithm will be at least $\rho\epsilon|S|$. To see this note that, on each iteration of the algorithm, there are at least $\rho\epsilon n$ nodes in $D(OPT) \backslash D(S)$ (since the algorithm has not yet completed). Thus, with probability at least $\rho\epsilon$, a node from $D(OPT) \backslash D(S)$ will be chosen on line 2. Thus, at least a $\rho\epsilon$ fraction of iterations will generate a charge, and on algorithm termination the sum of the charges on all vertices is expected to be at least $\rho\epsilon|S|$.

Choose some $i \in OPT$ and consider set S_i. We will show that the total charge placed on the nodes of S_i during the execution of algorithm A_2 is at most $(2 + H(\Delta))$ in expectation. Since there are $|\mathcal{OPT}|$ such sets, and since only nodes in sets S_i ever receive charge, this will imply that the total expected charge over all nodes is at most $(2 + H(\Delta))|\mathcal{OPT}|$. We then conclude that $\rho\epsilon\mathbb{E}[|S|] \leq (2 + H(\Delta))|\mathcal{OPT}|$, completing the proof.

The analysis of the total charge placed on nodes of S_i is similar to the analysis in Theorem 4.1. In expectation, a total charge of 1 will be placed on the nodes of S_i before $i \in D(S)$ (this is phase 1 in the proof of Theorem 4.1). After $i \in D(S)$, all nodes in $S_i \backslash D(S)$ are marked **Active**. When a node is crawled on line 4, if $k > 0$ nodes in $S_i \backslash D(S)$ are **Active**, then it must be that $i \in D(S)$ but $i \notin S$. Thus, i is a valid choice for the node selected on line 4. So, on any such iteration, it must be that the node selected on line 4 dominates at least k new nodes. We conclude that each node that is charged on this iteration receives a charge of at most $1/k$.

To summarize, if k nodes of S_i are **Active** on a given iteration, then any nodes in S_i can be charged at most $1/k$ on that iteration. Since $|S_i| \leq \Delta + 1$, we conclude in the same manner as in Theorem 4.1 that the total expected charge allocated to nodes in S_i, after the first node in S_i becomes **Active**, is at most $\sum_{k=1}^{\Delta+1} \frac{1}{k} \leq (H(\Delta) + 1)$. We conclude that the total expected charge placed on all nodes in S_i is at most $(2 + H(\Delta))$

□

4.5 Discussion

We would like to end this chapter with a short discussion on the applicability of our techniques and results to other problems. We have shown that a greedy-random approach yields low-cost local information algorithms, with small radius of information. In its essence this approach is based on the idea of an exploitation step , namely a greedy step, followed by an exploration step, a random Crawl step. We believe that this idea could be beneficial in part to other optimization problem as well, leading to the design of low-cost local information algorithms with small radius of information.

Perhaps, as an example demonstrating the potential of the greedy-random approach, consider the minimum r-hop dominating set problem, where one wants to find a set of minimum size such that all nodes in the graph are at distance at most r from that set. We first observe that a set is r-hop dominating in G if that set is 1-hop dominating in the r-closure graph of G, namely the graph where any two nodes at distance at most r in G are neighbors. The r-closure graph of G can be induced from r-local information. Thus, following algorithm 4 and its analysis one can immediately design a r^+-local algorithm with $O(H(\Delta_G))$-approximation to the minimum r-hop dominating set problem. We note that similarly to the lower bound presented in section 4.3, one can show that any algorithm (possibly randomized) to the minimum r-hop dominating set problem with less local information would give a poor approximation (by replacing each edge in the lower bound construction with a path of length r).

4.6 Conclusions

We presented a model of computation in which algorithms are constrained in the information they have about the input structure, which is revealed over time as expensive exploration decisions are made. Our motivation lies in determining whether and how an external user in a network, who cannot make arbitrary queries of the graph structure, can efficiently solve optimization problems in a local manner.

Our results on local algorithms for coverage problems suggest that inherent structural properties of social networks may be crucial in obtaining strong performance bounds.

Another implication is that the designer of a network interface, such as an online social network platform, may gain from considering the power and limitations that come with the design choice of how much network topology to reveal to individual users. On one hand, revealing too little information may restrict natural social processes that users expect to be able to perform, such as searching for potential new connections. On the other hand, revealing too much information may raise privacy concerns, or enable unwanted behavior such as automated advertising systems searching to target certain individuals.

The results presented in this chapter suggest that even minor changes to the structural information made available to a user may have a large impact on the class of optimization problems that can be reasonably solved by the user.

Part II

Local Algorithms for Stringent Time Constraints

Chapter 5

Introduction

In this part of the dissertation we focus on the design and analysis of highly time efficient algorithms over large networks. In many examples, such as node ranking and influence maximization one must find a solution in the quickest time possible; methods that do not scale with network size are doomed to be intractable in practice. As the size of networks is huge, there is prominent interest in local methods that do not traverse the whole network to solve the problem at hand. Such methods would only rely on Jump and Crawl queries that are natural and easy to implement in many domains, instead of classical approaches that rely on processing and traversing the whole network multiple times. Information about the network is only provided for the nodes accessed, namely there is no local information, and one must pay a unit (of time) for each new piece of information traversed. We will refer to such methods as *local*.

In chapter 6 we focus on designing highly time efficient local algorithms for influence maximization under the standard Independent Cascades model. In chapter 7, we design highly time efficient local algorithms for computing the set of nodes with significant PageRank score. We will use very different techniques to tackle these problems than those used in previous chapters. While a novel technique to summarize large-scale diffusion information in a concise hypergraph representation will be our main technical contribution to solve the influence maximization problem, for PageRank computations a multi-scale node sampling method, that repeatedly invokes a novel random-walk based personalized PageRank approximation, would be rendered successful.

Chapter 6

Quasilinear and Sublinear Time Local Algorithms for Influence Maximization

6.1 Introduction

Diffusion is a fundamental process in the study of complex networks, modeling the spread of disease, ideas, or product adoption through a population. The common feature in each case is that local interactions between individuals can lead to epidemic outcomes. This is the idea behind word-of-mouth advertising, in which information about a product travels via links between individuals; see, for example, [7, 25, 27, 28, 50, 54, 105]. In recent years, as online social network structure has become increasingly visible, applications of diffusion models on networks to advertising have become increasingly relevant. A prominent application is a viral marketing campaign which aims to use a small number of targeted interventions to initiate cascades of influence that create a global increase in product adoption [38, 50, 71, 81].

A large part of this interest focuses on the algorithmic problem of inferring potential influencers from network topology. Given a network, how can we determine which individuals should be targeted to maximize the magnitude of a resulting cascade? [38, 71, 104]. Supposing that there is a limit k on the number of nodes to target (e.g. due to advertising budgets), the goal is to efficiently find an appropriate set of k nodes with which to "seed" a diffusion process.

In this chapter we develop fast local approximation algorithms for the above influence-maximization problem, under the standard *independent cascades* model of influence spread. Before describing out techniques and results, we would like to provide a detailed background into the problem at hand.

The Model: Independent Cascades We adopt the *independent cascades* (IC) model of diffusion, formalized by Kempe et al. [71]. In this model we are given a directed edge-weighted graph $\mathcal{G}$ with n nodes and m edges, representing the underlying network. Influence spreads via a random process that begins at a set S of seed nodes. Each node, once infected, has a chance of subsequently infecting its neighbors: the weight of edge $e = (v, u)$ represents the probability that the process spreads along edge e from v to u. If we write $I(S)$ for the (random) number of nodes that are eventually infected by this process, then we think of the expectation of $I(S)$ as the *influence* of set S. Our optimization problem, then, is to find set S maximizing $\mathbb{E}[I(S)]$ subject to $|S| \leq k$.

For the corresponding algorithmic problem we assume that the network topology is described in the *sparse* representation of an (arbitrarily ordered) adjacency list for each vertex, as is natural for representing complex networks. We are interested in developing efficient, and highly-local algorithms for the problem. The only queries to be used by our algorithm are accessing a vertex uniformly at random and traversing the edges incident to a previously-accessed vertex. Such algorithms access the network structure in a very limited way and are implementable in a wide array of graph access models, including the ones used for sublinear-time and property testing algorithms on sparse graphs [52, 106].

The IC model has become one of the prominent ways to model influence spread; see for example [31–33, 72, 83, 118]. One of the reasons is that it captures the common intuition that influence can spread stochastically through a network, much like a disease [36, 50, 71].

An important property of the IC model is its computational tractability. Indeed, Kempe et al. show that $\mathbb{E}[I(\cdot)]$ is a submodular monotone function [71], and hence the problem of maximizing $\mathbb{E}[I(\cdot)]$ can be approximated to within a factor of $(1-\frac{1}{e}-\epsilon)$ for any $\epsilon > 0$, in polynomial time, via a greedy hill-climbing method. In contrast, many other formulations of the influence maximization problem have been shown to have strong lower bounds on polynomial-time approximability [11, 30, 92, 103]

The greedy approach to maximizing influence in the IC model described above takes time $O(kn)$, *given oracle access to the function* $\mathbb{E}[I(\cdot)]$. In general, however, these influence values must be computed from the underlying network topology. A common approach is to simulate the random diffusion process many times to estimate influence values for each node. This ultimately[6] leads to a total runtime[7] of $\Omega(mnk \cdot \text{POLY}(\epsilon^{-1}))$. Due to the massive size and temporal volatility of many network dataset instances, this does not provide a fully satisfactory solution: it is crucial for an algorithm to scale well with network size [31, 83]. This requirement has spawned a large body of work aimed at developing more efficient methods of finding influential individuals in social networks. See, for example, [31–33, 65, 74, 83, 90, 118]. However, to date, this work has focused primarily on empirical methods; no algorithm has been found to provide provable approximation guarantees in time asymptotically less than $\Theta(nmk)$.

What is the bottleneck in developing better algorithms? One issue is the difficulty of estimating the expected influence of a given vertex. As an illustrating example, consider the simple network of a star on n nodes[8] in which each edge is bidirectional and has weight $p = n^{-1/4}$. In this example the center node has expected influence $n^{3/4}$ (infecting each leaf with probability p), whereas each leaf has expected influence $\Theta(n^{1/2})$. In this example, estimating the influence of a leaf via random sampling requires significant effort: one would have to realize the graph $\Omega(\frac{1}{p})$ times to obtain a reasonable estimate. As each realization requires linear time (to provide estimates for all leaves), a superlinear runtime would be

[6]After simple optimizations, such as reusing each simulation for multiple nodes.

[7]The best implementations appear to have running time $O(mnk\log(n) \cdot \text{POLY}(\epsilon^{-1}))$ [32], though to the best of our knowledge a formal analysis of this runtime has not appeared in the literature.

[8]Of course, the star graph is simple enough that there are obvious alternative methods for finding appropriate seed sets; we present it merely to illustrate the potential runtime requirements of estimating node influences directly.

required to estimate influences even if we are willing to tolerate multiplicative errors as large as $n^{1/4-\epsilon}$. A different approach is thus required if we are to achieve running time that is close to linear in the network size.

First Result: A Quasi-Linear Time Local Algorithm Our first and main result is a local algorithm for finding $(1 - 1/e - \epsilon)$-approximately optimal seed sets in arbitrary directed networks, which runs in time $O((m + n)\epsilon^{-3} \log n)$. Importantly, the runtime of our algorithm is independent of the number of seeds k and is essentially runtime optimal as we give a lower bound of $\Omega(m + n)$ on the runtime required to obtain a constant approximation for this problem (assuming an adjacency list representation). We also note that this approximation factor is nearly optimal, as no polytime algorithm achieves approximation $(1 - \frac{1}{e} + \epsilon)$ for any $\epsilon > 0$ unless P = NP [71, 72].

Our method is randomized, and it succeeds with probability $3/5$; moreover, failure is detectable, so this success probability can be amplified through repetition.

Our algorithm proceeds in two steps. First, we apply random sampling techniques to preprocess the network and generate a sparse hypergraph representation that estimates the influence characteristics of each node. Each hypergraph edge corresponds to a set of individuals that was influenced by a randomly selected node in the *transpose* graph. This preprocessing is done once, resulting in a structure of size $O((m + n)\epsilon^{-3} \log(n))$. This hypergraph encodes our influence estimates: for a set of nodes S, the total degree of S in the hypergraph is approximately proportional to the influence of S in the original graph. We can therefore run a standard greedy algorithm on this hypergraph to return a set of size k of approximately maximal total degree.

To make this approach work one needs to overcome several inherent difficulties. First, we show that the marginal influence of a node v is proportional to the probability that v is influenced by a randomly chosen node u in the transpose graph. These probabilities can be estimated more efficiently than influence itself. In particular, this transpose-graph formulation simplifies the process of estimating *marginal* influence, so that we need not repeat the estimation procedure when considering different partial solutions. This results in substantial savings in runtime.

The next difficulty to overcome is the stringent runtime constraint — we must construct our hypergraph in time $O((m+n)\epsilon^{-3}\log(n))$. We show that this number of steps suffices to approximate the influence of each set of nodes in the graph. This approximation comes, for each set, as a probabilistic guarantee with confidence $1 - 1/\text{POLY}(n)$. Finally, in order to prevent errors from accumulating when applying the greedy algorithm to the hypergraph, it is important that our estimator for influence (i.e. total hypergraph degree) is itself a monotone submodular function.

Second Result: A Sublinear Time Local Algorithm We extend our approximation algorithm to allow a provable tradeoff between runtime and approximation quality. Given a network $\mathcal{G}$ with arboricity[9] $a(\mathcal{G})$ and a parameter $\beta \in (1, n]$, our algorithm attains approximation $\Theta(\frac{1}{\beta})$ in time $O(\frac{n\log^3(n)\cdot a(\mathcal{G})}{\beta})$. To the best of our knowledge, ours is the first algorithm with such a tradeoff between runtime and approximation quality.

In particular, on networks of bounded arboricity, our algorithm finds a node of approximately maximal influence in sublinear time when $\beta = \omega(log^3(n))$. We note that many rich classes of graphs have bounded arboricity, including planar graphs, graphs with small maximum in-degree or maximum out-degree, other graphs with bounded treewidth (as the treewidth is at most twice the arboricity), many models of network formation and empirically observed social and technological networks [26, 49, 82].

We provide a lower bound of $\Omega((m+n)/\beta \cdot \min\{\beta, k\})$ on the runtime needed to obtain an $O(\beta)$-approximation. Thus, for networks with bounded arboricity, our algorithm is essentially runtime-optimal (up to logarithmic factors) given any fixed seed size k. Our method is randomized, and it succeeds with probability $3/5$; moreover, we show that this probability can be amplified through repetition.

The intuition behind our modified algorithm is that a tradeoff between execution time and approximation factor can be achieved by constructing fewer edges in our hypergraph representation. Given an upper bound on runtime, we can build edges until that time has

[9]The arboricity of a network is the minimum number of spanning forests needed to cover all edges [97].

expired, then run the influence maximization algorithm using the resulting (impoverished) hypergraph. We show that this approach generates a solution whose quality degrades gracefully with the preprocessing time, with an important caveat. If there are many individuals with high influence, it may be that a reduction in runtime prevents us from achieving enough concentration to estimate the influence of any node. If so, the highest-degree node(s) in the constructed hypergraph will not necessarily be the nodes of highest influence in the original graph. However, in this case, the fact that many individuals have high influence enables an alternative approach: a node chosen at random, according to the degree distribution of nodes in the hypergraph representation, will have high influence with high probability.

Given the above, our algorithm will proceed by constructing two possible seed sets: one using the greedy algorithm applied to the constructed hypergraph, and the other is a singleton selected at random according to the hypergraph degree distribution. To decide between the two, we design a procedure for efficienctly estimating the influence of a given set, up to a maximum of n/β. We then return the set with higher tested influence.

Our test for estimating influence proceeds by repeatedly constructing depth-first trees of various sizes, rooted at the nodes to be tested. This is done in a careful way in order to keep the overall runtime small, which depends on the density of the densest subgraph in the network: if the nodes to be tested lie in a particularly dense region of the graph, extra time may be required to build partial spanning trees, even if the graph is sparse on average. This dependency is what motivates the arboricity term in the approximation factor of the algorithm.

Finally, we emphasize that our solution concept is to return a set with high expected influence, with respect to the influence diffusion process, with probability greater than $1/2$ over randomness in the approximation algorithm. A potential relaxation would be to develop an algorithm that returns a set with high expected influence, where the expectation is with respect to both the diffusion process and the randomness in the approximation algorithm. For this weaker notion of approximability, the final testing phase of the algorithm described above is unnecessary: we could simply return one of our two potential solutions at random. This modified algorithm would have a runtime of $O(\frac{(n+m)\log^3(n)}{\beta})$, dropping the dependence on arboricity. As the lower bound of $\Omega((m+n)/\beta \cdot \min\{\beta, k\})$ holds even for this relaxed solution concept, this algorithm is nearly optimal (up to logarithmic factors) for arbitrary networks, given a fixed k. While we feel that the original (stronger) approximation notion is more relevant, this alternative formulation may be of interest in cases where variability in solution quality can be tolerated.

Related Work

Models of influence spread in networks, covering both cascade and threshold phenomena, are well-studied in the sociology and marketing literature [50, 56, 105]. The problem of finding the most influential set of nodes to target for a diffusive process was first posed by Domingos and Richardson [38, 104]. A formal development of the IC model, along with a greedy algorithm based upon submodular maximization, was given by Kempe et al. [71]. Many subsequent works have studied the nature of diffusion in online social networks, using empirical data to estimate influence probabilities and infer network topology; see [54, 84, 85].

It has been shown that many alternative formulations of the influence maximization problem are computationally difficult. The problem of finding, in a *linear threshold* model, a set of minimal size that influences the entire network was shown to be inapproximable within $O(n^{1-\epsilon})$ by Chen [30]. The problem of determining influence spread given a seed set in the IC model is #P-hard [31].

There has been a large line of work aimed at improving the runtime of the algorithm by Kempe et al. [71]. These have focused mainly on heuristics for special settings, such as assuming that all nodes have relatively low influence or that the input graph is clustered [31, 33, 74, 118], as well as empirically-motivated implementation improvements [32, 83].

One particular approach of note involves first attempting to sparsify the input graph, then estimating influence on the reduced network [33, 90]. However, these sparsification problems are shown to be computationally intractible in general.

Various alternative formulations of influence spread as a submodular process have been proposed and analyzed in the literature [72, 93], including those that include iterations between multiple diffusive processes [16, 55]. We focus specifically on the IC model, and leave open the question of whether our methods can be extended to apply to these alternative models.

The influence estimation problem shares some commonality with the problems of local graph partitioning, as well as estimating pagerank and personalized pagerank vectors [3, 4, 111]. These problems admit local algorithms based on sampling short random walks. However, these methods do not seem directly applicable to influence maximization due to the inherently non-local nature of influence cascades. To elaborate on this, consider a large rooted d-regular tree in which all edges point away from the root towards the leaves. The root of such tree will influence most of the network if all edges have weight at least $1/d$, but this is not apparent from any local neighborhood around the root. We should therefore not expect to be able to efficiently estimate influence via short random walks.

6.2 Model and Preliminaries

The Independent Cascade Model In the independent cascade (IC) model, influence spreads via an edge-weighted directed graph $\mathcal{G}$. An infection begins at a set S of seed nodes, and spreads through the network in rounds. Each infected node v has a single chance, upon first becoming infected, of subsequently infecting his neighbors. Each directed edge $e = (v, u)$ has a weight $p_e \in [0, 1]$ representing the probability that the process spreads along edge e to node u in the round following the round in which v was first infected.

As noted in [71], the above process has the following equivalent description. We can interpret $\mathcal{G}$ as a distribution over unweighted directed graphs, where each edge e is independently realized with probability p_e. If we realize a graph g according to this probability distribution, then we can associate the set of infected nodes in the original process with the set of nodes reachable from seed set S in g. We will make use of this alternative formulation of the IC model throughout this chapter.

Notation We let m and n denote the number of edges and nodes, respectively, in the weighted directed graph $\mathcal{G}$. We write $g \sim \mathcal{G}$ to mean that g is drawn from the random graph distribution $\mathcal{G}$. Given set S of vertices and (unweighted) directed graph g, write $C_g(S)$ for the set of nodes reachable from S in g. When g is drawn from $\mathcal{G}$, we will refer to this as the set of nodes influenced by S. We write $I_g(S) = |C_g(S)|$ for the number of nodes influenced by S, which we call the influence of S in g. We write $\mathbb{E}_{\mathcal{G}}[I(S)] = \mathbb{E}_{g\sim\mathcal{G}}[I_g(S)]$ for the expected influence of S in $\mathcal{G}$.

Given two sets of nodes S and W, we write $C_g(S|W)$ for the set of nodes reachable from S but not from W. That is, $C_g(S|W) = C_g(S) \setminus C_g(W)$. As before, we write $I_g(S|W) = |C_g(S|W)|$; we refer to this as the marginal influence of S given W. The expected marginal influence of S given W is $\mathbb{E}_{\mathcal{G}}[I(S|W)] = \mathbb{E}_{g\sim\mathcal{G}}[I_g(S|W)]$.

In general, a vertex in the subscript of an expectation or probability denotes the vertex being selected uniformly at random from the set of vertices of $\mathcal{G}$. For example, $\mathbb{E}_{v,\mathcal{G}}[I(v)]$ is the average, over all graph nodes v, of the expected influence of v.

For a given graph g, define g^T to be the *transpose graph* of g: $(u,v) \in g$ iff $(v,u) \in g^T$. We apply this notation to both weighted and unweighted graphs.

The Influence Maximization Problem Given graph $\mathcal{G}$ and integer $k \geq 1$, the influence maximization problem is to find a set S of at most k nodes maximizing the value of $\mathbb{E}_{\mathcal{G}}[I(S)]$. For $\beta > 1$, we say that a particular set of nodes T with $|T| \leq k$ is a $\frac{1}{\beta}$-approximation to the influence maximization problem if $\mathbb{E}_{\mathcal{G}}[I(T)] \geq \frac{\max_{S:|S|=k} \mathbb{E}_{\mathcal{G}}[I(S)]}{\beta}$.

We assume that graph $\mathcal{G}$ is provided in adjacency list format, with the neighbors of a given vertex v ordered arbitrarily.

A Simulation Primitive Our algorithms we will make use of a primitive that realizes an instance of the nodes influenced by a given vertex u in weighted graph $\mathcal{G}$, and returns this set of nodes. Conceptually, this is done by realizing some $g \sim \mathcal{G}$ and traversing $C_g(u)$.

Let us briefly discuss the implementation of such a primitive. Given node u, we can run a depth first search in $\mathcal{G}$ starting at node u. Before traversing any given edge e, we perform a random test: with probability p_e we traverse the edge as normal, and with probability $1 - p_e$ we do not traverse edge e and ignore it from that point onward. The set of nodes traversed in this manner is equivalent to $C_g(u)$ for $g \sim \mathcal{G}$, due to deferred randomness. We then return the set of nodes traversed. The runtime of this procedure is precisely the sum of the degrees (in $\mathcal{G}$) of the vertices in $C_g(u)$.

We can implement this procedure for a traversal of g^T, rather than g, by following in-links rather than out-links in our tree traversal. Also, we will sometimes wish to run this procedure with an upper bound on the number of nodes to traverse; in this case we simply abort the depth-first traversal when the bound has been reached and return the set of nodes explored up to that point.

6.3 A Local Approximation Algorithm for Influence Maximization

In this section we present a local algorithm for the influence maximization problem on arbitrary directed graphs. Our algorithm returns a $(1 - \frac{1}{e} - \epsilon)$-approximation to the influence maximization problem, with success probability $3/5$, in time $O((m+n)\epsilon^{-3}\log n)$. We note that this algorithm is a simplification of a more general version that permits a tradeoff between runtime and approximation, which appears in Section 6.4.

The algorithm is described formally as Algorithm 6, but let us begin by describing our construction informally. Our approach proceeds in two steps. The first step, BuildHypergraph, stochastically generates a sparse, undirected hypergraph representation $\mathcal{H}$ of our underlying graph g. This is done by repeatedly simulating the influence spread process on the transpose of the input graph, g^T. This simulation process is performed as described in Section 6.2: we begin at a random node u and proceed via depth-first search, where each encountered edge e is traversed independently with probability p_e. The set of nodes encountered becomes an edge in $\mathcal{H}$. We then repeat this process, generating multiple hyperedges. The BuildHypergraph subroutine takes as input a bound R on its runtime; we continue building edges until a total of R steps has been taken by the simulation process. (Note that the number of steps taken by the process is equal to the number of edges considered by the depth-first search process). Once R steps have been taken in total over all simulations, we return the resulting hypergraph.

In the second step, BuildSeedSet, we use our hypergraph representation to construct our output set. This is done by repeatedly choosing the node with highest degree in $\mathcal{H}$, then removing that node and all incident edges from $\mathcal{H}$. The resulting set of k nodes is the generated seed set.

While Algorithm 6 is relatively simple to describe, it is not obvious at all why such an algorithm should work well under its imposed, stringent time constraints. We now turn to provide a detailed analysis of Algorithm 6. Fix k and a weighted directed graph $\mathcal{G}$. Let $\mathcal{OPT} = \max_{S:|S|=k}\{\mathbb{E}_{\mathcal{G}}[I(S)]\}$, the maximum expected influence of a set of k nodes.

Our goal is to bound the approximation factor of the set returned by Algorithm 6.

Theorem 6.1. *Fix $\epsilon > 0$. Algorithm 6 returns a set S with $\mathbb{E}_{\mathcal{G}}[I(S)] \geq (1 - \frac{1}{e} - \epsilon)\mathcal{OPT}$, with probability at least $3/5$, and runs in time $O(\frac{(m+n)\log(n)}{\epsilon^3})$.*

To prove Theorem 6.1, we first observe that the influence of a set of nodes S is precisely n times the probability that a randomly selected node u influences any node from S in the transpose graph g^T.

Observation 6.2. *For each subset of nodes $S \subseteq \mathcal{G}$,*

$$\mathbb{E}_{g\sim\mathcal{G}}[I_g(S)] = n\mathrm{Pr}_{u,g\sim\mathcal{G}}[S \cap C_{g^T}(u) \neq \emptyset].$$

Proof.

$$\begin{aligned}
\mathbb{E}_{g\sim\mathcal{G}}[I_g(S)] &= \sum_{u\in g} \mathrm{Pr}_{g\sim\mathcal{G}}[\exists v \in S \text{ such that } u \in C_g(v)] \\
&= \sum_{u\in g} \mathrm{Pr}_{g\sim\mathcal{G}}[\exists v \in S \text{ such that } v \in C_{g^T}(u)] \\
&= n\mathrm{Pr}_{u,g\sim\mathcal{G}}[\exists v \in S \text{ such that } v \in C_{g^T}(u)] \\
&= n\mathrm{Pr}_{u,g\sim\mathcal{G}}[S \cap C_{g^T}(u) \neq \emptyset].
\end{aligned}$$

□

Algorithm 6 Maximize Influence

```
Require: Precision parameter $\epsilon \in (0,1)$, directed edge-weighted graph $\mathcal{G}$.
1: $R \leftarrow 72(m+n)\epsilon^{-3}\log(n)$
2: $\mathcal{H} \leftarrow$ BuildHypergraph$(R)$
3: return BuildSeedSet$(\mathcal{H},k)$

BuildHypergraph$(R)$:
1: Initialize $\mathcal{H} = (V,\emptyset)$.
2: repeat
3:     Choose node $u$ from $\mathcal{G}$ uniformly at random.
4:     Simulate influence spread, starting from $u$, in $\mathcal{G}^T$. Let $T$ be the set of nodes discovered.
5:     Add $T$ to the edge set of $\mathcal{H}$.
6: until $R$ steps have been taken in total by the simulation process.
7: return $\mathcal{H}$

BuildSeedSet$(\mathcal{H},k)$:
1: for $i = 1,\ldots,k$ do
2:     $v_i \leftarrow \text{argmax}_v\{deg_{\mathcal{H}}(v)\}$
3:     Remove $v_i$ and all incident edges from $\mathcal{H}$
4: end for
5: return $\{v_1,\ldots,v_k\}$
```

Observation 6.2 implies that we can estimate $\mathbb{E}_{\mathcal{G}}[I(S)]$ by estimating the probability of the event $S \cap C_{g^T}(u) \neq \emptyset$. The degree of a node v in $\mathcal{H}$ is precisely the number of times we observed that v was influenced by a randomly selected node u. We can therefore think of $\mathcal{H}$ as encoding an approximation to the influence function in graph $\mathcal{G}$.

We now show that the algorithm takes enough samples to accurately estimate the influences of the nodes in the network. This requires two steps. First, we show that runtime $R = 72(m+n)\epsilon^{-3}\log(n)$ is enough to build a sufficiently rich hypergraph structure, with high probability over the random outcomes of the influence cascade model.

Lemma 6.3. *Hypergraph $\mathcal{H}$ will contain at least $\frac{24n\log(n)}{\mathcal{OPT}\epsilon^3}$ edges, with probability at least $\frac{2}{3}$.*

Proof. Given a vertex u and an edge $e = (v,w)$, consider the random event indicating whether edge e is checked as part of the process of growing a depth-first search rooted at u in the IC process corresponding to graph $g^T \sim \mathcal{G}^T$. Note that edge e is checked if and only if node v is influenced by node u in this invocation of the IC process. In other words, edge $e = (v,w)$ is checked as part of the influence spread process on line 4 of BuildHypergraph if and only if $v \in T$. Write $m_{g^T}(u)$ for the random variable indicating the number of edges that are checked as part of building the influence set T starting at node u in g^T.

Let $X = \frac{24n\log(n)}{OPT\epsilon^3}$ for notational convenience. Consider the first (up to) X iterations of the loop on lines 2-6 of BuildHypergraph. Note that $\mathcal{H}$ will have at least X edges if the total runtime of the first X iterations is at most R. The expected runtime of the algorithm over these iterations is

$$\begin{aligned}
X \cdot \mathbb{E}_{u,g\sim\mathcal{G}}[1 + m_{g^T}(u)] &= X + \frac{X}{n}\mathbb{E}_{g\sim\mathcal{G}}\left[\sum_u m_{g^T}(u)\right] \\
&= X + \frac{24\log(n)}{OPT\epsilon^3}\mathbb{E}_{g\sim\mathcal{G}}\left[\sum_u m_{g^T}(u)\right] \\
&= X + \frac{24\log(n)}{OPT\epsilon^3}\sum_{e=(v,w)\in\mathcal{G}^T}\mathbb{E}_{g\sim\mathcal{G}}\left[|\{u : v \in C_{g^T}(u)\}|\right] \\
&= X + \frac{24\log(n)}{OPT\epsilon^3}\sum_{e=(v,w)\in\mathcal{G}^T}\mathbb{E}_{g\sim\mathcal{G}}\left[|\{u : u \in C_g(v)\}|\right] \\
&\le \frac{24n\log(n)}{\epsilon^3} + \frac{24\log(n)}{OPT\epsilon^3}\sum_{e=(v,w)\in\mathcal{G}^T} OPT \\
&= \frac{24(m+n)\log(n)}{\epsilon^3}.
\end{aligned}$$

Here , the second equality (line 4 from above) follows by noting that an edge $(v,w) \in \mathcal{G}^T$ is traversed as part of $m_{g^T}(u)$ if and only if v appears in $C_{g^T}(u)$.

Thus, by the Markov inequality, the probability that the runtime over the first X iterations is greater than $R = 72(m+n)\epsilon^{-3}\log(n)$ is at most $\frac{1}{3}$. The probability that at least X edges are present in hypergraph $\mathcal{H}$ is therefore at least $\frac{2}{3}$, as required. $\square$

Next, we show that the resulting hypergraph is of sufficient size to estimate the influence of each set, up to an additive error that shrinks with ϵ, with probability $1 - 1/\text{POLY}(n)$. We then show that such estimation guarantees suffice to find good approximations to the influence maximization problem.

Write $m(\mathcal{H})$ and $deg_{\mathcal{H}}(S)$ for the number of edges of $\mathcal{H}$ and the number of edges from $\mathcal{H}$ incident with a node from S, respectively. An important subtlety is that the value of $m(\mathcal{H})$ is a random variable, determined by the stopping condition of BuildHypergraph. We must be careful to bound the effect on our influence estimation. We show that the value of $m(\mathcal{H})$ is sufficiently concentrated that the resulting bias is insignificant.

Lemma 6.4. *Suppose that $m(\mathcal{H}) \ge \frac{24n\log(n)}{\mathcal{OPT}\epsilon^3}$. Then, for any set of nodes $S \subseteq V$,*

$$\Pr[|\mathbb{E}_{\mathcal{G}}[I(S)] - \frac{n}{m(\mathcal{H})}deg_{\mathcal{H}}(S)| > \epsilon\mathcal{OPT}] < \frac{1}{n^3},$$

with probability taken over randomness in $\mathcal{H}$.

Proof. By assumption we have that $m(\mathcal{H}) \geq 24n\log(n)/(\mathcal{OPT}\epsilon^3)$, and we also know $m(\mathcal{H}) \leq R = 72(m+n)\log(n)/\epsilon^3$ (since each edge has size at least 1). Fix some arbitrary

$$J \in [\frac{24n\log(n)}{\mathcal{OPT}\epsilon^3}, \frac{72(m+n)\log(n)}{\epsilon^3}].$$

We will study the probability that

$$|\mathbb{E}_{\mathcal{G}}[I(S)] - \frac{n}{m(\mathcal{H})} deg_{\mathcal{H}}(S)| > \epsilon\mathcal{OPT}$$

conditional on $m(\mathcal{H}) = J$.

Suppose first that $\mathbb{E}_{\mathcal{G}}[I(S)] \geq \epsilon OPT$. Let D_S denote the degree of S in $\mathcal{H}$, for notational convenience. Thinking of D_S as a random variable, we have that D_S is the sum of $m(\mathcal{H}) = J$ identically distributed Bernoulli random variables each with probability $\mathbb{E}_{\mathcal{G}}[I(S)]/n \geq \epsilon OPT/n$, by Observation 6.2.
In particular, since $m(\mathcal{H}) = J \geq 24n\log(n)/(OPT\epsilon^3)$,

$$\mathbb{E}[D_S] \geq \frac{\epsilon OPT}{n} m(\mathcal{H}) \geq 24\epsilon^{-2}\log n.$$

The Multiplicative Chernoff bound (A.1) then implies that

$$\Pr\left[D_S < (1-\epsilon)\frac{m(\mathcal{H})}{n}\mathbb{E}_{\mathcal{G}}[I(S)]\right] \leq \Pr[D_S < (1-\epsilon)\mathbb{E}[D_S]]$$
$$< e^{-\mathbb{E}[D_S]\epsilon^2/2} < e^{-12\log(n)} = \frac{1}{n^{12}}.$$

Similarly, we can use the Multiplicative Chernoff bound (A.1) to conclude that

$$\Pr\left[D_S > (1+\epsilon)\frac{m(\mathcal{H})}{n}\mathbb{E}_{\mathcal{G}}[I(S)]\right] < e^{-\mathbb{E}[D_S]\epsilon^2/2}$$
$$\leq e^{-12\log(n)} = \frac{1}{n^{12}}.$$

Next suppose that $\mathbb{E}_{\mathcal{G}}[I(S)] < \epsilon OPT$, so that $\mathbb{E}[D_S] < \epsilon \cdot OPT \cdot m(\mathcal{H})/n$. In this case, the Multiplicative Chernoff bound (A.1) implies that

$$\begin{aligned}
&\Pr\left[D_S > \mathbb{E}[D_S] + \frac{\epsilon OPT}{n} m(\mathcal{H})\right] \\
&= \Pr\left[D_S > \mathbb{E}[D_S]\left(1 + \frac{\epsilon OPT}{n}\frac{m(\mathcal{H})}{\mathbb{E}[D_S]}\right)\right] \\
&< e^{-\frac{\epsilon OPT}{n} m(\mathcal{H})/4} \\
&\le e^{-6\log(n)} \le \frac{1}{n^6}.
\end{aligned}$$

Thus, in all cases, the probability that the event of interest occurs is at most $\frac{1}{n^6}$. Since we conditioned on $m(\mathcal{H}) = J$, and since there are no more than $72(m+n)\log(n)/\epsilon^3 = o(n^3)$ potential values of J, the law of total probability implies that the unconditional probability that $|\mathbb{E}_{\mathcal{G}}[I(S)] - \frac{n}{m(\mathcal{H})} deg_{\mathcal{H}}(S)| > \epsilon \mathcal{OPT}$, is at most $\frac{1}{n^3}$ as required. □

Finally, we must show that the greedy algorithm applied to $\mathcal{H}$ in BuildSeedSet returns a good approximation to the original optimization problem. Recall that, in general, the greedy algorithm for submodular function maximization proceeds by repeatedly selecting the singleton with maximal contribution to the function value, up to the cardinality constraint. The following lemma shows that if one submodular function is approximated sufficiently well by a distribution of submodular functions, then applying the greedy algorithm to a function drawn from the distribution yields a good approximation with respect to the original.

Lemma 6.5. *Choose $\delta > 0$ and suppose that $f \colon 2^V \to \mathbb{R}_{\ge 0}$ is a non-decreasing submodular function. Let D be a distribution over non-decreasing submodular functions with the property that, for all sets S with $|S| \le k$, $\Pr_{\hat{f}\sim D}[|f(S) - \hat{f}(S)| > \delta] < 1/n^3$. If we write $S_{\hat{f}}$ for the set returned by the greedy algorithm on input $\hat{f}$, then*

$$\Pr_{\hat{f}\sim D}\left[f(S_{\hat{f}}) < (1 - 1/e)\left(\max_{S\colon |S|=k} f(S)\right) - 2\delta\right] < 1/n.$$

Proof. Choose $S^* \in \operatorname{argmax}_{|S|=k}\{f(S)\}$. With probability at least $1 - 1/n^3$, $\hat{f}(S^*) \ge f(S^*) - \delta$. So, in particular, $\max_{|S|=k}\hat{f}(S) \ge f(S^*) - \delta$.

We run the greedy algorithm on function $\hat{f}$; let S_i be the set of nodes selected up to and including iteration i (with $S_0 = \emptyset$). On iteration i, we consider each set of the form $S_{i-1} \cup \{x\}$ where x is a singleton. There are at most n of these sets, and hence the union bound implies that f and $\hat{f}$ differ by at most δ on each of these sets, with probability at least $1 - 1/n^2$. In particular, $|f(S_i) - \hat{f}(S_i)| < \delta$. Taking the union bound over all iterations, we have that $|f(S_k) - \hat{f}(S_k)| < \delta$ with probability at least $1 - 1/n$. We therefore have

$$f(S_k) \geq \hat{f}(S_k) - \delta \geq (1 - 1/e) \max_S \hat{f}(S) - \delta \geq (1 - 1/e) f(S^*) - 2\delta$$

conditioning on an event of probability $1 - 1/n$. □

We are now ready to complete our proof of Theorem 6.1.

Proof of Theorem 6.1. Lemma 6.3 and 6.4 together imply that, conditioning on an event of probability at least $3/5$, we will have

$$\Pr\left[\left|\mathbb{E}_{\mathcal{G}}[I(S)] - \frac{n \cdot deg_{\mathcal{H}}(S)}{m(\mathcal{H})}\right| > \epsilon \mathcal{OPT}\right] < \frac{1}{n^3}$$

for each $S \subseteq V$. We then apply Lemma 6.5 with $f(S) := \mathbb{E}_{\mathcal{G}}[I(S)]$, $\hat{f}(S) := \frac{n \cdot deg_{\mathcal{H}}(S)}{m(\mathcal{H})}$ (drawn from distribution corresponding to distribution of $\mathcal{H}$ returned by BuildHypergraph), and $\delta = \epsilon \mathcal{OPT}$. Lemma 6.5 implies that, with probability at least $1 - \frac{1}{n}$, the greedy algorithm applied to $\mathcal{H}$ returns a set S with $\mathbb{E}_{\mathcal{G}[I(S)]} \geq (1 - 1/e)\mathcal{OPT} - 2\epsilon\mathcal{OPT} = (1 - 1/e - 2\epsilon)\mathcal{OPT}$. Noting that this is precisely the set returned by BuildSeedSet gives the desired bound on the approximation factor (rescaling ϵ by a factor of 2). Thus the claim holds with probability at least $2/3 - 1/n \geq 3/5$ (for $n \geq 20$).

Finally, we argue that our algorithm can be implemented in the appropriate runtime. The fact that BuildHypergraph executes in the required time follows from the explicit bound on its runtime. For BuildSeedSet, we will maintain a list of vertices sorted by their degree in $\mathcal{H}$; this will allow us to repeatedly select the maximum-degree node in constant time. The initial sort takes time $O(n \log n)$. We must bound the time needed to remove an edge from $\mathcal{H}$ and correspondingly update the sorted list. We will implement the sorted list as a doubly linked list of groups of vertices, where each group itself is implemented as a doubly linked list containing all vertices of a given degree (with only non-empty groups present). Each edge of $\mathcal{H}$ will maintain a list of pointers to its vertices. When an edge is removed, the degree of each vertex in the edge decreases by 1; we modify the list by shifting any decremented vertex to the preceding group (creating new groups and removing empty groups as necessary). Removing an edge from $\mathcal{H}$ and updating the sorted list therefore takes time proportional to the size of the edge. Since each edge in $\mathcal{H}$ can be removed at most once over all iterations of BuildSeedSet, the total runtime is at most the sum of node degrees in $\mathcal{H}$, which is at most $R = O((m+n)\epsilon^{-3}\log(n))$. □

6.3.1 Amplifying the Success Probability

Algorithm 6 returns a set of influence at least $(1 - \frac{1}{e} - \epsilon)$ with probability at least $3/5$. The failure probability is due to Lemma 6.3: hypergraph $\mathcal{H}$ may not have sufficiently many edges after R steps have been taken by the simulation process in line 4 of the BuildHypergraph subprocedure. However, note that this failure condition is detectable via repetition: we can repeat Algorithm 6 multiple times, and use only the iteration that generates the most edges. The success rate can then be improved by repeated invocation, up to a maximum of $1 - 1/n$ with $\log(n)$ repetitions (at which point the error probability due to Lemma 6.4 becomes dominant).

We next note that, for any $\ell > 1$, the error bound in Lemma 6.4 can be improved to $\frac{1}{n^\ell}$, by increasing the value of R by a factor of ℓ, since this error derives from Chernoff bounds. This would allow the success rate of the algorithm to be improved up to a maximum of $1 - \frac{1}{n^\ell}$ by further repeated invocation. To summarize, the error rate of the algorithm can be improved to $1 - \frac{1}{n^\ell}$ for any ℓ, at the cost of increasing the runtime of the algorithm by a factor of $\ell^2 \log(n)$.

6.4 Approximation in Sublinear Time

In this section we describe a modified local algorithm that provides a tradeoff between runtime and approximation quality. For an an arbitrary $\beta > 1$, our algorithm will obtain a $\Theta(1/\beta)$-approximation to the influence maximization problem, in time $O(\frac{n \cdot a(\mathcal{G}) \log^3(n)}{\beta})$, where $a(\mathcal{G})$ is the arboricity of graph $\mathcal{G}$, with probability at least $3/5$. The success rate can be improved by standard amplification techniques; we discuss this in Section 6.4.1.

Our algorithm is listed as Algorithm 7 below. The intuition behind our construction is as follows. We wish to find a set of nodes with high expected influence. One approach would be to apply Algorithm 6 and simply impose a tighter constraint on the amount of time that can be used constructing our hypergraph representation. This might correspond to reducing the value of parameter R by, say, a factor of β. Unfortunately, the precision of our sampling method does not always degrade gracefully with β: if β is sufficiently large, we may not have enough data to guess at a maximum-influence node (even if we allow ourselves a factor of β in the approximation ratio). In these cases, the sampling approach fails to provide a good approximation.

However, as we will show, our sampling fails precisely because many of the edges in our hypergraph construction were large, and (with constant probability) this can occur only if many of the nodes that make up those edges have high influence. In this case, we could proceed by selecting a node from the hypergraph at random, with probability proportional to its hypergraph degree. We prove that this procedure is likely to return a node of very high influence precisely in settings where the original approach is not.

However, for this to work we need to figure out which of the two approaches will succeed (since, in particular, the required bound on the hypergraph size is a function of $\mathcal{OPT}$, which is unknown). We therefore apply both methods, then directly estimate the influence of each solution. We must do so carefully in order to keep our runtime small. To this end, the TestInfluence procedure generates a rough sketch of the probability distribution over influence values by repeatedly growing depth-first trees of various sizes. By balancing tree sizes with sampling precision, we are able to give an estimate of influence up to a maximum of n/β. We then return whichever solution has the greater estimated influence.

We note that, since our final estimation step proceeds by repeatedly exploring subgraphs of an input graph $\mathcal{G}$, its runtime will be tied to the *arboricity* of $\mathcal{G}$. Indeed, the arboricity is used to explicitly bound the time needed to construct a depth-first tree of a given size.

Definition 7. *The arboricity number of an undirected graph $\mathcal{G}$, denoted as $a(\mathcal{G})$, is the minimum number of spanning forests needed to cover all the edges of the graph. The arboricity of a directed, weighted graph is the arboricity of the undirected, unweighted version of the graph.*

Theorem 6.6. *For any $\beta > 1$, Algorithm 7 returns, with probability of at least $3/5$, a node with expected influence at least min$\{\frac{1}{2}, \frac{1}{\beta}\} \cdot OPT$. Its runtime is $O(\frac{n \cdot a(\mathcal{G}) \log^3(n)}{\beta})$.*

Before proving Theorem 6.6, let us discuss its statement and some minor variations.

Discussion 1. *It will turn out that the runtime of Algorithm 7 is dominated by subprocedure TestInfluence: the time needed to estimate the influence of a given solution. Given access to an oracle that estimates the influence of a given set, up to a maximum of n/β, the runtime of Algorithm 7 becomes $O(\frac{(n+m)\log^3(n)}{\beta} + X(n,m,\beta))$, where $X(n,m,\beta)$ is a bound on the runtime of the oracle. We provide an implementation of such an oracle, based on growing spanning forests. Our analysis is given in terms of the arboricity of the underlying graph and shows that $X(n,m,\beta) \subseteq O(\frac{n \cdot a(\mathcal{G}) \log^3(n)}{\beta})$.*

Another important note is in regards to the probabilistic guarantee we require from our algorithm. Algorithm 7 returns a set that has high influence (in expectation over the random diffusion process), with probability at least $3/5$ over the random bits of the algorithm. The runtime of Algorithm 7 can be improved if our goal is relaxed to returning a set of high influence in expectation both over the random diffusion process and the randomness in the algorithm. For this relaxed solution concept, it becomes unnecessary to test the influence of the two potential solutions: the algorithm can simply choose between them at random. The expected influence of the set returned would then be at least $\mathcal{OPT}/2\beta$. The runtime of this modified algorithm is $O(\frac{(n+m)\log^3(n)}{\beta})$.

Algorithm 7 Influence Maximization with Tradeoff

```
Require: Approximation parameter $\beta > 1$, directed weighted graph $\mathcal{G}$.
1: $R \leftarrow \frac{(24\cdot 36)(n+m)\log^2(n)}{\beta}$
2: $\mathcal{H} \leftarrow$ BuildHypergraph$(R)$
3: $S \leftarrow$ BuildSeedSet$(\mathcal{H}, k)$
4: Choose $v \in V$ with probability proportional to degree in $\mathcal{H}$
5: if TestInfluence$(S) >$ TestInfluence$(v)$ then return $S$
6: else return $\{v\}$
```

TestInfluence(S)**:**

– τ: our guess at β times the influence of S.

– L: our guess at the realized influence size that contributes most to the expected influence.

```
1: for $\tau = n, n/2, n/4, \ldots, 1$ do
2:     for $L = n, n/2, n/4, \ldots, \tau$ do
3:         for $j = 1, \ldots, 32(L/\tau)\log(n)$ do
4:             Simulate influence in $\mathcal{G}$, starting from $S$, to a maximum of $L/\beta$ distinct nodes.
5:             Let $T_j$ be the set of nodes discovered.
6:             if $\sum_j |T_j| > 96(n/\beta)\log(n)$ then return $\tau/\beta$
7:         end for
8:         if at least $256\log(n)$ sets $T_j$ satisfy $|T_j| \geq L/\beta$ then return $\tau/\beta$
9:     end for
10: end for
11: return 1
```

Let us now turn to the proof of Theorem 6.6. Observe that when $\beta = O(\log n)$, the conditions of Theorem 6.6 are satisfied by Algorithm 6 with (say) $\epsilon = 1/6$ (to get an approximation factor bigger than $1/2$), since its runtime is $O((m+n)\log(n)) \subseteq O(\frac{n \cdot a(\mathcal{G})\log^3(n)}{\beta})$ (see Fact 6.10 in the appendix). We can therefore assume that $\beta > \log(n)$.

Our analysis proceeds via two cases, depending on whether $\mathcal{H}$ has sufficiently many edges as a function of $\mathcal{OPT}$. We first show that, subject to $\mathcal{H}$ having many edges, set S from line 3 is likely to have high influence. This follows the analysis from Theorem 6.1 almost exactly.

Lemma 6.7. *Suppose that* $m(\mathcal{H}) \geq \frac{24n\log(n)}{OPT}$*. Then, with probability at least* $1 - \frac{1}{n}$*, set* S *satisfies* $\mathbb{E}_{\mathcal{G}}[I(S)] \geq \frac{1}{2}OPT$*, with probability taken over randomness in* $\mathcal{H}$*.*

Proof. This follows by applying Lemma 6.4 with $\epsilon = \frac{1}{6}$, followed by the analysis of BuildSeedSet$(\mathcal{H}, k)$ from the proof of Theorem 6.1. □

Note that β does not appear explicitly in the statement of Lemma 6.7. The (implicit) role of β in Lemma 6.7 is that as β becomes large, Algorithm 7 uses fewer steps to construct hypergraph $\mathcal{H}$ and hence the condition of the lemma is less likely to be satisfied.

We next show that if $m(\mathcal{H})$ is small, then node v from line 4 is likely to have high influence. Recall our assumption that $\beta > \log(n)$.

Lemma 6.8. *Suppose that* $m(\mathcal{H}) < \frac{24n\log(n)}{OPT}$*. Then, with probability at least* $2/3$*, node* v *satisfies*
$\mathbb{E}_{\mathcal{G}}[I(v)] \geq \mathcal{OPT}\log(n)/\beta$*, with probability taken over randomness in* $\mathcal{H}$*.*

Proof. Let random variable X denote the number of times that a node with influence at most $\mathcal{OPT}\log(n)/\beta$ was added to a hyperedge of $\mathcal{H}$. Since $\mathcal{H}$ has fewer than $\frac{24n\log(n)}{\mathcal{OPT}}$ edges, the expected value of X is at most

$$\begin{aligned} E[X] &\leq \frac{24n\log(n)}{\mathcal{OPT}} \sum_{u \in V} \frac{1}{n} \min\{\mathbb{E}_{\mathcal{G}}[I(u)], \mathcal{OPT}\log(n)/\beta\} \\ &\leq \frac{24n\log^2(n)}{\beta}. \end{aligned}$$

Markov inequality then gives that $\Pr[X > \frac{(24\cdot 6)n\log^2(n)}{\beta}] < 1/6$. Conditioning on this event, we have that at most $\frac{(24\cdot 6)n\log^2(n)}{\beta}$ of the nodes touched by BuildHypergraph have influence less than $OPT\log(n)/\beta$. Since at least $\frac{(24\cdot 36)n\log^2(n)}{\beta}$ nodes were touched in total, the probability that node v from line 4 has influence less than $OPT\log(n)/\beta$ is at most $1/6$. The union bound then allows us to conclude that v has $\mathbb{E}[I(v)] \geq OPT/\beta$ with probability at least $1-(1/6+1/6) \geq 2/3$. □

Next, we show that procedure TestInfluence(S) returns a sufficiently good estimate of the influence of a given set of nodes. The proof, which is somewhat technical, is deferred to Section 6.6.1.

Lemma 6.9. *Given a set of nodes S procedure TestInfluence returns a value from the range* $[x/\log(n), x]$ *with probability at least* $1-\frac{1}{2n}$, *where* $x = \min\{\mathbb{E}_{\mathcal{G}}[I(S)], n/\beta\}$.

Finally, we show that the runtime of Algorithm 7 is at most $O(\frac{n\cdot a(\mathcal{G})\log^3(n)}{\beta})$. The following fact about arboricity will be particularly useful.

Fact 6.10 (Nash-Williams [97])**.** *Given any graph g and subgraph h of g (containing at least two nodes),* $\frac{m(h)}{n(h)-1} \leq a(g)$.

As noted by Nash-Williams this characterization of arboricity is tight as there is always one such h with $\lceil \frac{m(h)}{n(h)-1} \rceil = a(g)$. We are now ready to bound the runtime of Algorithm 7.

Lemma 6.11. *Algorithm 7 runs in time* $O(\frac{n\cdot a(\mathcal{G})\log^3(n)}{\beta})$.

Proof. Similarly to the analysis of Algorithm 6, lines 1-3 take at most $R = O(\frac{(n+m)\log^2(n)}{\beta})$. Fact 6.10 implies that this is $O(\frac{n\log^2(n)\cdot a(\mathcal{G})}{\beta})$.

We must now show that calling TestInfluence() costs at most $O(\frac{n\log^3(n)\cdot a(\mathcal{G})}{\beta})$ which completes the proof. On each choice of L and τ TestInfluence() grows several trees, where the sum over those tree sizes is at most $O(\frac{n}{\beta})\log n$. By the Nash-Williams theorem [97], each subgraph explored during this process has arboricity at most $a(\mathcal{G})$.

Thus the total time to build the trees (for a given τ and L) is most $O(\frac{n\cdot a(\mathcal{G})}{\beta}\log(n))$. To implement the test on line 7, for each value of L, one needs to keep track on the number of trees with size at least $\frac{L}{\beta}$. This can easily be done online by keeping a counter, updating the counter after each tree is realized. Since we also iterate over $\log(n)$ values for L and τ, the total runtime is $O(\frac{n\log^3(n)\cdot a(\mathcal{G})}{\beta})$. $\square$

We are now ready to complete the proof of Theorem 6.6.

Proof of Theorem 6.6. Lemma 6.7 and Lemma 6.8 imply that, with probability at least $2/3 - 1/n^2 \geq 3/5$ (for $n \geq 5$), one of S or $\{v\}$ has influence at least $OPT\log(n)/\beta$ (recalling that $\beta > \log n$). Since TestInfluence determines the influence of each set up to a potential under-valuation of a factor of $\log(n)$, the set for which TestInfluence returns the highest estimate must therefore have influence at least $\frac{\mathcal{OPT}\log(n)}{\beta} \cdot \frac{1}{\log(n)} = \mathcal{OPT}/\beta$. The required bound on the runtime of Algorithm 7 follows directly from Lemma 6.11. $\square$

6.4.1 Amplifying the Success Probability

Algorithm 7 returns, with probability $3/5$, a set that approximates the maximum expected influence over sets of size k in $\mathcal{G}$. We note that this returned set comes with an estimate of the optimal influence in the network; the "failure" conditions correspond to selecting this estimate incorrectly. To amplify success probability, we could return this estimate along with the set S. One could then call the algorithm multiple times; each successful invocation would generate an estimate at least OPT/β, whereas failed invocations would potentially generate smaller estimates. Amplification of success probability then corresponds to accepting whichever output is associated with the highest estimate. This would allow success probability to be raised up to $1 - 1/n$, at which point sampling error in Lemma 6.7 becomes the dominant error factor.

As in our original algorithm, the error bound in Lemma 6.7 can be improved to $1 - \frac{1}{n^\ell}$ by increasing the value of R by a factor of ℓ, since this error derives from Chernoff bounds. This would allow the success rate of the algorithm to be improved up to a maximum of $1 - \frac{1}{n^\ell}$ by further repeated invocation. To summarize, the error rate of the algorithm can be improved to $1 - \frac{1}{n^\ell}$ for any ℓ, at the cost of increasing the runtime of the algorithm by a factor of $\ell^2 \log(n)$.

6.4.2 A Lower Bound

We now turn to provide a lower bound on the time it takes an algorithm to compute a β-approximation for the maximum expected influence problem. In particular, for any given budget k, at least $\Omega(n/\beta)$ queries are required to obtain approximation factor $\frac{1}{\beta}$ with constant probability.

Theorem 6.12. *Let $0 < \epsilon < \frac{1}{10e}$, $\beta > 1$ be given. Any randomized algorithm for the maximum influence problem that has runtime of $\frac{m+n}{24\beta \min\{k,\beta\}}$ cannot return, with probability at least $1 - \frac{1}{e} - \epsilon$, a set of nodes with approximation ratio better than $\frac{1}{\beta}$.*

Proof. Note first that for a graph consisting of n singletons, an algorithm must return at least k/β nodes to obtain an approximation ratio of $\frac{1}{\beta}$. Doing so in at most $n/2\beta^2$ queries requires that $2k/\beta \leq n/\beta^2$, which implies $2k\beta \leq n$. We can therefore assume $2k\beta \leq n$ for the remainder of the proof.

The proof will invoke the application of Yao's Minimax Principle for the performance of Las Vegas randomized algorithms on a family of inputs [119]. The lemma states that the least expected cost of deterministic Las Vegas algorithms on a distribution over a family of inputs is a lower bound on the expected cost of the optimal randomized Las Vegas algorithm over that family of inputs.

We define the cost of the algorithm as 0 if it returns a set nodes with approximation ratio better than $\frac{1}{\beta}$ and 1 otherwise. Note that the cost of an algorithm equals its probability of failure and we can think of any Monte-Carlo randomized algorithm (with runtime at most $\frac{m+n}{24\beta \min\{k,\beta\}}$) as a Las Vegas one.

Assume for notational simplicity that β is an integer. We will build a family of lower bound graphs, one for each value of n (beginning from $n = \beta + 1$); each graph will have $m \leq n$, so it will suffice to demonstrate a lower bound of $\frac{n}{12\beta \min\{k,\beta\}}$.

We now consider the behavior of a deterministic algorithm A with respect to the uniform distribution on the constructed family of inputs. For a given value β the graph would

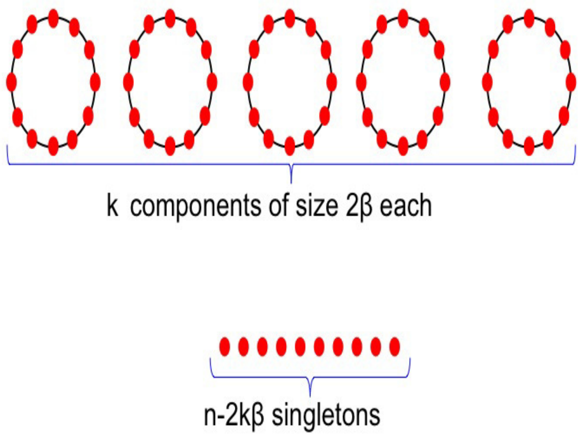

Figure 6.1: An illustration of the lower bound construction for approximate influence maximization.

be made from k components of size 2β and $n - 2k\beta$ singleton components (recall that $2k\beta \leq n$). See Figure 6.1 for an illustration of the construction.

If algorithm A returns nodes from ℓ of the k components of size 2β, it achieves a total influence of $2\ell\beta + (k - \ell)$. Thus, to attain approximation factor better than $\frac{1}{\beta}$, we must have $2\ell\beta + (k - \ell) \geq \frac{1}{\beta}2k\beta$, which implies $\ell \geq \frac{k}{2\beta - 1}$ for any $\beta > 1$.

Suppose $k > 12\beta$. The condition $\ell \geq \frac{k}{2\beta-1}$ implies that at least $\frac{k}{2\beta-1}$ of the large components must be queried by the algorithm, where each random query has probability $\frac{2k\beta}{n}$ of hitting a large component. If the algorithm makes fewer than $\frac{n}{12\beta^2}$ queries, then the expected number of components hit is $\frac{n}{12\beta^2} \cdot \frac{2k\beta}{n} = \frac{k}{6\beta}$. The Multiplicative chernoff bound (Lemma A.1, part 3) then imply that the probability hitting more than $\frac{k}{2\beta}$ components is no more than $e^{-\frac{k}{6\beta} \cdot 2/3} \leq \frac{1}{e^{4/3}} < 1 - \frac{1}{e} - \epsilon$, a contradiction.

If $k \leq 12\beta$ then we need that $\ell \geq 1$, which occurs only if the algorithm queries at least one of the $k\beta$ vertices in the large components. With $\frac{n}{2k\beta}$ queries, for n large enough, this happens with probability smaller than $\frac{1}{e} - \epsilon$, a contradiction.

We conclude that, in all cases, at least $\frac{n}{12\beta\min\{k,\beta\}}$ queries are necessary to obtain approximation factor better than $\frac{1}{\beta}$ with probability at least $1 - \frac{1}{e} - \epsilon$, as required.

By Yao's Minimax Principle this gives a lower bound of $\Omega(\frac{nd}{24\beta\min\{k,\beta\}})$ on the expected performance of any randomized algorithm, on at least one of the inputs.

Finally, we note that our construction can be modified to apply to non-sparse networks, as follows. For any $d \leq n$, we can augment our graph by overlaying a d-regular graph with exponentially small weight on each edge. This does not significantly impact the influence of any set of nodes, but it increases the time to determine whether a node is in a large component by a factor of $O(d)$ (as edges must be traversed until one with non-exponentially-small weight is found). Thus, for each $d \leq n$, we have a lower bound of $\frac{nd}{24\beta\min\{k,\beta\}}$ on th expected performance of A on a distribution of networks with $m = nd$ edges. $\square$

6.5 Discussion

We would like to end this chapter with a short discussion on the applicability of our results and techniques to other problems.

We would like to start by mentioning that although we have focused on the influence maximization problem under the independent cascades model, our results apply as they are to the probabilistic k-coverage problem as well. In the probabilistic k-coverage problem, each edge comes with an exposure probability, and the goal is to choose a set of k nodes such that its one-hop expected exposure (namely coverage) in maximized. One can clearly run our algorithms, growing diffusion trees only one hop away, namely stars. As now we grow stars instead of deep diffusion trees, the dependency of our results (for sublinear-time approximation) on the arboricity of the graph disappears.

We next turn to discuss how our results could potentially be used for addressing the influence maximization problem under other diffusion models. Our algorithms were based on designing a concise submodular function, the degree of a set in the hypergraph $\mathcal{H}$, such that the influence of a set S can be computed much faster from the concise submodular function representation. Looking back at the proof of theorem 6.1, only two ingredients need to hold in order to get a $(1 - 1/e - \epsilon)$-approximation in quasilinear time: the first, is that one can compute a distribution of concise submodular functions that approximate influence with weak probabilistic guarantee: only with probability at least $1 - n^3$, the concise-submodular family approximates well the influence of every set of size at most k. That is the guarantee of lemma 6.4. As we have shown, Lemma 6.5 can then be used for any such approximation to get good expected approximation! The second ingredient is to be able to compute, in at most quasilinear time, a maximizing set in the concise representation used (the hypergraph H in our results). As long as we use some type of graph as the concise-representation and are interested in node degrees in it, this could potentially be possible.

Unfortunately, it is not clear how to get the first ingredient (the equivalent of Lemma 6.4) from the methods we developed in this work. In particular, Lemma 6.4 heavily relies on key observation 6.2, which does not hold for non-independent cascading effects. Proving (or disproving) an equivalent to Lemma 6.4 for other diffusion models, as well as developing appropriate concise-representations of other diffusion processes is an interesting direction for future work.

6.6 Omitted Proofs

6.6.1 Proof of Lemma 6.9

First recall the statement of Lemma 6.9. We must show that, given a set of nodes S, with probability at least $1 - \frac{1}{2n}$, procedure TestInfluence returns a value from the range $[x/\log(n), x]$, where $x = \min\{\mathbb{E}_{\mathcal{G}}[I(S)], n/\beta\}$. The key to algorithm TestInfluence is to efficiently decide, for a given set S and potential influence value $\tau \leq n/\beta$, whether S has influence at least $\tau \log(n)$ or less than τ. We begin with two useful observations that will assist us in this task.

Lemma 6.13. *For any $v \in \mathcal{G}$, there exists some t, $1 \leq t \leq \log n$, such that $2^t \Pr_{g\sim\mathcal{G}}[I_g(v) \geq 2^t] \geq \mathbb{E}_{g\sim\mathcal{G}}[I_g(v)]/\log(n)$.*

Proof.

$$\begin{aligned}\mathbb{E}_g[I(v)] &= \sum_{y=1}^{n} \Pr_{g\sim\mathcal{G}}[I_g(v) \geq y] \\ &= \sum_{t=0}^{\log n} \sum_{y=2^t}^{2^{t+1}} \Pr_{g\sim\mathcal{G}}[I_g(v) \geq y] \\ &\leq \sum_{t=0}^{\log n} 2^t \Pr_{g\sim\mathcal{G}}[I_g(v) \geq 2^t]\end{aligned}$$

and hence $\mathbb{E}_{g\sim\mathcal{G}}[I_g(v)] \leq \log(n) \max_t\{2^t \Pr_{g\sim\mathcal{G}}[I_S(v) \geq 2^t]\}$, as required. □

Lemma 6.14. *For any $v \in \mathcal{G}$ and any $1 \leq t \leq \log n$, $2^t \Pr_{g\sim\mathcal{G}}[I_g(v) \geq 2^t] \leq \mathbb{E}_{g\sim\mathcal{G}}[I_g(v)]$.*

Proof.

$$\begin{aligned}\mathbb{E}_{g\sim\mathcal{G}}[I_g(v)] &= \sum_{y=1}^{n} y \Pr_{g\sim\mathcal{G}}[I_g(v) = y] \\ &\leq \sum_{y=2^t}^{n} 2^t \Pr_{g\sim\mathcal{G}}[I_g(v) = y] \\ &= 2^t \Pr_{g\sim\mathcal{G}}[I_g(v) \geq 2^t].\end{aligned}$$

□

Note that we think of L in algorithm TestInfluence(S) as a guess at the value of 2^t from the statement of Lemma 6.13. We are now ready to proceed with the proof of Lemma 6.9.

Consider a given iteration of TestInfluence, corresponding to a value of τ. We think of τ as a guess at the value of $\mathbb{E}_{\mathcal{G}}[I(S)]\beta$. We will first show that, with high probability, we do not return on an iteration in which $\mathbb{E}_{g\sim\mathcal{G}}[I(S)] < \tau/\beta$. We must consider the return conditions on line 6 and line 7.

Suppose $\mathbb{E}_{g\sim\mathcal{G}}[I_g(S)] < \tau/\beta$. Consider the return condition on line 7. For set S, $\Pr_{g\sim\mathcal{G}}[I_g(S) \geq L/\beta] < (\tau/L)$ for each choice of L, by Lemma 6.14. Thus, for any given choice of L, the probability that the event $[I_g(S) \geq L/\beta]$ occurs more than $256\log(n)$ times in $(L/\tau)32\log(n)$ trials is, by multiplicative Chernoff bound (A.1), at most $e^{-6(32\log(n))/3} = 1/n^{64}$. Taking the union bound over all values of L, we conclude that with probability at least $1 - 1/n^{63}$ we will not return on an iteration for which $\mathbb{E}_{g \sim \mathcal{G}}[I_g(S)] < \tau/\beta$ on line 7.

Next consider the condition on line 6. Suppose $\mathbb{E}_{g \sim \mathcal{G}}[I_g(S)] < \tau/\beta$, and pick some $L' \in \{n, n/2, \ldots, \tau\}$. As above, $\Pr_{g \sim \mathcal{G}}[I_g(S) \geq L'/\beta] < (\tau/L')$. Consider the sets of nodes built for node S on the iteration corresponding to some choice of L. If $L \leq L'$, then each set T_j will have size at most L'/β. For any $L > L'$, we know that by the multiplicative chernoff bound (A.1), the probability that event $[I_g(S) \geq L'/\beta]$ occurs more than $2(L/\tau)32\log(n) \cdot (\tau/L') = 64(L/L')\log(n)$ times is at most $e^{-32\log(n)/3} \leq 1/n^{10}$. Taking the union bound over all choices of L and L', we have that with probability at least $1 - 1/n^8$, for each L and $L' \leq L$, fewer than $2(L/L')32\log(n)$ trees built on iteration L have size greater than L'/β. Thus, conditioning on this event, we have that the sum of tree sizes is less than

$$\sum_{t=1}^{\log L} (2^t/\beta) 2(L/2^t) \log^2(n) \leq 2\frac{n}{\beta} 32 \log^3(n).$$

Taking the union bound over all values of L, we conclude that with probability at least $1 - 1/n^9$ we will not return on an iteration for which $\mathbb{E}_{g \sim \mathcal{G}}[I_g(S)] < \frac{\tau}{\beta}$, on line 6.

We will now show that, in an iteration in which $\tau \log(n)/\beta \leq \mathbb{E}_{\mathcal{G}}[I(S)]$, the algorithm returns with high probability. Suppose S has $\mathbb{E}_{g \sim \mathcal{G}}[I_g(S)] \geq (\tau/\beta)\log(n)$. Then, by Lemma 6.13, there exists some $L \geq \tau$, L a power of 2, such that $\Pr_{g \sim \mathcal{G}}[I_g(S) \geq L/\beta] \geq \frac{\tau}{2L}\log(n)$. The Multiplicative chernoff bound (A.1) implies that, during the $32(L/\tau)\log(n)$ trees built for set S, the probability that fewer than $\frac{1}{2}32\log(n) = 16\log(n)$ reach size L/β is at most $e^{-32\log(n)/8} = 1/n^4$. Thus with probability $1 - O(1/n^4)$, if our algorithm does not return on line 6, it will return on line 7, as required.

Chapter 7

Sublinear Time Local Algorithms for PageRank Computations

7.1 Introduction

A basic problem in network analysis is to identify the set of network nodes that are "significant." For example, they could be the significant Web pages that provide the authoritative contents in Web search; they could be the critical proteins in a protein interaction network; and they could be the set of people (in a social network) most effective to seed the influence for online advertising. As the networks become larger, we need more efficient algorithms to identify these "significant" nodes.

Identifying Nodes with Significant PageRanks

The meanings and measures of significant vertices depend on the semantics of the network and the applications. In this chapter, we focus on a particular measure of significance — the *PageRank* of the vertices.

Formally, the PageRank (with restart constant, also known as the teleportation constant, α) of a Web page is proportional to the probability that the page is visited by a random surfer who explores the Web using the following simple random walk: at each step, with probability $(1-\alpha)$ go to a random Web page linked to from the current page, and with probability α, restart the process from a uniformly chosen Web page. For the ease of presentation of our later results, we consider a normalization of PageRank so that the sum of the PageRank values over all vertices is equal to n, the number of vertices in the network,

$$\sum_{u \in V} \text{PageRank}(u) = n.$$

PageRank has been used by the Google search engine and has found applications in a wide range of data analysis problems [15, 24]. In this chapter, we consider the following natural problem of finding vertices with "significant" PageRank:

> SIGNIFICANTPAGERANKS: *Given a network $G = (V, E)$, a threshold value $1 \leq \Delta \leq |V|$ and a positive constant $c > 1$, compute a subset $S \subseteq V$ with thWe property that S contains all vertices of PageRank at least Δ and no vertex with PageRank less than Δ/c.*

For the corresponding algorithmic problem we assume that the network topology is described in the *sparse* representation of an (arbitrarily ordered) adjacency list for each vertex, as is natural for sparse graphs such as social and information networks. We are interested in developing an efficient local algorithm [3, 4, 112] for the problem in the context of Web applications. The algorithm is only allowed to randomly sample out-links of previously accessed nodes in addition to sampling nodes uniformly at random from the network. This model is highly suitable for PageRank maintenance in Web graphs and online information networks.

As the main contribution of this chapter, we present a nearly optimal, local algorithm for SIGNIFICANTPAGERANKS.

The running time of our algorithm is $\tilde{O}(n/\Delta)$. We also show that any algorithm for SIGNIFICANTPAGERANKS must have query complexity as well as runtime complexity $\Omega(n/\Delta)$. Thus, our algorithm is optimal up to a logarithmic factor.

Note that when $\Delta = \Omega(n^{\delta})$, for some constant $0 < \delta < 0$, our algorithm has sublinear time complexity.

Our SIGNIFICANTPAGERANKS algorithm applies a multiscale matrix sampling scheme that uses a fast Personalized PageRank estimator (see below) as its main subroutine.

Personalized PageRanks

While the PageRank of a vertex captures the importance of the vertex as collectively assigned by all vertices in the network, one can use the distributions of the following random walks to define the pairwise contributions of significance [60]: given a teleportation probability α and a starting vertex u in a network $G = (V, E)$, at each step, with probability $(1 - \alpha)$ go to a random neighboring vertex, and with probability α, restarts the process from u. For $v \in V$, the probability that v is visited by this random process, denoted by $\text{PersonalizedPageRank}_u(v)$, is u's personal PageRank contribution of significance to v. It is not hard to verify that

$$\begin{aligned} &\forall u \in V, \quad \sum_{v \in V} \text{PersonalizedPageRank}_u(v) = 1; \text{ and} \\ &\forall v \in V, \quad \text{PageRank}(v) = \sum_{u \in V} \text{PersonalizedPageRank}_u(v). \end{aligned}$$

Personalized PageRanks has been widely used to describe personalized behavior of Web users [100] as well as for developing good network clustering techniques [4]. As a result, fast algorithms for computing or approximating personalized PageRank are quite useful. One can approximate PageRanks and personalized PageRanks by the power method [15], which involves costly matrix-vector multiplications for large scale networks. Applying effective truncation, Jeh and Widom [64] and Andersen, Chung, and Lang [4] developed personalized PageRank approximation algorithms that can find an ϵ-additive approximation in time proportional to the product of ϵ^{-1} and the maximum in-degree in the graph.

Multi-Scale Matrix Sampling

Following the matrix view of the personalized PageRank formulation of Haveliwala [60] and the subsequent approximation of algorithms [4, 64], we introduce a matrix problem whose solution would lead to fast PageRank approximation and sublinear-time algorithms for SIGNIFICANTPAGERANKS.

In the basic form of this matrix problem, we consider a blackbox model for accessing an unknown $n \times n$ matrix with entries from $[0,1]$, in which we can only make a query of the following form: *matrixAccess*(i,ϵ), where $1 \leq i \leq n$ and $\epsilon \in (0,1]$. This query will return, with high probability, a list of entry-index pairs that provide an ϵ-precise approximation of row i in the unknown matrix: For each $1 \leq j \leq n$, if (p,j) is in the list of entry-index pairs returned by *matrixAccess*(i,ϵ), then $|p - m_{i,j}| \leq \epsilon$, where $m_{i,j}$ is the $(i,j)^{th}$ entry of the unknown matrix; otherwise if there is no entry containing index j, then $m_{i,j}$ is guaranteed to be at most ϵ. Importantly, we require that such query cost is propositional[10] to $1/\epsilon$. We will refer to this blackbox model as the *sparse and approximate row access model*, or SARA model for short.

We now define the basic form of our matrix problem:

> SIGNIFICANTMATRIXCOLUMNS**:** *Given an $n \times n$ matrix M, with entries from $[0,1]$, in the SARA model, a threshold Δ and a positive constant $c > 1$, return a subset of columns $S \subseteq V$ with the property that S contains all columns of sum at least Δ and no column with sum less than Δ/c.*

There is a straightforward connection between SIGNIFICANTMATRIXCOLUMNS and SIGNIFICANTPAGERANKS. Following [3], we define a matrix PPR (short for PersonalizedPageRank) to be the $n \times n$ matrix, whose u^{th} row is

$$\text{PersonalizedPageRank}_u(\cdot).$$

Clearly PPR is a matrix with entries from $[0,1]$ and for $1 \leq v \leq n$, $\text{PageRank}(v)$ is equal to the sum of the v^{th} column in PPR. Therefore, if we can solve the SIGNIFICANTMATRIXCOLUMNS problem with cost $\tilde{O}(n/\Delta)$ and also solve the problem of computing an ϵ-additive approximation of personalized PageRank in $\tilde{O}(\log(n)/\epsilon)$ time, then we are able to solve SIGNIFICANTPAGERANKS in $\tilde{O}(n/\Delta)$ time.

In this chapter, we analyze a multi-scale sampling algorithm for SIGNIFICANTMATRIXCOLUMNS. The algorithm selects a set of precision parameters $\{\epsilon_1,...,\epsilon_h\}$ where h

[10]Note that for this to hold there must be at most $O(1/\epsilon)$ entries bigger than ϵ in that row. This property is true, for example, for all right-stochastic matrices.

grows linearly with n/Δ and $\epsilon_i = i/h$. It then makes use of the sparse-and-approximate-row-access queries to obtain approximations of randomly sampled rows. For each i in range $1 \leq i \leq h$, the algorithm makes $\tilde{O}(1)$ (depending on the desired success probability) row-access queries to get a good approximation to the contribution of column elements of value of order ϵ_i. We show that with probability $1 - o(1)$, the multi-scale sampling scheme solve SIGNIFICANTMATRIXCOLUMNS with cost $\tilde{O}(n/\Delta)$.

While we could present our algorithm directly on PPR, we hope this matrix abstraction enables us to better highlight the two key algorithmic components in our fast PageRank approximation algorithm:

- multi-scale sampling, and
- robust approximation of personalized PageRanks.

Robust Approximation of PersonalizedPageRanks

For networks with constant maximum degrees, we can simply use personalized PageRank approximation algorithms developed by Jeh-Widom [64] or Andersen-Chung-Lang [4] inside the multi-scale scheme to obtain an $\tilde{O}(n/\Delta)$ time algorithm for SIGNIFICANT-PAGERANKS. However, for networks such as Web graphs and social networks that may have nodes with large degrees, these two approaches are not sufficient for our needs.

We develop a new local algorithm for approximating personized PageRank that satisfies the desirable robustness property that our multiscale sample scheme requires. Given $\lambda, \epsilon > 0$ and a starting vertex u in a network $G = (V, E)$, our algorithm estimates each entry in the Pprsonalized PageRank vector defined by u,

$$\text{PersonalizedPageRank}(u, .)$$

to a $[1-\lambda, 1+\lambda]$ multiplicative approximation around its value plus an additive error of at most ϵ. The time complexity of our algorithm is $O\left(\frac{\log^2 n \log(\epsilon^{-1})}{\epsilon\lambda^2}\right)$. Our algorithm requires a careful simulation of random walks from the starting node u to ensure that its complexity does not depend on the degree of any node. Together with the multi-scale sampling scheme, this algorithm leads to an $\tilde{O}(n/\Delta)$ time algorithm for SIGNIFICANTPAGERANKS. We conclude our analysis by showing that our algorithm for solving SIGNIFICANTPAGERANKS is optimal up to a polylogarithmic factor.

Related Work

Our research is inspired by the body of work on local algorithms [3, 4, 112], sublinear-time algorithms [106], and property testing [51] which study algorithm design for finding relevant substructures or estimating various quantities of interest without examining the entire input. Particularly, we focus on identifying nodes with significant PageRanks and approximating personalized PageRanks without exploring the entire input network. In addition, our framework is based on a combination of uniform crawling and uniform sampling of vertices in a graph and hence it can be viewed as a sublinear algorithm (when $\Delta = n^{\Omega(1)}$) in a rather general access model as discussed in [106].

It is well-known that in a directed graph, high in-degree of a node does not imply high PageRank for that node and vice versa. In fact, even in real-world Web graphs, only weak correlations have been reported between PageRank and in-degree [101]. One therefore needs to use methods for PageRank estimation that are not solely based on finding high in-degree nodes. Indeed, over the past decade, various beautiful methods have been developed to approximate the PageRank of all nodes. The common thread is that they all run in time at least linear in the input (See [15] for a survey of results). Perhaps the closest ones to our framework are the following two Monte-Carlo based approaches. The PageRank estimation method of [5] conducts simulation of a constant number of random walks from each of the nodes in the network and therefore it requires linear time in the size of the network. A similar approach is analyzed in [6], where a small number of random walks are computed from each network node, which shows that a tight estimate for the PageRank of a node with a large enough PageRank can be computed from the summary statistics of these walks. In addition, the paper shows how these estimates can be kept up to date, with a logarithmic factor overhead, on a certain type of a dynamic graph in which a fixed set of edges is inserted in a random order.

Our scheme is suitable to handle any network with arbitrary changes in it as well, including addition or removal of edges and nodes, with the necessary computation being performed "on the spot" as needed. But in contrast to the above approaches, for $\Delta = n^{\Omega(1)}$, our construction gives a sublinear-time algorithm for identifying all nodes whose PageRanks are above threshold Δ and approximating their PageRanks.

We have benefited from the intuition of several previous works on personalized PageRank approximation. Jeh and Widom developed a method based on a deterministic simulation of random walks by pushing out units of mass across nodes [64]. Their algorithm gives an ϵ-additive approximation with runtime cost of order of $\log n/\epsilon$ times the maximum out-degree of a node in the network. Andersen, Chung, and Lang [4] provided a clever implementation of the approach of Jeh and Widom that removes the $\log n$ factor from the runtime cost, still stopping when the residual amount to push out per node is at most[11] ϵ. We note, however, that for networks with large out-degrees, the complexity of this algorithm may not be sublinear.

Andersen *et al.* [3] developed a "backwards-running" version of the local algorithm of [4]. Their algorithm finds an ϵ-additive approximation to the PageRank vector with runtime proportional to $\frac{1}{\epsilon}$, times the maximum in-degree in the network, times the PageRank value. The authors show how it can be used to provide some reliable estimate to a node's PageRank: for a given k, with runtime proportional to $\tilde{\Theta}(k)$ times the maximum in-degree in the network (and no dependency on the PageRank value), it can bound the total contribution from the k highest contributors to a given node's PageRank. However, for networks with large in-degrees, its complexity may not be sublinear even for small values of k. We also note that the method does not scale well for estimating the PageRank values of multiple nodes, and needs to be run separately for each target node.

The problem of SIGNIFICANTMATRIXCOLUMNS can also be viewed as a matrix sparsification or matrix approximation problem, where the objective is to remove all columns

[11]Thus at termination the infinity norm of the residual vector is at most ϵ, which can easily be shown to bound from above the infinity norm of the difference between the true personalized PageRank vector and the estimation computed.

with l_1 norm less than Δ/c while keep all columns with l_1 norm at least Δ. To achieve time-efficiency, it is essential to allow the algorithm the freedom in deciding whether to keep or delete columns whose l_1 norm is in the range $[\Delta/c, \Delta]$.

While there has been a large body of work of finding a low complexity approximation to a matrix (such as a low-rank matrix) that preserves some desirable properties, many of the techniques developed are not directly applicable to our task.

First, we would like our algorithms to work even if the graph does not have a good low rank approximation; indeed, all of our algorithms work for any input graph. Second, our requirement to approximately preserve l_1 norm only for significant columns enable us to achieve $\tilde{O}(n/\Delta)$ complexity for any matrix with entries from the unit interval, whereas all low-rank matrix approximations run in time at least linear in the number of rows and columns of the matrix in order explicitly reconstruct a low-rank approximation; see [68, 69] for recent surveys on low-rank approximations.

On a high level, the problem of SIGNIFICANTMATRIXCOLUMNS may seem to share some resemblance to the *heavy-hitters* problem considered in the data streaming literature [35]. In the heavy-hitter problems, the goal is to identify all elements in a vector stream that have value bigger than the sum of all elements. The main difficulty to overcome is the sequential order by which items arrive and the small space one can use to store information about them. The main technique used to overcome these difficulties is the use of multiple hash functions which allows for concise summary of the frequent items in the stream. However, in SIGNIFICANTMATRIXCOLUMNS we are faced with a completely different type of constraints — access to only a small fraction of the input matrix (in order to achieve sublinear runtime) and having a precision-dependent cost of matrix row-approximations. As a result, hashing does not seem to be a useful avenue for this goals and one needs to develop different techniques in order to solve the problem.

Organization In Section 7.2, we introduce some notations that will be used in this paper. In Section 7.3, to better illustrate the multi-scale framework, we present a solution to a somewhat simpler abstract problem that distills the computational task we use to solve

SIGNIFICANTMATRIXCOLUMNS. In particular, we consider a blackbox model accessing an unknown vector that either returns an exact answer or 0 otherwise. Like the access model in SIGNIFICANTMATRIXCOLUMNS, higher precision costs more. In Section 7.4, we present our multi-scale sampling algorithm for SIGNIFICANTMATRIXCOLUMNS. In Section 7.5, we address the problem of finding significant columns in a PageRank matrix by giving a robust local algorithm for approximating personalized PageRank vectors. The section ends with a presentation of a tight lower bound for the cost of solving SIGNIFICANT MATRIX COLUMNS over PageRank matrices.

7.2 Preliminaries

In this section, we introduce some basic notations that we will frequently use in the paper. For a positive integer n, $[1:n]$ denotes the set of all integers j such that $1 \leq j \leq n$. If $M \in \mathbb{R}^{n\times n}$ is an $n \times n$ real matrix, for $v \in [1:n]$, we will use $M(v,\cdot)$ and $M(\cdot,v)$ to denote v^{th} row and the v^{th} column of M, respectively. We denote the sum of the column v in M by $\text{ColumnSum}(M,v)$. When the context is clear we shall suppress M in this notation and denote it by $\text{ColumnSum}(v)$.

Most graphs considered in this paper are directed. For a given directed graph $G = (V,E)$, we usually assume $V = [1:n]$. We use an $n \times n$ matrix $A(G)$ to denote the adjacency matrix of G. In other words, $A(i,j) = 1$ if and only $(i,j) \in E$.

The PageRank vector of a graph G is the (unique) stationary point of the following equation [60, 100]:

$$\text{PageRank}(\cdot) = \alpha \cdot \mathbf{e}_n + (1-\alpha)\text{PageRank}(\cdot) \cdot D^{-1}A(G),$$

where $\mathbf{e}_n$ is the n-place row vector of all 1's, $0 < \alpha < 1$ is a teleportation probability constant, and D is a diagonal matrix with the out-degree of v at entry (v,v).

Similarly, the personalized PageRank vector of u in the graph G is the (unique) stationary point of the following equation [60]:

$$\text{PersonalizedPageRank}_u(\cdot) = \alpha \cdot \mathbf{1}_u + (1-\alpha)\text{PersonalizedPageRank}_u(\cdot) \cdot D^{-1}A(G),$$

where $\mathbf{1}_u$ is the indicator function of u.

Note that with the above definition of PageRank, the sum of the entries of the PageRank vector is normalized to n. This normalization is more natural in the context of personalized PageRank than the traditional normalization in which the sum of all PageRank entries is 1.

For any x, $\log(x)$ means $\log_2(x)$ and $\ln(x)$ denotes the natural logarithm of x.

7.3 Multi-Scale Approximation of Vector Sum

Before presenting our algorithms for SIGNIFICANTMATRIXCOLUMNS, we give a multi-scale algorithm for a much simpler problem that, we hope, captures the essence of the general algorithm.

We consider the following blackbox model for accessing an unknown vector $\mathbf{p} = (p_1, ..., p_n) \in [0,1]^n$: we can only access the entries of $\mathbf{p}$ by making a query of the form *vectorAccess*(i, ϵ). If $p_i \geq \epsilon$, the query *vectorAccess*(i, ϵ) returns p_i, otherwise when $p_i < \epsilon$, *vectorAccess*(i, ϵ) returns 0. Furthermore, *vectorAccess*(i, ϵ) incurs a *cost* of $1/\epsilon$. In this subsection, we consider the following abstract problem:

> VECTORSUM**:** *Given a blackbox model* vectorAccess() *for accessing an unknown vector* $\mathbf{p} = (p_1, ..., p_n) \in [0,1]^n$*, a threshold* $\Delta \in [1:n]$ *and a positive constant* $c > 1$*, return* ***PASS*** *if* $\sum_i p_i \geq \Delta$*, return* ***FAIL*** *if* $\sum_i p_i < \frac{\Delta}{c}$*, and otherwise return either* ***FAIL*** *or* ***PASS****.*

To motivate our approach, before describing our multi-scale algorithm to solve this problem, let us first analyze the running time of a standard sampling algorithm. In such an algorithm, one would take h i.i.d. samples $s_1, \ldots, s_h$ uniformly from $[1:n]$ and query p_{s_t} at some precision ϵ to obtain an estimator

$$\frac{n}{h} \sum_{t=1}^{h} p_{s_t} \mathbf{I}[p_{s_t} \geq \epsilon]$$

for the sum $\sum_i p_i$. The error stemming from querying at precision ϵ would be of order $n\epsilon$, so we clearly will have to choose ϵ of order Δ/n or smaller not to drown our estimate in the query error, leading to a run time of order hn/Δ. The number of samples, h, on the other hand, has to be large enough to guarantee concentration, which at a minimum requires that the expectation of the sum $\sum_{t=1}^{h} p_{s_t}\mathbf{I}[p_{s_t} \geq \epsilon]$ is of order at least unity. But the expectation of this sum is upper bounded by $(h/n)\sum p_i$ which is of order $h\Delta/n$ in the most interesting case where $\sum p_i$ is roughly equal to Δ. We thus need h to be of order at least n/Δ, giving a running time of order $(n/\Delta)^2$, while we are aiming for a sublinear running time of order $\tilde{O}(n/\Delta)$.

Our algorithm is based on a different idea by querying p_t at a different precision each time, namely, by querying p_{s_t} at precision $\epsilon_t = t/h$ in the t^{th} draw, and considering the estimator

$$\frac{n}{h}\sum_{t=1}^{h}\mathbf{I}[p_{s_t} \geq \epsilon_t] \tag{7.1}$$

for the sum $\sum_i p_i$. In expectation, this estimator is equal to n times

$$\frac{1}{h}\sum_{t=1}^{h}\mathbf{P}[p_{s_t} \geq \epsilon_t] = \frac{1}{h}\sum_{t=1}^{h}\mathbf{P}\left[p_{s_t} \geq \frac{t}{h}\right] \tag{7.2}$$

with s_t denoting an integer chosen uniformly at random from $[1:n]$. This is a Riemann sum approximation to the well known expression

$$\mathbf{E}[p_s] = \int_0^1 dx\mathbf{P}[p_s \geq x]$$

and differs from this integral by an error $O(\frac{1}{h})$. In the most interesting case where $\sum_i p_i$ is of order Δ, concentration again requires h to be of order at least n/Δ, which also guarantees that the error $O(1/h)$ from the Riemann sum approximation does not dominate the expectation $\mathbf{E}[p_s] = \frac{1}{n}\sum_i p_i$. But now we only query p_s at the highest resolution $\epsilon_1 = 1/h$ once, leading to a much faster running time. In fact, up to log factors, the running time will be dominated by the first few queries, giving a running time of $\tilde{O}(h) = \tilde{O}(n/\Delta)$, as desired.

In the next section we proceed with the algorithm's formal description and analysis.

7.3.1 A Multi-Scale Algorithm for Approximating Vector Sum

The following algorithm, *MultiScaleVectorSum*, replaces the standard sampling to estimate the sum $\sum_i p_i$ by a multi-scale version which spends only a small amount of time at the computation intensive scales requiring high precision. In addition to the blackbox oracle $vectorAccess()$, this algorithm takes three other parameters: $\Delta \in (1,n)$ and $c > 1$ as defined in VECTORSUM, and a confidence parameter $\delta \in (0,1)$: this algorithm uses randomization and we will show that it correctly solves VECTORSUM with probability at least $1-\delta$. Our algorithm implements the strategy discussed above except for one modification: instead of sampling at a different precision ϵ_t each time, we sample at each precision a constant number of times τ, where τ depends on the desired success probability, given a total number of queries equal to $L = \tau h$, where $h = \Theta(n/\Delta)$ with the implicit constant in the Θ-symbol depending on c in such a way that it grows with $(c-1)^{-2}$ as $c \to 1$ (somewhat arbitrary, but convenient for our notation and proofs, we introduce the c dependence of our constructions through the variable $\beta = \frac{c-1}{4c}$; in terms of this variable, we write the lower cutoff Δ/c as $\Delta(1-4\beta)$, and use the midpoint $\Delta(1-2\beta)$ between Δ and Δ/c as the cutoff for the algorithm to decide between **PASS** and **FAIL**).

Theorem 7.1 (Multi-Scale Vector Sum)**.** *For any* $\mathbf{p} \in (0,1)^n$ *accessible by* $vectorAccess()$*, threshold* $\Delta \in (1,n)$*, robust parameter* $c > 1$*, and failure parameter* $\delta \in (0,1)$*, the method* MultiScaleVectorSum $(vectorAccess(), \Delta, c, \delta)$ *correctly solves* VECTORSUM *with probability at least* $(1-\delta)$ *and costs*

$$O\left(\frac{n}{\Delta}\left(\frac{1}{c-1}\right)^2 \log\Big(\frac{n}{\Delta(c-1)}\Big)\log\Big(\frac{2}{\delta}\Big)\right).$$

Proof. By Steps 3-7, for any constant $c > 1$, the cost of the algorithm is

$$\sum_{t=1}^{L}\frac{1}{\epsilon_t} = \tau\sum_{i=1}^{h}\frac{h}{i} \le L(1+\log h) = O\left(\frac{n}{\Delta}\left(\frac{1}{c-1}\right)^2 \log\Big(\frac{n}{\Delta(c-1)}\Big)\log\Big(\frac{2}{\delta}\Big)\right).$$

We now prove the correctness of the algorithm.

Algorithm MultiScaleVectorSum, after the initialization Steps 1 and 2, computes the

Algorithm 8 MultiScaleVectorSum

```
Require: $vectorAccess(\cdot,\cdot)$, threshold $\Delta \in (1,n)$, cutoff parameter $c > 1$, failure probability $\delta \in (0,1)$.
1: $\beta = \frac{c-1}{4c}$; $\tau = \lceil \log(1/\delta) \rceil$; $h = \lceil \frac{3n}{\Delta\beta^2} \rceil$; $L = \tau h$
2: $sum = 0$.
3: for $t = 1 : L$ do
4:     $\epsilon_t = \frac{1}{h} \lceil \frac{t}{\tau} \rceil$.
5:     Let $s_t$ be an uniform random element from $[1 : n]$.
6:     $z_t = vectorAccess(s_t, \epsilon_t)$.
7:     $sum = sum + z_t$.
8: end for
9: if $sum \geq (1-2\beta)\frac{L\Delta}{n}$ then
10:     Return PASS.
11: else
12:     return FAIL.
13: end if
```

multi-scale parameters ϵ_t and applies sampling to calculate the sum

$$Q = \sum_{t=1}^{L} z_t = \sum_{t=1}^{L} \mathbf{I}\left[p_{s_t} \geq \epsilon_t\right]$$

where $s_1, \ldots, s_L$ are chosen i.i.d. uniformly at random from $[1:n]$. The expectation of Q is easily estimated in terms of the bounds

$$\begin{aligned}\mathbf{E}[Q] &= \frac{1}{n}\sum_{k=1}^{n}\sum_{t=1}^{L}\mathbf{I}\left[\epsilon_t \leq p_k\right] = \frac{1}{n}\sum_{k=1}^{n}\sum_{t=1}^{L}\mathbf{I}\left[\lceil t/\tau\rceil \leq hp_k\right] \\ &= \frac{\tau}{n}\sum_{k=1}^{n}\sum_{i=1}^{h}\mathbf{I}\left[i \leq hp_k\right] = \frac{\tau}{n}\sum_{k=1}^{n}\lfloor hp_k\rfloor \leq \frac{\tau}{n}\sum_{k=1}^{n} hp_k = \frac{L}{n}\sum_{k=1}^{n} p_k \qquad (7.3)\end{aligned}$$

and

$$\mathbf{E}[Q] \geq \frac{\tau}{n}\sum_{k=1}^{n}\Big(hp_k - 1\Big) = \frac{L}{n}\sum_{k=1}^{n} p_k - \tau. \qquad (7.4)$$

We thus use $\frac{n}{L}Q$ as an estimate of $\sum_{k=1}^{n} p_k$ when we decide on whether to output **PASS** in Step 9.

Assume first that $\sum p_k \geq \Delta$. Since $\tau \leq \beta^2 \frac{L\Delta}{3n} \leq \beta\frac{L\Delta}{n}$, we then have

$$\mathbf{E}[Q] \geq \frac{L\Delta}{n} - \tau \geq (1-\beta)\frac{L\Delta}{n},$$

implying that

$$(1-\beta)\mathbf{E}[Q] \geq (1-2\beta)\frac{L\Delta}{n}.$$

This allows us to use the multiplicative Chernoff bound in the form of Lemma A.1 to conclude that

$$\mathbf{Pr}\left[Q \leq (1-2\beta)\frac{L\Delta}{n}\right] \leq \mathbf{Pr}\left[Q \leq (1-\beta)\mathbf{E}[Q]\right] \leq$$
$$\exp\left(-\frac{\beta^2}{2}\mathbf{E}[Q]\right) \leq \exp\left(-\frac{3}{8}\frac{\beta^2 L\Delta}{n}\right) \leq \delta,$$

where we used $\beta \leq 1/4$ in the last step.

On the other hand, if $\sum p_k \leq \Delta/c = (1-4\beta)\Delta$, we bound

$$\mathbf{E}[Q] \leq \frac{L\Delta}{n}(1-4\beta)$$

which in turn implies that

$$(1+2\beta)\mathbf{E}(Q) \leq (1-2\beta)\frac{L\Delta}{n}.$$

Using the multiplicative Chernoff bound in the form A.1 (part 3), this gives

$$Pr\left[Q \geq (1-2\beta)\frac{L\Delta}{n}\right] \leq \exp\left(-\beta^2\frac{L\Delta}{n}\frac{1-2\beta}{1+2\beta}\right) \leq \exp\left(-\frac{\beta^2 L\Delta}{3n}\right) \leq \delta.$$

where we again used $\beta \leq 1/4$.

Thus, MultiScaleVectorSum $(vectorAccess(), \Delta, c, \delta)$ correctly solves VECTORSUM with probability at least $1-\delta$.

$\square$

7.4 Multi-Scale Matrix Sampling

In this section, we consider SIGNIFICANTMATRIXCOLUMNS in a slightly more general matrix access model than what we defined in Section 7.3. The extension of the model is also needed in our PageRank approximation algorithm, which we will present in the next section.

7.4.1 Notation: Sparse Vectors

To better specify this model and the subsequent algorithms, we first introduce the notation of *sparse vector* introduced by Gilbert, Moler, and Schreiber [48] for Matlab. Suppose $\mathbf{a} = (a_1, ..., a_n) \in \mathbb{R}^n$ is a vector. Let $\text{nnz}(\mathbf{a})$ denotes the number of nonzero elements in $\mathbf{a}$. Let $\text{Sparse}(\mathbf{a})$ denote the *sparse form* of vector $\mathbf{a}$ by "squeezing out" any zero elements in $\mathbf{a}$. Conceptually, one can view $\text{Sparse}(\mathbf{a})$ as a list of $\text{nnz}(\mathbf{a})$ index-entry pairs, one for each nonzero element and its index in $\mathbf{a}$. For example, we can view $\text{Sparse}([0, 0.3, 0.5, 0, 0.2])$ as $((2, 0.3), (3, 0.5), (5, 0.2))$.

A sparse vector can be easily implemented using a binary search tree [12]. Throughout the paper we shall make use of the following simple proposition:

[12]For average case rather than worst-case guarantees, a hash table is a typical implementation choice.

Proposition 7.2. *For* $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$*,* $\mathbf{a} + \mathbf{b}$ *can be implemented in time* $O(\text{nnz}(\mathbf{b}) \cdot \log n)$ *saving the result in the data structure of* $\mathbf{a}$.

Proof. Each sparse vector can be implemented as a balanced binary search tree, where the index of an entry serves as the entry's key. When performing the addition, we update the binary search tree of **a** by inserting one by one the elements of **b** into it (and updating existing entries whenever needed). By the standard theory of binary search trees, each such insertion operation takes $O(\log n)$ time. □

In the rest of the paper, without further elaboration, we assume all vectors are expressed in this sparse form. We also adpot the following notations: let $\text{Sparse}([\,])$ denote the all zero's vector in the sparse form, and for any $i \in [1:n]$ and $b \in \mathbb{R} - \{0\}$, let $\text{Sparse}(i,b)$ denote the sparse vector with only one nonzero element b located in the i^{th} place in the vector. In addition, we will use the following notation: for two vectors n-place vectors $\mathbf{a} = (a_1, ..., a_n)$ and $\mathbf{b} = (b_1, ..., b_n)$, and parameters $\epsilon \in \mathbb{R}$ and $C > 0$, we use $\mathbf{a} \le C \cdot \mathbf{b} + \epsilon$ to denote $a_i \le C\dot{b}_i + \epsilon$, $\forall i \in [1:n]$.

7.4.2 The Matrix Access Model

In the model that we will consider in the rest of this section, we can access an unknown $n \times n$ matrix $M = (m_{i,j})$, with entries from $[0,1]$, by only using queries of the form *matrixAccess*$(i, \epsilon, \lambda, p)$, where $i \in [1:n]$ specifies a row, $\epsilon \in (0,1]$ specifies a required additive precision, $\lambda \in (0,1]$ specifies a multiplicative precision, and $p \in (0,1]$ specifies the probability requirement. This query will return a sparse vector $\tilde{\mathbf{m}}_i = \text{Sparse}([\tilde{m}_{i,1}, ..., \tilde{m}_{i,n}])$ such that

- with probability at least $1 - p$,

$$(1 - \lambda) \cdot \mathbf{m}_i - \epsilon \le \tilde{\mathbf{m}}_i \le (1 + \lambda) \cdot \mathbf{m}_i + \epsilon, \tag{7.5}$$

  where $\mathbf{m}_i = M(i, \cdot)$ denotes the i^{th} row of matrix M, and

- with probability at most p (the query may fail), $\tilde{\mathbf{m}}_i$ could be an arbitrary sparse vector.

We refer to this blackbox model as the *probabilistic sparse-and-approximate row-access model with additive/multiplicative errors*. For constant integers $c_1, c_2, c_3, c_4 > 0$, we say that *matrixAccess* is an (c_1, c_2, c_3, c_4)-SARA model if for all $i \in [1:n]$, $\epsilon \in (0,1)$, $\lambda \in (0,1)$, and $p \in (0,1)$, both the cost of calling $\tilde{\mathbf{m}}_i$ =*matrixAccess*$(i, \epsilon, \lambda, p)$ and $\text{nnz}(\tilde{\mathbf{m}}_i)$ are bounded from above by

$$c_1 \left(\frac{1}{\lambda}\right)^{c_2} \left(\frac{\log^{c_3}(1/\epsilon)}{\epsilon}\right) (\log^{c_4} n) \log(1/p).$$

7.4.3 The Matrix Problem

In this section, we give a solution to the following abstract problem.

> SIGNIFICANTMATRIXCOLUMNS: *Given an $n \times n$ matrix M, with entries from $[0,1]$, in the (c_1, c_2, c_3, c_4)-SARA model, a threshold Δ and a positive constant $c > 1$, return a sparse vector cSum with the property that for all $j \in [1:n]$, if* $\text{ColumnSum}(M,i) \geq \Delta$, *then $cSum(j) \neq 0$ and if* $\text{ColumnSum}(M,i) < \Delta/c$, *then $cSum(j) = 0$.*

7.4.4 Understanding the Impact of Additive/Multiplicative Errors

Our algorithm for SIGNIFICANTMATRIXCOLUMNS is straightforward. At a high level, it simultaneously applies Algorithm 8 to all columns of the unknown matrix. It uses a sparse-vector representation for efficient bookkeeping of the columns with large sum according to the sampled data. Our analysis of this algorithm is similar to the one presented that in Theorem 7.1 for VECTORSUM as we can use the union bound over the columns to reduce the analysis to a single column. The only technical difference is the handling of the additive/multiplicate errors.

To understand the impact of these errors, we consider a vector $\mathbf{p} = (p_1, ..., p_n) \in [0,1]^n$ and chose ϵ_t, $t = 1, \ldots, L$ as in Algorithm 8. Fix $\phi, \lambda \in (0, 1/2)$, and suppose that we access p_i with multiplicative error λ and additive error $\phi \cdot \epsilon_t$. We will show that if this returns a number $\tilde{p}_i \geq \epsilon_t$, the actual value of p_i is at least $\rho\epsilon_t$, where $\rho = 1 - \lambda - \phi$. To see this, we bound

$$p_i \geq (1+\lambda)^{-1}(\tilde{p}_i - \phi \cdot \epsilon_t) \geq (1+\lambda)^{-1}(1-\phi)\epsilon_t.$$

Since $(1-\phi)/(1+\lambda) \geq (1-\lambda-\phi)$, this implies $p_i \geq \rho\epsilon_t$, as desired.

In a similar way, it is easy to see that $p_i \geq \rho^{-1}\epsilon_t$ implies that $\tilde{p}_i \geq \epsilon_{t,}$. Indeed, if $p_i \geq \rho^{-1}\epsilon_t$ then

$$\tilde{p}_i \geq (1-\lambda)p_i - \phi \cdot \epsilon_t \geq \left(\frac{1-\lambda}{1-\lambda-\phi} - \phi\right)\epsilon_t.$$

The lower bound is clearly larger than ϵ_t, , showing that $\tilde{p}_i \geq \epsilon_t$.

For $s_1 \ldots, s_L \in [1:n]$, the sum

$$\tilde{Q} = \sum_{t=1}^{L} \mathbf{I}[\tilde{p}_{s_t} \geq \epsilon_t] \tag{7.6}$$

can therefore be bounded from below and above by

$$Q_- = \sum_{t=1}^{L} \mathbf{I}[p_{s_t} \geq \rho^{-1}\epsilon_t] \quad \text{and} \quad Q_+ = \sum_{t=1}^{L} \mathbf{I}[p_{s_t} \geq \rho\epsilon_t], \tag{7.7}$$

respectively:

$$Q_- \leq \tilde{Q} \leq Q_+. \tag{7.8}$$

Finally, we also note that if we access p_i with multiplicative error λ and additive error $\phi \cdot \epsilon_t$, then this a returns a number which is never larger than ρ^{-1}. Indeed, this follows by bounding $\tilde{p}_i$ by $1 + \lambda + \phi \cdot \epsilon_t \leq 1 + \lambda + \phi \leq \rho^{-1}$.

7.4.5 A Multi-Scale Algorithm

In this section we present the multi-scale algorithm in full details and proceed with an analysis of its runtime and correctness. The algorithm is essentially an extension of Algorithm 8, applying the VectorSum algorithm to all columns in parallel. As now the call to vectorAcesss has been replaced by a combined additive-multiplicative method, the constant β is set to a slightly smaller value than in Algorithm 8. In addition to the constants β, τ, h, L

that are used in Algorithm 8, we also have the constant λ for the value of multiplicative approximation needed and ϕ for the additive-approximation needed; and last, p is the wanted success probability of the row approximation procedure (matrixAccess) invoked throughout the algorithm. We note that these constants are defined to allow complete and rigorous analysis of our algorithm and its correctness. As the multi-scale algorithm will essentially be implementing Algorithm 8 over all columns, we will need a method that can return all elements in a row that fall within a certain bin; we call it the *rangeIndicator* method.

rangeIndicator(): for a sparse vector $\mathbf{a}$, and $l,u \in \mathbb{R}$ such that $l < u$, the call $\mathbf{b} = \text{rangeIndicator}(\mathbf{a},l,u)$ returns a sparse vector $\mathbf{b}$ such that for all $i \in [1:n]$,

$$\mathbf{b}(i) = \begin{cases} 1 & \text{if } l \leq \mathbf{a}(i) \leq u \\ 0 & \text{otherwise} \end{cases}$$

For example, rangeIndicator(Sparse([0,0.3,0.5,0,0.2]),0.1,0.3) returns the sparse form of [0,1,0,0,1]. We shall use the following simple proposition:

Proposition 7.3. rangeIndicator$(\mathbf{a},l,u)$ *takes* $O(\text{nnz}(\mathbf{a})\log n)$ *time.*

Proof. The sparse vector $\text{nnz}(\mathbf{a})$ is implemented using a binary search tree; one can therefore scan its content using, say, an inorder scan and insert each element in the range $[l,u]$ to a sparse vector $\mathbf{b}$, initially empty. The inorder scan costs $O(\text{nnz}(\mathbf{a}))$ time and each insertion into b costs $O(\log n)$ time, giving the desired result. □

We are now ready to state our main theorem.

Theorem 7.4 (Multi-Scale Column Sum)**.** *For any matrix M, with entries from* $[0,1]$*, accessible by matrixAccess, threshold* $\Delta \in (1,n)$*, robust parameter* $c > 1$*, and failure parameter* $\delta \in (0,1)$*, with probability at least* $(1-\delta)$*,*

$$cSum = \text{MultiScaleVectorSum}\left(vectorAccess(),\Delta,c,\delta\right).$$

correctly solves SIGNIFICANTMATRIXCOLUMNS.

Algorithm 9 MultiScaleColumnSum

```
Require: $matrixAccess(\cdot,\cdot,\cdot,\cdot)$, threshold $\Delta \in (1,n)$, cutoff $c > 1$, failure probability $\delta \in (0,1)$.
1: $\beta = \frac{c-1}{5c}$;  $\tau = \lceil \log(2n/\delta) \rceil$;  $h = \lceil \frac{3n}{\Delta\beta^2} \rceil$;  $L = \tau h$;  $p = \delta/(2L)$;  $\lambda = \beta/2$;  $\phi = \beta/2$;  $\rho = 1 - \lambda - \phi$.
2: $cSum = \text{Sparse}([\,])$.
3: for $t = 1 : L$ do
4:     $\epsilon_t = \frac{1}{h} \lceil \frac{t}{\tau} \rceil$.
5:     Let $s_t$ be an uniform random element from $[1:n]$;  $q_t = matrixAccess(s_t, \phi \cdot \epsilon_t, \lambda, p)$.
6:     $\mathbf{z}_t = \text{rangeIndicator}\left(\mathbf{q}_t, \epsilon_t, \rho^{-1}\right)$.
7:     $cSum = cSum + \mathbf{z}_t$.
8: end for
9: $cSum = \text{rangeIndicator}\left(cSum, (1-2\beta)\frac{L\Delta}{n}, L\right)$;
10: Return $cSum$.
```

Furthermore, if $matrixAccess$ is an (c_1,c_2,c_3,c_4)-SARA model, then the cost of MultiScaleVectorSum $(vectorAccess(),\Delta,c,\delta)$ *is*

$$O\left(c_1\left(\frac{n}{\Delta}\right)\left(\frac{1}{c-1}\right)^{c_2+3}\log^{c_3+2}\left(\frac{1}{c-1}\right)\log^{c_3+c_4+3}n\log^2\left(\frac{2}{\delta}\right)\right).$$

Proof. The cost of the algorithm is dominated by the sparse matrix operations in line 5-7, plus the cost of the last operation in line 9. Using our access model together with Propositions 7.2 and 7.3, the cost of the steps in line 5-7 at time t are of order

$$O\left(c_1\left(\frac{1}{\beta}\right)^{c_2}\frac{\log^{c_3}(\frac{1}{\beta\epsilon_t})}{\beta\epsilon_t}\log^{c_4+1}n\log(2L/\delta)\right)\leq$$

$$O\left(c_1\frac{h}{\lceil t/\tau\rceil}\log^{c_3}h\left(\frac{1}{\beta}\right)^{c_2+1}\log^{c_4+1}n\log(\frac{2n}{\Delta\beta\delta})\right).$$

Note that this includes the extra factor of $\log n$ from Proposition 7.2, a factor which is absent in the sparseness of $\mathbf{q}_t$ and $\mathbf{z}_t$.

Summing over t gives a running time of order

$$\begin{aligned}O&\left(c_1L\log^{c_3+1}h\left(\frac{1}{\beta}\right)^{c_2+1}\log^{c_4+1}n\log(\frac{2n}{\Delta\beta\delta})\right)\\ &=O\left(c_1\left(\frac{n}{\Delta}\right)\log^{c_3+1}\left(\frac{n}{\Delta\beta}\right)\left(\frac{1}{\beta}\right)^{c_2+3}\log^{c_4+1}n\log\left(\frac{2n}{\Delta\beta\delta}\right)\log\left(\frac{2}{\delta}\right)\right)\\ &=O\left(c_1\left(\frac{n}{\Delta}\right)\left(\frac{1}{\beta}\right)^{c_2+3}\log^{c_3+2}\left(\frac{1}{\beta}\right)\log^{c_3+c_4+3}n\log^2\left(\frac{2}{\delta}\right)\right).\end{aligned}$$

To estimate the cost of the last step of the algorithm, we bound the sparseness of $cSum$ at the completion of the FOR loop by $\mathrm{nnz}(cSum)\leq\sum_t\mathrm{nnz}(\mathbf{z}_t)$ and then apply Proposition 7.3 once more, giving a cost which of the same order as the total cost of the algorithm accrued up to this step.

To prove the correctness of the algorithm, we first note that with probability at least $(1-p)^L\geq 1-pL$, each of the L calls of $matrixAccess$ in line 5 will return a sparse vector obeying the bound (7.5). Next, we apply the union bound to reduce the focus of the analysis to a single column:

$$\Pr\left[\text{MULTISCALECOLUMNSUM is unsuccessful}\right] \leq$$
$$\textstyle\sum_{i=1}^{n} \Pr\left[\text{MULTISCALECOLUMNSUM is unsuccessful on column } i\right].$$

When considering column i, we now let $\mathbf{p} = (p_1, ..., p_n)^T = M(\cdot, i)$, the i^{th} column of M. In other words, $p_j = m_{i,j}$ for all $j \in [1 : n]$. Note that the i^{th} entry of $cSum$ after step 8 is of the form (7.6). Taking into account the bound (7.8), our proof will be very similar to that of Theorem 7.1.

We first consider the case that $\sum_i p_i \geq \Delta$ in which case we bound

$$\mathbf{E}[Q_-] \geq \frac{L\rho}{n}\sum_k p_k - \tau \geq \frac{\Delta L}{n}\left[\rho - \frac{\beta^2}{3}\right] = \frac{\Delta L}{n}\left[1 - \beta - \frac{\beta^2}{3}\right].$$

Multiplying both sides by $(1-\beta)$, we obtain

$$(1-\beta)\mathbf{E}[Q_-] \geq \frac{\Delta L}{n}(1-2\beta).$$

Combined with the bound (7.8) and the multiplicative Chernoff bound (Lemma A.1), this shows that conditioned on $matrixAccess$ returning a sparse vector obeying the bound (7.5) in each instance in line 5, we get

$$\mathbf{Pr}\left\{cSum(i) \leq (1-2\beta)\frac{\Delta L}{n}\right\} \leq \exp\left(-\frac{\beta^2}{2}\mathbf{E}[Q_-]\right) \leq \exp\left(-\frac{3}{8}\frac{\beta^2 L\Delta}{n}\right) \leq \frac{\delta}{2n}.$$

In a similar way, if $\sum_k p_k \leq \Delta/c = \Delta(1-5\beta)$, we bound

$$\mathbf{E}[Q_+] \leq \frac{\Delta L}{n\rho}(1-5\beta) = \frac{\Delta L}{n}\frac{1-5\beta}{1-\beta},$$

implying that

$$(1+2\beta)\mathbf{E}[Q_+] \leq \frac{\Delta L}{n}(1-2\beta)$$

and hence

$$\mathbf{Pr}\left\{cSum(i) \geq (1-2\beta)\frac{\Delta L}{n}\right\} \leq \exp\left(-\beta^2\frac{\Delta L}{n}\frac{1-2\beta}{1+2\beta}\right) \leq \exp\left(-\frac{\Delta L\beta^2}{3n}\right) \leq \frac{\delta}{2n},$$

again conditioned on $matrixAccess$ returning a sparse vector obeying the bound (7.5) in each instance in line 5.

Thus the total failure probability is at most $Lp + n\frac{\delta}{2n} = \delta$, as desired.

$\square$

7.5 Identifying Nodes with Significant PageRank

7.5.1 Robust Approximation of Personalized PageRanks

We now present our main subroutine for SIGNIFICANTPAGERANKS which, we recall, addresses the following problem: Given a directed graph $G = (V, E)$, a threshold value $1 \leq \Delta \leq |V|$ and a positive constant $c > 1$, compute a subset $S \subseteq V$ with the property that S contains all vertices of PageRank at least Δ and no vertex with PageRank less than Δ / c.

Let PPR denote the personalized PageRank Matrix of G defined in the Introduction, where we recall that $PPR(i, j)$ is equal to the personalized PageRank contribution of node i to node j in G. Under this notation, the SIGNIFICANTPAGERANKS can be viewed as a SIGNIFICANTMATRIXCOLUMNS problem, if we can develop an efficient procedure for accessing the rows of PPR. This procedure, which we refer to as *PPRmatrixAccess()*, takes a row number i, an additive precision parameter ϵ, a multiplicative precision parameter λ and success probability p, and returns a sparse vector $\tilde{\mathbf{m}}_i = \text{Sparse}([\tilde{m}_{i,1}, ..., \tilde{m}_{i,n}])$ such that

- with probability at least $1 - p$,

  $$(1 - \lambda) \cdot \mathbf{m}_i - \epsilon \leq \tilde{\mathbf{m}}_i \leq (1 + \lambda) \cdot \mathbf{m}_i + \epsilon,$$

  where $\mathbf{m}_i = PPR(i, \cdot)$, and

- with probability at most p, $\tilde{\mathbf{m}}_i$ can be any sparse vector.

Our algorithm for *PPRmatrixAccess()* uses the following key observation that connects personalized PageRank with the hitting probability of a Markov model.

Observation 7.5. *$PPR(v, j)$ is equal to the success probability that a random walk starting at v and independently terminating at each time step with probability α, hits j just before termination.*

Proof. Let 1_v be the indicator vector of v. Solving the system given by

$$\text{PersonalizedPageRank}(v, \cdot) = \alpha \mathbf{1}_v + (1 - \alpha)\text{PersonalizedPageRank}(v, \cdot) D^{-1} A,$$

Algorithm 10 PPRmatrixAccess

```
Require: node $v$, additive approximation $\epsilon$, multiplicative approximation $\lambda$.
 1: $cSum = \text{Sparse}([\,])$.
 2: Set $length = \lceil \log_{\frac{1}{(1-\alpha)}}(\frac{4}{\epsilon}) \rceil$.
 3: Set $r = \lceil \frac{1}{\epsilon\lambda^2} \cdot 4\ln(n/p) \rceil$.
 4: for $r$ rounds do
 5:     Run one realization of a restarting random walk from $v$. Artificially stop the walk
        after $length$ steps if it has not terminated already.
 6:     if the walk visited a node $j$ just before making a termination step then
 7:         $cSum = cSum + Sparse(j, 1/r)$    //namely, add $1/r$ to $j$'s value.
 8:     end if
 9:     Return $cSum$.
10: end for
```

one obtains

$$\text{PersonalizedPageRank}(v, \cdot) = \alpha \mathbf{1}_v (I - (1-\alpha)D^{-1}A)^{-1} = \alpha \mathbf{1}_v \sum_{i=0}^{\infty} ((1-\alpha)D^{-1}A)^i.$$

The observation then follows directly from the last equation. □

Our algorithm for *PPRmatrixAccess* given below conducts a careful simulation of such restarting random walks. As such it only needs an oracle access to a random out-link of a given node.

Theorem 7.6. *For any node* v*, values* $0 < \epsilon < 1$, $0 < \lambda < 1$, $0 < \alpha < 1$, *and success probability* $0 < p < 1$, PPRmatrixAccess$(v, \epsilon, \lambda, p)$ *is a* $(10 \max\{\log^{-1}(\frac{1}{1-\alpha}), 1\}, 2, 1, 2)$-*SARA model. In particular, its runtime is upper bounded by*

$$O\left(\frac{\ln^2(n)\ln(1/p)\log(\epsilon^{-1})}{\epsilon\lambda^2}\right).$$

Proof. We start by analyzing the runtime guarantee. Algorithm PPRmatrixAccess performs $\lceil \frac{1}{\epsilon\lambda^2} \cdot 4\ln(n/p) \rceil$ rounds where at each round it simulates a random walk with termination probability of α for at most $length$ steps. Each step is simulated by taking uniform sample ('termination' step) with probability α and by choosing a random out-link with probability $1-\alpha$. The update of $cSum$ in line 7 takes at most $\log n$ (see proposition 7.2). Thus the total number of queries used is

$$\left\lceil \frac{4\ln(n/p)}{\epsilon\lambda^2} \right\rceil \cdot \left\lceil \log_{\frac{1}{(1-\alpha)}} \left(\frac{4}{\epsilon}\right) \right\rceil \log(n) \le \left\lceil \frac{4\ln(n/p)}{\epsilon\lambda^2} \right\rceil \cdot \left\lceil \frac{\log(\frac{4}{\epsilon})}{\log(\frac{1}{1-\alpha})} \right\rceil \log(n) \le$$

$$(8+2)\max\left\{\log^{-1}(\frac{1}{1-\alpha}),1\right\} \frac{\ln^2(n)\log(1/p)\log(\epsilon^{-1})}{\epsilon\lambda^2}.$$

We now prove the guarantees on the returned vector $cSum$ (line 9 in the algorithm). Given a node j, denote by $p_k(v,j)$ the contribution to j from restarting walks originating at v that are of length at most k, namely,

$$p_k(v,j) = \alpha \mathbf{1}_v \sum_{i=0}^{k} (1-\alpha)D^{-1}A)^i.$$

We ask how much is contributed to j's entry from restarting walks of length bigger or equal to k. The contribution is at most $(1-\alpha)^k$ since the walk needs to survive at least k consecutive steps. Taking $(1-\alpha)^k \le \frac{\epsilon}{4}$ will guarantee that at most $\frac{\epsilon}{4}$ is lost by only considering walks of length smaller than k, namely:

$$PPR(v,j) - \frac{\epsilon}{4} \le p_k(v,j) \le PPR(v,j).$$

For this to hold it suffices to take $k = \lceil \log_{\frac{1}{(1-\alpha)}} (\frac{4}{\epsilon}) \rceil$, the value the parameter $length$ is set to in step 2.

Next, the algorithm computes an estimate of $p_k(v,j)$ by realizing walks of length at most k. This is the value of $cSum$ at index j returned by the algorithm. Denote this by $\hat{p}_k(v,j)$. The algorithm computes such an estimation (in line 7) by taking the average number of hits over r trials (adding $1/r$ per hit).

Now, if $PPR(v,j) \ge \frac{\epsilon}{2}$ then $p_k(v,j) \ge \frac{\epsilon}{4}$ and by the multiplicative Chernoff bound (Lemma A.1),

$$Pr\left(\hat{p}_k(v,j) > (1+\lambda)p_k(v,j)\right) \le \exp(-\ln(n/p))$$

and

$$Pr\left(\hat{p}_k(v,j) < (1-\lambda)p_k(v,j)\right) \leq \exp(-\ln(n/p)).$$

By the union bound we can conclude that with probability $1-\frac{2p}{n}$,

$$(1-\lambda)(PPR(v,j)-\frac{\epsilon}{4}) \leq \hat{p}_k(v,j) \leq (1+\lambda)PPR(v,j).$$

Similarly, if $PPR(v,j) < \frac{\epsilon}{2}$ then $p_k(v,j) < \frac{\epsilon}{2}$ and by the multiplicative Chernoff bound (Lemma A.1, part 3),

$$Pr\left(\hat{p}_k(v,j) > (1+\lambda)\frac{\epsilon}{2}\right) \leq \exp(-\ln(n/p)) = p/n.$$

As $\lambda < 1$ we therefore have $0 \leq \hat{p}_k(v,j) \leq \epsilon$ with probability at least $1-p/n$. And as $PPR(v,j) < \frac{\epsilon}{2}$ we clearly have, with probability $1-p/n$,

$$(1-\lambda)PPR(v,j) - \epsilon \leq \hat{p}_k(v,j) \leq (1+\lambda)PPR(v,j) + \epsilon,$$

as needed.

By the union bound, the complete claim holds with probability at least $1-p$. □

7.5.2 A Tight Lower Bound for Solving SIGNIFICANTPAGERANKS

In this subsection, we present a corresponding lower bound for identifying all nodes with significant PageRank values. Our lower bound holds under the stringent model where one can access any node of interest in the graph in one unit of cost and that the PageRank of the node accessed is given for free. We call such a model the *strong query model.* We first give a lower bound to illurstrate the challenge for identifying nodes with significant PageRanks, even in graphs where there is only one significant node.

We then show that for any integral threshold Δ and precision c there are instances where the output size of SIGNIFICANTPAGERANKS is $\Omega(n/\Delta)$. Clearly, this also serves as a lower bound for the runtime of any algorithm that solves the SIGNIFICANTPAGERANKS problem, regardless of the computational model used to compute the required output. We note that the runtime of our algorithmic solution to SIGNIFICANTPAGERANKS is at most only a small polylogarithmic factor away from this bound.

For clarity of exposition we present our lower bounds for $\alpha = 0.5$. Similar lower bounds hold for any fixed $0 < \alpha < 1$.

Theorem 7.7 (Hardness for Identifying One Significant Node)**.** *Let $\alpha = 0.5$. For n large enough, any algorithm making less than $\frac{n}{6\Delta}$ queries in the strong query model on graphs on n nodes and threshold $\Delta \leq \frac{n}{9}$, would fail with probability at least $1/e$ to find a node with PageRank at least Δ, on at least one graph on n nodes.*

Proof. The proof will apply Yao's Minimax Principle for analyzing randomized algorithms [119], which uses the average-case complexity of the deterministic algorithms to derive a lower bound on the randomized algorithms for solving a problem.

Given positive integers n and $\Delta \leq \frac{n}{9}$, we construct a family $\mathcal{F}$ of undirected graphs on n nodes by taking a cycle subgraph on $n - d - 1$ nodes and an isolated star subgraph on the remaining $d + 1$ nodes, where we set $d = 3\Delta - 1$. To complete the construction we take a random labeling of the nodes. See Figure 7.1 for an illustration.

Let A be a deterministic algorithm for the problem. We shall analyze the behavior of A on a uniformly random graph from $\mathcal{F}$.

First, by solving the PageRank equation system it is easy to check that each node on the cycle subgraph has PageRank value of 1, the hub of the subgraph has PageRank $\frac{d}{3} + \frac{2}{3}$, and a leaf of the star subgraph has PageRank $\frac{2}{3} + \frac{1}{3d}$. The only node with PageRank at least Δ is the hub of the star subgraph.

Let T be the number of queries the algorithm make. The probability that none of the nodes of the star subgraph are found after T queries by A is at least

$$(1 - \frac{d+1}{n})^T \geq \exp(-2T\frac{d+1}{n}) \geq \exp(-1),$$

for $T \leq \frac{n}{6\Delta} = \frac{n}{2(d+1)}$. Here we used the fact that $1 - x \geq \exp(-2x)$, for $0 \leq x \leq 1/3$.

We define the cost of the algorithm as 0 if it has found a node of Pagerank at least Δ and 1 otherwise. Note that the cost of an algorithm equals its probability of failure and we

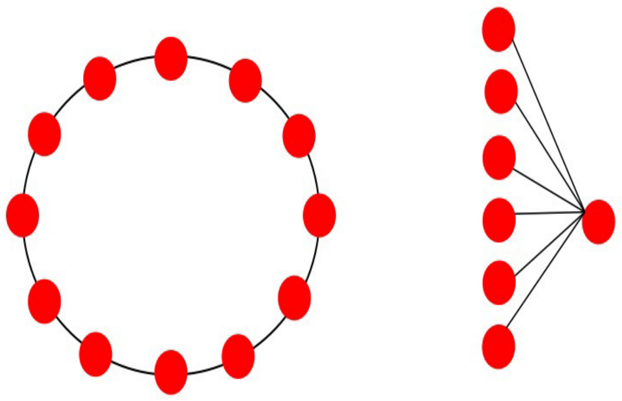

Figure 7.1: An example illustrating the "cycle and star" lower bound construction for PageRank computations.

can think of any Monte Carlo randomized algorithm (performing less than $\frac{n}{6\Delta}$ queries) as a Las Vegas one. Then by Yao's Minimax Principle, any randomized algorithm that makes at most $\frac{n}{6\Delta}$ queries will have an expected cost of at least $1/e$, i.e., a failure probability of at least $1/e$ on at least one of the inputs. □

Theorem 7.8 (Graphs with Many Significant Nodes)**.** *Let $\alpha = 0.5$, Δ be integral and c be given. Then, there are infinitely many n such that there exists a graph on n nodes where the output to* SIGNIFICANTPAGERANKS *on that graph has size $\Omega(\frac{n}{\Delta})$.*

Proof. The construction is a variant of the one used in the proof of Theorem 7.8. The graph is made of $\frac{n}{(3\Delta+1)}$ identical copies of an undirected star graph on $d+1 = 3\Delta$ nodes. An easy calculation with the PageRank equations shows that each hub has PageRank of $\Delta + \frac{1}{3}$ and each leaf has PageRank of $\frac{2}{3} + \frac{1}{(9\Delta-3)} \leq 1$. The number of nodes with PageRank at least Δ is therefore $\frac{n}{d} = \Omega(\frac{n}{\Delta})$.

□

7.6 Other Applications

In this chapter we will focus on other applications of our algorithm for SIGNIFICANTMATRIXCOLUMNS, beyond PageRank approximation.

We will start with a few definitions. Let R_i be a a random walk that starts at i and independently terminate at each time step with probability α. Let $E_{\text{stop}}(i,j)$ denote the event of the walk R_i hits j and then terminate. Let the event $F_{i,j}$ be some event associated with the $(i,j)^{th}$ matrix entry. Assume that $F_{i,j}$ can be decided in $O(t)$ time, where t is the length of the walk before termination. We will be interested in matrices where the $(i,j)^{th}$ matrix entry is the probability that both $E_{\text{stop}}(i,j)$ and $F_{i,j}$ occurred. For example, one may be interested in random walks that visit a specific node k sometime before visiting j and terminating.

Claim 7.9. *Let M be a matrix such that $\forall i,j : M(i,j) = \Pr(E_{stop}(i,j)$ and $F(i,j))$. Then one can solve* SIGNIFICANTMATRIXCOLUMNS *on the input matrix M in time $\tilde{O}(\frac{n}{\Delta})$.*

Proof. The proof closely follows that of Theorem 10. In that proof we have shown that the $(i,j)^{th}$ matrix entry is exactly the hitting probability of a restarting random walk at j. We then developed a combined additive and multiplicative approximation based on simulation of the random walk and the understanding that such walk has an exponential chance to decay as a function of time. One can clearly estimate $M(i,j)$, where $M(i,j) = \Pr(E_{\text{stop}}(i,j)$ and $F(i,j))$, by also checking in each simulation that event $F_{i,j}$ occurred on top of checking that the walk hit node j, which can be decided in the length of the walk. We can then use these estimates, exactly as done in Theorem 10, to complete the proof: the additive approximation follows from the fact that the walk still needs to survive the same number of steps before termination, and the multiplicative bound follows from the Chernoff bounds as before. □

We now turn to ask whether we could use the algorithms developed in this chapter to estimate other eigenvectors of the PageRank system. We start with a simple observation.

Observation 7.10. *Let M be a non-negative matrix such that for entries (i,j): $M(i,j) = b \cdot \Pr(E_{stop}(i,j))$, where $b > 1$. Then one can solve* SignificantMatrixColumns *on the input matrix M in time $\tilde{O}(\frac{nb}{\Delta})$.*

Proof. The proof is immediate: run SignificantMatrixColumns on the matrix M/b with threshold $\Delta' = \frac{\Delta}{b}$. Multiply the output by b. $\square$

We note that for this observation to hold it is important to have an input matrix which is exactly b times a hitting-probability matrix. Consider for example a matrix where all entries equal $1/n$, but one "special" entry (i,j) has value n (thinking of $b = n^2$). Any algorithm that runs in $\tilde{O}(n/\Delta)$ time (including ours) would surly miss sampling the entries from the "special" row and would therefore produce poor approximations to the j^{th} column.

Interestingly, observation 7.6 can be used for providing approximations to other eigenvectors of the PageRank system. Recall that for PageRank computation we would like to approximate a probability vector u with the property

$$u = \alpha \cdot \mathbf{e}_n + (1-\alpha) u \cdot D^{-1} A(G),$$

where $\mathbf{e}_n$ is the n-place row vector of all 1's, $0 < \alpha < 1$ is a teleportation probability constant, and D is a diagonal matrix with the out-degree of v at entry (v,v). This can be written as:

$$u = u\left(\alpha J_n + (1-\alpha) \cdot D^{-1} A(G)\right),$$

where J_n is the $n \times n$ matrix of all 1s.

Namely, the PageRank vector can be thought of as a left eigenvector of the above matrix, corresponding to eigenvalue "1". Since the matrix is right-stochastic, "1" is the largest eigenvalue of it and one might want to solve the system for other, real eigenvalues $\lambda \neq 0$ assuming they exist and are known:

$$\lambda u = u\left(\alpha J_n + (1-\alpha) \cdot D^{-1} A(G)\right).$$

By manipulating the equation and writing $\beta = \frac{1-\alpha}{\lambda}$, one gets

$$u\left(I - \beta D^{-1} A(G)\right) = \frac{\alpha}{\lambda + \alpha - 1}(1-\beta)\mathbf{e}_n.$$

Now if $\lambda > (1-\alpha)$ then $\beta < 1$ and so one can invoke Theorem 7.10 (with b taken as $\frac{\alpha}{\lambda+\alpha-1} > 1$) to get:

Corollary 7.11. *Let $b = \frac{\alpha}{\lambda+\alpha-1}$. Then on can solve* SIGNIFICANTMATRIXCOLUMNS *on the PageRank system corresponding to a given eigenvalue $(1-\alpha) < \lambda < 1$ in time $\tilde{O}(\frac{nb}{\Delta})$.*

Part III

Appendix

Appendix A

Tools from Probability Theory

Here we state several fundamental results from probability theory that are used throughout the dissertation.

A.1 Chernoff Bounds

The Chernoff bounds, also known as Chernoff-Hoeffding bounds, provide concentration results for the sum of independent Bernoulli random variable around its mean. These bounds are credited to Chernoff [34] and Hoeffding [61].

Lemma A.1. *(Multiplicative Chernoff Bound) Let* $X = \sum_{i=1}^{n} X_i$ *be a sum of independent (but not necessarily identical) Bernoulli random variables. Then,*

1. *For* $0 < \lambda < 1$,

$$\begin{aligned} Pr[X < (1-\lambda)E[X]] &< \exp(-\tfrac{\lambda^2}{2}E[X]) \\ Pr[X > (1+\lambda)E[X]] &< \exp(-\tfrac{\lambda^2}{4}E[X]). \end{aligned}$$

2. *For* $\lambda \geq 1$,

$$\Pr[X > (1+\lambda)E[X]] < \exp(-\frac{\lambda E[X]}{3}).$$

3. *For any constant* $\Delta \geq (1+\lambda)E[X]$,

$$Pr[X > \Delta] < \begin{cases} \exp(-\frac{\lambda^2}{4} \cdot \frac{\Delta}{1+\lambda}) & \textit{if } 0 < \lambda < 1 \\ \exp(-\frac{\lambda}{3} \cdot \frac{\Delta}{(1+\lambda)}) & \textit{if } \lambda \geq 1 \end{cases}$$

Proof. The case of $0 < \lambda < 1$ is standard and a proof can be found, for example, in chapter 4 of [94]. For any λ, it is also shown therein that

$$\Pr[X > (1+\lambda)\mu n] \leq \left(\frac{e^{\lambda}}{(1+\lambda)^{(1+\lambda)}} \right)^{\mu n}.$$

Now for $\lambda \geq 1$,

$$\frac{e^{\lambda}}{(1+\lambda)^{(1+\lambda)}} < \exp\left(-\frac{\lambda^2}{2+\lambda}\right) \leq \exp\left(-\frac{\lambda}{3}\right),$$

and the second claimed item follows.

We now prove the last claimed item. Assume that $\frac{\Delta}{(1+\lambda)} - E[X] > 0$ (otherwise the proof follows immediately from part 1). Define $k = \lceil \frac{\Delta}{(1+\lambda)} - E[X] \rceil$ and $Y = \sum_{i=1}^{n+k} Y_i$, where for $1 \leq i \leq n$, $Y_i = X_i$ and for $n < i \leq n+k$, Y_i are independently distributed Bernoulli random variables with expectation $(\frac{\Delta}{(1+\lambda)} - E[X])/k$ each. Note that $k \geq 1$, Y_i are indeed Bernoulli random variables as $0 < (\frac{\Delta}{1+\lambda} - E[X])/k \leq 1$, and that $E[Y] = E[X] + (\frac{\Delta}{(1+\lambda)} - E[X]) = \frac{\Delta}{(1+\lambda)}$. Now,

$$Pr(X > \Delta) = Pr(X > (1+\lambda)\Delta/(1+\lambda)) \leq$$

$$Pr(Y > (1+\lambda)\Delta/(1+\lambda)) < \begin{cases} \exp(-\frac{\lambda^2}{4} \cdot \frac{\Delta}{1+\lambda}) & \text{if } \lambda < 1 \\ \exp(-\frac{\lambda}{3} \cdot \frac{\Delta}{(1+\lambda)}) & \text{if } \lambda \geq 1 \end{cases}$$

The next to last inequality follows from the fact that Y first-order stochastically dominates X, and the last inequality follows from parts 1 and 2 of the lemma. □

Lemma A.2. *(Additive Chernoff Bound) Let* $X = \sum_{i=1}^{n} X_i$ *be a sum of independent (but not necessarily identical) Bernoulli random variables. Then for* $\lambda > 0$,

$Pr[X < E[X] - \lambda] < \exp(-2\lambda^2/n)$.

$Pr[X > E[X] + \lambda] < \exp(-2\lambda^2/n)$.

Lemma A.2 is standard and a proof can be found, for example, in chapter 1 of [39].

A.2 Concentration of Geometric Random Variables

The following lemma gives concentration result for a sum of i.i.d. Geometric random variables around its mean.

Lemma A.3. *(Concentration of Geometric Random Variables) Let Y_i be n i.i.d. Geometric random variables. Define $Y = \sum_{i=1}^{n} Y_i$. Then for $\lambda > 0$,*

$$Pr[Y > (1+\lambda)E[Y]] \leq \exp(-2\lambda^2 n).$$

Proof. Denote $\mu = E[Y_1]$ and define $W(n,p)$ to be the a random variable for the number of independent Bernoulli experiments, with bias $p = \frac{1}{\mu}$ each, to get n successes. Denote by $B(t,p)$ a Binomial random variable on a sequence of t trials and success probability p on each trial. First, by definition, Y is identically distributed to $W(n,p)$. Next, it easily follows that

$$Pr[W(n,p) \geq t] = Pr[B(t,p) \leq n]\ , \tag{1}$$

see for example exercise 2.4 in [39]. Now set $t = \lceil (1+\lambda)\mu n \rceil$. Then using equation (1) and the integrality of $W(n,p)$ we get,

$$Pr[Y > (1+\lambda)\mu n] = Pr[W(n,p) > (1+\lambda)\mu n] = Pr[W(n,p) \geq t] =$$

$$Pr[B(t,p) \leq n] = Pr[B(t,p) \leq (1+\lambda)n - \lambda n].$$

As $E[B(t,p)] = tp \geq (1+\lambda)n$, we get,

$$Pr[B(t,p) \leq n] \leq Pr[B(t,p) \leq E[B(t,p)] - \lambda n].$$

Last, by lemma A.2 we get,

$$Pr[B(t,p) \leq E[B(t,p)] - \lambda n] \leq \exp(-2((\lambda n)^2/n) = \exp(-2\lambda^2 n).$$

□

Bibliography

[1] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[2] Noga Alon, Ronitt Rubinfeld, Shai Vardi, and Ning Xie. Space-efficient local computation algorithms. In *SODA*, pages 1132–1139, 2012.

[3] Reid Andersen, Christian Borgs, Jennifer T. Chayes, John E. Hopcroft, Vahab S. Mirrokni, and Shang-Hua Teng. Local computation of pagerank contributions. *Internet Mathematics*, 5(1):23–45, 2008.

[4] Reid Andersen, Fan R. K. Chung, and Kevin J. Lang. Local graph partitioning using pagerank vectors. In *FOCS*, pages 475–486, 2006.

[5] 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*, 45, 2007.

[6] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. Fast incremental and personalized pagerank. *PVLDB*, 4(3):173–184, 2010.

[7] Eytan Bakshy, Brian Karrer, and Lada A. Adamic. Social influence and the diffusion of user-created content. In *ACM EC*, pages 325–334, 2009.

[8] Egon Balas. The prize collecting traveling salesman problem: II. polyhedral results. *Networks*, 25(4):199–216, 1995.

[9] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[10] József Beck. An algorithmic approach to the lovász local lemma. i. *Random Struct. Algorithms*, 2(4):343–366, 1991.

[11] Oren Ben-Zwi, Danny Hermelin, Daniel Lokshtanov, and Ilan Newman. Treewidth governs the complexity of target set selection. *Disc. Opt.*, 8(1):87–96, 2011.

[12] András A. Benczúr and David R. Karger. Approximating *s*-*t* minimum cuts in $\tilde{o}(n^2)$ time. In *STOC*, pages 47–55, 1996.

[13] Itai Benjamini and Oded Schramm. Recurrence of distributional limits of finite planar graphs. *Electronic Journal of Probability*, 6(23), 2001.

[14] Itai Benjamini, Oded Schramm, and Asaf Shapira. Every minor-closed property of sparse graphs is testable. In *STOC*, pages 393–402, 2008.

[15] Pavel Berkhin. Survey: A survey on pagerank computing. *Internet Mathematics*, 2(1), 2005.

[16] Shishir Bharathi, David Kempe, and Mahyar Salek. Competitive influence maximization in social networks. In *WINE*, pages 306–311, 2007.

[17] Béla Bollobás. Mathematical results on scale-free random graphs. *in Handbook of Graphs and Networks: From the Genome to the Internet*, 2003.

[18] Béla Bollobás and Oliver Riordan. The diameter of a scale-free random graph. *Combinatorica*, 24(1):5–34, 2004.

[19] Béla Bollobás, Oliver Riordan, Joel Spencer, and Gábor E. Tusnády. The degree sequence of a scale-free random graph process. *Random Struct. Algorithms*, 18(3):279–290, 2001.

[20] Christian Borgs, Michael Brautbar, Jennifer T. Chayes, Sanjeev Khanna, and Brendan Lucier. The power of local information in social networks. In *WINE*, pages 406–419, 2012.

[21] Christian Borgs, Michael Brautbar, Jennifer T. Chayes, and Brendan Lucier. Influence maximization in social networks: Towards an optimal algorithmic solution. *CoRR*, abs/1212.0884, 2012.

[22] Christian Borgs, Michael Brautbar, Jennifer T. Chayes, and Shang-Hua Teng. A sublinear time algorithm for pagerank computations. In *WAW*, pages 41–53, 2012. Journal version to appear in *Internet Mathematics*.

[23] Mickey Brautbar and Michael Kearns. Local algorithms for finding interesting individuals in large networks. In *Innovations in Computer Science (ICS)*, pages 188–199, 2010.

[24] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

[25] J. J. Brown and P. H. Reingen. Social ties and word of mouth referral behavior. *Journal of Consumer Research*, 14(3):350–362, 1987.

[26] S. Carmi, S. Havlin, S. Kirkpatrick, Y. Shavitt, and E. Shir. A model of internet topology using k-shell decomposition. *PNAS*, pages 11150–11154, 2007.

[27] Damon Centola and Michael Macy. Complex contagions and the weakness of long ties. *American Journal of Sociology*, 113(3):702–734, 2007.

[28] Meeyoung Cha, Alan Mislove, and P. Krishna Gummadi. A measurement-driven analysis of information propagation in the flickr social network. In *WWW*, pages 721–730, 2009.

[29] Bernard Chazelle, Ronitt Rubinfeld, and Luca Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM J. Comput.*, 34(6):1370–1379, 2005.

[30] Ning Chen. On the approximability of influence in social networks. In *SODA*, pages 1029–1037, 2008.

[31] Wei Chen, Chi Wang, and Yajun Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *KDD*, pages 1029–1038, 2010.

[32] Wei Chen, Yajun Wang, and Siyu Yang. Efficient influence maximization in social networks. In *KDD*, pages 199–208, 2009.

[33] Wei Chen, Yifei Yuan, and Li Zhang. Scalable influence maximization in social networks under the linear threshold model. In *ICDM*, pages 88–97, 2010.

[34] Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23(4):493–507, 1952.

[35] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.

[36] P. Dodds and D. Watts. Universal behavior in a generalized model of contagion. *Phys Rev Lett*, 92(21):218701, 2007.

[37] Benjamin Doerr, Mahmoud Fouz, and Tobias Friedrich. Social networks spread rumors in sublogarithmic time. In *STOC*, pages 21–30, 2011.

[38] Pedro Domingos and Matthew Richardson. Mining the network value of customers. In *KDD*, pages 57–66, 2001.

[39] D. P. Dubhashi and A. Panconesi. *Concentration of Measure for the Analysis of Randomised Algorithms*. Cambridge University Press, 2009.

[40] D. Easley and J. Kleinberg. *Networks, Crowds, and Markets, reasoning about a Highly Connected World.* Cambridge University Press, 2010.

[41] Matthias Englert, Anupam Gupta, Robert Krauthgamer, Harald Räcke, Inbal Talgam-Cohen, and Kunal Talwar. Vertex sparsifiers: New results from old techniques. In *APPROX-RANDOM*, pages 152–165, 2010.

[42] Eyal Even-Dar and Asaf Shapira. A note on maximizing the spread of influence in social networks. *Inf. Process. Lett.*, 111(4):184–187, 2011.

[43] Christos Faloutsos, Kevin S. McCurley, and Andrew Tomkins. Fast discovery of connection subgraphs. In *KDD*, pages 118–127, 2004.

[44] Uriel Feige. A threshold of $ln(n)$ for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[45] Linton C. Freeman. Centrality in social networks conceptual clarification. *Social Networks*, 1(3):215–239, 1978.

[46] George Giakkoupis and Thomas Sauerwald. Rumor spreading and vertex expansion. In *SODA*, pages 1623–1641, 2012.

[47] George Giakkoupis and Nicolas Schabanel. Optimal path search in small worlds: dimension matters. In *STOC*, pages 393–402, 2011.

[48] John R. Gilbert, Cleve Moler, and Robert Schreiber. Sparse matrices in matlab: design and implementation. *SIAM J. Matrix Anal. Appl.*, 13(1):333–356, January 1992.

[49] G. Goel and J. Gustedt. Bounded arboricity to determine the local structure of sparse graphs. In *WG*, pages 159–167, 2006.

[50] J. Goldenberg, B. Libai, and E. Mulle. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing Letters*, pages 221–223, 2001.

[51] Oded Goldreich. Introduction to testing graph properties. In *Property Testing*, pages 105–141, 2010.

[52] Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002.

[53] Daniel Golovin and Andreas Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *J. Artif. Intell. Res. (JAIR)*, 42:427–486, 2011.

[54] Manuel Gomez-Rodriguez, Jure Leskovec, and Andreas Krause. Inferring networks of diffusion and influence. *TKDD*, 5(4):21, 2012.

[55] Sanjeev Goyal and Michael Kearns. Competitive contagion in networks. In *STOC*, pages 759–774, 2012.

[56] M. Granovetter. Threshold models of collective behavior. *American Journal of Sociology*, (83):1420–1443, 1978.

[57] Sudipto Guha and Samir Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20(4):374–387, 1998.

[58] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.

[59] Avinatan Hassidim, Jonathan A. Kelner, Huy N. Nguyen, and Krzysztof Onak. Local graph partitions for approximation and testing. In *FOCS*, pages 22–31, 2009.

[60] T.H Haveliwala. Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. In *Trans. Knowl. Data Eng*, volume 15(4), pages 784–796, 2003.

[61] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[62] Piotr Indyk and David P. Woodruff. Optimal approximations of the frequency moments of data streams. In *STOC*, pages 202–208, 2005.

[63] M. Jackson. *Social and Economic Networks*. Princeton University Press, 2008.

[64] Glen Jeh and Jennifer Widom. Scaling personalized web search. In *WWW*, pages 271–279, 2003.

[65] Qingye Jiang, Guojie Song, Gao Cong, Yu Wang, Wenjun Si, and Kunqing Xie. Simulated annealing based influence maximization in social networks. In *AAAI*, 2011.

[66] David S. Johnson, Maria Minkoff, and Steven Phillips. The prize collecting steiner tree problem: theory and practice. In *SODA*, pages 760–769, 2000.

[67] Viggo Kann. On the approximability of np-complete optimization problems, 1992. PhD thesis, Department of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm.

[68] Ravi Kannan and Santosh Vempala. Spectral algorithms. *Foundations and Trends in Theoretical Computer Science*, 4(3-4):157–288, 2009.

[69] Ravindran Kannan. Spectral methods for matrices and tensors. In *STOC*, pages 1–12, 2010.

[70] Michael J. Kearns. *Computational complexity of machine learning*. ACM distinguished dissertations. MIT Press, 1990.

[71] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *KDD*, pages 137–146, 2003.

[72] David Kempe, Jon M. Kleinberg, and Éva Tardos. Influential nodes in a diffusion model for social networks. In *ICALP*, pages 1127–1138, 2005.

[73] Myunghwan Kim and Jure Leskovec. The network completion problem: Inferring missing nodes and edges in networks. In *SDM*, pages 47–58, 2011.

[74] Masahiro Kimura and Kazumi Saito. Tractable models for information diffusion in social networks. In *PKDD*, pages 259–271, 2006.

[75] Jon M. Kleinberg. The small-world phenomenon: an algorithm perspective. In *STOC*, pages 163–170, 2000.

[76] Joachim Kneis, Daniel Mölle, and Peter Rossmanith. Partial vs. complete domination: t-dominating set. In *SOFSEM (1)*, pages 367–376, 2007.

[77] Jochen Könemann, Ojas Parekh, and Danny Segev. A unified approach to approximating partial covering problems. *Algorithmica*, 59(4):489–509, 2011.

[78] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. What cannot be computed locally! In *PODC*, pages 300–309, 2004.

[79] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. The price of being near-sighted. In *SODA*, pages 980–989, 2006.

[80] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Local computation: Lower and upper bounds. *CoRR*, abs/1011.5470, 2010.

[81] Jure Leskovec, Lada A. Adamic, and Bernardo A. Huberman. The dynamics of viral marketing. *TWEB*, 1(1), 2007.

[82] Jure Leskovec and Eric Horvitz. Planetary-scale views on a large instant-messaging network. In *WWW*, pages 915–924, 2008.

[83] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne M. VanBriesen, and Natalie S. Glance. Cost-effective outbreak detection in networks. In *KDD*, pages 420–429, 2007.

[84] Jure Leskovec, Mary McGlohon, Christos Faloutsos, Natalie S. Glance, and Matthew Hurst. Patterns of cascading behavior in large blog graphs. In *SDM*, 2007.

[85] D. Liben-Nowell and J. Kleinberg. Tracing information flow on a global scale using internet chain-letter data. *PNAS*, 105(12):4633–4638, 2008.

[86] Nathan Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992.

[87] László Lovász. *Large Networks and Graph Limits*, volume 60 of *Colloquium publications*. American Mathematical Society, Providence, R.I., 2012.

[88] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[89] Yishay Mansour, Aviad Rubinstein, Shai Vardi, and Ning Xie. Converting online algorithms to local computation algorithms. In *ICALP (1)*, pages 653–664, 2012.

[90] Michael Mathioudakis, Francesco Bonchi, Carlos Castillo, Aristides Gionis, and Antti Ukkonen. Sparsification of influence networks. In *KDD*, pages 529–537, 2011.

[91] Andrew McGregor. Graph mining on streams. In *Encyclopedia of Database Systems*, pages 1271–1275. 2009.

[92] Stephen E. Morris. Contagion. *Review of Economic Studies*, 67:57–78, 2000.

[93] Elchanan Mossel and Sebastien Roch. On the submodularity of influence in social networks. In *STOC*, pages 128–134, 2007.

[94] Rajeev Motwani and Prabhaker Raghavan. *Randomized Algorithms*. Cambridge University Pres, 1995.

[95] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

[96] Moni Naor and Larry Stockmeyer. What can be computed locally? In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, STOC '93, pages 184–193, New York, NY, USA, 1993. ACM.

[97] C. St.J. A. Nash-Williams. Decomposition of finite graphs into forests. *Journal of the London Mathematical Society*, 39(1):12, 1964.

[98] M.E.J. Newman. *Networks: An Introduction*. Oxford University Press, 2010.

[99] Huy N. Nguyen and Krzysztof Onak. Constant-time approximation algorithms via local improvements. In *FOCS*, pages 327–336, 2008.

[100] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Stanford University 1998.

[101] Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal. Using pagerank to characterize web structure. *Internet Mathematics*, 3(1):1–20, 2006.

[102] Michal Parnas and Dana Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theor. Comput. Sci.*, 381(1-3):183–196, 2007.

[103] David Peleg. Local majorities, coalitions and monopolies in graphs: a review. *Theor. Comput. Sci.*, 282(2):231–257, 2002.

[104] Matthew Richardson and Pedro Domingos. Mining knowledge-sharing sites for viral marketing. In *KDD*, pages 61–70, 2002.

[105] Everett M. Rogers. *Diffusion of Innovations*. Free Press, 5th edition, 2003.

[106] Ronitt Rubinfeld and Asaf Shapira. Sublinear time algorithms. *SIAM Journal on Discrete Math*, 25:1562–1588, 2011.

[107] Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. Fast local computation algorithms. In *ITCS*, pages 223–238, 2011.

[108] Eldar Sadikov, Montserrat Medina, Jure Leskovec, and Hector Garcia-Molina. Correcting for missing data in information cascades. In *WSDM*, pages 55–64, 2011.

[109] Yaron Singer. How to win friends and influence people, truthfully: influence maximization mechanisms for social networks. In *WSDM*, pages 733–742, 2012.

[110] Petr Slavík. Improved performance of the greedy algorithm for partial cover. *Inf. Process. Lett.*, 64(5):251–254, 1997.

[111] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *STOC*, pages 81–90, 2004.

[112] Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM J. Comput.*, 40(4):981–1025, 2011.

[113] Daniel A. Spielman and Shang-Hua Teng. A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning. *SIAM J. Comput.*, 42(1):1–26, 2013.

[114] Kyoungwon Suh, Yang Guo, James F. Kurose, and Donald F. Towsley. Locating network monitors: complexity, heuristics, and coverage. In *INFOCOM*, pages 351–361, 2005.

[115] Jukka Suomela. Survey of local algorithms. *ACM Comput. Surv.*, 45(2):24, 2013.

[116] The Facebook Blog, October 2012. `http://www.facebook.com/zuck/posts/10100518568346671`.

[117] WorldWideWebSize.com: Daily Estimated Size of the World Wide Web. `http://www.worldwidewebsize.com`.

[118] Yu Wang, Gao Cong, Guojie Song, and Kunqing Xie. Community-based greedy algorithm for mining top-k influential nodes in mobile social networks. In *KDD*, pages 1039–1048, 2010.

[119] Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *FOCS*, pages 222–227, 1977.