MODELS AND ALGORITHMS FOR PAGERANK SENSITIVITY

A DISSERTATION
SUBMITTED TO THE INSTITUTE OF COMPUTATIONAL
AND MATHEMATICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

David Francis Gleich
September 2009

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Michael Saunders) Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Chen Greif)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Amin Saberi)

Approved for the University Committee on Graduate Studies.

ABSTRACT

The PageRank model helps evaluate the relative importance of nodes in a large graph, such as the graph of links on the world wide web. An important piece of the PageRank model is the teleportation parameter α. We explore the interaction between α and PageRank through the lens of sensitivity analysis. Writing the PageRank vector as a function of α allows us to take a derivative, which is a simple sensitivity measure. As an alternative approach, we apply techniques from the field of uncertainty quantification. Regarding α as a random variable produces a new PageRank model in which each PageRank value is a random variable. We explore the standard deviation of these variables to get another measure of PageRank sensitivity. One interpretation of this new model shows that it corrects a small oversight in the original PageRank formulation.

Both of the above techniques require solving multiple PageRank problems, and thus a robust PageRank solver is needed. We discuss an inner-outer iteration for this purpose. The method is low-memory, simple to implement, and has excellent performance for a range of teleportation parameters.

We show empirical results with these techniques on graphs with over 2 billion edges.

PREFACE

For a fun preface to this document, I decided to reflect back on the first PageRank code I ever wrote. Program 1 is a poor implementation of PageRank for all the following reasons:

it uses an ugly `;` to finish `end` commands;
it uses a terribly inefficient construction to normalize `A`;
it has poor documentation of the parameter `e`;
it doesn’t use compensated summation;
it doesn’t include a teleportation parameter; and
it doesn’t compute PageRank.

These elements constitute a terribly inauspicious start to my work with PageRank. I can only hope I’ve learned since writing this first code.

Program 1 – An incorrect PageRank implementation. The first PageRank program I wrote, trivially altered to fit on this page. It’s incorrect. See program 2 for a working version.

```
function pgv = pagerank(g, e)
% PAGERANK Compute pagerank for graph g.
% pgv = pagerank(g) returns the pagerank vector for graph g with default parameters
% pgv = pagerank(g, e) returns the pagerank vector for graph g with epsilon = e
n = get(g, ’size’);
A = g.adj;
% normalize A by the degrees of each node
degs = degree(g);
[i j v] = find(A);
for ii = 1:length(i)
  A(i(ii), j(ii)) =  v(ii)./degs(i(ii));
end;
init = 1/n*ones(n, 1);
pgv = init;
delta = Inf;
while (delta > e)
  pgvn = A*pgv;
  d = norm(pgv, 1) - norm(pgvn, 1);
  pgvn = pgvn + d*init;
  delta = norm(pgvn - pgv);
  pgv = pgvn;
end;
```

ACKNOWLEDGMENTS

A thesis reflects an individual's work. It should show a capacity for novel research at the brink of human knowledge. It should demonstrate deep insight into a problem. And all of this should translate into a document written by one.

The singularity of this view feels wrong. Research does not exist in a vacuum. During my time at Stanford, I've had the pleasure of talking and working with many others. This thesis is my work, but the influence of all these interactions is considerable.

It will be difficult to acknowledge everyone. Nevertheless, I will try.

Before getting started, I need to acknowledge my funding. In my second year, I worked under the Department of Energy Advanced Simulation and Computing (DOE ASC) grant. NSF grant CCF-0430617 funded my research during my third year. Support for my fourth and fifth years was generously provided by Microsoft through a Microsoft Research/Live Labs Fellowship. Finally, two quarters of summer work were under the Computational Approaches to Digital Stewardship grant from the Library of Congress.

First, I'd like to thank the members of my thesis committees. Art Owen, Hector Garcia Molina, Chen Greif, Amin Saberi, and Michael Saunders patiently listened to my defense and gave me useful feedback. Also, Chen, Amin, and Michael read through this document and provided essential input. I'm extremely grateful for the time each member spent working to make this thesis better.

Second, the ICME staff has been amazing. Seth Tornborg and Brian Tempero—the ICME computer guys—always ensured I had the latest computers. This relationship helped when I needed to experiment on some of these big graphs. Deb Michaels, Priscilla Williams, and Mayita Romero kept ICME running during my graduate career. They occasionally arranged pizza or other treats. Finally, I would never have graduated from Stanford without Indira Choudhury. Indira helped me wade through the bureaucracy. She made it easy for me to focus on research.

Third, the ICME students are equally amazing. We had bar nights, barbecues, beach trips, baseball, and basketball—and those are just the b's. The students provided a social component that made ICME a great place to spend five years studying. Of course, the students are extremely sharp. It was great

to be able to sit at a white board with almost anyone and debate merits and demerits of almost anything in computational mathematics.

Some of my closest friends and classmates are: Esteban Arcaute, Michael Atkinson, Andrew Bradley, Paul Constantine, Adam Guetz, Nick Henderson, Jeremy Kozdon, Chris Maes, Nicole Taheri, Ying Wang, and Nick West. In particular, Jeremy Kozdon, Michael Atkinson, Paul Constantine and I survived five years together in the four different offices (Durand 110, 102, 112, and Terman 396). We all developed different research, different advisers, and different girlfriends (or wifes!), but managed to stick together until graduation. Chris Maes and I ended up as roommates for two years. Again, I cannot sufficiently thank these folks for all the help they've provided. I can only hope I wasn't too much of a burden.

Let me continue with my research collaborators. Without them, it would be difficult to imagine fitting as much material into the thesis as I have.

It's quite likely that Leslie Ward, Erin Bodine, and the rest of the Harvey Mudd Overture Services clinic team are directly (or perhaps only indirectly) responsible for the topic of this thesis. The present topic clearly follows a trajectory from our initial investigations into web search. I owe them my thanks for sending me down this path—although, at points I may have used less flattering and more colorful language to describe my thesis.

Reid Andersen, Pavel Berkhin, Kevin Lang, Peter Glynn, and Sebastiano Vigna all helped shape my view of the PageRank problem through our discussions. I'm extremely grateful to Marzia Polito for working through the intricacies of personalized PageRank and HostRank with me over the course of a summer internship. Likewise, Amy Langville was incredibly helpful during my visits to her in Charleston.

Abraham Flaxman merits a special acknowledgment. You see, he gave me the data about α I always wanted. It is gratifying to receive data that fits initial assumptions so cleanly; see figure 4.6 for more about his data.

Chen Greif also merits mention. Not only did he fly down to Stanford to attend my dissertation defense in person, he also arranged for three trips to Vancouver, and a post-doctoral position too. He's been a great research collaborator and always makes sure I address every little detail.

At Stanford, Margot Gerritsen and Amin Saberi have been constant fixtures throughout my education. Soon after I came to Stanford, Margot asked me to prepare a homework assignment on PageRank for her numerical linear algebra class. Both advise the computational approaches to digital stewardship group at Stanford, which has funded part of my research. I'm grateful for their involvement in my research and general education at Stanford.

And then there's Paul. For those reading this who don't know, Paul Constantine was a fellow Ph.D. student at Stanford. Over a fateful beer, we realized that the random alpha PageRank ideas in chapter 4 are possible and make sense. Subsequently, this research blossomed into almost two theses: his thesis on parameterized matrix equations and mine on PageRank. It's been a wonderful collaboration, and this thesis would be rather different without these pieces.

During my research, I've had three mentors: Leonid Zhukov, Gene H. Golub, and Michael Saunders.

Leonid was my first academic mentor, although he was officially my mentor as an intern at Overture and Yahoo!. He never seemed to quite agree with my philosophy on ordering authors (I try to use alphabetical ordering) and kept claiming that I was making him "less important" on papers (we all know where Z lies in the alphabet). In this case, he's first because the order is chronological. Leonid patiently set me straight on matters of matrix computations. He took me through latent semantic indexing (LSI), principle component analysis (PCA), PageRank (well, he started me on PageRank), spectral clustering, semi-definite programming, and visualization, all with explanations that were clear, correct, and physically motivated. I'm indebted to him for this introduction.

Gene H. Golub was my first Ph.D. adviser. He opened doors for me during my stint as his student. Our initial work on a least squares problem never went anywhere useful. But we soared with PageRank—literally, in fact. Gene's stature ensured I could attend a workshop on linear algebra for the web at Schloss Dagstuhl in Germany. With a few small exceptions, everyone working on PageRank was there. It was a tremendous opportunity to learn about PageRank from the experts. Later, Gene helped me arrange a visit to the numerical analysis group at Oxford while he was on his sabbatical there.

Gene was famous for claiming that becoming his student was a lifetime commitment. He died shortly into my fourth year. I'm still saddened that my tenure under him was so short. I believe he would have loved the interplay between PageRank and quadrature that developed in chapter 4. He was a great adviser.

After Gene's death, Michael Saunders graciously stepped in to fill Gene's big shoes (both literally and figuratively). He's done a marvelous job of guiding me through the end stages of a Ph.D. I'm grateful to him for his help during this difficult transition. Michael ensured that everything I did was perfect. In particular, he poured over this thesis to make it immeasurably better. There are no dangling participles or split infinitives left!

Finally, we come to my family and friends. These individuals helped support me throughout all the research.

Let's begin with the Fletchers. Hugh and Jane Fletcher opened their doors to me on many occasions. In one case, I was homeless and they sheltered me. In many other cases, I was hungry and they fed me (grilled filet mignon, yum!); I was thirsty and they gave me drink (California Pinot Noir and Napa Cabernet, yum yum!). Lindsey Fletcher, their daughter, also became one of my few "non-technical" friends. She wouldn't let us talk "math" around her—and sometimes, that's refreshing. Of course, I need to specially acknowledge Les Fletcher, their son. After living with him for four years: two in graduate school and two at Harvey Mudd College, he knows me better than almost anyone (and relishes reminding my wife he's lived with me longer). I suspect he identified the topic of this thesis long before I settled on it. He was always there to listen to any of my crazy ideas. I often think of the Fletchers as a second family that happens to live down the street in Mountain View.

Next, Debbie Heimowitz and Jason Azicri are two unique friends I had the pleasure of meeting at Stanford. They've been constant founts of wisdom, advice, and silliness.

Anything I write about my parents is woefully inadequate for their influence on me. For this thesis, this influence manifests itself at least two ways. First, Kristin Leiferman (my mother—I've used my parents first names) ensured that I knew how to edit a manuscript. She helped review many of my school reports. Each was always carefully edited. Over time, I believe my writing improved as the number of edits decreased. To wit, she only found two mistakes in an early draft of this thesis. Second, Gerald Gleich always expatiated the details of his medical research during any free time, including the middle of the family dinner (much to the chagrin of some other siblings squeamish about such things). His stories drove me to pursue my own research and the excitement that follows from discovering something new!

Last, but not at all least, is Laura Bofferding (my wife). Laura has been my savior. She's quick to try and help whenever I need something. She's put up with far too many long nights at the office while I've been working on this thesis and she's never complained about it once. Also, her carrot cake is a tremendous incentive to get my work done so she'll let me eat it. I'm beholden to her for all her help and support.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF PROGRAMS

LIST OF ALGORITHMS

I think this is the beginning of a beautiful friendship.

—Rick from Casablanca

1 INTRODUCTION

Page et al. [1999] is a technical report about a new way to identify important web pages. The system is known as PageRank™ and models a way to take advantage of how people might browse the web. Let us leave the technical details of PageRank until chapter 2 and explore the setting and application background for this thesis in this chapter. For those familiar with PageRank, please proceed to section 1.5.

1.1 PAGERANK AND WEB SEARCH

The PageRank system [Page et al., 1999] is widely viewed as a critical reason for the success of the Google™ search engine [Langville and Meyer, 2006a]. To understand where PageRank fits within web search, let's examine a rather high-level view of a search engine. This description is meant to convey useful knowledge of how web search might work, and we fully admit that many details are highly simplified. Good references for more information are Witten et al. [1999]; Baeza-Yates and Ribeiro-Neto [1999]; Manning et al. [2008].

CRAWLING Before any search engine can return search results, it must have a set of potential results. The process of identifying all web pages is known as *crawling* by analogy to the way a spider "crawls" around a web by supporting itself on "edges" or "links." On the web, hyperlinks connect pages, and crawlers discover new pages via these links. This process may continue indefinitely as pages constantly change and new pages spring to life. Nevertheless, the set of pages for a web search engine is almost entirely determined by its crawlers.[1]

TEXT ANALYSIS To retrieve results, the engine must "understand" the text on a page. Text analysis or term indexing produces a database listing every single page containing a term.[2] These databases are huge and contain more than just individual words. Phrases, such as "new york," are also indexed. One way to think about these databases is like a huge index at the back of a truly huge book.

[1] In a recent lecture, a Google engineer reported that their crawling operation can often detect changes on some pages, and update the search engine, *within minutes* [Dean, 2009].

[2] As of early May 2009, claims to know about 22,450,000,000 pages that contain the word "a" and 26,350,000,000 pages that contain the digit "1."

LINK ANALYSIS In addition to textual analysis of each web page, search engines examine the hyperlinks, more often just called links, between the pages to extract information about the quality of the page. PageRank is one such measure. It identifies pages that have a large number of votes from pages that also have a large number of votes. We'll see more about PageRank in a moment. Link analysis also includes measures that examine how much a web page looks like a genuine web page instead of a potential "spam" webpage. *The contributions of this thesis fall into the category of new link analysis algorithms and metrics.*

RANKING REGRESSION The text and link analyses are used in a ranking function that determines the final order of the results. This function is often generated by a machine learning approach, which selects features that produce rankings corresponding to the pages people think are most important. The details of these ranking functions are not readily available: they are the real trade secrets of the search engines.

PRODUCE RANKINGS Of course, "the last step" is to integrate all the previous analyses on a set of documents that contain the words in the query—and to produce this list in milliseconds.

PageRank, then, is one of a series of link analysis algorithms employed by a search algorithm to help with a single component of the search engine.[3]

It is time to define PageRank informally. Consider someone browsing the web. At every page, the surfer either follows a link on the page to another page or does something else.[4] When following a link, without any other information, the surfer picks a link from the page at random and follows that one. When "doing something else," the surfer moves to a random page on the web and restarts the surfing. A technical term for "doing something else" is *teleporting* or *resetting* by analogy with teleporting to a location after entering a URL or resetting the browser by closing and opening it. To generate a simple model, we assume both of these behaviors even though they may seem ridiculous. When stated mathematically, this random surfer model is called a *Markov chain* because the behavior of the surfer only depends upon the current page and not the history of previous pages.

Let α be the probability that the surfer follows a link; then $1 - \alpha$ is the probability that the surfer "does something else." Pictorially, the model is illustrated by figure 1.1, where the big black circle represents the current page. With probability α the surfer follows one of the three links, represented as arrows, to the bottom three pages, represented as circles. With probability $1 - \alpha$ the surfer moves to one of the pages in the blob.[5] Jumping into the

[3] As a personal aside, I hope this setup properly contextualizes my research for those who ask if my plan is to "start the next Google" once I tell them I'm working on PageRank. I'm not.

[4] Modern web-browsers open the possibility of doing both of these activities through the use of tabbed browsing.

[5] For those deeply familiar with the PageRank model, this explanation is slightly inaccurate but it contains the essential pieces. We will formalize everything in due course.

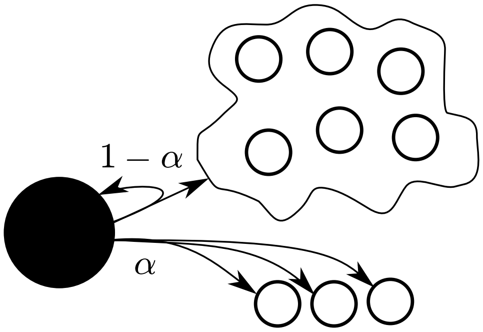


Figure 1.1 – A pictorial illustration of the PageRank model.

blob is also known as *teleporting* or *resetting* because it can move the surfer anywhere in the web.

Now imagine that we let the surfer run for a long time. The *PageRank* of a page is the probability of finding the surfer at that page as the surfing time becomes infinite. A key assumption behind PageRank is that pages where we are more likely to find the random surfer are more important pages and thus we can view the PageRank as a measure of the page's importance. In reality, the PageRank problem is expressed as a mathematical equation that generates a number between 0 and 1 for each page. We'll delve into the mathematics of PageRank in chapter 2.

The focus of my thesis is investigating what happens when varying the α parameter.

For a preview, let's look only at the pages in Wikipedia [Various, 2009b]. In this case, the surfer ignores all the links to the actual source material outside of the Wikipedia system. Table 1.1 shows the titles of the 10 pages with highest PageRank in Wikipedia. A few things change with the α parameter. When α is 0.5, the pages are focused on countries, whereas when α is 0.99, the pages are focused on the encyclopedia infrastructure. For example, the page "Category:Wikipedia administration" describes the administration of the encyclopedia itself.

1.2 PAGERANK ON A GRAPH

Although PageRank models a random surfer on the web and computes the probability of finding this surfer at any given page, the output of PageRank is a number for each page on the web related to its importance. In the previous section, we explained PageRank where a surfer sits at a current page in the web. The proper mathematical abstraction of the linked nature of the web is a

α = 0.50	α = 0.85	α = 0.99
United States	United States	C:Contents
C:Living people	C:Main topic classif.	C:Main topic classif.
France	C:Contents	C:Fundamental
Germany	C:Living people	United States
England	C:Ctgs. by country	C:Wikipedia admin.
United Kingdom	United Kingdom	P:List of portals
Canada	C:Fundamental	P:Contents/Portals
Japan	C:Ctgs. by topic	C:Portals
Poland	C:Wikipedia admin.	C:Society
Australia	France	C:Ctgs. by topic

Table 1.1 – Highest PageRank pages in Wikipedia. The set of pages in Wikipedia with the highest PageRank scores for three values of the parameter α. The prefix "C:" denotes a category page and any term with a period is abbreviated.

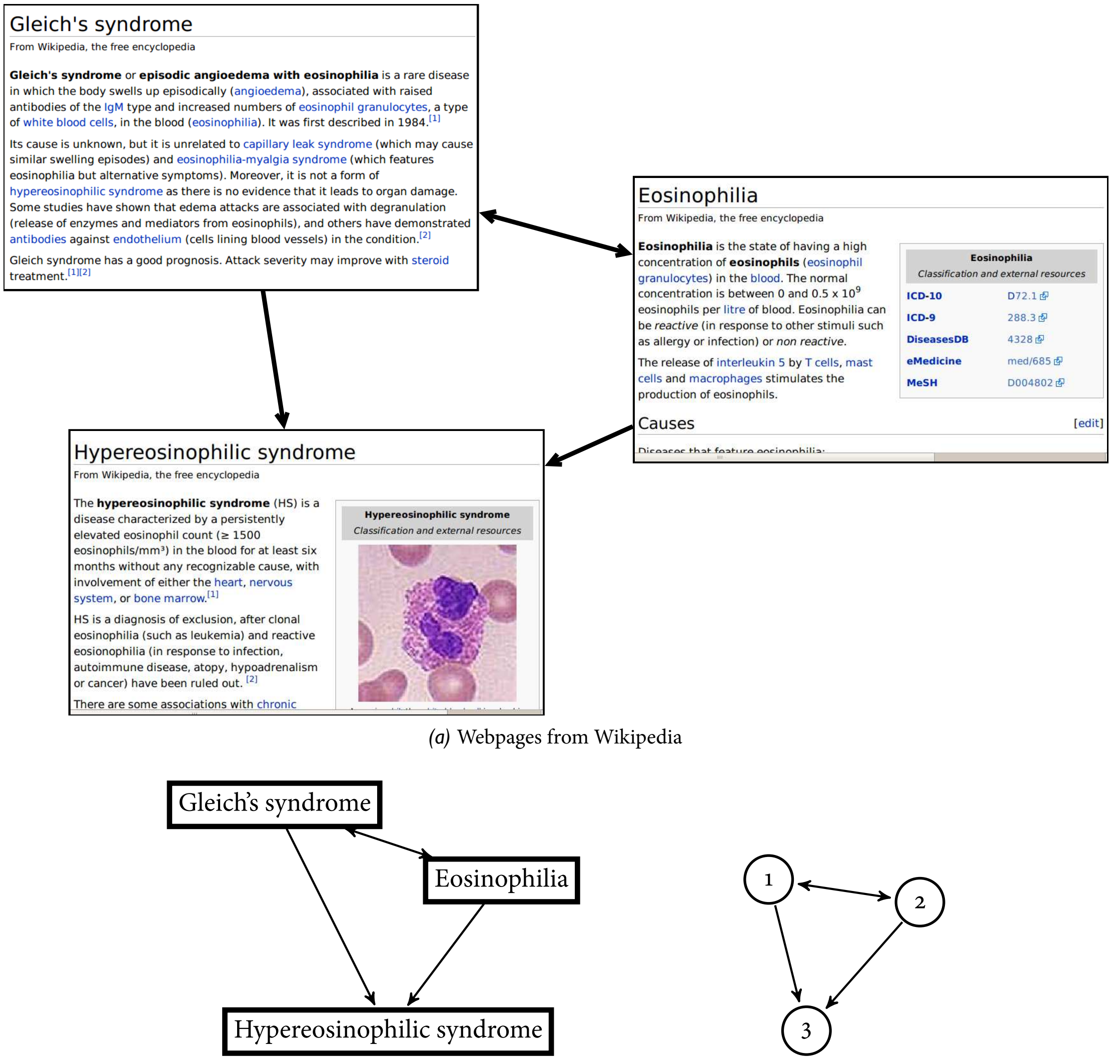
graph. A graph is a set of nodes and connections. For the web, the nodes are web pages and the connections are the links. Figure 1.2 shows this relationship pictorially on a small subset of Wikipedia. This mathematical abstraction is relevant because it means that PageRank exists for any graph and not just the web graph. PageRank on a graph produces an importance score for each node, and this places PageRank amongst a class of *network analysis* techniques [Brandes and Erlebach, 2005] known as centrality measures or indices [Koschützki et al., 2005].

Instead of looking at a "random surfer" on the web, the non-web PageRank models a random walk on the graph. The behavior of the walk is the same as the random surfer: with probability α the walk continues along an edge of the graph and with probability $1 - \alpha$ the walk jumps to a random node in the graph. Random walks are a common technique to analyze graphs, with a rich history predating PageRank.

Because they apply when looking at PageRank on a general graph, the results of this thesis are not limited to web search. See section 4.8.3 for one example, but do read the background material first.

1.3 VARIATIONS ON THE PAGERANK THEME

PageRank is a simple model for the random surfer. After hearing about this model, someone invariably approaches and asks: "Why doesn't the surfer do . . . , instead?" Sometimes, the answer is: "So-and-so looked at that already, they found . . . " Often, it's: "That's a great idea! It hasn't been looked at yet." The key thing to remember is that PageRank is *just a model*. Of course there are potential improvements to the model and there have been many proposed extensions to the model. We cover some of them in the next chapter. An important extension is the *personalized PageRank* model, in which the surfer does not randomly restart browsing anywhere on the web after choosing



Gleich's syndrome

From Wikipedia, the free encyclopedia

Gleich's syndrome or **episodic angioedema with eosinophilia** is a rare disease in which the body swells up episodically (angioedema), associated with raised antibodies of the IgM type and increased numbers of eosinophil granulocytes, a type of white blood cells, in the blood (eosinophilia). It was first described in 1984.[1]

Its cause is unknown, but it is unrelated to capillary leak syndrome (which may cause similar swelling episodes) and eosinophilia-myalgia syndrome (which features eosinophilia but alternative symptoms). Moreover, it is not a form of hypereosinophilic syndrome as there is no evidence that it leads to organ damage. Some studies have shown that edema attacks are associated with degranulation (release of enzymes and mediators from eosinophils), and others have demonstrated antibodies against endothelium (cells lining blood vessels) in the condition.[2]

Gleich syndrome has a good prognosis. Attack severity may improve with steroid treatment.[1][2]

Eosinophilia

From Wikipedia, the free encyclopedia

Eosinophilia is the state of having a high concentration of **eosinophils** (eosinophil granulocytes) in the blood. The normal concentration is between 0 and 0.5 x 10^9 eosinophils per litre of blood. Eosinophilia can be *reactive* (in response to other stimuli such as allergy or infection) or *non reactive.*

The release of interleukin 5 by T cells, mast cells and macrophages stimulates the production of eosinophils.

Eosinophilia	
Classification and external resources	
ICD-10	D72.1
ICD-9	288.3
DiseasesDB	4328
eMedicine	med/685
MeSH	D004802

Causes [edit]

Diseases that feature eosinophilia:

Hypereosinophilic syndrome

From Wikipedia, the free encyclopedia

The **hypereosinophilic syndrome** (HS) is a disease characterized by a persistently elevated eosinophil count (≥ 1500 eosinophils/mm³) in the blood for at least six months without any recognizable cause, with involvement of either the heart, nervous system, or bone marrow.[1]

HS is a diagnosis of exclusion, after clonal eosinophilia (such as leukemia) and reactive eosionophilia (in response to infection, autoimmune disease, atopy, hypoadrenalism or cancer) have been ruled out. [2]

There are some associations with chronic

Hypereosinophilic syndrome
Classification and external resources

(a) Webpages from Wikipedia

(b) A labeled graph

(c) A graph

Figure 1.2 – Relationships between web pages and graphs. The linking relationships between web pages define a mathematical object called a graph. Mostly, we'll look at what happens with structures like figure (c), where the information about the pages is abstracted to a unique number that identifies the page. In this case, 1 identifies the page "Gleich's syndrome" on Wikipedia.

not to follow a link. Rather, the surfer in this new model restarts at one of only a few pages. If the pages relate to one person, then the resulting PageRank vector is called a personalized PageRank vector. If the pages are topically related, then the vector may be called a topic-specific PageRank vector. In either case, *what* the surfer does when not following links on a page determines the *type* of pages that are important.

The random surfer model is but one interpretation of the PageRank model. In Higham [2005], it is shown that PageRank is related to playing pinball. Place a ball at a page on the web. Suppose that the ball moves according on these rules:

> with probability β, it stays put;
>
> with probability γ, it moves to a page that links to the current page; and
>
> with probability $1 - \beta - \gamma$, the game ends.

Although this model does not use α, the probabilities β and γ are defined from the PageRank parameters. See the paper for the definitions, as they are needlessly tangential. The average length of time that this game lasts is the PageRank of that page.

Another view of PageRank is as "Google Juice."[6] Consider pages with high PageRank values. This means that a random surfer is quite likely to be on these pages. Why? There are two possibilities: either many pages link to them, or a few high PageRank pages—where the surfer is already likely to be found—link to them. It is this latter case that gives rise to the notion of "Google Juice." If a page already has a high PageRank value, it can contribute its influence to another page. Thus, another interpretation of PageRank is a system where importance "flows" along links between pages.

We hope that these alternate views of the PageRank model provide further intuition for the PageRank method. Having multiple viewpoints on a problem is an important aspect of any numerical research. What is obvious or trivial from one perspective is often difficult to perceive from another. In some cases, these viewpoints provide the intuition necessary to close important open problems.

1.4 OTHER USES FOR PAGERANK

So far, we've seen that PageRank on the web models where we find a random surfer, that this process generalizes to a centrality measure on an

[6] We feel obliged to document this term. Please see `http://c2.com/cgi/wiki?GoogleJuice` for a corroborating definition.

arbitrary graph, and that there are many ways to change and interpret the PageRank model. There are still other uses for PageRank.

CLUSTERING The problem of clustering is to find ways to divide a graph into pieces by separating the nodes into cohesive groups. One approach is to find a set of strongly related nodes, call that a group, remove it from the graph, and repeat until the graph is empty. PageRank helps find a group of strongly related nodes, as Andersen et al. [2006] demonstrate. They show that a modified personalized PageRank, where the surfer only resets to a single page (called the target), produces a group of pages near the target. Further, they show that they can use a customized PageRank algorithm to compute these groups of nodes extremely quickly.

SPORTS RANKING PageRank also helps to rank sports teams. In two recent contributions, both Langville [2009] and Govan et al. [2008] extend ideas originating from Keener [1993] by using PageRank to compute a ranking of sports teams. Instead of a random surfer, they posit a "fair-weather fan" who picks favorites between teams based on some set of statistics. A simple case is to use the win and loss records between teams to create a graph where two teams are connected from the loser to the winner. The PageRank vector of that graph gives a useful ordering of the teams.

BIOINFORMATICS The GeneRank method by Morrison et al. [2005] produces lists of genes that may be relevant to a microarray experiment. Because some of the entries in the microarray data are noisy, the experiment may not reveal *all* of the interesting genes activated in different conditions. GeneRank uses a surfer over known relationships between genes, where the surfer "restarts" with probability proportional to the activation level of the genes in the experiment. Its output is a set of genes "near" the genes with high activation levels. The goal of the method is to aid researchers working with microarray datasets to see which other genes are nearby using known relationships.

Freschi [2007] uses a similar idea, which they call ProteinRank, to predict protein functions. The nodes of the relevant graph are proteins, and two proteins are connected if they physically interact, which yields a protein-protein interaction (PPI) network. Instead of gene annotations from a microarray experiment driving the random surfer behavior in the reset step, the ProteinRank algorithm uses known functional behaviors to direct the surfer to the relevant portion of the graph.

NETWORK ALIGNMENT Singh et al. [2007] introduce the IsoRank algorithm[7] to identify functionally similar nodes between two different graphs. The random surfer analogy does not directly apply in this case, and the "Google Juice" formulation is more appropriate. Instead of working on the graphs in the original problem, IsoRank uses a combined graph based on all pairs of vertices. The PageRank of this new graph helps identify which nodes are potential mates. Section 5.6.5 discusses this application further.

1.5 CONTRIBUTIONS

At this point, we have motivated the PageRank problem from its original use as a description of a random web surfer and shown how the same model yields many different applications. The aspect of PageRank we address in this thesis is the *sensitivity* of the PageRank vector with respect to the parameter α.

Some aspects of the sensitivity of PageRank were previously understood. For example, as the value α gets close to 1, the random surfer is typically following links in the graph. The impact of the *graph* is exaggerated in this case. Also, as α gets close to 1, the vector may change rapidly. We'll review the relevant background material for sensitivity in chapter 2.

A simple approach to investigating the sensitivity of the PageRank vector is to look at the derivative with respect to α, which we explore in chapter 3.[8] Algorithms to compute the derivative of PageRank were already known, but we propose a new algorithm that can use any existing PageRank solver without modification. The results from our algorithm (algorithm 1) on the pages from Wikipedia are shown in table 1.2.

Table 1.2 – Pages in Wikipedia with the largest derivative. Pages in Wikipedia with the largest derivative, by value, not by magnitude.

$\alpha = 0.50$	$\alpha = 0.85$	$\alpha = 0.99$
United States	C:Main topic classif.	C:Main topic classif.
C:Living people	C:Contents	C:Contents
United Kingdom	C:Fundamental	C:Fundamental
Race in the US. Census	C:Wikipedia admin.	C:Wikipedia admin.
C:Ctgs. by country	C:Ctgs. by topic	C:Ctgs. by topic
France	C:Society	C:Society
England	Por:List of portals	Por:List of portals
Canada	C:Articles	C:Articles
Germany	Por:Contents/Portals	Por:Contents/Portals
World War II	C:Ctgs. by location	C:Ctgs. by location
List of sovereign states	C:Categories	C:Ctgs. by country

[7] There is another algorithm called IsoRank based on isotonic regression [Zheng et al., 2008].

[8] Although PageRank was described as a random surfer model, it also has a nice expression as a function of the parameter α. Our investigation is of the derivative of this function with respect to α.

After investigating the derivative, we develop a new model for PageRank in chapter 4 along with a significant new approach to sensitivity analysis. Instead of using a derivative, which just measures the effect of small change in α, this new model examines an approach based on the variance of the PageRank vector as a function of its parameter over a wide range of values of α. Another interpretation shows that this sensitivity measure corresponds to replacing α in PageRank with a random variable and studying the standard deviation of a set of random PageRank variables. Hence, we call our new method RAPr—**R**andom **A**lpha **P**age**r**ank. Figure 1.3 illustrates how sensitivity works in our new model. Our best-performing algorithm on this problem only involves computing PageRank vectors.

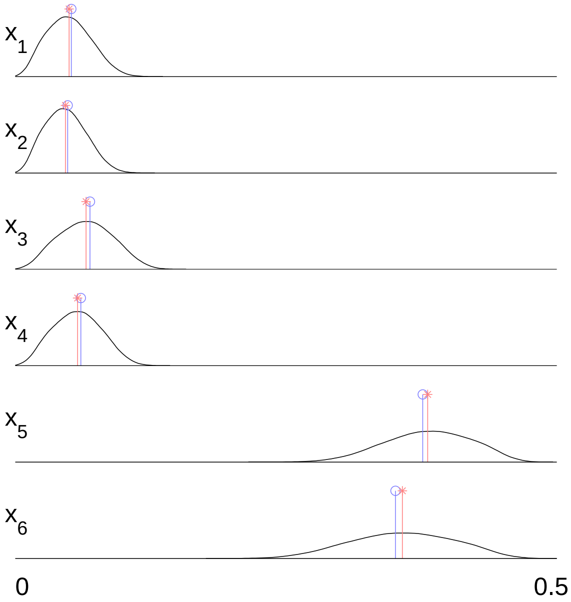


Figure 1.3 – PageRank with a random variable as the teleportation parameter. This plot shows how likely each PageRank value between 0 and 0.5 is when the constant parameter α is replaced with a random variable. Rather than computing the value of PageRank at the circle stem points, we look at the entire range of values it might take. The width of the major portion of the curves is the new sensitivity parameter. See chapter 4 for more information about this plot and the details of the model.

In both of the previous contributions (the derivative and the standard deviation), the key algorithmic step is a method to compute a PageRank vector. In chapter 2, we review existing algorithms to compute PageRank. For many of the experiments, we used an inner-outer algorithm from Gray et al. [2007]. The performance of this algorithm is excellent and the algorithm is reliable; see figure 1.4. Subsequently, we developed a parallel implementation of this inner-outer iteration for the PageRank problem along with extensions of the idea to other PageRank solvers and new convergence analysis (chapter 5).

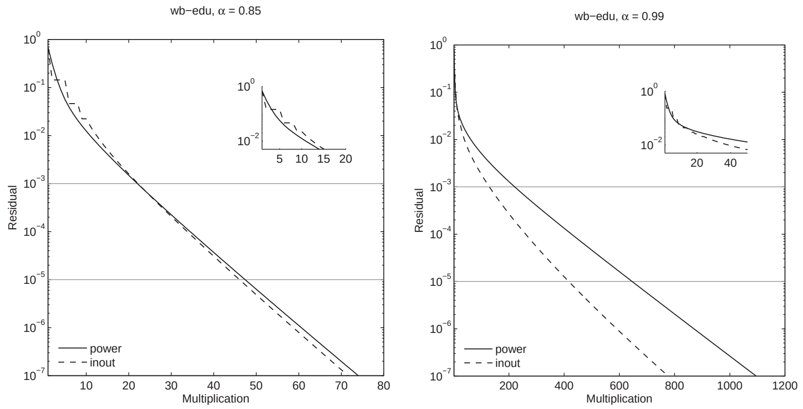


Figure 1.4 – Performance of the PageRank algorithm. The inner-outer method clearly outperforms the power method on wb-edu with $\alpha = 0.85$ on the left and $\alpha = 0.99$ on the right. The small inner figure shows the convergence in the first few iterations. See chapter 5 for details on the inner-outer method, and table 2.2 for information about the wb-edu graph.

1.5.1 *Publications*

My original paper on the sensitivity analysis arising from the derivative of PageRank was written up in the proceedings from a Dagstuhl workshop [Gleich et al., 2007]. I still intend to submit these results to a journal, perhaps in combination with a few other ideas.

A preliminary version of the random α PageRank idea was presented at the 2007 Workshop on Algorithms for the Web Graph [Constantine and Gleich, 2007]. We extended that manuscript to a journal-length article, which we plan to submit this summer.

Although the inner-outer algorithm was originally proposed in [Gray et al., 2007], those authors graciously included my contributions into the current journal manuscript, which will be published in the SIAM Journal of Scientific Computing.[9]

[9] A draft is available from www.cs.ubc.ca/~greif/Papers/gggl2009.pdf.

SUMMARY

In the remainder of this thesis, each chapter ends with a short summary of the major contributions. These discussions are intended to aid the browsability of the thesis. Reading only the introduction and summary of a chapter gives a flavor for the contributions. Readers should be able to determine if a chapter is worth their time from these.

In this chapter, PageRank and this thesis are presented informally through the random surfer model. From this model, we discuss a few other ways of viewing PageRank and a few other ways of using PageRank. The random surfer model posits a surfer that moves to a linked page with probability α and "does something else" with probability $1 - \alpha$. This thesis focuses on the effect of the α parameter and investigates sensitivity with respect to that parameter.

In the dim background of mind,
we know what we ought to be doing,
but somehow we cannot start.

—William James

2 PAGERANK BACKGROUND

PageRank literature exploded between 2004 and 2005.

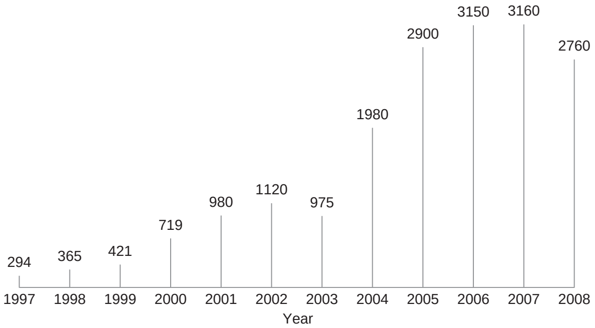


Figure 2.1 – PageRank papers by year. The number of Google Scholar results from the search “pagerank” listed by year. These counts are not accurate publication counts, but should correctly represent the publication trend.

Covering all the literature and background here is impossible. Thus, we focus on the relevant introduction for this thesis. Such a restriction is only partially helpful. PageRank is deceptively complicated and the details matter. It is worth noting that Langville and Meyer [2006a], besides being an excellent reference text for PageRank, has a reasonably comprehensive summary of the literature until 2005.

A big issue with the PageRank literature is that many communities write about the problem and its applications. Among them are numerical analysts [Serra-Capizzano, 2005], theoretical computer scientists [Andersen et al., 2006], information retrieval scientists [Gyöngyi et al., 2004], and even biologists [Morrison et al., 2005] and physicists [Shepelyansky and Zhirov, 2009]! Standard introductions to PageRank involve the following idea: important web pages link to other important web pages [Page et al., 1999; Langville and Meyer, 2006a]. Let s_i be the importance of a page indexed by i. This idea suggests the definition

$$s_i = \sum_{j \text{ links to } i} \frac{s_j}{\text{total links from } j}.$$

Thus page j distributes a fraction of its importance to page i when j links to i. This definition is subsequently adjusted to account for a few difficulties that immediately arise. The prior definition, which is often given as *the* definition of PageRank, lacks *the most distinguishing feature* of PageRank: the adjustments themselves. It is the adjustments that define PageRank, and not the flow of importance. Certainly, the flow of importance is an important aspect of PageRank, but not its defining feature. Establishing this defining feature is where we begin.

2.1 MATRIX COMPUTATION PRELIMINARIES

Before getting to PageRank, we need some notation for operations involving matrices and vectors. Unless otherwise noted, the following conventions hold:

bold capital letter	$\mathbf{A}, \mathbf{G}, \mathbf{P}$	for matrices,
bold lower case letters	$\mathbf{b}, \mathbf{v}, \mathbf{x}$	for vectors,
lower case Greek letters	α, β, γ	for scalars,
subscripted capital letters	A_{ij}, G_{ij}, P_{ij}	for matrix elements,
subscripted lower case letters	b_i, v_i, x_i	for vector elements, and
calligraphic capital letters	$\mathcal{G}, \mathcal{S}$	for graphs and sets,

which is a variation on Householder notation used in a few recent textbooks [Meyer, 2000; Trefethen and Embree, 2005]. The following symbols represent standard matrix or vector operations:

$\mathbf{A}^T, \mathbf{x}^T$	is the transpose of a matrix or vector,
$\mathbf{A}^+$	is the pseudo-inverse of a matrix,
$\mathbf{e}$	is the vector of all ones of appropriate length,
$\lVert\cdot\rVert$	is the 1-norm of a matrix or vector,
$\mathbf{A} \otimes \mathbf{B}$	for the Kronecker product between matrices or vectors, and
$\mathbf{A} \bullet \mathbf{B}$	for the Hadamard, or elementwise, product between matrices or vectors.

Horn and Johnson [1991] have a nice background on the Kronecker and Hadamard products. These are less well known than the standard product operations between matrices and have a few fascinating properties.

Throughout this thesis, $\mathbf{P}$ represents a square, column stochastic matrix. Formally, column stochastic implies $P_{ij} \geq 0$ and $\mathbf{e}^T\mathbf{P} = \mathbf{e}^T$. Taking $\mathbf{P}$ as column stochastic is contrary to the notation in probability, where $\mathbf{P}$ is a row stochastic matrix. The matrix $\bar{\mathbf{P}}$ is a column sub-stochastic matrix (henceforth called a *sub-stochastic matrix*) where

$$(\mathbf{e}^T\bar{\mathbf{P}})_j = \begin{cases} 0 & \bar{P}_{ij} = 0 \text{ for all } i \\ 1 & \bar{P}_{ij} \neq 0 \text{ for some } i. \end{cases} \tag{2.1}$$

We use two definitions consistently in the remainder of the text, except where explicitly noted. One common exception is that $\mathbf{e}$ may also denote an error vector when we discuss approximations to exact solutions.

2.2 THE PAGERANK PROBLEM

Discussing PageRank from both a theoretical and a practical view is hard. There are many slight variants of the PageRank problem, and this section enumerates three of them after introducing the *core* PageRank problem. The distinctions among the variants are important, although this core formulation of the PageRank problem masks them. Hiding the distinctions is mathematically advantageous as most properties of the PageRank problem are preserved for all variants and thus it simplifies analysis.

Figure 2.2 provides guidance for the discussion of the next few sections. This section introduces the mathematical PageRank formulations. The subsections describe the variations.

Without further ado, what distinguishes PageRank? PageRank begins with any idea that defines a stochastic matrix $\mathbf{P}$. Ideally, the importance of items is proportional to the dominant eigenvector

$$\mathbf{P}\mathbf{x} = \mathbf{x}$$

of the stochastic matrix, but this may not be unique. (We'll see that $\mathbf{x}$ is not unique for the "importance" model given in the introduction.) Given *any stochastic matrix* $\mathbf{P}$, PageRank modifies it to produce a new problem with a unique answer. These modifications, then, define PageRank and not the starting stochastic matrix. In a slightly hyperbolic sense, PageRank is a technique to take any cockamamied idea and fix it. The data for the PageRank problem are

$\mathbf{P}$ a column stochastic matrix that defines the transitions of a Markov chain;

α a *teleportation parameter* or *damping parameter*, $0 \leq \alpha < 1$; and

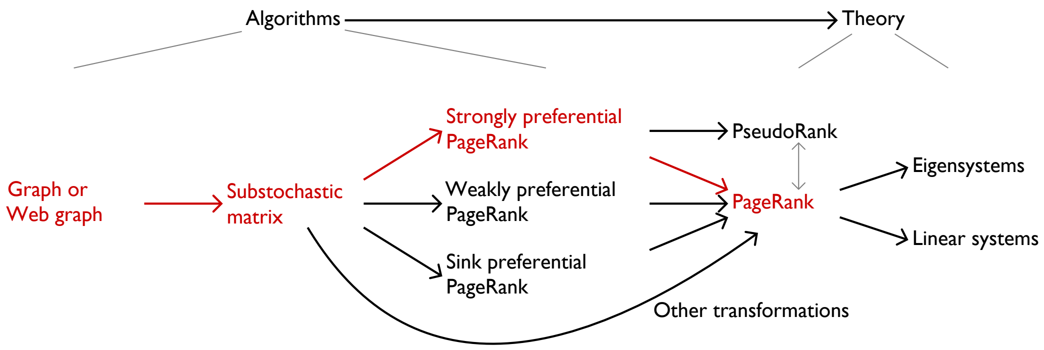


Figure 2.2 – Overview of PageRank formulations. Most derivations of PageRank begin with a graph and proceed through a sub-stochastic matrix and a PageRank variant before getting to the PageRank problem. Instead, starting with the PageRank problem yields a mathematically pure look at the problem. Algorithms and implementations for PageRank often need to take advantage of the graph structure and begin with a graph or sub-stochastic matrix. The bold path illustrates the most common PageRank formulation.

$\mathbf{v}$ a *teleportation distribution*, where $v_i \ge 0$ and $\mathbf{e}^T\mathbf{v} = 1$, also known as the *preference vector*.

To fix the scheme, PageRank modifies it so that it does something predictable from $\mathbf{v}$. The parameter α controls the trade-off between $\mathbf{P}$ and $\mathbf{v}$. Transitions of the PageRank process are given by a modified Markov matrix

$$\mathbf{M} = \alpha \underbrace{\mathbf{P}}_{\text{follow transitions}} + (1-\alpha)\underbrace{\mathbf{v}\mathbf{e}^T}_{\text{reset}} = \mathbf{M}(\alpha, \mathbf{P}, \mathbf{v}). \tag{2.2}$$

Subsequently, we will omit the explicit dependence on the parameters when they are clear from context. Interpreted as a Markov chain, PageRank is a process that follows transitions in the original process $\mathbf{P}$ with probability α or *resets* according to a known distribution over the states with probability $(1-\alpha)$.

In contrast with $\mathbf{P}$, the dominant eigenvector $\mathbf{M}\mathbf{x} = \mathbf{x}$ is always unique. This $\mathbf{x}$ is the PageRank vector (with a slight detail addressed below). Uniqueness of the eigenvector is trivial when $v_i > 0$ and follows from the Perron-Frobenius theorem because $\alpha < 1$ implies that $M_{ij} > 0$. A detail often swept under the rug is what happens when $v_i \ge 0$. Without a completely positive $\mathbf{v}$, $\mathbf{M}$ is no longer irreducible and the simple theorems for a unique eigenvector do not apply. That said, the eigenvector *is still unique* because $\mathbf{M}$ has only a single ergodic class over the set of states reachable from the support of $\mathbf{v}$. Berman and Plemmons [1994, theorem 3.23] justifies this statement with the more general result that all unit eigenvectors of a stochastic matrix are convex combinations of unit eigenvectors of the ergodic classes extended to all states with 0 probability.[1]

PageRank values are also the stationary distribution probabilities for the modified Markov chain, namely the PageRank vector $\mathbf{x}$ is the stationary distribution vector. For this reason, the PageRank vector is a probability distribution vector and has the natural normalization

$$x_i \geq 0, \mathbf{e}^T\mathbf{x} = 1.$$

The discussion thus far is the *eigenvector* definition of PageRank:

$$\mathbf{M}\mathbf{x} = \mathbf{x} \qquad \text{and} \qquad \mathbf{e}^T\mathbf{x} = 1. \tag{2.3}$$

As a probability distribution, the PageRank vector is *also* the solution of the linear system

$$(\mathbf{I} - \alpha\mathbf{P})\mathbf{x} = (1 - \alpha)\mathbf{v}, \tag{2.4}$$

which follows from $\mathbf{e}^T\mathbf{x} = 1$ and $\mathbf{M}\mathbf{x} = \alpha\mathbf{P}\mathbf{x} + (1 - \alpha)\mathbf{v} = \mathbf{x}$. This system is non-singular for all $\alpha < 1$, and $(\mathbf{I} - \alpha\mathbf{P})$ is an M-matrix. We could hardly ask to be luckier! Note that there is no difficulty with non-negative $\mathbf{v}$ for the linear system. Both representations yield quite a bit of flexibility in working with the problem.

We summarize this section with the following definition.

Problem 1 (PageRank). *Given a column stochastic* $\mathbf{P}$, $0 \leq \alpha < 1$, *and a distribution vector* $\mathbf{v}$, *set* $\mathbf{M} = \alpha\mathbf{P} + (1 - \alpha)\mathbf{v}\mathbf{e}^T$. *Solving PageRank is computing or approximating the unique vector* $\mathbf{x}$ *in*

$$\mathbf{M}\mathbf{x} = \mathbf{x} \textit{ and } \mathbf{e}^T\mathbf{x} = 1 \qquad \textit{or} \qquad (\mathbf{I} - \alpha\mathbf{P})\mathbf{x} = (1 - \alpha)\mathbf{v}. \tag{2.5}$$

All the relevant pieces of the PageRank problem are present in this statement. Anything that calls itself PageRank will compute a vector that satisfies this property for some α, $\mathbf{P}$, and $\mathbf{v}$. Hence, this problem is the *core* PageRank problem at the heart of figure 2.2.

2.2.1 *PageRank variations*

An alternative starting point for PageRank is to begin with a sub-stochastic matrix $\bar{\mathbf{P}}$. As discussed in the next section, sub-stochastic matrices often arise from random walk or influence propagation definitions on graphs. For such $\bar{\mathbf{P}}$, there are two established formulations of the PageRank problem: *strongly*

[1] A special case is also not difficult to see. When $v_i \geq 0$ then $\mathbf{M}$ can be symmetrically permuted and partitioned so that $\mathbf{M} = \begin{bmatrix} \mathbf{M}_1 & \\ \mathbf{M}_{12} & \mathbf{M}_2 \end{bmatrix}$, where the spectral radius of $\mathbf{M}_1 < 1$ and $\mathbf{M}_2$ is stochastic and irreducible. The rows and columns in $\mathbf{M}_2$ correspond to all states reachable from the support of $\mathbf{v}$. Solutions to $\begin{bmatrix} \mathbf{M}_1 & \\ \mathbf{M}_{12} & \mathbf{M}_2 \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}$ are unique because $\rho(\mathbf{M}_1) < 1$ implies that $\mathbf{x}_1 = 0$ and also stochasticity and irreducibility of $\mathbf{M}_2$ implies that $\mathbf{x}_2$ is unique.

preferential PageRank and *weakly preferential PageRank* [Boldi et al., 2007]. We also formalize a *sink preferential PageRank* model below. Each formulation corresponds to a different way of converting $\bar{\mathbf{P}}$ into a stochastic matrix and focuses on the columns of the matrix $\bar{\mathbf{P}}$ that are completely 0. The dangling indicator vector $\mathbf{d}$ is 1 for such columns and 0 for columns where $\bar{\mathbf{P}}$ is not completely zero. Formally,

$$\mathbf{d}^T = \mathbf{e}^T - \mathbf{e}^T\bar{\mathbf{P}}, \qquad \mathbf{d}_j = \begin{cases} 1 & \bar{P}_{ij} = 0 \text{ for all } i \\ 0 & \bar{P}_{ij} \neq 0 \text{ for some } i. \end{cases} \tag{2.6}$$

The strongly preferential PageRank problem uses the fully stochastic matrix

$$\mathbf{P}_{\mathbf{v}} = \bar{\mathbf{P}} + \mathbf{v}\mathbf{d}^T, \tag{2.7}$$

where $\mathbf{v}$ is the *same* vector as in the PageRank problem (2.3) or (2.4). To convince ourselves it is a column stochastic matrix, note that $\mathbf{d}^T$ is positive only in the columns that caused $\bar{\mathbf{P}}$ to fail to be stochastic, and so the matrix $\mathbf{P}_{\mathbf{v}}$ will be the matrix $\bar{\mathbf{P}}$ where each 0 column is replaced by $\mathbf{v}$.

Weakly preferential PageRank replaces each 0 column of $\bar{\mathbf{P}}$ with a *different* distribution vector $\mathbf{u}$, and uses the stochastic matrix

$$\mathbf{P}_{\mathbf{u}} = \bar{\mathbf{P}} + \mathbf{u}\mathbf{d}^T, \tag{2.8}$$

where $\mathbf{u}$ is an arbitrary distribution vector with $u_i \geq 0$, $\mathbf{e}^T\mathbf{u} = 1$, and $\mathbf{u} \neq \mathbf{v}$.

Instead of replacing 0 columns of $\bar{\mathbf{P}}$ with a distribution, sink preferential PageRank inserts a 1 into the diagonal for each of these columns, which corresponds to using the stochastic matrix

$$\mathbf{P}_{\mathbf{d}} = \bar{\mathbf{P}} + \operatorname{Diag}[\mathbf{d}], \tag{2.9}$$

where Diag[$\mathbf{d}$] is a diagonal matrix with the entries of d along the diagonal.

NOTA BENE *Unless otherwise noted, we use the strongly preferential PageRank formulation of the problem when* $\bar{\mathbf{P}}$ *is sub-stochastic. This choice is made in most of the literature.*

PSEUDORANK Recall that we defined a PageRank vector with $\mathbf{e}^T\mathbf{x} = 1$. Relaxing that requirement on the strongly preferential PageRank problem yields a vector called *PseudoRank* [Boldi et al., 2007].

Problem 2 (PseudoRank). *A PseudoRank vector* $\mathbf{y}$ *satisfies*

$$(\mathbf{I} - \alpha\bar{\mathbf{P}})\mathbf{y} = \sigma\mathbf{v} \tag{2.10}$$

for $\sigma = n, 1,$ *or* $(1 - \alpha)$.

PageRank and PseudoRank are related by $\mathbf{x} = \mathbf{y}/\mathbf{e}^T\mathbf{y}$. Proving it requires simple substitution. Note that $\sigma = \mathbf{e}^T\mathbf{y} - \alpha\mathbf{e}^T\bar{\mathbf{P}}\mathbf{y}$. Consider

$$(\mathbf{I} - \alpha\mathbf{P})\frac{\mathbf{y}}{\mathbf{e}^T\mathbf{y}} = \frac{(\mathbf{I} - \alpha\bar{\mathbf{P}})\mathbf{y} - \alpha\mathbf{v}\mathbf{d}^T\mathbf{y}}{\mathbf{e}^T\mathbf{y}} = \frac{\sigma - \alpha\mathbf{d}^T\mathbf{y}}{\mathbf{e}^T\mathbf{y}}\mathbf{v} \tag{2.11}$$

$$= \frac{(\mathbf{e}^T\mathbf{y} - \alpha\mathbf{e}^T\bar{\mathbf{P}}\mathbf{y}) - \alpha(\mathbf{e}^T\mathbf{y} - \mathbf{e}^T\bar{\mathbf{P}}\mathbf{y})}{\mathbf{e}^T\mathbf{y}}\mathbf{v} \tag{2.12}$$

$$= (1 - \alpha)\mathbf{v}. \tag{2.13}$$

Many authors define PageRank as PseudoRank [McSherry, 2005; Gyöngyi et al., 2004]. While they share some equivalence, there is an important distinction with regard to the limit when $\alpha \to 1$, and that's discussed in section 2.7.

2.2.2 *PageRank on a graph*

Most derivations of PageRank begin with PageRank on a graph, and most often it is the web graph. For an arbitrary directed graph $\mathcal{G}$ with adjacency matrix $\mathbf{A}$ ($A_{ij} = 1$ if node i has a directed edge to node j, and $A_{ij} = 0$ if there is no edge), the PageRank vector is commonly defined by applying one of the sub-stochastic algorithms to the matrix

$$\bar{\mathbf{P}} = \mathbf{A}^T\mathbf{D}^+, \tag{2.14}$$

where $\mathbf{D}$ is a diagonal matrix with diagonal entries $D_{ii} = (\mathbf{A}\mathbf{e})_i$ = outdegree of node i, and $\mathbf{D}^+$ is the pseudo-inverse [Golub and van Loan, 1996], another diagonal matrix with

$$(\mathbf{D}^+)_{ii} = \begin{cases} 1/D_{ii} & D_{ii} \neq 0 \\ 0 & D_{ii} = 0. \end{cases} \tag{2.15}$$

In the context of web search, each web page corresponds to a node in $\mathcal{G}$, and nodes u and v are connected with a directed edge if the page corresponding to node u links to the page corresponding to node v.

Despite appearances, the setup $\bar{\mathbf{P}} = \mathbf{A}^T\mathbf{D}^+$ does not appear out of thin air. We've seen it before. From the introduction to the chapter: important web pages link to important web pages and their scores

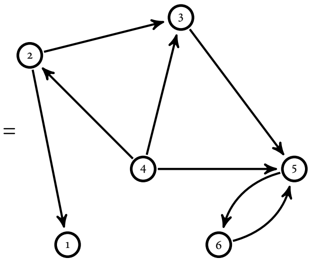
$$s_i = \sum_{j \text{ links to } i} \frac{s_j}{\text{total links from } j}.$$

In matrix form,

$$\mathbf{s} = \bar{\mathbf{P}}\mathbf{s}.$$

We could immediately apply the ideas of section 2.2 except that they require a stochastic matrix. Thankfully, section 2.2.1 tells us how to convert $\bar{\mathbf{P}}$ to $\mathbf{P}$ for PageRank.

But why do we need any of these ideas? Let's work through an example in some detail. For

the graph $\mathcal{G}$ = 

and the matrix $\bar{\mathbf{P}} = \begin{bmatrix} 0 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/3 & 0 & 0 \\ 0 & 1/2 & 0 & 1/3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1/3 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$

there is a single dangling column, so $\mathbf{d} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$. The only eigenvector $\bar{\mathbf{P}}\mathbf{x} = \mathbf{x}$ is $\mathbf{x} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1/2 & 1/2 \end{bmatrix}^T$. And so $\mathbf{x}$ is unique, but not that useful. Suppose that $\mathbf{v} = \begin{bmatrix} 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \end{bmatrix}^T$ and $\alpha = 5/6$. Using the strongly preferential PageRank model yields the PageRank vector

$$\mathbf{x} = \begin{bmatrix} 0.049 & 0.041 & 0.059 & 0.032 & 0.425 & 0.394 \end{bmatrix}^T$$

after rounding. It is the same two nodes that are the most important, but node 5 is more important than node 6. Also we learn that node 3 is the most important among the rest.

Even with all of these choices, there are still details left. Should *self-loops* in $\mathcal{G}$ be retained? Should multiple-links between pages be respected? At some point, we need to end the enumeration of PageRank variants. The answers to these questions ultimately depend upon the application.

2.2.3 *Other variants*

Our list here is not exhaustive. Langville and Meyer [2006a, section 8.4] define a *bounce-back* stochastic matrix from any sub-stochastic matrix where each edge into a 0 column produces a new vertex to return the Markov chain to the previous state. Another correction addresses a theoretical concern with the limit as $\alpha \to 1$ [Vigna, 2005]. Each variant yields a PageRank problem described completely by problem 1. It is for these reasons that the PageRank problem really is

$$(\mathbf{I} - \alpha \mathbf{P})\mathbf{x} = (1 - \alpha)\mathbf{v}.$$

2.2.4 *Historical note*

Astonishingly, an early paper on ranking the nodes of a social network proposed a method with surprising similarities to PageRank [Katz, 1953]. After renormalization and notation adjustment, the Katz model is

$$(\mathbf{I} - \alpha \mathbf{W}^T)\mathbf{x} = \alpha \mathbf{W}^T \mathbf{e} \tag{2.16}$$

with $\alpha = 0.5$.

2.2.5 *Summary of important properties*

We conclude our discussion of the PageRank problem with a summary of properties, not all of which have been explicitly mentioned so far:

the PageRank problem is problem 1 (page 17);

the PageRank vector $\mathbf{x}$ has unit sum ($\mathbf{e}^T\mathbf{x} = 1$);

the PageRank linear system is $(\mathbf{I} - \alpha \mathbf{P})\mathbf{x} = (1 - \alpha)\mathbf{v}$;

the PageRank eigensystem is $\mathbf{M}\mathbf{x} = \mathbf{x}$ where $\mathbf{M} = \alpha \mathbf{P} + (1 - \alpha)\mathbf{v}\mathbf{e}^T$;

the matrix $(\mathbf{I} - \alpha \mathbf{P})$ is a nonsingular M-matrix;

the PageRank variants are strongly preferential, weakly preferential, and sink preferential;

the PageRank vector $\mathbf{x}$ can be defined via a Neumann series when $\alpha < 1$, $\mathbf{x} = \sum_{n=0}^{\infty} (\alpha \mathbf{P})^n \mathbf{v}$;

the maximum eigenvalue of $\mathbf{M}$ is 1 and it has a unique eigenvector $\mathbf{x}$; and

the second largest eigenvalue of $\mathbf{M}$ is no larger than α [Eldén, 2004].

See Langville and Meyer [2006a] for formal derivations of these results.

2.3 CONNECTIONS WITH LANGVILLE AND MEYER'S NOTATION

Langville and Meyer [2006a] established a different set of notation for the PageRank problem. There are many relationships between our notations, but we prefer to separate PageRank from a web-ranking context.

The biggest difference between our notations is the column vs. row orientation of the matrices. Langville and Meyer use row stochastic matrices and then write the PageRank equations as

$$\boldsymbol{\pi}^T(\alpha \mathbf{S} + (1-\alpha)\mathbf{E}) = \boldsymbol{\pi}^T.$$

Such notation closely follows standard probability and Markov chain theory, although most of that literature also utilizes row vectors instead of the column vector, which would make it

$$\boldsymbol{\pi}(\alpha \mathbf{S} + (1-\alpha)\mathbf{E}) = \boldsymbol{\pi}.$$

Instead, our notation is designed to avoid unnecessary transpose symbols and retain column vectors. Thus we write

$$(\alpha \mathbf{P} + (1-\alpha)\mathbf{v}\mathbf{e}^T)\mathbf{x} = \mathbf{x}.$$

Table 2.1 summarizes the symbol relationships between our symbols.

2.4 ALGORITHMS

So far, we have seen how PageRank is formulated as the linear equation

$$(\mathbf{I} - \alpha \mathbf{P})\mathbf{x} = (1-\alpha)\mathbf{v}$$

or the eigensystem

$$\mathbf{M}\mathbf{x} = \mathbf{x}.$$

Both of these formulations correspond to extremely well studied problems: solving linear systems and computing eigenvectors, respectively [Golub and van Loan, 1996]. So why do we need to write about algorithms for PageRank? The property that makes PageRank an interesting problem is that the matrices are HUGE! Recent reports about the size of the web establish that there are over one trillion (1,000,000,000,000) pages [Alpert and Hajaj, 2008]—although many are duplicates—and at least one search engine has crawled over 180 billion pages [Cuil, 2009]. Algorithms to compute PageRank, then,

Table 2.1 – Relationship to Langville and Meyer's PageRank notation. A *Rosetta stone* to translate my notation for readers familiar with Langville and Meyer's popular book *Google's PageRank and Beyond*.

Their symbol	Our symbol	Discussion
$\mathbf{a}$	$\mathbf{d}$	Our initiation to PageRank was through Kamvar's papers in which **d** is the dangling node vector.
$\mathbf{E}$	$\mathbf{e}\mathbf{v}^T$	We always make the matrix **E** explicit to emphasize its rank-1 structure.
$\mathbf{G}$	$\mathbf{M}^T$	Here the **G** stands for the Google matrix. We use the application neutral **M** for the PageRank *modified* matrix.
$\mathbf{H}$	$\bar{\mathbf{P}}^T$	Langville and Meyer use the symbol **H** to suggest the hyperlink matrix without any sort of correction. We use $\bar{\mathbf{P}}$ to suggest "$\mathbf{P}^{-}$" and that $\bar{\mathbf{P}}$ needs an eventual correction to a stochastic matrix.
$\mathbf{L}$	$\mathbf{A}$	Using **A** follows common notation in graph theory where A is the adjacency matrix; L hints at the link matrix.
$\boldsymbol{\pi}$	$\mathbf{x}$	To keep consistent with solving linear systems ($\mathbf{Ax} = \mathbf{b}$) and eigenvalue problems ($\mathbf{Ax} = \lambda\mathbf{x}$), we use **x** to denote the unknown PageRank variables in the problem.
$\mathbf{S}$	$\mathbf{P}^T$	The symbol **S** suggests stochastic, whereas we use **P** to denote a standard Markov chain transition matrix.

must cope with matrices derived from such graphs. In this case, classic iterative algorithms for linear systems and eigenvectors actually perform well, partly because they use only one or two working vectors. In order to take advantage of additional structure in the strongly personalized PageRank problem, we derive all the algorithms for a *sub-stochastic matrix* $\bar{\mathbf{P}}$ and a graph. In terms of figure 2.2, this structure is why *algorithms* lie before the theory. To discuss specializations of these algorithms on the PageRank problem properly, we need one surprising preliminary discussion.

2.4.1 *Important implementation details*

Let's begin with a silly question: given a list of positive numbers, how should we compute their sum? Given this task, most people would produce a simple code that resembles the following four lines of MATLAB.

```
sumx = 0;
for i=1:numel(x)
   sumx = sumx + x(i);
end
```

This routine correctly sums the numbers in exact arithmetic. When computed in floating-point arithmetic, every individual sum contributes an error of at most ε because

$$\mathrm{fl}(\texttt{sumx + x(i)}) = (1+\delta)(\texttt{sumx + x(i)}), \qquad |\delta| \le \varepsilon$$

for some machine ε. After n of these sums, the output `sumx` has error bounded by $n\varepsilon$ where n is the number of summands [Higham, 2002]. For a large PageRank problem $n \sim 10^9$ and for double precision arithmetic $\varepsilon = 2.2 \cdot 10^{-16}$, in which case $n\varepsilon$ is *not small*. Computing sums is a common operation in PageRank algorithms. We need a better algorithm.

Nearly since the dawn of computation, this problem has been studied [Higham, 2002]. (In fact, the reference for the remainder of this section is Higham [2002], which contains an exhaustive treatment of the problem.) The proposed solutions range from phenomenally complicated to subtly simple, and it is the latter approach that is most appropriate for the PageRank context. One of the simplest techniques is called *compensated summation*.

The compensated summation algorithm requires storing one extra floating-point value to accumulate an approximation of the *error* in the current summation. As given in Higham [2002, section 4.3], the following algorithm is due to Kahan.

```
sumx = 0; err = 0;
for i=1:numel(x)
  temp = sumx;         % save the current sum
  y = x(i) + err;      % add the error to the summand
  sumx = temp + y;     % increase the sum by the summand and the error
  err = (temp - sumx); % compute the exact difference after adding y
  err = err + y;       % err should be -y, add y to find the true error
end
```

Instead of $n\varepsilon$ error, this computation has error $2\varepsilon + O(n\varepsilon^2)$. For most conceivable PageRank problems, this accuracy should be sufficient.

This difference is not academic. If we call the simple summation algorithm `ssum` and the compensated summation algorithm `csum` then the difference appears even for 10^7 summands.

```
rand('state',1); x = rand(1e7,1); % ensure repeatable results
y = x./ssum(x); z = x./csum(x);   % normalize for comparison
ssum(y)                           % 1.0000000000000302
csum(y)                           % 0.9999999999999633
ssum(z)                           % 1.0000000000000664
csum(z)                           % 1.0000000000000000
```

The problem with `ssum` is that it cannot reproduce its own normalization! Normalization is an important part of a PageRank code (as we'll see in a moment). We therefore *need* compensated summation for these computations.

Wills and Ipsen [2009] originally highlighted the need for compensated summation in their discussion of the stability of the power method for PageRank. The `law` codes for PageRank [Vigna et al., 2008] implement this feature as well. We have not checked for any further history on this aspect of PageRank computation. Quite amazingly, the problem of accurate summation is still studied [Zhu and Hayes, 2009].

2.4.2 *The power method*

For an eigenvalue problem $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$, the power method is a classic algorithm to compute the eigenvector with largest eigenvalue (in magnitude) [Golub and van Loan, 1996]. Given an almost arbitrary $\mathbf{x}^{(0)}$[2], then

$$\mathbf{x}^{(k+1)} = \rho^{(k+1)}\mathbf{A}\mathbf{x}^{(k)} \qquad \rho^{(k+1)} = 1/\left\|\mathbf{A}\mathbf{x}^{(k)}\right\| \tag{2.17}$$

converges to the eigenvector with maximum eigenvalue when the largest eigenvalue (in magnitude) of $\mathbf{A}$ is unique and real. The scalar quantities ρ normalize the iterates so that they do not grow arbitrarily large.

Consider the power method on the PageRank eigensystem $\mathbf{M}\mathbf{x} = \mathbf{x}$. The largest eigenvalue is unique and equal to 1. Normalization at each step is not required because the largest eigenvalue of $\mathbf{M}$ is 1. Eliminating that step and expanding $\mathbf{M}$ (with (2.2)) yields the iteration

$$\mathbf{x}^{(k+1)} = \alpha\mathbf{P}\mathbf{x}^{(k)} + (1-\alpha)\mathbf{v}\mathbf{e}^T\mathbf{x}^{(k)}. \tag{2.18}$$

Recall that $\mathbf{e}^T\mathbf{x} = 1$. A quick computation shows that when $\mathbf{e}^T\mathbf{x}^{(0)} = 1$, then $\mathbf{e}^T\mathbf{x}^{(k)} = 1$ for all iterations and thus

$$\mathbf{x}^{(k+1)} = \alpha\mathbf{P}\mathbf{x}^{(k)} + (1-\alpha)\mathbf{v}. \tag{2.19}$$

[2] Theory requires us to state that $\mathbf{x}^{(0)}$ must not be orthogonal to the desired eigenvector. Experience with floating point approximations tells us exact orthogonality is not required. This is one of the rare cases when round-off actually helps. An exactly orthogonal vector quickly loses its orthogonality during the floating-point computation. After that happens, the power method succeeds.

For strongly preferential PageRank on $\bar{\mathbf{P}}$, a further optimization is

$$\begin{aligned}\mathbf{y}^{(k+1)} &= \alpha\bar{\mathbf{P}}\mathbf{x}^{(k)} \\ \mathbf{x}^{(k+1)} &= \mathbf{y}^{(k)} + (1 - \mathbf{e}^T\mathbf{y}^{(k+1)})\mathbf{v}.\end{aligned} \tag{2.20}$$

This optimization follows from $\mathbf{P} = \bar{\mathbf{P}} + \mathbf{v}\mathbf{d}^T$ and $\mathbf{d}^T = \mathbf{e}^T - \mathbf{e}^T\bar{\mathbf{P}}$ and is the iteration given in many PageRank papers [Page et al., 1999; Kamvar et al., 2003]. We regard (2.20) as the standard iteration, but prefer (2.19) for analysis purposes. As an algorithm, the power method continues this iteration until $\left\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right\| \leq \tau$ for a user-provided tolerance τ.

Deciding how to begin the power method is easy: follow the advice below.

NOTA BENE *The power method always starts with* $\mathbf{x}^{(0)} = \mathbf{v}$.

No one has suggested a better starting vector for the power method for PageRank than the vector $\mathbf{v}$.

THE RICHARDSON ITERATION Surprisingly, the power method for the PageRank eigensystem is completely equivalent to the Richardson iteration [Varga, 1962] on the linear system $(\mathbf{I} - \alpha\mathbf{P})\mathbf{v} = (1 - \alpha)\mathbf{v}$. The Richardson iteration for $\mathbf{A}\mathbf{x} = \mathbf{b}$ is

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \omega(\mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}), \tag{2.21}$$

and equivalence with (2.19) follows after substituting $\mathbf{A} = (\mathbf{I} - \alpha\mathbf{P})$, $\mathbf{b} = (1 - \alpha)\mathbf{v}$, and $\omega = 1$.[3]

ERROR ANALYSIS All error analysis below uses the 1-norm and examines the difference $\left\|\mathbf{x}^{(k)} - \mathbf{x}\right\|$ for the exact solution $\mathbf{x}$ and the current approximation $\mathbf{x}^{(k)}$.

Lemma 3. *Let* $\mathbf{x}$ *be the exact PageRank vector satisfying* $(\mathbf{I} - \alpha\mathbf{P})\mathbf{x} = (1 - \alpha)\mathbf{v}$. *When computing PageRank, the power method ((2.19) or (2.20)) satisfies*

$$\left\|\mathbf{x}^{(k+1)} - \mathbf{x}\right\| \leq \alpha \left\|\mathbf{x}^{(k)} - \mathbf{x}\right\|.$$

Proof. If we expand both $\mathbf{x} = \alpha\mathbf{P}\mathbf{x} + (1 - \alpha)\mathbf{v}$ and $\mathbf{x}^{(k+1)} = \alpha\mathbf{P}\mathbf{x}^{(k+1)} + (1 - \alpha)\mathbf{v}$ and then take the difference:

$$\left\|\mathbf{x}^{(k+1)} - \mathbf{x}\right\| = \left\|\alpha\mathbf{P}(\mathbf{x}^{(k)} - \mathbf{x})\right\| \leq \alpha \left\|\mathbf{x}^{(k)} - \mathbf{x}\right\|. \qquad \square$$

[3] Those familiar with the Richardson method are likely wondering if $\omega = 1$ is optimal. Good question. We cannot say and believe it to be an open problem.

Thus, using the power method (or Richardson iteration) on PageRank converges linearly (or geometrically) with rate α.

MATLAB Often, program code is the best way of understanding an algorithm and program 2 shows a complete MATLAB implementation of the power method for the strongly preferential PageRank model. In this program, `P` = $\bar{\mathbf{P}}^T$ for efficiency. (MATLAB stores sparse matrices by columns, and so `P'*x` is faster than `P*x`.) Following Wills and Ipsen [2009], this implementation *includes* the additional normalization step in the power method that we said was not required.

Program 2 – The PageRank Power Method. PageRank (with the strongly preferential correction section 2.2.1) on a sub-stochastic matrix `P` is just a few lines of MATLAB, even with optimizations for a constant $\mathbf{v}$. The function `csum` computes a compensated sum of a vector with the algorithm from section 2.4.1, and the `normdiff` function computes $\|\mathbf{x} - \mathbf{y}\|_1$ without computing the difference as a separate vector.

```
function [x flag reshist]=powerpr(P,a,v,tol,maxit,verbose)
% POWERPR Solve a PageRank system using the power method
%
% x=powerpr(P) solve a PageRank system with the row (sub-)stochastic matrix
% P with alpha=0.85 and uniform teleportation (in a strongly preferential
% sense) to an accuracy of 1e-12.
%
% If d = ones(n,1) - P*ones(n,1), then the output x satisifes
%   ||x - alpha*(P + dv')'*x + (1-alpha)*v||_1 ≤ 2*tol
% or (for small tol)
%   x = alpha*(P + dv')*x + (1-alpha)*v.
%
% [x flag reshist]=powerpr(P,a,v,tol,maxit) provides extra output and options
% for the value of alpha, the teleportation distribution vector v, the
% tolerance, and the maximum number of iterations.  The output flag is 0 if
% the system converged and 1 otherwise.  reshist is the vector of
% residuals from each iteration.
%
% Example:
%   x=powerpr(P);

n=size(P,1);
if ~exist('a','var') || isempty(a), a=0.85; end
if ~exist('v','var') || isempty(v), v=1./n; end
if ~exist('tol','var') || isempty(tol), tol=1e-12; end
if ~exist('maxit','var') || isempty(maxit), maxit=10000; end
if ~exist('verbose','var') || isempty(verbose), verbose=0; end
x=zeros(n,1)+v; flag=0; delta=2; iter=0; reshist=zeros(maxit,1);
if verbose, dp=delta; end
while iter<maxit && delta>tol
    y=a*(P'*x); w = 1-csum(y); y = y + w*v;
    delta=normdiff(x,y); reshist(iter+1)=delta; iter=iter+1; x=y./csum(y);
    if verbose, fprintf('power: m=%7i d=%8e r=%8e\n',iter,delta,delta/dp); dp=delta;
    end
end
flag=delta>tol; reshist=reshist(1:iter);
if flag, s='finished'; else s='solved'; end
fprintf('%8s %10s(a=%6.4f) in %5i multiplies to %8e tolerance\n', ...
    s, mfilename, a, iter, delta);
```

ON A GRAPH Further optimizations of the power method are possible on standard graph data structures in other languages. First, constructing the sub-stochastic matrix explicitly is not required on most graph structures. Implicitly using the degree normalization saves memory. Second, in program 2, the quantity `delta` is computed after `y` is completely updated. Instead, the program could accumulate this quantity while updating the vector `y`. Section 6.5.2 provides an optimized iteration on a compressed web graph structure.

2.4.3 *Gauss-Seidel*

For the PageRank linear system $(\mathbf{I} - \alpha\mathbf{P})\mathbf{v} = (1-\alpha)\mathbf{v}$, an extremely simple linear solver is the Jacobi method. It is nearly the same as the Richardson iteration, but a few subtle differences confuse the two. (Looking at these differences is a fun exercise that won't be covered here.) A close cousin of the Jacobi method is the Gauss-Seidel iteration, which is asymptotically faster [Varga, 1962]. See [Arasu et al., 2002; Del Corso et al., 2005] for comprehensive evaluations of the performance of Gauss-Seidel. It is fast and often takes only half the iterations of the power method with exactly the same work per iteration (or nearly so).

The easiest way to understand the Gauss-Seidel iteration is to imagine a particular programming error in the Jacobi iteration. Thus there is no avoiding the Jacobi iteration, and we begin by describing it. For $\mathbf{Ax} = \mathbf{b}$, split $\mathbf{A} = \mathbf{D_A} - \mathbf{N_A}$ into its diagonal and negated off-diagonal components and then iterate:

$$\mathbf{x}^{(k+1)} = \mathbf{D_A}^{-1}(\mathbf{b} + \mathbf{N_A}\mathbf{x}^{(k)}).$$

If $\mathbf{A}$ is stored by rows, then a simple implementation is

$$x_i^{(k+1)} = \left(b_i - \sum_{j \neq i}^{n} A_{ij} x_j^{(k)} \right) / A_{ii} \qquad i = 1, \ldots, n.$$

Under well studied conditions, this iteration converges. Implementing this iteration requires allocating both $\mathbf{x}^{(k)}$ and $\mathbf{x}^{(k+1)}$. Gauss-Seidel follows by "forgetting" to allocate $\mathbf{x}^{(k+1)}$ and updating $\mathbf{x}^{(k)}$ in place, that is

$$x_i^{(k)} \leftarrow \left(b_i - \sum_{j \neq i}^{n} A_{ij} x_j^{(k)} \right) / A_{ii} \qquad i = 1, \ldots, n. \tag{2.22}$$

This change is reasonable and corresponds to using the most recent values of all the variables while solving the equations.

To solve the PageRank linear system, do not compute $\mathbf{A} = (\mathbf{I} - \alpha\mathbf{P})$ and apply the iteration. Instead, proceed implicitly, just like for the power method. With just $\bar{\mathbf{P}}$, working implicitly is easy; with $\mathbf{P}$, it is hard. The *law* codes for PageRank [Vigna et al., 2008] describe the implicit implementation in some detail. In is implemented in the not-quite-complete program 3. Arguably the key step is separately accumulating $\mathbf{d}^T\mathbf{x}$ on the current iterate (`dsum`) and the updated entries (`dsumn`). Beyond that detail, the implementation is straightforwardly an implicit version of (2.22).

But let's outline the update at each iteration anyway! For Gauss-Seidel, there is no large benefit to working with strongly vs. weakly preferential PageRank and so the following iteration actually computes the update for the weakly preferential PageRank formulation. That is,

$$\mathbf{A} = (\mathbf{I} - \alpha\bar{\mathbf{P}} - \alpha\mathbf{u}\mathbf{d}^T).$$

Suppose that $\theta = \mathbf{d}^T\mathbf{x}$ before any changes to $\mathbf{x}$, and that $\bar{\theta} = \mathbf{d}^T\mathbf{x}$ after we change an element of $\mathbf{x}$. Let $\bar{x}_i$ be a temporary update value and $d_i = 1$ if page i is dangling. To update x_i, the steps follow.

1. Compute $\tau = \sum_{j \neq i} \bar{P}_{ij} x_j$.
2. Add $(\theta + \bar{\theta})u_i$ to τ but subtract u_i if page i is a dangling page.
3. Finalize the value of $\bar{x}_i = \frac{\alpha\tau + (1-\alpha)v_i}{1 - \alpha\bar{P}_{ii} - \alpha u_i d_i}$.
4. If page i is dangling, subtract x_i from θ and add $\bar{x}_i$ to $\bar{\theta}$.
5. Accumulate the difference between x_i and $\bar{x}_i$.
6. Set $x_i = \bar{x}_i$.

These steps correspond to the labeled steps in the Gauss-Seidel sweep function.

Gauss-Seidel has a few subtleties. Suppose $\mathbf{P}$ has no diagonal entries, then the power method, Richardson, and Jacobi iterations all coincide. Although the asymptotic performance of Gauss-Seidel is better than that of the Jacobi method, a simple test shows that this need not hold for the average performance.

```
n = 10000; rand('state',0); % setup a 10000x10000 graph with adjacency matrix A
A = sprand(n,n,10/n); A = spones(A); A = A - diag(diag(A));
P = normout(A); % normalize the out-degrees
alpha = 0.99;   % exacerbate the issue.
[xp,flag,histp] = powerpr(P,alpha);
[xp,flag,histgs] = gspr(P,alpha);
fprintf('power method takes %i iterations\n', length(hist));
fprintf('gauss seidel takes %i iterations\n', length(histgs));
```

The results are startling:

```
power method takes 25 iterations
gauss seidel takes 304 iterations
```

Such results, however, are not common. Figure 2.3 shows typical behavior on web graphs.

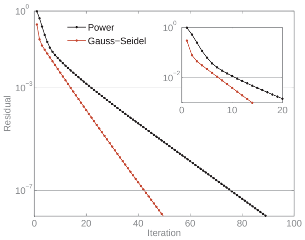


Figure 2.3 – Gauss-Seidel vs. the power method. On the **ubc-cs** graph with $\alpha = 0.85$, the Gauss-Seidel method handily beats the power method.

THE PROBLEM Gauss-Seidel is a fast algorithm for PageRank. Its fatal flaw is that it requires access to $\bar{\mathbf{P}}$ *by rows*, but standard data structures provide access to $\bar{\mathbf{P}}$ by columns. While transposing a matrix in Matlab is as easy as `Pt = P'`, for a gigantic matrix it is not as easy. So Gauss-Seidel imposes some restrictions on the data structures. Another problem with Gauss-Seidel is that it cannot be parallelized effectively. Parallel variants of Gauss-Seidel exist [Saad, 2003] but they require a good multicoloring of the graph structure underlying the matrix to be effective. Thus it is not an appropriate algorithm for really large-scale problems. Nonetheless, it is often the best-performing serial algorithm. *Use it when possible.*

2.4.4 *Summary*

Gauss-Seidel concludes our discussion of classic algorithms for PageRank. Algorithms for PageRank do not live with the problem $(\mathbf{I} - \alpha\mathbf{P})\mathbf{x} = (1 - \alpha)\mathbf{v}$ but at the higher level of a weakly or strongly preferential framework. Some algorithms even operate at the graph level. The PageRank problem requires these optimizations.

See the discussion in section 5.1 for more about *new* algorithms developed for PageRank. As explained there, unfortunately, these new algorithms have little to recommend them over the classic power method and Gauss-Seidel iterations.

2.5 PAGERANK PARAMETERS

Recall that the data for PageRank (problem 1) are $\mathbf{P}, \mathbf{v},$ and α. Varying these parameters often has a large effect on the PageRank vector $\mathbf{x}$. Many of these effects are well understood.

For example, when $\mathbf{P}$ comes from a graph in the strongly personalized PageRank model, then adding a new edge from node i to node j increases x_i [Chien et al., 2004].[4] Other results often focus on applications of link manipulation to increase PageRank values [Zhang et al., 2004; de Kerchove

[4] The result in the article is slightly more general, but this statement is the motivation.

Program 3 – Gauss-Seidel PageRank. This highly compressed implementation of Gauss-Seidel on a sub-stochastic matrix P = $\bar{\mathbf{P}}^T$ shows all steps necessary to compute (2.22) implicitly for the strongly or weakly preferential PageRank problem. It omits a few lines and is not "cut-and-paste" ready, but it retains the details in the essential pieces. See section 6.6 for information on where to get a complete implementation.

(a) Gauss-Seidel iteration

```
function [x flag reshist] = gspr(P,a,v,tol,maxit,verbose,u)
% GSPR Compute PageRank with the Gauss Seidel algorithm
% P: a row substochastic matrix; a=alpha; v=teleportation vector;
% tol=stopping tolerance; maxit=max iterations; u=weakly personalized vector

n = size(P,1); Ps=P; % make a copy so that gssweeppr computes extra info once
x=zeros(n,1)+v; normed=true; extra=0; flag=0; delta=2; iter=0;
reshist=zeros(maxit,1); t=0; z=0; dsum=[];
while iter<maxit && delta>tol
    [x rdiff dsum Ps]=gssweeppr(x,Ps,v,a,(1-a),dsum,u); % rdiff is the difference
    if normed,  nx = csum(x); x=x./nx; dsum = dsum./nx; end
    % evaluate the residual to make sure we are correctly converged
    if rdiff < tol, delta = prresid(x,P,a,v,dsum,u); extra = extra + 1; end
    if verbose, fprintf('gs : m=%7i nm=%7i d=%8e c=%8e\n', iter, ...
        iter+extra, delta, rdiff);
    end
    reshist(iter+1)=rdiff; iter=iter+1;
end
flag=delta>tol; reshist=reshist(1:iter);

function delta=prresid(x,P,a,v,dsum,u) % compute the residual (I-aP)x - (1-a)v
    h = a*(P'*x); h = h + a*dsum*u + ((1-a)*csum(x))*v; delta = normdiff(h,x);
```

(b) Gauss-Seidel sweep

```
function [x ndiff dsumn Ps] = gssweeppr(x,P,v,a,g,dsum,u)
if isstruct(P), n=P.n; ri=P.ri; cp=P.cp; id=P.id; Ps=P; else [cp ri]=sparse_to_csc(P);
    n=length(cp)-1; d=zeros(n,1);  % compute CSC/inv degs
    for i=1:length(ri), d(ri(i))=d(ri(i))+1; end, d(d>0)=1./d(d>0); id=d;
    Ps = struct('n',n,'cp',cp,'ri',ri,'id',id); % save for next iteration
end
if isscalar(u), uscalar=true; else uscalar=false; end
if isscalar(v), vscalar=true; else vscalar=false; end
dsumn1=0; dsumn2=0; ndiff1=0; ndiff2=0; vals=false; dsum1=dsum; dsum2=0;
if isempty(dsum), dsum1=0; dsum2=0; % compute initial dangling sum
  for i=1:n, if id(i)==0, t=dsum1; z=x(i)+dsum2; dsum1=t+z; dsum2=(t-dsum1)+z; end,end
end
for i=1:n
    xn=0; pii=0;
    for cpi=cp(i):cp(i+1)-1                                   % Step 1 handle Pbar
        j=ri(cpi);
        if vals, pji = ai(cpi); else pji=id(j); end
        if i==j, pii = pji; continue; end
        xn=xn+x(j)*pji;
    end
    dsums = (dsumn1+dsumn2+dsum1+dsum2);                      % Step 2, u*d'
    if uscalar, xn=xn+dsums*u; ucurr=u; else xn=xn+dsums*u(i); ucurr=u(i); end
    if id(i)==0, xn = xn - x(i)*ucurr; pii = pii + ucurr; end % page i is dangling
    if vscalar, vi = v; else vi=v(i); end                     % Step 3 update x
    xn=(a*xn+g*vi)/(1-a*pii);
    if id(i)==0                                               % Step 4 update dsum
        t=dsum1; z=-x(i)+dsum2; dsum1=t+z; dsum2=(t-dsum1)+z;
        t=dsumn1; z=xn+dsumn2; dsumn1=t+z; dsumn2=(t-dsumn1)+z;
    end
    xi=x(i);                                                  % Step 5 accumulate
    if xn>xi, dxi = xn-xi; else dxi = xi-xn; end              %    the difference
    t=ndiff1; z=dxi+ndiff2; ndiff1=t+z; ndiff2=(t-ndiff1)+z;  %    between iterations
    x(i)=xn;                                                  % Step 6 set x
end
dsumn = dsumn1+dsumn2; ndiff = ndiff1+ndiff2; % finalize sums
```

et al., 2008]. In terms of pure PageRank theory, a clear statement about the effect of **P** follows.

Theorem 4 (Bianchini et al. [2005], theorem 5.3). *Suppose* **P** *and* $\hat{\mathbf{P}}$ *are two stochastic matrices of the same size. Given fixed* α *and* **v**, *the PageRank vectors for* **P** *and* $\hat{\mathbf{P}}$ *are* **x** *and* $\hat{\mathbf{x}}$. *Let* $\mathcal{U}$ *be the set of columns where* **P** *and* $\hat{\mathbf{P}}$ *differ. Then*

$$\|\mathbf{x} - \hat{\mathbf{x}}\| \leq \frac{2\alpha}{1-\alpha} \sum_{i \in \mathcal{U}} x_i .$$

Although the proof of this result is not long, it's a diversion from the topic of this thesis.[5]

The gist is that what happens with **P** is relatively well studied. The same holds for **v** and even in the original PageRank paper, the impact of **v** is clear [Page et al., 1999]. Page et al. [1999] defined $E = \mathbf{v}$ and wrote:

> However, aside from solving the problem of rank sinks, E turns out to be a powerful parameter to adjust the page ranks. Intuitively the E vector corresponds to the distribution of web pages that a random surfer periodically jumps to. As we see below, it can be used to give broad general views of the Web or views which are focussed [sic] and personalized to a particular individual.

Manipulating **v** begets the *personalized* or *topic specific* PageRank variants alluded to by chapter 1. These problems have been extensively studied [Haveliwala, 2002; Jeh and Widom, 2003] and personalized PageRank forms the basis of novel clustering algorithms [Andersen et al., 2006] and (loosely interpreted) interpolation schemes for graphs [Zhou et al., 2005].

That leaves α, which merits its own section.

2.6 THE PAGERANK FUNCTION OF THE DAMPING PARAMETER

Much of the initial research on α is motivated by the idea expressed in the following statement [Langville and Meyer, 2006a, page 58]:

> But the larger values of α are the ones of most interest because they give more weight to the true link structure of the Web while smaller values of α increase the influence of the artificial probability vector $\mathbf{v}^T$. Since the PageRank concept is predicated on taking advantage of the Web's link structure, it is natural to choose α closer to 1.

[5] Although Gleich and Polito [2007] use this theorem with considerable bravura.

Simply put, the idea is that $\alpha < 1$ is introducing a distortion into the rankings. As we will see, this sentiment is incorrect for the web.[6]

To look at what happens with α in PageRank, we study the implicitly defined PageRank function of α,

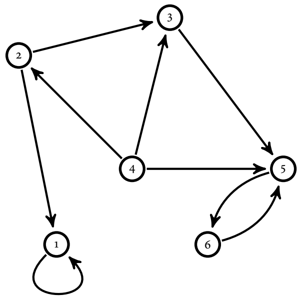
$$(\mathbf{I} - \alpha\mathbf{P})\mathbf{x}(\alpha) = (1 - \alpha)\mathbf{v}, \tag{2.23}$$

which people sometimes write as

$$\mathbf{x}(\alpha) = (1 - \alpha)(\mathbf{I} - \alpha\mathbf{P})^{-1}\mathbf{v}.$$

Of course, we do not mean to suggest actually computing such functions explicitly and analyzing their properties, although, we are going to do so for purely expository purposes. For instance,

$\mathcal{G}$ =



with $\mathbf{v} = \mathbf{e}/6$, yields

$$\mathbf{x}(\alpha) = \begin{bmatrix} \frac{1}{6} - \frac{\alpha\,(\alpha+3)}{3\,\alpha^3+6\,\alpha^2-36} \\ \frac{2\,(\alpha-1)\,(\alpha+3)}{3\,\alpha^3+6\,\alpha^2-36} \\ \frac{(\alpha-1)\,(\alpha+2)\,(\alpha+3)}{3\,\alpha^3+6\,\alpha^2-36} \\ \frac{1}{2} - \frac{(\alpha+2)\,(\alpha-4)}{2\,\alpha^3+4\,\alpha^2-24} \\ \frac{1}{6} - \frac{\frac{\alpha^3}{3}+\frac{7\,\alpha^2}{6}+\frac{5\,\alpha}{3}}{(\alpha+1)\,(\alpha^3+2\,\alpha^2-12)} \\ -\frac{\frac{\alpha^3}{6}+\alpha^2+4\,\alpha+4}{(\alpha+1)\,(\alpha^3+2\,\alpha^2-12)} - \frac{1}{6} \end{bmatrix}.$$

While this expression looks challenging to interpret, figure 2.5 shows the PageRank function of a single node, $x_1(\alpha)$ on the 335-node largest strong component of the harvard500 graph. The expression in that figure looks nearly impossible to understand, and thus we need a more rational (pun fully intended) approach to the problem. Both of these examples were computed by the MATLAB symbolic toolbox.

A few groups started studying the effect of α around the same time. One of the first was Boldi et al. [2005]. Among other observations, they noted that $\mathbf{x}(\alpha)$ is a *rational vector function* of α. Rational functions are ratios of polynomials. In figure 2.5 the caption notes that $x_1(\alpha) = (-23/6030)f(\alpha)/g(\alpha)$. Carefully examining the functions $f(\alpha)$ and $g(\alpha)$ shows that they are indeed polynomials.

It is not difficult to see why PageRank is a rational function. Consider Cramer's rule [Meyer, 2000, page 476] for the solution of $\mathbf{Ax} = \mathbf{b}$:

$$x_i = \frac{\det(\mathbf{A_i})}{\det(\mathbf{A})},$$

[6] See the last paragraph of this section for another viewpoint. Also, we do not mean to suggest that such ideas were misguided. Newer research just provides better guidance.

where $\mathbf{A_i} = \mathbf{A}$ with the ith column replaced by $\mathbf{b}$. So each component of the solution of a linear system is a ratio of determinants. It so happens that the determinant is a polynomial in the matrix entries. When each entry in the matrix depends on a single parameter, say α as in $(\mathbf{I} - \alpha\mathbf{P})$, then the determinant is going to be a polynomial in α. Replacing a column with $\mathbf{b} = (1 - \alpha)\mathbf{v}$ does not change the story for $\mathbf{A_i}$ and hence, each entry in the PageRank vector is a rational function of α.

The second property that results from studying the function is that as α gets closer to 1, the PageRank vector becomes useless [Boldi et al., 2005]. Precisely, the web graph has a single large connected component and many terminal components. If any terminal components have size larger than 1—an example is nodes 5 and 6 in the graph above—then the PageRank values in the largest strong component are 0 when $\alpha = 1$. The largest strong component is roughly 25% of the web and includes most of the interesting pages. These pages include important things like `yahoo.com`, `microsoft.com`, and many popular blogs. As α gets closer to 1, then, the PageRank vector degrades the PageRank value of these important pages, which renders it useless.

Later, Avrachenkov et al. [2007] explored what α should be in light of this behavior. Using a theoretical model, they argue that α should be $1/2$, and no larger. A cartoon version of the argument is that $\alpha = 0$ produces the trivial ranking $\mathbf{v}$ and $\alpha = 1$ produces a useless ranking. A good ranking should be far from both of these locations, and hence $\alpha = 1/2$. To illustrate their point, they plot the PageRank mass, the sum of PageRank values at a subset of states, in the largest strong component as a function of α. In their theory, the mass in the largest strong component ought to begin decreasing around $\alpha = 1/2$. These plots disagree and show that α can be larger than $1/2$ before this happens. The plots themselves show interesting phenomena. In figure 2.4, we see that the strongly preferential PageRank model admits a larger α before significantly shedding mass in the largest strong component.

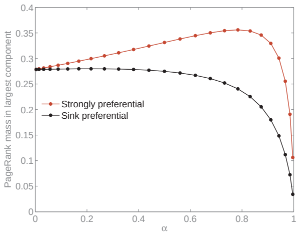


Figure 2.4 – Strong component PageRank mass. For values of α between 0 and 1, this plots the sum of PageRank values inside the largest strong component of wb.cs-stan. The results differ for the strongly preferential model and show that it allows a larger value of α before the strong component starts losing mass.

Separately, Langville and Meyer included ideas in their book about the sensitivity with respect to α. A key tool in their analysis is $\mathbf{x}'(\alpha)$, the derivative of the PageRank function. We return to a discussion of the derivative in chapter 3. Based on the derivative, their summary was that as α gets closer to 1, the PageRank vector becomes sensitive to small changes. Golub and Greif [2006] looked at computing the derivative to get a cheap sensitivity result about PageRank but found that it was just as expensive as computing PageRank and therefore not cheap. Also Boldi et al. [2005] explored using the derivatives to extrapolate the PageRank vector to values of α close to 1.

Meanwhile, people continued to work on what happens when $\alpha \to 1$. Their efforts are useful: we use their results in this thesis. Although it is wrong for the web it is mathematically interesting and the story as $\alpha \to 1$ is our next topic.

COUNTERPOINT The preceding discussion makes simplifying assumptions—it is a modeling argument. From it, we conclude that taking α close to 1 is not a good idea if the goal is to produce a useful ordering of web pages. There are other goals, and we do not mean to imply that PageRank computations with α near 1 are entirely useless. Indeed, in chapter 4, we use computations with α close to 1 inside a variation on the PageRank model. Furthermore, the argument gave no practical guidance about when α is too close to 1 beyond the simple advice $\alpha = 0.5$. Our point is simply that setting α large should be considered carefully. Using a small α (0.5 – 0.9) is not a mere matter of computational convenience, there are important reasons why it should be so.

2.7 THE LIMIT CASE

For all $\alpha < 1$, the PageRank vector is unique. Yet there may be many $\mathbf{x}$ that satisfy $\mathbf{Px} = \mathbf{x}$ (the PageRank equation when $\alpha = 1$). From the previous section, PageRank is a rational vector function of α, so what happens when $\alpha = 1$? The limit exists! That is,

$$\lim_{\alpha \to 1} \mathbf{x}(\alpha)$$

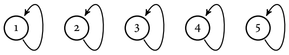
exists and is unique.[7]

2.7.1 *The linear system*

Looking at $(\mathbf{I} - \alpha\mathbf{P})\mathbf{x} = (1-\alpha)\mathbf{v}$ is the easiest way to find the limit. Consider the Jordan canonical form $\mathbf{P} = \mathbf{XJX}^{-1}$. Because $\mathbf{P}$ is a stochastic matrix, all the eigenvalues λ that have $|\lambda| = 1$ are semisimple [Meyer, 2000, page 696] and thus

$$\mathbf{J} = \begin{bmatrix} \mathbf{I} & & \\ & \mathbf{D_1} & \\ & & \mathbf{J_2} \end{bmatrix}, \tag{2.24}$$

[7] For trivial loop-only graph



$\mathbf{P} = \mathbf{I}$ and the PageRank vector $\mathbf{x}(\alpha) = \mathbf{v}$ for all $\alpha < 1$. The limit vector is also $\mathbf{v}$ but *any* vector satisfies $\mathbf{Px} = \mathbf{x}$ (the PageRank equation when $\alpha = 1$).

where $\mathbf{D_1}$ is a diagonal matrix for all the eigenvalues on the unit circle and $\mathbf{J_2}$ is a matrix of Jordan blocks for all the interior eigenvalues. Substituting the Jordan form into the PageRank equation produces

$$\begin{aligned}(\mathbf{I} - \alpha\mathbf{P})\mathbf{x} &= (1-\alpha)\mathbf{v}\\ (\mathbf{I} - \alpha\mathbf{X}\mathbf{J}\mathbf{X}^{-1})\mathbf{x} &= (1-\alpha)\mathbf{v}\\ (\mathbf{I} - \alpha\mathbf{J})\underbrace{\mathbf{z}}_{=\mathbf{X}^{-1}\mathbf{x}} &= (1-\alpha)\underbrace{\mathbf{u}}_{=\mathbf{X}^{-1}\mathbf{v}}\\ \left(\mathbf{I} - \alpha\begin{bmatrix}\mathbf{I} & & \\ & \mathbf{D_1} & \\ & & \mathbf{J_2}\end{bmatrix}\right)\mathbf{z} &= (1-\alpha)\mathbf{u}.\end{aligned}$$

These equations are decoupled and give

$$\begin{aligned}(1-\alpha)\mathbf{z_0} &= (1-\alpha)\mathbf{u_0}\\ (\mathbf{I} - \alpha\mathbf{D_1})\mathbf{z_1} &= (1-\alpha)\mathbf{u_1}\\ (\mathbf{I} - \alpha\mathbf{J_2})\mathbf{z_2} &= (1-\alpha)\mathbf{u_2}.\end{aligned}$$

Both $\mathbf{D_1}$ and $\mathbf{J_2}$ have no eigenvalues equal to 1, and then as $\alpha \to 1$, $\mathbf{z_1} \to 0$ and $\mathbf{z_2} \to 0$; but $\mathbf{z_0} = \mathbf{u_0}$ for all $\alpha \neq 1$ and in the limit, then, $\mathbf{z_0}$ is still $\mathbf{u_0}$.

Suppose $\mathbf{X} = [\, \mathbf{X_0} \; \mathbf{X_1} \; \mathbf{X_2} \,]$ and $\mathbf{X}^{-1} = \begin{bmatrix}\mathbf{Y_0}\\ \mathbf{Y_1}\\ \mathbf{Y_2}\end{bmatrix}$ are partitioned conformally with $\mathbf{J}$. We have now established that

$$\lim_{\alpha\to 1}\mathbf{x}(\alpha) = \mathbf{X_0}\mathbf{Y_0}\mathbf{v}. \qquad (2.25)$$

Although this technique makes it easy to see that the limit value exists, it is *not* insensitive to the formulation of the problem. For PseudoRank (problem 2) with $\sigma = 1$ the linear system

$$(\mathbf{I} - \alpha\bar{\mathbf{P}})\mathbf{y}(\alpha) = \mathbf{v}$$

has no limit for $\mathbf{y}(\alpha)$ as $\alpha \to 1$ because the right-hand side is not normalized to be consistent. For this reason, and others, we prefer the *core* PageRank formulation (problem 1).

2.7.2 *Jordan canonical form*

An alternate derivation of the limit vector uses the Jordan canonical form of $\mathbf{M}(\alpha)$ instead. Serra-Capizzano [2005] proposed this idea and we repeat that derivation here to elucidate the Jordan form of $\mathbf{M}$ and the *eigenvalues* after the modification. In the derivation of the Jordan form, Serra-Capizzano used a *row-stochastic* **P**. To keep the results comparable (and to keep readers on their toes) we revert to Langville and Meyer's notation for this subsection (which uses a row-stochastic **S** instead). At the end, we'll double check that the limit value is the same when derived from the eigensystem (here) and the linear system (the last section).

Recall that the Google matrix is

$$\mathbf{G} = \alpha\mathbf{S} + (1-\alpha)\mathbf{e}\mathbf{v}^T = \mathbf{M}^T. \tag{2.26}$$

Let $\mathbf{S} = \mathbf{V}\mathbf{J}\mathbf{V}^{-1}$ be the Jordan canonical form of **S**.[8] We'll state the Jordan form without the presence of a final scaling matrix to transform the off-diagonal elements in the Jordan blocks to unit values. (Recall that the choice of off-diagonal values in the Jordan blocks is arbitrary.) Without further ado, set

$$\begin{aligned}\mathbf{R} &= \mathbf{I} - \mathbf{e}_1\mathbf{w}^T, &(2.27)\\ \mathbf{w}^T\mathbf{e}_1 &= 0, &(2.28)\\ \mathbf{w}^T &= (1-\alpha)(\mathbf{e}_1^T - \mathbf{v}^T\mathbf{V})(\mathbf{I}-\alpha\mathbf{J})^{-1} &(2.29)\end{aligned}$$

and then

$$\mathbf{G} = \mathbf{V}\mathbf{R}(\alpha\mathbf{J} + (1-\alpha)\mathbf{e}_1\mathbf{e}_1^T)\mathbf{R}^{-1}\mathbf{V}^{-1}. \tag{2.30}$$

Stated as such, this result is somewhat opaque. The derivation is straightforward, but needs a few useful facts about stochastic matrices and eigenvalues. So let's work through it.

From $\mathbf{S} = \mathbf{V}\mathbf{J}\mathbf{V}^{-1}$, we have

$$\mathbf{V}^{-1}\mathbf{G}\mathbf{V} = \alpha\mathbf{J} + (1-\alpha)\mathbf{V}^{-1}\mathbf{e}\mathbf{v}^T\mathbf{V}. \tag{2.31}$$

We simplify the above expression through $\mathbf{V}^{-1}\mathbf{e} = \mathbf{e}_1$, which follows from the fact that **S** is a stochastic matrix.[9] At this point, we simply guess the structure of the matrix that reduces the right-hand side of the previous expression to a Jordan matrix. Let $\mathbf{R} = \mathbf{I} + \mathbf{e}_1\mathbf{w}^T$. We'll show how to pick **w** so that

$$\mathbf{R}^{-1}\mathbf{V}^{-1}\mathbf{G}\mathbf{V} = (\alpha\mathbf{J} + (1-\alpha)\mathbf{e}_1\mathbf{e}_1^T)\mathbf{R}^{-1}. \tag{2.32}$$

[8] Serra uses **X** here, but we've replaced it by **V** to avoid confusion with **X** in the previous section.

[9] To be precise, we need the property that 1 is a non-defective eigenvalue of a stochastic matrix and thus the Jordan block has no off-diagonal elements.

To begin, we require that $\mathbf{e}_1^T\mathbf{w} = 0$ so that $\mathbf{R}^{-1} = \mathbf{I} - \mathbf{e}_1\mathbf{w}^T$. Our expanded equation is

$$(\mathbf{I} - \mathbf{e}_1\mathbf{w}^T)(\alpha\mathbf{J} + (1-\alpha)\mathbf{e}_1\mathbf{v}^T\mathbf{V}) = (\alpha\mathbf{J} + (1-\alpha)\mathbf{e}_1\mathbf{e}_1^T)(\mathbf{I} - \mathbf{e}_1\mathbf{w}^T). \quad (2.33)$$

A few steps of algebra using $\mathbf{e}_1^T\mathbf{w} = 0$ and $\mathbf{J}\mathbf{e}_1 = \mathbf{e}_1$ yield the equivalent expression

$$(1-\alpha)\mathbf{e}_1\mathbf{v}^T\mathbf{V} - \alpha\mathbf{e}_1\mathbf{w}^T\mathbf{J} = (1-\alpha)\mathbf{e}_1\mathbf{e}_1^T - \mathbf{e}_1\mathbf{w}^T, \quad (2.34)$$

where everything shares the common $\mathbf{e}_1$. This expression encodes only a single vector

$$(1-\alpha)\mathbf{v}^T\mathbf{V} - \alpha\mathbf{w}^T\mathbf{J} = (1-\alpha)\mathbf{e}_1^T - \mathbf{w}^T \quad (2.35)$$

or more elegantly the linear system

$$\mathbf{w}^T(\mathbf{I} - \alpha\mathbf{J}) = (1-\alpha)(\mathbf{e}_1^T - \mathbf{v}^T\mathbf{V}). \quad (2.36)$$

This last step completes the derivation of the Jordan form. Note, however, that the eigenvalues of $\mathbf{G}$, which are also the eigenvalues of $\mathbf{M}$, are given by the diagonal of $\alpha\mathbf{J} + (1-\alpha)\mathbf{e}_1\mathbf{e}_1^T$. Recall that $J_{11} = 1$, which corresponds to a semi-simple eigenvalue of $\mathbf{S}$. Write $\mathbf{J} = \left[\begin{smallmatrix} 1 & \\ & \mathbf{J_1} \end{smallmatrix}\right]$ so that[10]

$$\alpha\mathbf{J} + (1-\alpha)\mathbf{e}_1\mathbf{e}_1^T = \begin{bmatrix} 1 & \\ & \alpha\mathbf{J_1} \end{bmatrix}.$$

This analysis confirms the following theorem.

Theorem 5 (Eldén [2004]; Brauer [1952] theorem 29). *If the eigenvalues of* $\mathbf{P}$ *are* $1, \lambda_2, \ldots, \lambda_n$ *then the eigenvalues of* $\mathbf{M}(\alpha)$ *are* $1, \alpha\lambda_2, \ldots, \alpha\lambda_n$.

This result is a corollary of the old theorem from Brauer about eigenvalues of combinations of matrices. Elden's proof is specific to the PageRank case and more modern.

FINDING THE LIMIT With the Jordan form from eqs. (2.27) to (2.30) we can work out the PageRank vector $\mathbf{x}(1)$ in the limit sense.[11] For $\alpha < 1$, the PageRank vector $\boldsymbol{\pi}(\alpha)$ satisfies

$$\begin{aligned} \boldsymbol{\pi}(\alpha)^T &= \boldsymbol{\pi}(\alpha)^T\mathbf{G}, && (2.37) \\ &= \boldsymbol{\pi}(\alpha)^T\mathbf{V}\mathbf{R}(\alpha)(\alpha\mathbf{J} + (1-\alpha)\mathbf{e}_1\mathbf{e}_1^T)\mathbf{R}(\alpha)^{-1}\mathbf{V}^{-1}, && (2.38) \end{aligned}$$

[10] Note that $\mathbf{J_1}$ is different from $\mathbf{J_2}$ in the previous section.

[11] In the remainder of the document, $\mathbf{x}(1)$ means this limiting value.

which implies

$$\begin{aligned}\boldsymbol{\pi}(\alpha)^T &= \mathbf{e}_1^T\mathbf{R}(\alpha)^{-1}\mathbf{V}^{-1}, && (2.39)\\ &= \mathbf{e}_1^T\mathbf{V}^{-1} - \mathbf{w}(\alpha)^T\mathbf{V}^{-1}. && (2.40)\end{aligned}$$

The only dependence on α is in $\mathbf{w}(\alpha)$. This fact appears unfortunate because $\mathbf{w}(\alpha)$ solves the system $\mathbf{w}(\alpha)^T(\mathbf{I} - \alpha\mathbf{J}) = (1 - \alpha)(\mathbf{e}_1^T - \mathbf{v}^T\mathbf{V})$, *but* $\mathbf{J}$ has structure that makes the solution obvious. An additional concern is that $\boldsymbol{\pi}(\alpha)^T$ includes the first row of $\mathbf{V}^{-1}$, a vector that may be arbitrary!

Let's resolve these concerns.

We can decompose[12]

$$\mathbf{J} = \begin{bmatrix}\mathbf{I} & & \\ & \mathbf{D}_1 & \\ & & \mathbf{J}_2\end{bmatrix}, \qquad (2.41)$$

where $\mathbf{D}_1$ is a diagonal matrix of all eigenvalues on the unit circle not equal to 1 and $\mathbf{J}_2$ is a Jordan matrix with all eigenvalues on the interior of the unit circle. If we conformally partition $\mathbf{w}(\alpha)^T = \begin{pmatrix}\mathbf{w}_0(\alpha)^T & \mathbf{w}_1(\alpha)^T & \mathbf{w}_2(\alpha)^T\end{pmatrix}$ and $\mathbf{V} = \begin{pmatrix}\mathbf{V}_0 & \mathbf{V}_1 & \mathbf{V}_2\end{pmatrix}$ [13] then

$$\begin{aligned}\mathbf{w}_0(\alpha)^T &= \mathbf{e}_1 - \mathbf{v}^T\mathbf{V}_0, && (2.42)\\ \mathbf{w}_1(\alpha)^T &= -(1-\alpha)\mathbf{v}^T\mathbf{V}_1(\mathbf{I} - \alpha\mathbf{D}_1)^{-1}, \text{ and} && (2.43)\\ \mathbf{w}_2(\alpha)^T &= -(1-\alpha)\mathbf{v}^T\mathbf{V}_2(\mathbf{I} - \alpha\mathbf{J}_2)^{-1}. && (2.44)\end{aligned}$$

Both of the linear systems for $\mathbf{w}_1(\alpha)$ and $\mathbf{w}_2(\alpha)$ are non-singular for $0 \le \alpha \le 1$ and $\mathbf{w}_0(\alpha)$ is a constant function!

The solution of $\mathbf{w}(\alpha)^T$ also resolves our second concern, the $\mathbf{e}_1^T\mathbf{V}^{-1}$ factor in $\boldsymbol{\pi}(\alpha)^T$. After partitioning

$$\mathbf{V}^{-1} = \begin{bmatrix}\mathbf{Z}_0\\ \mathbf{Z}_1\\ \mathbf{Z}_2\end{bmatrix} \qquad (2.45)$$

we have

$$\boldsymbol{\pi}(\alpha)^T = \mathbf{v}^T\mathbf{V}_0\mathbf{Z}_0 + (1-\alpha)\mathbf{v}^T\mathbf{V}_1(\mathbf{I}-\alpha\mathbf{D}_1)^{-1}\mathbf{Z}_1 + (1-\alpha)\mathbf{v}^T\mathbf{V}_2(\mathbf{I}-\alpha\mathbf{J}_2)^{-1}\mathbf{Z}_2, \qquad (2.46)$$

[12] You might think this next step is incorrect because it asserts a form on **J** that possibly requires reordering **V**. The algebra, however, still works if we assert this form on step 1.

[13] Remember that **X** and **V** are going to slightly different because they are Jordan forms of transposed matrices.

which is valid for all $0 \le \alpha \le 1$ and actually all α other than the set $\alpha = \frac{1}{\lambda_i}$ where λ_i is an eigenvalue of $\mathbf{S}$ and $\lambda_i \neq 0$.

CHECKING BACK From the linear system, we found that $\mathbf{x}(1) = \mathbf{X_0}\mathbf{Y_0}\mathbf{v}$ and from the eigensystem, we found $\boldsymbol{\pi}(1)^T = \mathbf{v}^T\mathbf{V_0}\mathbf{Z_0}$. In the former,

$$\mathbf{P} = \begin{bmatrix} \mathbf{X}_0 & \mathbf{X}_1 & \mathbf{X}_2 \end{bmatrix} \begin{bmatrix} \mathbf{I} & & \\ & \mathbf{D}_1 & \\ & & \mathbf{J}_2 \end{bmatrix} \begin{bmatrix} \mathbf{Y}_0 \\ \mathbf{Y}_1 \\ \mathbf{Y}_2 \end{bmatrix},$$

and the latter,

$$\mathbf{S} = \begin{bmatrix} \mathbf{V}_0 & \mathbf{V}_1 & \mathbf{V}_2 \end{bmatrix} \begin{bmatrix} \mathbf{I} & & \\ & \mathbf{D}_1 & \\ & & \mathbf{J}_2 \end{bmatrix} \begin{bmatrix} \mathbf{Z}_0 \\ \mathbf{Z}_1 \\ \mathbf{Z}_2 \end{bmatrix}.$$

The relationship is $\mathbf{P} = \mathbf{S}^T$, so that

$$\mathbf{X_0} = \mathbf{Z_0}^T \text{ and } \mathbf{Y_0} = \mathbf{V_0}^T.$$

Thus we find the same limit vectors in each formulation.

2.8 PAGERANK DATASETS

Most of this chapter has discussed the theory of PageRank. But a large portion of this thesis involves *actually computing PageRank*. Thus, we need data to compute PageRank, and in this case: bigger really is better. Table 2.2 shows a series of properties for the datasets used in the forthcoming experiments. Each dataset is a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. I either collected the dataset myself or took it from a publicly available source.

The graphs aa-stan, ee-stan, and cs-stan correspond to the web graphs for the hosts `aa.stanford.edu`, `ee.stanford.edu`, and `cs.stanford.edu`, respectively. These graphs were formed as a subset of the Webbase 2001 crawl [Hirai et al., 2000] compressed with the Webgraph framework [Boldi and Vigna, 2005]. The graph cnr-2000 is the result of an Ubicrawler crawl [Boldi et al., 2004].

2.8.1 *Wikipedia matrices*

Wikipedia provides access to copies of its English page database collected at almost-periodic intervals. We downloaded a series of these database dumps from 2005 until 2009. From each database we formed an article-article link graph, where an article is

a page in the main Wikipedia namespace, for example
`http://en.wikipedia.org/wiki/PageRank`;

a category page, for example
`http://en.wikipedia.org/wiki/Category:Matrix_theory`; or

a portal page, for example
`http://en.wikipedia.org/wiki/Portal:Mathematics`.

We removed all other pages and links.

2.8.2 *Flickr crawl*

Using the techniques in Gleich and Polito [2007], we built a snapshot of the flickr social network starting from a particular node and crawling until the PageRank on the boundary was less than 0.0001. These techniques are related to the RankMass crawler from Cho and Schonfeld [2007].

$$\begin{aligned}
f(\alpha) = {}& 1724683103168320512000\alpha^{102} - 351689859974563275916800\alpha^{101} + 1046657678560756011923040\alpha^{100} \\
&+332821515558986503317268308\alpha^{99} + 202994690094545539249274953458\alpha^{98} + 701216550622104187641429941160\alpha^{97} \\
&+38942435173273232195508862504752\alpha^{96} - 5204876256969489587508598423780757\alpha^{95} - 53419116345848724180375395029139614\alpha^{94} \\
&+1621997105501543781796265745838677670\alpha^{93} + 17992097277595516775992937444966323725\alpha^{92} \\
&-228388738389199148614341585444680228464\alpha^{91} - 2572935401339464873388154472765864295466\alpha^{90} \\
&-18662047188535851000868073690251020472621\alpha^{89} - 155192964832717622674637679380949267008397\alpha^{88} \\
&+13633798075806927018912795365187923947976816\alpha^{87} + 153692481592717017931843564092779914769739855\alpha^{86} \\
&-2424702525231324896856434133527720085459106818\alpha^{85} - 34112664906875644324640001664890877920583430935\alpha^{84} \\
&+222921632950502905446093540571509314548545319158\alpha^{83} + 4458381340774458139955262362762709170337141183042\alpha^{82} \\
&-9722398912749159172830586061232227612575398195577\alpha^{81} - 402863595222192101330043246404750577170418624210463\alpha^{80} \\
&-241296146875962767748365749082981265577900593669099\alpha^{79} + 26884891161116233003550134767867058390000240645389885\alpha^{78} \\
&+75002935639704657680175868562515328344632861061620026\alpha^{77} - 1355245718493528694128677343628002432897202221776993666\alpha^{76} \\
&-6666337432948865424681896342751813538288258918631143898\alpha^{75} + 50876562123828411130342908134923596879946044492587906688\alpha^{74} \\
&+385972738637461890892793659070699381929652086327544953064\alpha^{73} - 1324370012053495348856190918458325441254102678707139546912\alpha^{72} \\
&-16416792980158036153780188009203628703318521649963318398744\alpha^{71} + 17510197624369310054645143199845105805941154913191274775360\alpha^{70} \\
&+533320137070985354296793454864336229974212018883255863520736\alpha^{69} + 275502212308122569075672900514808641788656066608417565862128\alpha^{68} \\
&-13429082722840051523544458153489421210623008268881676515202688\alpha^{67} \\
&-23110058843365910555627839838104471746030299594537756688223008\alpha^{66} \\
&+262081257818502675810469542460738736851208401216965512926700160\alpha^{65} \\
&+729407390179003876249104385055674850942454472967192021090685376\alpha^{64} \\
&-3847937179452929633833233710422322341537775007885518269634539392\alpha^{63} \\
&-15488141989129507247130473020571135237573107436265881323677072000\alpha^{62} \\
&+36050325771659567239591241663693950811960305821938730156334667776\alpha^{61} \\
&+246707867322513330007744656494007568641366676837744833157870986240\alpha^{60} \\
&+66698815198854350338382524697115939758820557665663603703007667712\alpha^{59} \\
&-2959446110396107328472639479854607457433633185566140760490226286592\alpha^{58} \\
&-12528512804728910558071029225789548204605758683928995029146000314368\alpha^{57} \\
&+19985525277247932558760938212461479524515746377831707793868714172416\alpha^{56} \\
&+343866190600408921247069416527135879796528858737524668958998645633024\alpha^{55} \\
&+237159992339459130849980507259488489676582642639199883151854812422144\alpha^{54} \\
&-6150352682504179603648657901968989091083378789857325448622418220859392\alpha^{53} \\
&-12507084588874068660420542622454441021005365876210831205762085535989760\alpha^{52} \\
&+76052343558405304817491728967709919562879906814237879556140479278219264\alpha^{51} \\
&+281657470545819893901842735393494111347269819443029672934492155921629184\alpha^{50} \\
&-524010169549932716315240835391286383538294517356494888193446880264060928\alpha^{49} \\
&-4283228548253488673520351046009849054273946705738400536855052450584985600\alpha^{48} \\
&-2155194129185085332436034710334032595487897368550943059587873095183237120\alpha^{47} \\
&+44942983365390912258646063248936155917171235534162037124027584790839951360\alpha^{46} \\
&+123764976043225311633569878034493895722302903722502785220748272524591104000\alpha^{45} \\
&-263604612819883334094471942440378055857630908721587551326277602165812887552\alpha^{44} \\
&-2043045823645899057845901056050369454115577248500633141166053687383937777664\alpha^{43} \\
&-883572534249006235663814128436259426227447113734226469390794110452279279616\alpha^{42} \\
&+22029266389692672474905374638580604237511322238870051881693348503640495620096\alpha^{41} \\
&+45203159614332573226167349621344476004471313288020398240113991699259941978112\alpha^{40} \\
&-168198634626680009003513480377236264968641977685259854545270514440488513175552\alpha^{39} \\
&-668594708420193863217346925249650551196858552245852383052928679191604052885504\alpha^{38} \\
&+829995196451920004299651167659513171123326408698056871202815263749436350660608\alpha^{37} \\
&+6805400890411122172338081288981379379115027947251954438964848554500327026458624\alpha^{36} \\
&-839859147076619012613401783607878586283917926703478867476334483102478263910400\alpha^{35} \\
&-54336251411672379109173054554388944990018972031681985156883655345205770838867968\alpha^{34} \\
&-31763834543511199735483407052389951464492348704450435677017768682913434678853632\alpha^{33} \\
&+357712343186400835247921272739995225258056636329416844164038875886993432486346752\alpha^{32} \\
&+394894109850616441422196163643656479874423531345017994904270039571808903743143936\alpha^{31} \\
&-1993929054800515710688917066299914269693286626662952457319746685784090804001701888\alpha^{30} \\
&-3002267549064744794430368624087097289757148076091004127530245571997364275264880640\alpha^{29} \\
&+9573037450950832796546125489519791559144293205440801001596502044790259906531819520\alpha^{28} \\
&+17344649689902103638748302705765490194768583990372876266091126135709005379492904960\alpha^{27} \\
&-40109860118705371377719161262775470420310263138301806878152530252877875499258347520\alpha^{26} \\
&-81164940713776050502710413692301000793918577563223455903690236298808582388129464320\alpha^{25} \\
&+148564684652598057008901304730992665142722743799406464890491019151228896289384038400\alpha^{24} \\
&+316011966716392521139260824696069224379619965509016982919437611079336611648687308800\alpha^{23} \\
&-493937443242584182232311411058386151572960694882377097092991520649901348015308800000\alpha^{22} \\
&-1032631097012698995004666052463769745257461602028357530776684844670222403998056448000\alpha^{21} \\
&+1496051498205595023212876520305710378404801491740260076675413316113755884612485120000\alpha^{20} \\
&+2808040259722605050478570986966436499493340536522637266197921902172159568196403200000\alpha^{19} \\
&-4136197022520781923456607241837348242573554483478641216258357714108027813809356800000\alpha^{18} \\
&-6160485939298474432256897143548698073388765535732087612799636625216399725507379200000\alpha^{17} \\
&+10181878863815582516096533223816217477281300683613861533575784672221983047837286400000\alpha^{16} \\
&+10194856369622478439949806821168034096091795201083524149176484646680561362088755200000\alpha^{15} \\
&-21187043154586589769777874169878395124445179056372063547781637907554829911195648000000\alpha^{14} \\
&-10797617499303349106653965603456976243130791900284770633819420554804763717271552000000\alpha^{13} \\
&+34810029936836090031365778846622873044044684351940766602929819682347633942200320000000\alpha^{12} \\
&+2444726911101623695480273766948648572307801537702233726799335232903066482114560000000\alpha^{11} \\
&-41556351242381300546605413427086996396993985304666948472681225427743791067955200000000\alpha^{10} \\
&+13235190618796698664164720739564065289559223880035423917829189953498612603289600000000\alpha^{9} \\
&+31707117886734781934293206235313206269589256456457305316683904435212337020928000000000\alpha^{8} \\
&-22902862215982163769314078339007120966645769612414912118902053217891120054272000000000\alpha^{7} \\
&-108614472314935279647963811607975778475067743866290587662452063452941778944000000000000\alpha^{6} \\
&+167023406144409967263213225195804783777841967624560134653553688921837142016000000000000\alpha^{5} \\
&-24846557002993909429620971708349332904274137038359512435388331176828600320000000000000\alpha^{4} \\
&-44133290475782082257151448326460233618416074004025428691689175005528064000000000000000\alpha^{3} \\
&+24877807310589964539391042460645398662644987789332289326870352822796288000000000000000\alpha^{2} \\
&-40214815854114377103803069242671282026506242510354083123536738438348800000000000000000\alpha \\
&-52038087132641691932831070631369958870257591306470635457082294272000000000000000000
\end{aligned}$$

Figure 2.5 – A PageRank function. $x_1(\alpha) = (-23/6030)f(\alpha)/g(\alpha)$, see section 2.6

$$
\begin{aligned}
&g(\alpha) = 21252680112847680000\alpha^{102}\\
&-3542775096896042918400\alpha^{101} - 377301357230918051819160\alpha^{100} + 62030166204003769204027938\alpha^{99} + 301903572553392042618587937\alpha^{98}\\
&-27515144995670593102754792187\alpha^{97} - 1391342388530090922919905979557\alpha^{96} - 11397010225845179645798293856049\alpha^{95}\\
&+487046819801240647260974920877667\alpha^{94} + 8641748415645906110710596472701695\alpha^{93} - 14615573868254463557271968794871527\alpha^{92}\\
&-1455304405730842808585234463006780870\alpha^{91} - 16140532952116322684344866986683755014\alpha^{90}\\
&-107685923577790689207116358432796101348\alpha^{89} + 3574857500140390342079726927167132783327\alpha^{88}\\
&+76245995916566900197088870723441134067760\alpha^{87} - 320477613697118756563592647774688786780579\alpha^{86}\\
&-14315018719450474212530996756919665488506623\alpha^{85} - 12271042346558183829899943919127664848771235\alpha^{84}\\
&+1538719934896052457300693234469902122130588440\alpha^{83} + 7259823837632938466306787148779956756499503259\alpha^{82}\\
&-91383277962053778179963631846131934198363974003\alpha^{81} - 912158632690159715631486922494993985581191177254\alpha^{80}\\
&+1124589169570249225316595386438810701468062018941\alpha^{79} - 55599491760340084897708205765116975153096053881206\alpha^{78}\\
&+254197028878341726795811304127085084201803714274594\alpha^{77} - 1155102780712932745491921904562487673324953687625090\alpha^{76}\\
&-19623309116424352882311523132748440745863270150867432\alpha^{75} - 72367264828688457023192884699324797029606326773402260\alpha^{74}\\
&+510591330662979105902331311824358111451756310585317896\alpha^{73} + 6560635654785580651459993551515346226540950556472012168\alpha^{72}\\
&+11841946546859350197679256661965428675545845230913012752\alpha^{71} - 222422692257166102165445803087102201095333519552710152624\alpha^{70}\\
&-1447290325427425453794609658098719385231428839474861685840\alpha^{69} + 2125011726240928873652963898522501443619028980101705108896\alpha^{68}\\
&+56163879158282775333105949842095267377034088228166264755488\alpha^{67} + 133653341840138472687713523321901358136789047544268798190144\alpha^{66}\\
&-1165851790876533575106055126719543401792990924852555883239232\alpha^{65}\\
&-7205045167922126127366881708591461911830986630512778219907200\alpha^{64}\\
&+8196149623293434725419276185048399130126199483584663609965696\alpha^{63}\\
&+190347290617372900092754118891814664663338859287254054095265536\alpha^{62}\\
&+296403177926940870392191966640325276665391672647048523475737600\alpha^{61}\\
&-3179986962227253427695124755087565566711837258936975824737021952\alpha^{60}\\
&-12273950891286672757637149571293897139589064857886165164957404160\alpha^{59}\\
&+31408962973625270006925545397999409094566386715881351869322999808\alpha^{58}\\
&+253177395609699067378776631302481890469651122338031051366108686336\alpha^{57}\\
&-15354832074031738521204442047058295183786064138590507845987942400\alpha^{56}\\
&-3457076532174502560822426326142749948730584183953208907119801098240\alpha^{55}\\
&-6661437625275114934838338879511817915494254490727882100057772130304\alpha^{54}\\
&+28704083600179676384022705580143799967745682382583318411010759639040\alpha^{53}\\
&+173119877625293135511416194747967318688771201702803231109775079243776\alpha^{52}\\
&+42285615967170654345485778244291908234053330314299949447131636826112\alpha^{51}\\
&-3092545165791022831669116892040565590342926023532342815170675350831104\alpha^{50}\\
&-7385454932946443098573906964601689327710122151758555775183630113177600\alpha^{49}\\
&+44090325705050939960465955316629060665099648652920301218388343039721472\alpha^{48}\\
&+180430494757250498411208705426475191214202221095549279916495110854934528\alpha^{47}\\
&-493525709032718650057526281767644848135900953167613963100373354560880640\alpha^{46}\\
&-3036843091999605016958964058463815080108229170215714733277797291506270208\alpha^{45}\\
&+3865732160987803528525842299699004166440912343665407865787648656852123648\alpha^{44}\\
&+40165478772124194334610082404062794103423683134161618111009172215000203264\alpha^{43}\\
&-11270446090439842262616868429380066718469755470664378191173671836048162816\alpha^{42}\\
&-431725269187383778295706776607285692623377582173153891079752971949306806272\alpha^{41}\\
&-223578855128847742913688810087057318022143978462109332025481258567127269376\alpha^{40}\\
&+3806641102807223385639875513891980988734164017656312910101180605432770592768\alpha^{39}\\
&+4698338022493830197418469777664958098209079184719648168122484318843776794624\alpha^{38}\\
&-27472779560617412642244986656083233718762546534015558981997009063520073940992\alpha^{37}\\
&-53681346508826005770227174053581590059283954164048929404839105532796000534528\alpha^{36}\\
&+159792483519832871643195761447614587325418857137220582772566606510963040452608\alpha^{35}\\
&+447775073289651418862702364745936934030540232799739862181009845955145918054400\alpha^{34}\\
&-716151822637851063198942928932119452580573299424788537816341142171636199325696\alpha^{33}\\
&-2933014614963404405624949533910517712184375976693976408790612422895031925342208\alpha^{32}\\
&+2123830137329614973540541687269913350581043300869459472923500012177964595675136\alpha^{31}\\
&+15491595398748844916213727820453788960246908641990943232584972825253134896988160\alpha^{30}\\
&-567958048418299255333286711763252835749000069031930133372386182151207554908160\alpha^{29}\\
&-66470511672905973490254270449160748571544305482918584099892594998203682442444800\alpha^{28}\\
&-41709961606955286961348486645761651227147583272088758133872408100389592005345280\alpha^{27}\\
&+230054579604523153712298391601390663928014143964616089795553744517711724229427200\alpha^{26}\\
&+329047428589773383037144315393721888182438735406281384979987048470391313714380800\alpha^{25}\\
&-624457510685469088854461981456149137717339107570818384916469113052631000311398400\alpha^{24}\\
&-1677023335298418194342571458068169568589073430891247365956379137661470030954496000\alpha^{23}\\
&+1230550173656441248007569837874753909716280107131735470279305802166985335767040000\alpha^{22}\\
&+6678146820080249693682249156290720809288474225484848277349949390581823092817920000\alpha^{21}\\
&-1335521590342284671869409797110636836566621705504540007316206486982151246970880000\alpha^{20}\\
&-22059887560957847625176129319162020059098234073297049383363906396128975739944960000\alpha^{19}\\
&-785340364420012115414030768139171427530706940353196376588668778473179840512000000\alpha^{18}\\
&+61717145472396641090916430698897769773248344842243911905796114382139890755174400000\alpha^{17}\\
&+4618799817652795890614174914969648296550799276151584724886711273496204279808000000\alpha^{16}\\
&-145443881953486865648263190807202565800657985019098154597005260614892420857856000000\alpha^{15}\\
&+3895092842622840658053685865827455168729291218619441351061760037262624555008000000\alpha^{14}\\
&+280377685657177839855779204679112256388859412881172644774038185688216297799680000000\alpha^{13}\\
&-61886949354628165807560200683179015577169820467161436162087652305392411607040000000\alpha^{12}\\
&-419675757995547385956754793581818014152422427747599875638509945701343323750400000000\alpha^{11}\\
&+198444685626856286689595946806119633184708804987305884557282973568786130534400000000\alpha^{10}\\
&+448747751865602411338231508161295374031102511536034615950378207124016594944000000000\alpha^{9}\\
&-354225411849996408405676297836399354389793212596699228226946778751972671488000000000\alpha^{8}\\
&-289553838601814478908147100882111896550771868124112559407400778696805580800000000000\alpha^{7}\\
&+380193432519284724415876033554186663453423948344477630293719517144232755200000000000\alpha^{6}\\
&+498686387317498369534970359416974094935860609530687522431122340440965120000000000000\alpha^{5}\\
&-225214852583720088017543526212238701302651117601148886021831714815344640000000000000\alpha^{4}\\
&+657048203705194150644873621884638637600633656280985659999477783592960000000000000000\alpha^{3}\\
&+496488645341739551712753878887713942931184684832027306458656054181888000000000000000\alpha^{2}\\
&-357568569847705837270936787698491051277201721504762920085037986611200000000000000000\alpha\\
&+664931113361532730252841458067505030008847000027124786396051537920000000000000000
\end{aligned}
$$

Figure 2.5 (continued).

Table 2.2 – The percent values are the percentage of total pages with the property in the preceding column.

Name	Vertices	Edges	Strong components	Largest component	%	Max out-degree	Out-degree 0 nodes	%	Max in-degree	In-degree 0 nodes	%	Largest core	Size of 1-core	%
flickr-0.02[a]	33,607	124,867	33,070	538	1 %	7,043	33,064	98 %	174	0	0 %	25	18,371	54 %
flickr-0.0001[a]	820,878	9,837,214	277,277	527,476	64 %	10,272	265,189	32 %	8,549	0	0 %	351	483,062	58 %
generank[k]	4,047	339,596	10	4,026	99 %	493	0	0 %	493	0	0 %	129	38	<1 %
ubc-cs[b]	51,681	673,010	7,217	17,807	34 %	3,723	4,732	9 %	15,272	0	0 %	34	28,805	55 %
ubc[b]	339,147	4,203,811	65,141	249,872	73 %	24,759	53,549	15 %	90,577	0	0 %	80	204,641	60 %
wb-cs.stan[c,d,e]	9,914	36,854	4,391	2,759	27 %	277	2,861	28 %	340	699	7 %	13	4,357	43 %
wb-aa.stan[c,d,e]	114	229	3	112	98 %	109	2	1 %	110	0	0 %	1	114	100 %
wb-ee.stan[c,d,e]	1,615	7,046	531	653	40 %	210	465	28 %	221	38	2 %	6	660	40 %
www[f]	325,729	1,497,134	203,609	53,968	16 %	3,445	187,788	57 %	10,721	0	0 %	152	240,171	73 %
stan-berk[g]	683,446	7,583,376	109,238	333,752	48 %	249	4,735	<1 %	83,448	68,062	9 %	162	303,227	44 %
au2005[h]	4,060,729	21,019,271	3,021,497	1,008,816	24 %	7,620	2,819,272	69 %	48,027	10,887	<1 %	177	3,242,363	79 %
au2006[h]	3,907,649	23,782,896	2,854,693	1,016,992	26 %	16,605	2,682,095	68 %	56,944	2,315	<1 %	208	3,040,678	77 %
nz2006[h]	604,913	3,777,080	455,317	144,020	23 %	3,281	435,563	72 %	20,864	0	0 %	81	492,870	81 %
uk2004[h]	9,744,400	42,586,558	7,153,046	2,489,828	25 %	20,000	6,630,536	68 %	45,495	2,820	<1 %	152	7,914,970	81 %
uk2005[h]	10,037,216	47,993,341	7,338,097	2,604,479	25 %	19,710	6,837,983	68 %	63,624	9,764	<1 %	165	8,082,737	80 %
us2004[h]	6,411,252	23,883,438	5,243,332	1,084,200	16 %	20,000	4,826,195	75 %	116,393	602	<1 %	143	5,384,310	83 %
us2004scc[h]	1,084,200	11,554,007	1	1,084,200	100 %	12,479	0	0 %	45,309	0	0 %	143	607,287	56 %
cnr-2000[d,e]	325,557	3,216,152	100,977	112,023	34 %	2,716	78,056	23 %	18,235	0	0 %	81	165,919	50 %
eu-2005[d,e]	862,664	19,235,140	90,768	752,725	87 %	6,985	71,675	8 %	68,922	0	0 %	378	215,580	24 %
in-2004[d,e]	1,382,908	16,917,053	367,675	593,687	42 %	7,753	282,306	20 %	21,866	86	<1 %	473	672,031	48 %
wb-edu[c,d,e]	9,845,725	57,156,537	4,269,022	2,916,860	29 %	3,841	2,925,419	29 %	25,762	717,705	7 %	447	4,870,109	49 %
wiki-20051105[i]	1,634,989	19,753,078	528,698	1,103,453	67 %	4,970	72,556	4 %	75,547	464,135	28 %	60	331,451	20 %
wiki-20060925[i]	2,983,494	37,269,096	975,731	2,003,668	67 %	5,852	88,970	2 %	159,378	873,634	29 %	68	581,383	19 %
wiki-20061104[i]	3,148,440	39,383,235	1,040,035	2,104,115	66 %	6,576	91,462	2 %	168,685	932,906	29 %	69	605,593	19 %
wiki-20070206[i]	3,566,907	45,030,389	1,203,340	2,358,755	66 %	7,061	101,303	2 %	187,342	1,078,682	30 %	71	674,097	18 %
wiki-20070402[i]	3,799,337	48,193,953	1,292,985	2,500,292	65 %	7,004	104,511	2 %	199,158	1,158,045	30 %	73	715,174	18 %
wiki-20070527[i]	4,011,714	51,114,161	1,375,774	2,629,388	65 %	5,338	112,452	2 %	208,855	1,231,908	30 %	74	753,833	18 %
wiki-20070802[i]	4,335,341	55,177,586	1,500,443	2,827,699	65 %	8,546	124,484	2 %	220,757	1,342,058	30 %	76	826,126	19 %
wiki-20070908[i]	4,500,343	57,296,613	1,562,056	2,929,856	65 %	7,079	129,126	2 %	228,860	1,396,369	31 %	77	865,744	19 %
wiki-20080103[i]	4,982,964	63,242,904	1,799,291	3,175,527	63 %	5,742	139,301	2 %	258,035	1,609,050	32 %	79	943,480	18 %
wiki-20080312[i]	5,284,673	66,939,152	1,927,455	3,348,819	63 %	5,611	149,626	2 %	273,529	1,721,505	32 %	80	991,149	18 %
wiki-20081008	6,233,375	75,712,300	2,397,234	3,826,461	61 %	5,777	182,592	2 %	304,430	2,141,321	34 %	81	1,169,591	18 %

Continued on Next Page…

Table 2.2 – Continued

Name	Vertices	Edges	Strong components	Largest component	%	Max out-degree	Out-degree 0 nodes	%	Max in-degree	In-degree 0 nodes	%	Largest core	Size of 1-core	%
wiki-20090306[i]	6,750,292	80,990,219	2,596,053	4,143,840	61 %	5,640	206,685	3 %	321,426	2,319,993	34 %	78	1,278,802	18 %
uk-2006-host[j]	11,402	730,774	2,935	7,945	69 %	5,994	2,434	21 %	2,750	260	2 %	67	655	5 %
uk-2007-host[j]	114,529	1,836,441	54,822	59,160	51 %	51,692	49,379	43 %	3,518	4,766	4 %	259	10,710	9 %
wb-2001[d,e]	118,142,155	1,019,903,190	41,126,852	53,891,939	45 %	3,841	27,659,673	23 %	816,127	5,946,889	5 %	1,218	60,654,989	51 %
arabic-2005[d,e]	22,744,080	639,999,458	4,000,414	15,177,163	66 %	9,905	3,301,085	14 %	575,618	369	<1 %	3,126	9,487,719	41 %
incn-2004[d,e]	7,414,866	194,109,311	1,749,052	3,806,327	51 %	6,985	1,309,127	17 %	256,425	287	<1 %	6,822	3,389,134	45 %
sk-2005[d,e]	50,636,154	1,949,412,601	8,815,057	35,874,412	70 %	12,870	6,903,532	13 %	8,563,808	401	<1 %	4,502	19,301,703	38 %
uk-2005[d,e]	39,459,925	936,364,282	5,811,041	25,711,307	65 %	5,213	4,407,986	11 %	1,776,852	15,661	<1 %	585	16,807,640	42 %
uk-2002[d,e]	18,520,486	298,113,762	3,887,634	12,090,163	65 %	2,450	2,761,116	14 %	194,942	117,975	<1 %	943	8,765,833	47 %
it-2004[d,e]	41,291,594	1,150,725,436	6,753,961	29,855,421	72 %	9,964	5,268,869	12 %	1,326,745	2,538	<1 %	3,199	18,130,880	43 %
uk-2006[j]	77,741,046	2,965,197,340	10,789,143	49,710,330	63 %	20,178	8,439,830	10 %	4,070,239	994,082	1 %	4,986	34,780,373	44 %
uk-2007[j]	105,896,555	3,738,733,648	21,398,038	68,582,555	64 %	15,402	12,949,672	12 %	975,418	6,554,828	6 %	5,664	49,760,746	46 %

[a] See section 2.8.2.

[b] Gray et al. [2007]

[c] Subset of webbase-2001 dataset [Hirai et al., 2000], determined by domain.

[d] Boldi et al. [2004]

[e] Boldi and Vigna [2005]

[f] Barabási et al. [1999a],Barabási et al. [1999b]

[g] Davis [2007], the Kamvar collection

[h] Thelwall [2003]

[i] See section 2.8.1.

[j] Castillo et al. [2006]

[k] Morrison et al. [2005]

SUMMARY

This chapter is long. It covers massive detail about PageRank. It is, however, necessary: subsequent chapters use almost every property mentioned. To be precise,

Chapter 3 weakly and strongly preferential PageRank formulations, PageRank algorithms;

Chapter 4 PageRank at $\alpha = 1$, sensitivity to α, PageRank function of α;

Chapter 5 PageRank algorithms and PseudoRank; and

Chapter 6 implementation details and PageRank optimized on a graph.

All subsequent chapters rely heavily on the notation established here.

There is nothing wrong with change,
if it is in the right direction.

—Winston Churchhill

3 THE PAGERANK DERIVATIVE

The aim of this thesis is to study the sensitivity of PageRank with respect to the damping parameter α. Sensitivity, though more general, is often examined via perturbation. And perturbation theory, applied to PageRank, attacks the question: will a small change in α produce a large change in the PageRank vector? For a sufficiently small change, it is *the derivative* that determines the behavior of any smooth function,[1] and the examination begins there.

Using the derivative is a fine starting point, provided it exists. Does it? As discussed in the previous chapter (sections 2.6 and 2.7), PageRank is a rational function of α for all $0 \le \alpha \le 1$. Thus, the derivative exists. It even exists for *complex* α where $|\alpha| < 1$, though that is not an important fact for this thesis.

Existence sets the stage for the exploration in this chapter. Because the derivative exists, section 3.1 evaluates different algebraic formulations for the derivative vector. With an algebraic expression in hand, section 3.2 next demonstrates a few ways to *compute* the derivative. PageRank derivatives are remarkably close problems to PageRank and the best algorithm involves only computing PageRank, using any algorithm, and computing a second strongly personalized PageRank vector, again using any algorithm.

Algorithms, especially efficient ones, enable experiments. Sometimes, experiments even expose theory. The experiments with the PageRank derivative in section 3.3 follow this trajectory and expose a nice property of a Taylor step along the PageRank derivative. Theory, of course, is not everything and the final section investigates the predictive power of the PageRank derivative.

Studying the derivative began independently around 2004 in three papers. First, Golub and Greif [2006] mentioned it in a 2004 preprint. Second, Boldi et al. [2005] included the derivative for an extrapolation technique. Third, and finally, Berkhin [2005] includes methods to compute the derivative of PageRank.[2]

Throughout the chapter, we freely inject discussions of related background material, though the algorithm to compute the derivative is novel, as are pieces

[1] This follows from the Taylor series $f(x) = f(x_0) + (x - x_0)f'(x) + (1/2)(x - x_0)^2 f''(x) + \dots$ when $x - x_0$ is small. In such a case $(x - x_0)^2$ is minuscule and $f'(x)$ determines the behavior.

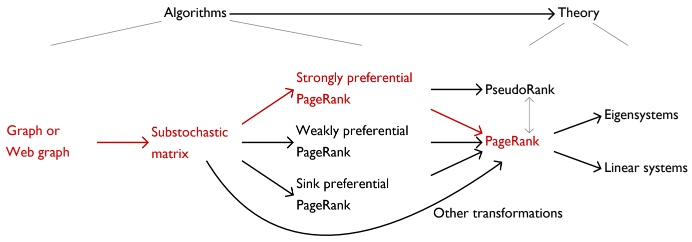
[2] Although two of these are 2005 publications, most would have been submitted in 2004.

of the theoretical discussion. A lackluster conclusion is that the derivative seems too correlated with PageRank, and does not appear to provide any sufficient guidance with regard to appropriate value of α; although, experiments with web spam in the next chapter (section 4.8.4) show that the derivative does have some useful properties.

3.1 FORMULATIONS

Given that the introduction to the chapter mentions that the derivative of the PageRank vector exists, we first address the burning question: what is it?

Remember figure 2.2 and all the different ways of looking at the PageRank problem from the previous chapter?



Must we compute a derivative for *all* of these formulations? As hinted by the top of the figure, only a few formulations are theoretically relevant. The difference between strongly, weakly, and sink preferential are irrelevant for the derivative: all that matters is **P**.[3] With **P** from any of these variations, the derivative vector satisfies the same formulation in terms of **P**.[4] Thus, it suffices to look at the derivative of the *core* PageRank problem alone. The core problem is still either a linear system or an eigensystem, and thus the difference between that choice may matter. Though, as shown shortly, it does not.

The core PageRank problem has enough structure to support the following lemma. It is important to have this lemma about the derivative, because it uses only properties of the PageRank problem—nothing else. It could tell us if a formulation were wrong, for instance.

Lemma 6. *Let* $\mathbf{x}(\alpha)$ *be the solution of a PageRank problem (problem 1) for* $\mathbf{P}, \mathbf{v}$*, and* α*. Then the derivative of PageRank with respect to* α*, denoted* $\mathbf{x}'(\alpha)$*, sums to* 0.

Proof. By definition,

$$\mathbf{x}'(\alpha) = \lim_{\omega \to 0} \frac{\mathbf{x}(\alpha + \omega) - \mathbf{x}(\alpha)}{\omega}.$$

[3] The matrix **P** is the fully column stochastic matrix in the definition of PageRank.

[4] This statement should not be surprising. All the variations converted $\bar{\mathbf{P}}$ to **P** and did not involve α. The conversion has no effect on differentiating with respect to α.

Because the limit of each component exists, we can move the summation inside the limit operation:

$$\mathbf{e}^T\mathbf{x}'(\alpha) = \lim_{\omega\to 0} \mathbf{e}^T \frac{\mathbf{x}(\alpha+\omega) - \mathbf{x}(\alpha)}{\omega}.$$

But $\mathbf{x}(\alpha+\omega)$ and $\mathbf{x}(\alpha)$ are both distribution vectors, which implies $\mathbf{e}^T\mathbf{x}(\alpha+\omega) = \mathbf{e}^T\mathbf{x}(\alpha) = 1$. The difference of these scalars $\mathbf{e}^T(\mathbf{x}(\alpha+\omega) - \mathbf{x}(\alpha)) = 0$. Consequently, the derivative sums to 0. □

Whether or not this result merits the full formal lemma-and-proof treatment is debatable.[5] In the ensuing discussion of the derivative, it is used repeatedly and thus deserves more than passing attention.

Finally, let's see an equation for the derivative.

3.1.1 *The linear system*

The PageRank derivative is easy to determine from the linear system. Recall the system $(\mathbf{I} - \alpha\mathbf{P})\mathbf{x} = (1-\alpha)\mathbf{v}$ from (2.4). Let's make the dependence on α explicit:

$$(\mathbf{I} - \alpha\mathbf{P})\mathbf{x}(\alpha) = (1-\alpha)\mathbf{v}.$$

Separating the left-hand side shows that, implicitly,

$$\mathbf{x}(\alpha) = \alpha\mathbf{P}\mathbf{x}(\alpha) + (1-\alpha)\mathbf{v}.$$

Standard rules of matrix calculus give $\mathbf{x}'(\alpha) = \alpha\mathbf{P}\mathbf{x}'(\alpha) + \mathbf{P}\mathbf{x}(\alpha) - \mathbf{v}$, or more conveniently,

$$(\mathbf{I} - \alpha\mathbf{P})\mathbf{x}'(\alpha) = \mathbf{P}\mathbf{x}(\alpha) - \mathbf{v}. \tag{3.1}$$

The PageRank derivative is extraordinarily close to a PageRank system! It is not a PageRank system because $\mathbf{P}\mathbf{x}(\alpha) - \mathbf{v}$ has some components less than 0 and $\mathbf{e}^T\mathbf{P}\mathbf{x}(\alpha) - \mathbf{e}^T\mathbf{v} = 0$. That is good though. If it were a PageRank problem then $\mathbf{e}^T\mathbf{x}'(\alpha)$ would be 1, but we know it's zero (lemma 6). Indeed, (3.1) satisfies the property that $\mathbf{e}^T\mathbf{x}'(\alpha) = 0$ because it implies $\mathbf{e}^T\mathbf{x}'(\alpha) = \alpha\mathbf{e}^T\mathbf{x}'(\alpha)$. Only one solution is possible for $\alpha < 1 : \mathbf{e}^T\mathbf{x}'(\alpha) = 0$.

Langville and Meyer [2006a] write the derivative *ex nihilo* as

$$\mathbf{x}'(\alpha) = (\mathbf{I} - \alpha\mathbf{P})^{-2}(\mathbf{P} - \mathbf{I})\mathbf{v}.$$

Our preference is to emphasize the PageRank-like structure of the derivative in (3.1). As we shall see after getting around to algorithms, looking at the problem in this manner is highly suggestive of algorithms.

[5] Langville and Meyer [2006a] mention it incidentally, for example.

Next, we see what differs for the eigensystem formulation of the PageRank problem.

3.1.2 *The eigensystem*

Golub and Greif [2006] originally derived a formula for the derivative vector from the eigensystem formulation (2.3) of PageRank, namely

$$\mathbf{M}\mathbf{x}(\alpha) = \mathbf{x}(\alpha),$$

where $\mathbf{M} = \alpha\mathbf{P} + (1-\alpha)\mathbf{v}\mathbf{e}^T$. Differentiating with respect to α yields the singular linear system

$$(\mathbf{I} - \mathbf{M})\mathbf{x}'(\alpha) = \mathbf{P}\mathbf{x}(\alpha) - \mathbf{v}. \qquad (3.2)$$

We ought to be alarmed. Does this mean the eigensystem formulation admits *multiple* PageRank derivatives? In particular, given any solution $\mathbf{x}'(\alpha)$ to (3.2), then $\mathbf{x}'(\alpha) + \theta\mathbf{x}(\alpha)$ is also a solution.[6]

Lemma 6 saves the day. The problem with the eigensystem formulation is that the algebraic eigenvector $\mathbf{x}(\alpha)$ has no imposed norm. In the above analysis, we differentiated the problem independently of the imposed norm and thus we need to use it somehow. Remember that the proof of lemma 6 used the normalization of $\mathbf{x}(\alpha)$ extensively. Thus, not all solutions $\mathbf{x}'(\alpha) + \theta\mathbf{x}(\alpha)$ are derivatives. Only the vector with θ^* such that $\mathbf{e}^T\mathbf{x}'(\alpha) + \theta^*\mathbf{e}^T\mathbf{x}(\alpha) = 0$ is a PageRank derivative.

Finally, then, impose the property $\mathbf{e}^T\mathbf{x}'(\alpha) = 0$ and the right-hand sides of both (3.2) and (3.1) become identical:

$$(\mathbf{I} - \mathbf{M})\mathbf{x}'(\alpha) = (\mathbf{I} - \alpha\mathbf{P} - (1-\alpha)\mathbf{v}\mathbf{e}^T)\mathbf{x}'(\alpha) = (\mathbf{I} - \alpha\mathbf{P})\mathbf{x}'(\alpha).$$

And thus, as we have seen a few times now,[7] it does not matter whether we work with the eigensystem or the linear system. Though the linear system often tends to be less complicated.

3.1.3 *PseudoRank*

In the previous chapter, we introduced PseudoRank and mentioned that it is loosely equivalent to PageRank.[8] PageRank and PseudoRank are different—and importantly so with regard to the derivative. The PseudoRank system ((2.10)) is

$$(\mathbf{I} - \alpha\bar{\mathbf{P}})\mathbf{y} = \sigma\mathbf{v},$$

[6] Remember that $\mathbf{M}\mathbf{x}(\alpha) = \mathbf{x}(\alpha)$ and so $\mathbf{x}(\alpha)$ is exactly the nullspace of $(\mathbf{I} - \mathbf{M})$—which is of dimension 1 because the PageRank vector is unique.

[7] The first example was the limiting vector in section 2.7.

[8] A normalized PseudoRank and PageRank vector are identical for the strongly preferential PageRank problem.

where σ may or may not depend on α. Most often, it does not [Gleich and Zhukov, 2005; McSherry, 2005; Langville and Meyer, 2006a]. When PseudoRank is constructed with σ independent of α, then its derivative satisfies

$$(\mathbf{I} - \alpha\bar{\mathbf{P}})\mathbf{y}'(\alpha) = \bar{\mathbf{P}}\mathbf{y}(\alpha).$$

This system has two important properties. First, while the PageRank derivative satisfies $\mathbf{e}^T\mathbf{x}'(\alpha) = 0$, this PseudoRank derivative has

$$\mathbf{y}'(\alpha) \geq 0,$$

which follows because $(\mathbf{I} - \alpha\bar{\mathbf{P}})$ is an M-matrix (with a non-negative inverse) and $\bar{\mathbf{P}}\mathbf{y}(\alpha)$ is positive. Second, PseudoRank has the property that the derivative of PseudoRank is another PseudoRank system, albeit for a possibility different value of θ.[9]

This second property is exploitable and is a component in an algorithm to compute the PageRank derivative given in the next section.

3.2 ALGORITHMS

So far, PageRank has been differentiated to construct the PageRank derivative algebraically. The next step is to analyze these derivatives, but experimentation is a powerful technique to suggest analysis. To experiment with the derivative requires computing a derivative—that is, an algorithm. Although the derivative vector in (3.1) is the solution of a linear system, can we solve the system more efficiently than a standard problem?

It seems likely. After all, PageRank solves

$$(\mathbf{I} - \alpha\mathbf{P})\mathbf{x}(\alpha) = (1 - \alpha)\mathbf{v},$$

whereas the derivative solves (3.1)

$$(\mathbf{I} - \alpha\mathbf{P})\mathbf{x}'(\alpha) = \mathbf{P}\mathbf{x}(\alpha) - \mathbf{v}.$$

These systems differ only in the right-hand side. Surely something efficient must be possible.

It is. As shown towards the end of this section, a strongly personalized PageRank solver suffices to compute the derivative. Despite the similarities of (3.1) to PageRank, an algorithm with this property is not entirely trivial,

[9] If $\bar{\mathbf{P}}$ is taken to be a stochastic matrix instead, then θ does not change.

unless the goal is merely to approximate the PageRank derivative, in which case consider the following.

A TRIVIAL IDEA A first idea for computing the derivative vector is a central finite difference method,

$$\mathbf{x}'(\alpha) \approx \frac{1}{2\varepsilon}(\mathbf{x}(\alpha+\varepsilon) - \mathbf{x}(\alpha-\varepsilon))$$

for small ε. While attractive for its simplicity, this method requires computing a PageRank vector for a value of α larger than the value of α at the derivative. Additionally, it yields only an approximation to the derivative vector. A first order backward finite difference formula

$$\mathbf{x}'(\alpha) \approx \frac{\mathbf{x}(\alpha) - \mathbf{x}(\alpha-\varepsilon)}{\varepsilon}$$

avoids computing PageRank at a larger value of α, but is less accurate—and dangerously so—when using inexact solutions $\mathbf{x}(\alpha)$. Rather simple analysis shows that using the first order difference is unwise unless the PageRank problems are solved accurately.[10]

A BIG LINEAR SYSTEM From equation (3.1) we can express both the PageRank vector and its derivative vector in the solution to a single linear system. By solving:

$$\begin{bmatrix} \mathbf{I} - \alpha\mathbf{P} & 0 \\ -\mathbf{P} & \mathbf{I} - \alpha\mathbf{P} \end{bmatrix} \begin{bmatrix} \mathbf{x}(\alpha) \\ \mathbf{x}'(\alpha) \end{bmatrix} = \begin{bmatrix} (1-\alpha)\mathbf{v} \\ -\mathbf{v} \end{bmatrix} \tag{3.3}$$

we simultaneously compute the solution vector for both PageRank and its derivative. Boldi et al. [2005] proposed computing the derivative in this manner by solving this linear system with a hybrid-Jacobi/block Gauss-Seidel procedure. One issue with this approach is that it requires solving a linear system twice the size of the PageRank system. Additionally, the linear system does not correspond to a PageRank problem, and it requires a general linear system solver.

TWO SMALLER SYSTEMS We could also, of course, solve the block-triangular linear system (3.3) by first solving $(\mathbf{I} - \alpha\mathbf{P})\mathbf{x}(\alpha) = (1-\alpha)\mathbf{v}$ and then solving $(\mathbf{I} - \alpha\mathbf{P})\mathbf{x}'(\alpha) = \mathbf{P}\mathbf{x}(\alpha) - \mathbf{v}$. But that algorithm is just solving the derivative linear system (3.1) directly.[11]

[10] If the PageRank problems are solved to a tolerance of γ, then each computed vector is roughly $\mathbf{x}(\alpha) + \gamma\mathbf{e}$ for an *error* vector $\mathbf{e}$. Both the central and backwards differences yield an error of γ/ε, and this suggests using a large ε. To get an accurate solution with a large ε requires central differencing.

[11] Techniques for multiple right-hand sides do not help in this case. The systems are generally too big for any factorizations, and other approaches are also inappropriate.

Instead of using these approaches, we devise a method to compute the PageRank vector analytically by solving two PageRank problems. The key to this result is an observation by Golub and Greif [2006]:

$$\mathbf{P}\mathbf{x}(\alpha) - \mathbf{v} = \frac{1}{\alpha}(\mathbf{x}(\alpha) - \mathbf{v}). \tag{3.4}$$

We use this result to rewrite equation (3.1) as

$$(\mathbf{I} - \alpha\mathbf{P})\mathbf{x}'(\alpha) = \frac{1}{\alpha}(\mathbf{x}(\alpha) - \mathbf{v}). \tag{3.5}$$

The vector $\mathbf{x}'(\alpha)$ then decomposes into a linear combination of two PageRank vectors:

$$\mathbf{x}'(\alpha) = \beta\mathbf{z}(\alpha) - \beta\mathbf{x}(\alpha), \tag{3.6}$$

where $(\mathbf{I} - \alpha\mathbf{P})\mathbf{z}(\alpha) = (1 - \alpha)\mathbf{x}(\alpha)$ and $\beta = \frac{1}{\alpha(1-\alpha)}$. This idea yields an algorithm for computing the PageRank derivative as the solution of two PageRank systems with different teleportation distribution vectors but the same value of α. Berkhin [2005] also made this observation.

This reduction is progress. It takes advantage of the structure of the derivative to express it as a combination of PageRank solutions. Recall, however, that efficient *algorithms* for PageRank operate at the level of PageRank variants. One concern with the previous approach is that it requires computing PageRank for a column stochastic matrix $\mathbf{P}$. As discussed in section 2.2.2, many codes for PageRank choose to work with the column sub-stochastic matrix $\bar{\mathbf{P}}$ and use the strongly preferential PageRank model with $\mathbf{P} = \bar{\mathbf{P}} + \mathbf{v}\mathbf{d}^T$.[12] To maximize our computational advantage, we want to solve only strongly-preferential PageRank problems when starting with a strongly-preferential PageRank problem.[13] Virtually all PageRank solvers work with $\bar{\mathbf{P}}$ and implicitly use the strongly preferential framework, whereas relatively few work with $\mathbf{P}$ or the weakly preferential framework.

[12] A column sub-stochastic matrix satisfies

$$(\mathbf{e}^T\bar{\mathbf{P}})_i = \begin{cases} 1 & P_{i,j} > 0 \text{ for any } i \\ 0 & P_{i,j} = 0 \text{ for all } i. \end{cases}$$

[13] For general problems, it will be hard to do better than using the reduction to problems with $\mathbf{P}$.

To be explicit, in the *strongly-preferential* case the PageRank vector satisfies

$$(\mathbf{I} - \alpha\bar{\mathbf{P}} - \alpha\mathbf{v}\mathbf{d}^T)\mathbf{x}(\alpha) = (1-\alpha)\mathbf{x}(\alpha)$$

and its derivative satisfies[14]

$$(\mathbf{I} - \alpha\bar{\mathbf{P}} - \alpha\mathbf{v}\mathbf{d}^T)\mathbf{x}'(\alpha) = \frac{1}{\alpha}(\mathbf{x}(\alpha) - \mathbf{v}).$$

Simply using strongly preferential solves does not work because the PageRank system for $\mathbf{z}(\alpha)$ from (3.6) is

$$(\mathbf{I} - \alpha\bar{\mathbf{P}} - \alpha\mathbf{v}\mathbf{d}^T)\mathbf{z}(\alpha) = (1-\alpha)\mathbf{x}(\alpha),$$

which is a weakly preferential PageRank system.

At this point, Boldi et al. [2007] provide a solution for a related problem They formalize that the strongly and weakly preferential PageRank systems are related by a rank-one change. Applying the Sherman-Morrison-Woodbury formula and an extra PageRank solve transitions between these formulations. For the derivative, applying this technique, however, then requires *three* PageRank solves. The extra solve is not necessary because a bit of algebra fixes the situation entirely, and there is no need for an explicit application of the Sherman-Morrison-Woodbury formula. Notice that

$$\begin{aligned}
(\mathbf{I} - \alpha\bar{\mathbf{P}})\mathbf{x}'(\alpha) &= \frac{1}{\alpha}\mathbf{x}(\alpha) + \underbrace{(\alpha\mathbf{d}^T\mathbf{x}'(\alpha) - 1/\alpha)}_{\delta}\mathbf{v}, \\
(\mathbf{I} - \alpha\bar{\mathbf{P}})\mathbf{x}(\alpha) &= (1 - \alpha + \alpha\mathbf{d}^T\mathbf{x}(\alpha))\mathbf{v}, \text{ and} \\
(\mathbf{I} - \alpha\bar{\mathbf{P}})\mathbf{z}(\alpha) &= (1 - \alpha + \alpha\mathbf{d}^T\mathbf{z}(\alpha))\mathbf{x}(\alpha).
\end{aligned}$$

Consequently, $\mathbf{x}'(\alpha)$ is still a linear combination of $\mathbf{z}(\alpha)$ and $\mathbf{x}(\alpha)$ where each is a strongly preferential PageRank vector. The coefficient for $\mathbf{z}(\alpha)$ is available, so

$$\mathbf{x}'(\alpha) = \frac{1}{\alpha(1 - \alpha + \alpha\mathbf{d}^T\mathbf{z}(\alpha))}\mathbf{z}(\alpha) + \eta\mathbf{x}(\alpha).$$

We now exploit $\mathbf{e}^T\mathbf{x}'(\alpha) = 0$ to compute η and present algorithm 1 to compute the derivative.

No algorithm in this thesis is missing Matlab code, and program 4 shows a simple implementation of this algorithm.

[14] This derivative is just the same as $(\mathbf{I} - \alpha\mathbf{P})\mathbf{x}'(\alpha)$ with $\mathbf{P} = \bar{\mathbf{P}} + \mathbf{v}\mathbf{d}^T$, which supports working with $\mathbf{P}$ for PageRank theory.

Algorithm 1 – Compute the derivative of PageRank.

```
1. Compute $\mathbf{x}(\alpha)$ as the solution to the original strongly preferential PageRank problem, $(\mathbf{I} - \alpha\bar{\mathbf{P}} - \alpha\mathbf{v}\mathbf{d}^T)\mathbf{x}(\alpha) = (1-\alpha)\mathbf{v}$.
2. Compute $\mathbf{z}(\alpha)$ as the solution to the strongly preferential PageRank problem with teleportation distribution $\mathbf{x}(\alpha)$, $(\mathbf{I} - \alpha\bar{\mathbf{P}} - \alpha\mathbf{x}(\alpha)\mathbf{d}^T)\mathbf{x}(\alpha) = (1-\alpha)\mathbf{x}(\alpha)$.
3. Set $\tilde{\mathbf{z}} = \frac{1}{\alpha(1-\alpha+\alpha\mathbf{d}^T\mathbf{z}(\alpha))}\mathbf{z}(\alpha)$.
4. Compute $\eta = \frac{-\mathbf{e}^T\tilde{\mathbf{z}}}{\mathbf{e}^T\mathbf{x}(\alpha)}$.
5. Return $\mathbf{x}'(\alpha) = \tilde{\mathbf{z}} + \eta\mathbf{x}(\alpha)$.
```

Program 4 – Strongly-preferential PageRank derivatives. Our PageRank codes for Matlab use *row sub-stochastic* matrices P.

```
function xp = derivpr(P,alpha,x,z)
% DERIVPR Compute the derivative of PageRank
% Given a PageRank vector x(alpha) and a PageRank vector y(alpha) that
% satisfy
%   (I - alpha P’)x = (1-alpha)v
%   (I - alpha P’)z = (1-alpha)x
% we produce the derivative of PageRank at alpha.
d = 1 - full(sum(P,2)); d = round(d); % compute the dangling vector
zt = z./(alpha*(1-alpha+alpha*d’*z));
g = -csum(zt)/csum(x);
xp = zt + g*x;
```

A full investigation of algorithms for PageRank derivatives ought to include a discussion about the stability of the algorithms. Such a discussion is a glaring omission of this chapter as the vectors $\mathbf{x}$ and $\mathbf{z}$ in algorithm 1 will not be accurately computed. As an ode to the missing analysis, let us note that algorithm 1 produces a derivative vector that satisfies $\mathbf{e}^T\mathbf{x}'(\alpha) = 0$ to machine precision. This property should be useful for a backward stability analysis. Such analyses are usually difficult to conduct and we do not expect this case to be an exception.

It is now time to address other properties of the derivative.

3.3 ANALYSIS

Studying algorithm 1 from the previous section to investigate the derivatives reveals a few interesting properties. The investigation begins with taking a Taylor step along the derivative.

3.3.1 *Taylor steps*

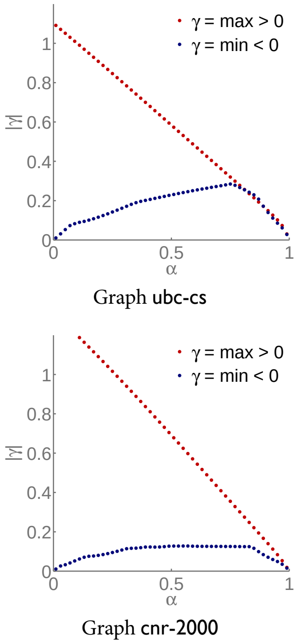
A key property of the PageRank vector is that it is a probability distribution. Thus, $\mathbf{e}^T\mathbf{x}(\alpha) = 1$ and $x(\alpha)_i \geq 0$. Consider approximations of PageRank vectors using the derivative

$$\mathbf{y}(\gamma) = \mathbf{x}(\alpha) + \gamma\mathbf{x}'(\alpha). \tag{3.7}$$

This equation is just a first order Taylor approximation of the function $\mathbf{x}(\alpha+\gamma)$ around α. When is $\mathbf{y}(\gamma)$ also a probability distribution? The answer to this question reveals when $\mathbf{y}(\gamma)$ should not be used as an approximate PageRank vector.

Figure 3.1 shows that $\gamma \approx 1-\alpha$ is the largest positive step until any component of $\mathbf{y}(\gamma)$ dips below 0 or exceeds 1. The minimum values of γ until $\mathbf{y}(\gamma)$ is no longer a probability distribution are always less than 0 but are not nearly as structured as the permissible positive set.

This experiment inspired the following theorem.



Graph ubc-cs

Graph cnr-2000

Figure 3.1 – Valid Taylor steps. The maximum values of γ until $\mathbf{y}(\gamma)$ loses positivity are nearly linear at $1-\alpha$. This figure inspired theorem 7.

Theorem 7. *Fix $\mathbf{P}, \mathbf{v}$, and α and let $\mathbf{x}(\alpha)$ and $\mathbf{x}'(\alpha)$ be the PageRank vector and derivative with respect to α. Then $\mathbf{y}(\gamma) = \mathbf{x}(\alpha) + \gamma\mathbf{x}'(\alpha)$ is a PageRank vector for $0 \leq \gamma < 1-\alpha$ with teleportation distribution vector $\mathbf{w}(\gamma) = \frac{1}{1-\alpha}((1-\alpha-\gamma)\mathbf{v} + \gamma\mathbf{P}\mathbf{x}(\alpha))$.*

Proof. The proof is straightforward and follows by computing

$$\begin{aligned}(\mathbf{I}-\alpha\mathbf{P})\mathbf{y}(\gamma) &= (\mathbf{I}-\alpha\mathbf{P})\mathbf{x}(\alpha) + \gamma(\mathbf{I}-\alpha\mathbf{P})\mathbf{x}'(\alpha)\\ &= (1-\alpha)\mathbf{v} + \gamma(\mathbf{P}\mathbf{x}(\alpha) - \mathbf{v})\\ &= (1-\alpha)\mathbf{w}(\gamma).\end{aligned} \tag{3.8}$$

First, notice that $\mathbf{e}^T\mathbf{w}(\gamma) = 1$. To verify $w(\gamma)_i \geq 0$ it suffices to show $(1-\alpha-\gamma)v_i + \gamma[\mathbf{P}\mathbf{x}(\alpha)]_i \geq 0$. From $v_i \geq 0$, we have $x(\alpha)_i \geq 0$ and then $\gamma[\mathbf{P}\mathbf{x}(\alpha)]_i \geq$ follows. From $0 \leq \gamma < 1-\alpha$, we have $(1-\alpha-\gamma)v_i \geq 0$ and then $w(\gamma)_i$ is the sum of two positive quantities. □

Showing non-negativity, and thus confirming figure 3.1, was the point of this theorem. It accomplishes this goal. For any $\gamma < 1-\alpha$, $\mathbf{y}(\gamma)$ is a non-negative probability distribution vector. It is, however, more than just any positive vector, it's a PageRank vector with the same α, just a different teleportation distribution.

To confirm theorem 7, we examine the difference between the approximation $\mathbf{y}(\gamma)$ and the PageRank vector with teleportation distribution $\mathbf{w}(\gamma)$,

Graph	$\gamma = 0.001$	$\gamma = 0.01$	$\gamma = 0.1$
aa-stan	1.72×10^{-10}	1.72×10^{-9}	4.30×10^{-8}
ee-stan	5.62×10^{-11}	5.62×10^{-10}	5.62×10^{-9}
cs-stan	5.31×10^{-11}	5.31×10^{-10}	2.90×10^{-10}
cnr-2000	1.79×10^{-10}	1.79×10^{-9}	5.35×10^{-9}

Table 3.1 – Experimental validation of theorem 7. The table entries show the value of $\|\mathbf{y}(\gamma) - \mathbf{x}(\mathbf{w}(\gamma), \alpha)\|_2$ using the notation from section 3.3.1 with $\alpha = 0.85$.

denoted $\mathbf{x}(\mathbf{w}(\gamma), \alpha)$. Note that $\mathbf{x}(\mathbf{w}(\gamma), \alpha)$ is *not* a strongly preferential PageRank vector. The norm of the differences are listed in table 3.1 for a few graphs and values of γ. These norms are quite small, demonstrating experimental evidence for the theorem.

3.3.2 *Bounds*

An immediate implication of the previous theorem is that

$$x'(\alpha)_i < \frac{1}{1-\alpha}.$$

Otherwise $y(1-\alpha-\varepsilon)_i > 1$ for some small but positive ε.[15] The same idea works to show that $x'(\alpha)_i > -\frac{1}{1-\alpha}$, because otherwise $y(1-\alpha-\varepsilon) < 0$ for some small ε. Thus

$$|x'(\alpha)| \leq \frac{1}{1-\alpha}.$$

Using rather different techniques, Langville and Meyer [2006a, p.66] establish this same fact.

As $\alpha \to 1$, these bounds become meaningless. What actually happens with the derivative at $\alpha = 1$ is discussed next.

3.3.3 *Limiting derivatives*

Section 2.7 establishes the PageRank vector when $\alpha = 1$. Now, we differentiate the explicit PageRank function to establish the PageRank derivative when $\alpha = 1$. Given $\mathbf{P} = \mathbf{X}\mathbf{J}\mathbf{X}^{-1}$ with $\mathbf{J} = \begin{bmatrix} \mathbf{I} & \\ & \mathbf{J}_1 \end{bmatrix}$,[16] $\mathbf{X} = \begin{bmatrix} \mathbf{X}_0 & \mathbf{X}_1 \end{bmatrix}$, and $\mathbf{X}^{-1} = \begin{bmatrix} \mathbf{Y}_0 & \mathbf{Y}_1 \end{bmatrix}^T$, then we have

$$\mathbf{x}(\alpha) = \mathbf{X}_0\mathbf{Y}_0\mathbf{v} + (1-\alpha)\mathbf{X}_1(\mathbf{I} - \alpha\mathbf{J}_1)^{-1}\mathbf{Y}_1\mathbf{v}$$

and

$$\mathbf{x}'(\alpha) = (\alpha - 1)\mathbf{X}_1(\mathbf{I} - \alpha\mathbf{J}_1)^{-1}\mathbf{J}_1(\mathbf{I} - \alpha\mathbf{J}_1)^{-1}\mathbf{Y}_1 - \mathbf{X}_1(\mathbf{I} - \alpha\mathbf{J}_1)^{-1}\mathbf{Y}_1\mathbf{v}.$$

This result matches Langville and Meyer [2006a, theorem 6.1.3], but with an explicit form for the group inverse of $(\mathbf{I} - \mathbf{P})$ using the Jordan form of $\mathbf{P}$.

[15] In a less succinct statement, the idea is to assume that $x(\alpha)_i = 0$ and use $y = 1 - \alpha$ to bound the maximum of $x'(\alpha)_i$ so that $y(\gamma)_i < 1$.

[16] In the last chapter, $\mathbf{J} = \begin{bmatrix} \mathbf{I} & & \\ & \mathbf{D}_1 & \\ & & \mathbf{J}_2 \end{bmatrix}$ for a diagonal $\mathbf{D}_1$ with all simple eigenvalues of $|\lambda| = 1$ and $\mathbf{J}_2$ with all other eigenvalues. Thus, $\mathbf{J}_1 = \begin{bmatrix} \mathbf{D}_1 & \\ & \mathbf{J}_2 \end{bmatrix}$.

3.4 EXPERIMENTS

Finally, we study the predictive power of the PageRank derivative.

3.4.1 *Does a negative derivative justify a change in ranking?*

One of the most promising uses of the derivative vector is to evaluate what happens in the PageRank vector at different values of α. Table 3.2 shows some results on this idea where we look at the fraction of pages with negative derivative that actually decrease in rank when α increases by a value γ. The fraction predicted by the derivative is higher than the average fraction predicted by a random vector. We do not consider the magnitude of the derivative with these predictions.

These results are mixed. For large values of γ, cnr-2000 shows a marked increase in predictive power using the derivative over a random vector. On the Wikipedia graphs, in contrast, there is almost no difference between using a random vector and the derivative.

3.4.2 *What are the pages with the largest derivative?*

For the largest strongly connected component of wiki-2006-11, table 3.3 lists the top 20 pages with largest derivative for a few values of α. Most of the pages that appear in the top 20 list are also highly ranked according to the PageRank value. Additionally, pages in the "category" namespace in Wikipedia are highly ranked by both PageRank and its derivative for the two largest values of α evaluated.

Table 3.2 – Prediction of rank change with the derivative. The $\mathbf{x}'$ entries show the fraction of pages with negative derivative that decreased in rank when α is increased by the value of γ in the table heading. These values are compared with the $\bar{\mathbf{r}}$ entries, which show the average fraction over 50 trials in which the derivative is replaced by a random vector generated with `randn` in MATLAB. For aa-stan, the ranking did not change and thus all predictions were incorrect.

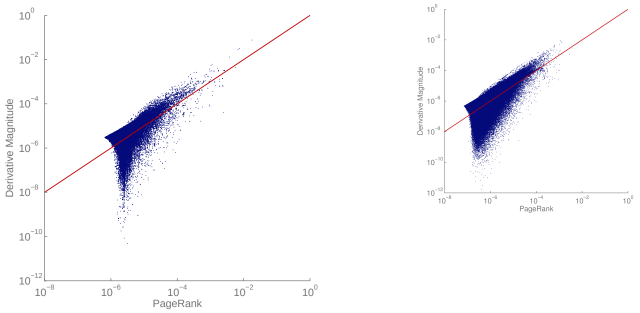
Graph	$\gamma = 0.001$		$\gamma = 0.01$		$\gamma = 0.1$	
	$\mathbf{x}'$	$\bar{\mathbf{r}}$	$\mathbf{x}'$	$\bar{\mathbf{r}}$	$\mathbf{x}'$	$\bar{\mathbf{r}}$
aa-stan	0.000	0.000	0.000	0.000	0.000	0.000
ee-stan	0.079	0.078	0.286	0.266	0.478	0.453
cs-stan	0.257	0.237	0.441	0.372	0.505	0.432
cnr-2000	0.557	0.477	0.621	0.527	0.641	0.553
wiki-2006-09	0.385	0.362	0.385	0.362	0.361	0.342
wiki-2006-11	0.385	0.361	0.383	0.360	0.360	0.341

Table 3.3 – Pages in Wikipedia with the largest derivative. The columns show the top 20 pages with largest derivative for three values of α computed on the largest strongly connected component in wiki-20061104. The pages are presented in order of the derivative, so "United States" has the largest derivative at $\alpha = 0.50$ and its rank according to PageRank is 1. Pages in the category namespace are abbreviated "C:" instead of the full "Category:". The page "Category:Main topic classification" is abbreviated "C:Main topic classif." Likewise "List of academic disciplines" is abbreviated "List of acad. disciplines."

α = 0.5		**α = 0.85**		**α = 0.95**	
Page	**Rank**	**Page**	**Rank**	**Page**	**Rank**
United States	1	Portal:List	6	Portal:List	2
Race (US Census)	6	C:Main topic classif.	4	C:Main topic classif.	3
C:Categories by country	23	C:Society	3	C:Society	4
United Kingdom	4	C:Political geography	10	Wikipedia	8
2006	3	Wikipedia	24	C:Fundamental	9
England	5	C:Fundamental	22	C:Science	21
Canada	7	C:Geography	25	C:Social sciences	12
2005	8	C:Social sciences	29	C:Geography	10
France	10	C:Politics	27	Portal:Browse	83
C:Society	108	C:Science	49	C:Portals	79
C:People	63	C:Human geography	41	C:Academic disciplines	36
C:Living people	2	C:Countries	28	C:Political geography	5
Germany	12	Human	36	C:Politics	16
Australia	11	C:Business	45	C:Nature	40
2004	9	C:People	16	List of acad. disciplines	63
World War II	18	C:Academic disciplines	98	Human	25
C:Political geography	188	C:Nature	94	C:Humans	41
Japan	14	C:Categories by country	5	Academia	75
Europe	32	C:Geography by place	38	Philosophy	64

3.4.3 *Is the derivative related to PageRank?*

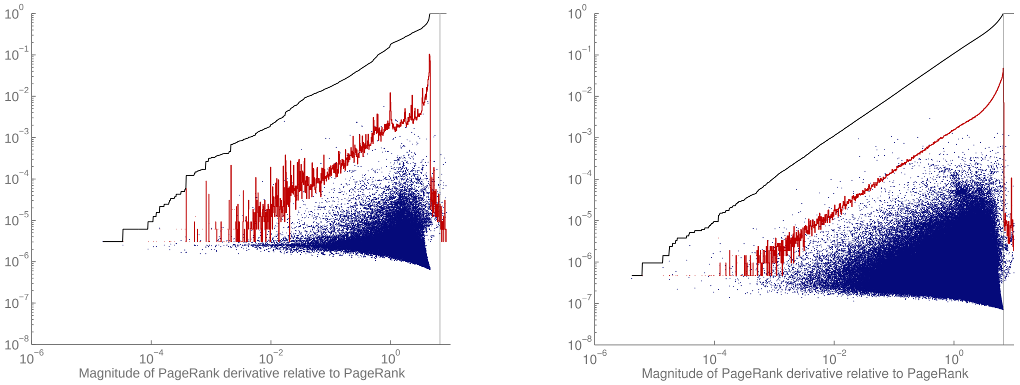
In the last table, many of the pages with large derivatives also had large PageRank values. The final experiment compares PageRank and its derivative to address whether $x'(\alpha)_i$ is proportional to $x(\alpha)_i$. The next two figures (figures 3.2 and 3.3) show the magnitude of the PageRank derivative as a function of the PageRank value.



(a) cnr-2000

(b) wiki-20061I-scc

Figure 3.2 – PageRank derivative magnitude. For both of these graphs, PageRank and the magnitude of its derivative are roughly proportional. The red line shows the equality relationship. Unfortunately, this scatter plot does not show the density of points inside the inner area.



(a) cnr-2000

(b) wiki-20061I-scc

Figure 3.3 – Relative magnitude of the PageRank derivative. The dots are values of $|x'(\alpha)_i/x(\alpha)_i|$ with height $x(\alpha)_i$, the red line is a histogram of the horizontal density, and the black line is the cumulative histogram. For this plot, the three lines share the same vertical axis, which is either a PageRank value, the fraction of total points in a histogram bin, or the total fraction of all points in a cumulative histogram. Note the log-scale on both axes. This figure implies that between 5 and 10% of the density of a linear fit is accounted for by a single relationship between PageRank and its derivative. Each vertical line is at the position $1/(1-\alpha)$.

These figures strongly support the idea that PageRank and its derivative are nearly proportional. Such a relationship is not entirely surprising. Pages with large PageRank may be more sensitive to changes in α because they depend on the PageRank values of all incoming links. Also, the PageRank derivative is a linear combination with the PageRank vector itself. Nevertheless, such a finding is disappointing.

3.5 DISCUSSION

These results are mixed. For some graphs, the derivative is successful at predicting rank changes; and for others, it is not. Considering the magnitude of the derivative along with its sign could improve these results.

For two graphs, the magnitude of PageRank and its derivative appear strongly correlated. Whether this result holds for a large set of graphs is not addressed, but our tacit experience from working with the codes and algorithms indicates that it does. The derivative, then, does not seem to be helpful as an additional ranking vector. But, there may be other uses for it. In the next chapter, we see that using information from the derivative helps a web spam identification task (section 4.8.4), and in the future work discussion (section 7.2), we propose using it to accelerate a computation from the next chapter.

SUMMARY

Computing the derivative of PageRank is not a problem. Algorithm 1 handily accomplishes this task. Given that algorithm, the derivative only helps us confirm that the PageRank vector is sensitive to α and does not help pick a value for α. Experiments with the derivative demonstrate variable accuracy at predicting rank decreases, and that the magnitude of derivative appears to be proportional to the PageRank value for many components.

Numquam ponenda est
pluralitas sine necessitate.

—William of Ockham

4 RANDOM ALPHA PAGERANK

Thus far in the thesis, all the PageRank computations used a particular value of α. Sometimes it was 0.85, sometimes 0.75 or 0.95. We even studied a range of values at one point (figure 2.4). These choices were arbitrary and largely motivated by the "standard" choice of α, namely, 0.85. This chapter begins by asking, *what should α be*?

Recall that the PageRank modification for a Markov chain transforms any input Markov chain into an irreducible, aperiodic chain with a unique stationary distribution. Elements of this unique stationary distribution give the importance of the nodes in the state space of the input Markov chain. Brin and Page proposed the PageRank method to measure the *global* importance of web pages under the behavior of a *random surfer*, which can be interpreted as a Markov chain on the web graph [Page et al., 1999]. We now focus on this random surfer model and show that it contains a slight error when interpreted over a set of "surfers."

Let us begin by revisiting the putative random surfer. With probability α, the surfer follows the links of a web page uniformly at random. With probability $1 - \alpha$, the surfer jumps to a different page according to a given probability distribution over web pages. Because of its influence on these random jumps, the value α is often called the *teleportation parameter*.

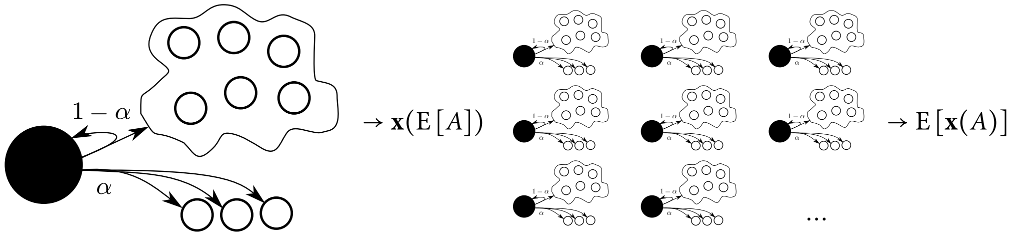
Thus, the PageRank value for a web graph depends on two quantities: the parameter α and the given distribution over the pages. The effect of both of these quantities on the mathematics of the PageRank vector are fairly well understood, but the choice of α is not well justified in the context of the random surfer model. Existing PageRank calculations use a single value of α and two choices stand out in the literature: $\alpha = 0.5$ [Katz, 1953; Avrachenkov et al., 2007; Chen et al., 2007] and $\alpha = 0.85$ [Page et al., 1999; Najork et al., 2007]. These choices are discussed in section 4.3.1.

Rather than trying and testing arbitrary values of α, suppose we pick α to make the random surfer model more accurate. Because α really ought to be the probability of following a link on a web page, let's make it so.

Empirically measured browsing patterns on the web show that individual users, unsurprisingly, have different browsing behavior [Huberman et al., 1998; White and Drucker, 2007]. We also confirm this result in section 4.5. If we assume that all users have their own probability α_i of teleporting, then the PageRank model suggests we should set $\alpha = \frac{1}{N}\sum_{i=1}^{N}\alpha_i$, i.e. the mean of these values. More generally, if A is a random variable with a density function encoding the distribution of teleportation parameters amongst multiple (perhaps infinite) surfers, then the PageRank model suggests $\alpha = \mathrm{E}[A]$, where $\mathrm{E}[\cdot]$ is the expectation operator.

The flaw in PageRank is that using $\alpha = \mathrm{E}[A]$ still does not yield the correct PageRank vector in light of the surfer values α_i. We will justify this statement shortly; intuitively it arises because a single value of α condenses all surfers into a single über-surfer. Instead, we propose to give a small vote to the PageRank vector $\mathbf{x}(\alpha_i)$ corresponding to each random surfer and create a global metric that amalgamates this information. In other words, we want to examine the random surfer model with "$\alpha = A$," where A is a random variable modeling the users' individual behaviors. Figure 4.1 gives a pictorial view of this change. If A is a random variable, then the PageRank vector $\mathbf{x}(A)$ is a random vector, and we can synthesize a new ranking measure from its statistics. We call this measure **R**andom **A**lpha **P**age**r**ank (RAPr), it is pronounced "wrapper."

An earlier work, Constantine and Gleich [2007], introduced a means of handling multiple surfers in PageRank. This chapter extends those ideas by clarifying the presentation, expanding the computational algorithms, and compiling additional results. In particular, the previous paper used the polynomial chaos approach to investigate the behavior of multiple surfers algorithmically. In Constantine et al. [2009], we showed that the polynomial chaos and quadrature methods are equivalent in the case of PageRank. The



(a) The PageRank Model

(b) Our random α PageRank model

Figure 4.1 – Differences between PageRank and the Random Alpha PageRank model. The PageRank model assumes a single random surfer representing an expected user. Our model assumes that each surfer is unique with a different value of α, which we represent as a random variable A. If the function $\mathbf{x}(\cdot)$ gives the PageRank vector for a deterministic or random α or A, respectively, we then compute the expected PageRank given the distribution for A.

presentation in this thesis eliminates the discussion of polynomial chaos beyond this paragraph. Finally, Constantine [2009] explores the general setting of parameterized matrix equations with one or many parameters.

In what follows, we explain and analyze the RAPr model. This model has strong connections with other path damping approaches to PageRank computation, which we show in sections 4.3.3 and 4.4.3. For the interested reader, we present our algorithms with actual MATLAB code from our RAPr suite of codes.

4.1 NOTATION

For this chapter, we'll need to introduce additional notation to handle the concepts from probability. See table 4.1 for a summary. Random variables are denoted by capital, non-subscribed, roman letters. Only A, the random α, is used frequently.

Another small change from the standard notation is that lower case roman letters a, b, l, r denote scalar parameters or bounded intervals.[1] These will be clear from context. For instance, they are used in the example below.

There are two other novelties: the expectation operator $\mathrm{E}[\cdot]$ and the standard deviation operator $\mathrm{Std}[\cdot]$. Given a continuous random variable[2] A with a density function $\rho(x)$ on the interval $[l, r]$, then we define

$$\mathrm{E}[A] = \int_l^r \rho(x)\,dx. \tag{4.1}$$

Evaluating the expectation of a function corresponds to

$$\mathrm{E}[f(A)] = \int_l^r f(x)\rho(x)\,dx. \tag{4.2}$$

The standard deviation operator is defined in terms of the expectation operator

$$\mathrm{Std}[A] = \sqrt{\mathrm{E}[(A - \mathrm{E}[A])^2]}. \tag{4.3}$$

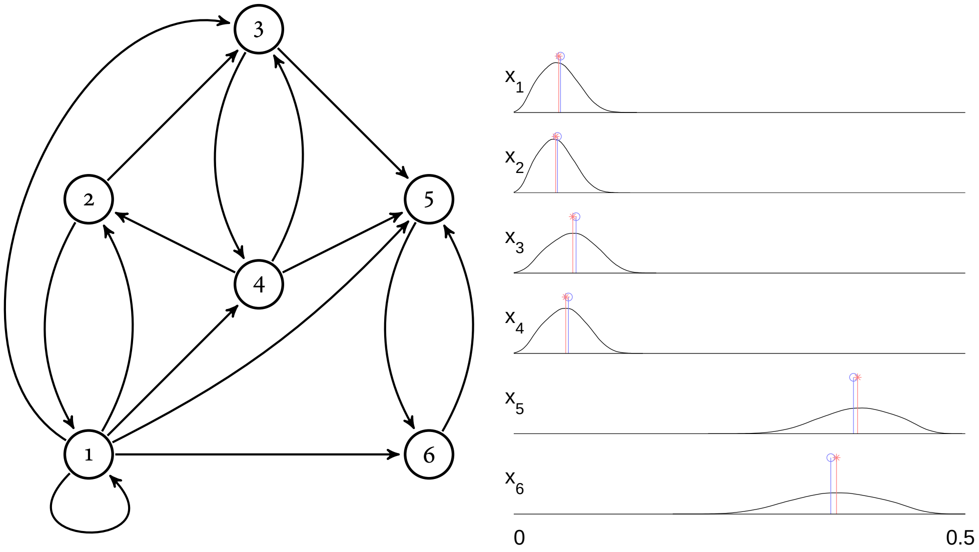
A good background on probability is Grinstead and Snell [1997].

Symbol	Meaning
A	a random variable for α
$\mathrm{Beta}(a, b, [l, r])$	the Beta distribution, see section 4.4.1
$\mathrm{Beta}(\cdot, \cdot)$	the Beta function
δ_{ij}	the Kronecker delta
$\mathrm{E}[\cdot]$	the expectation operator
$\mathrm{Std}[\cdot]$	the standard deviation operator

Table 4.1 – Additional notation for the random alpha PageRank model. Capital roman letters, such as A, denote random variables in this chapter.

[1] In the notation defined in section 2.1, scalars were Greek symbols. Switching to roman for a few scalars helps interpretability for this chapter.

[2] Such random variables are quite special, but they suffice for this chapter.



(a) A small web graph

(b) Density functions

Figure 4.2 – An example of the Random Alpha PageRank model. A simple web graph and approximate probability density functions of the corresponding PageRank random variables. This shows that pages 5 and 6 have the highest variance (widest density function). These pages are traps from which the random surfer cannot leave. In this plot, $A \sim \mathrm{Beta}(2, 16, [0, 1])$. In figure 4.2b, the circle stems show the PageRank value for $\alpha = \mathrm{E}[A] = 0.85$, whereas the star stems show the expectation according to the PageRank density.

4.2 VISION

The PageRank vector $\mathbf{x}(\mathrm{E}[A])$ does not incorporate the surfing behavior of all users; we propose to use $\mathrm{E}[\mathbf{x}(A)]$ instead. Because the PageRank vector is a nonlinear function of α, we do not expect $\mathrm{E}[\mathbf{x}(A)] = \mathbf{x}(\mathrm{E}[A])$, and section 4.4 gives a formal counterexample. For reasonable distributions of A, however, we expect

$$\mathbf{x}(\mathrm{E}[A]) \approx \mathrm{E}[\mathbf{x}(A)]\,. \tag{4.4}$$

Despite this similarity, moving from the deterministic $\mathbf{x}(\alpha)$ to the random $\mathbf{x}(A)$ yields more information. For a given page, its "PageRank" is now a random variable. Figure 4.2 shows the probability density functions for the PageRank random variables on a small graph.

We can use the *standard deviation* of the random variables to help "quantify the uncertainty" in the PageRank value. The standard deviation is a measure of the variability in the PageRank induced by the variability in A. For the graph in figure 4.2, the standard deviation vector is

$$\mathrm{Std}\left[\mathbf{x}(A)\right] = \begin{bmatrix} 0.021\,332 \\ 0.019\,883 \\ 0.026\,146 \\ 0.023\,193 \\ 0.041\,233 \\ 0.049\,304 \end{bmatrix}.$$

This vector shows that x_5 and x_6 have the highest standard deviation. In a traditional PageRank context, these pages are both in a sink-component and accumulate rank from the largest connected component (x_1, x_2, and x_4). A high standard deviation signals that the rank of these pages is "more likely" to change for different realizations of A.

Another interesting quantity derived from our model is the correlation coefficient between ranks. The correlation coefficient between two ranks x_i and x_j provides a measure of how x_i will vary as x_j varies with different realizations of A. If the correlation between x_i and x_j is positive then an increase in x_i from separate realizations of A implies that x_j tends to *increase* as well. If the correlation is negative, then an increase in x_i implies a *decrease* in x_j is likely. Here, we have computed the correlation coefficient between all pages:[3]

$$\begin{bmatrix} 1.000\,000 & 0.999\,996 & 0.998\,844 & 0.999\,211 & -0.999\,951 & -0.999\,373 \\ 0.999\,996 & 1.000\,000 & 0.998\,764 & 0.999\,149 & -0.999\,936 & -0.999\,313 \\ 0.998\,844 & 0.998\,764 & 1.000\,000 & 0.999\,963 & -0.999\,261 & -0.999\,920 \\ 0.999\,211 & 0.999\,149 & 0.999\,963 & 1.000\,000 & -0.999\,550 & -0.999\,989 \\ -0.999\,951 & -0.999\,936 & -0.999\,261 & -0.999\,550 & 1.000\,000 & 0.999\,667 \\ -0.999\,373 & -0.999\,313 & -0.999\,920 & -0.999\,989 & 0.999\,667 & 1.000\,000 \end{bmatrix}.$$

The sign pattern in the correlation structure shows that there are effectively two groups of pages, (x_1, x_2, x_3, x_4) and (x_5, x_6).

Thus far, we have seen a few useful quantities that we can derive from the RAPr model. In practice, we anticipate many problem-dependent uses for these quantities. Our experiments show that the standard deviation vector is uncorrelated (in a Kendall-τ sense[4]) with the PageRank vector itself. Because pages with a high standard deviation have highly variable PageRank values,

[3] This matrix is symmetric, so we could simply present one half of it.

[4] Kendall's τ difference in *concordant* and *discordant* pairs between two lists relative to an identical ordering and an inverted ordering. The τ value is 1 for identical lists and -1 for inverted lists.

the standard deviation vector could be an important input to a machine learning framework for web search or web page categorization.

The correlation structure between the random ranks indicates that some of the pages form natural groups. One may explore connections between negatively correlated ranks to glean information from the underlying graph. We do not pursue this idea further, though it may aid in applications such as spam detection.[5]

Another application for these techniques is local site analysis. On a website such as Wikipedia, the entire graph structure is available. Further, site usage logs contain the information necessary to generate the vector **v** based on incoming searches. These same logs also contain the information necessary to estimate the distribution of A. With the RAPr formulation, extra information is then available to help the site owner understand how people use the site.[6]

More generally, the PageRank model has become a key tool for network and graph analysis. It has been used to find graph cuts [Andersen et al., 2006], infer missing values on a partially labeled graph [Zhou et al., 2005], find interesting genes [Morrison et al., 2005], and help match graph structures in protein networks [Singh et al., 2007].[7] In all of these cases, the random surfer model does not directly apply. Each paper picks a particular value for α and computes a PageRank vector from that value. With RAPr, each case will have a natural random variable. For most, it may be a uniform distribution. Rather than reporting just a single number, the algorithms could use the standard deviation as natural error bounds representing uncertainty or sensitivity in the resulting PageRank vector. The sensitivity using the standard deviation accounts for fluctuations in the function over a wider interval than the derivative.

4.3 RELATED WORK

Our ideas have strong relationships with a few other classes of literature. Before delving into the details of the RAPr model, we'd like to discuss these relationships.

4.3.1 *Teleportation parameters in literature*

Algorithmic papers on PageRank tend to investigate the behavior of PageRank algorithms for multiple values of α [Kamvar et al., 2003; Golub and Greif, 2006], whereas evaluations of the PageRank vector tend to use the canonical value $\alpha = 0.85$ [Najork et al., 2007].

[5] There is already a paper on using a closely related idea for spam detection. See section 4.3.5.

[6] One of the most useful observations from this model is when people use the site in a way that is not predicted by the random surfer model with fitted parameters. This indicates that random surfer models are not appropriate and could suggest monitoring a different set of statistics.

[7] See section 1.4 for an informal description of these topics.

Katz [1953] used $\alpha = 0.5$ in a model closely related to PageRank.[8]

More recently, two papers suggested $\alpha = 0.5$ for PageRank. Back in section 2.6, we discussed Avrachenkov et al. [2007]. They argue that $\alpha = 1/2$ is the right choice.[9]

The second paper applies the random surfer model to a graph of literature citations [Chen et al., 2007]. They claim that citation behavior on literature networks contain very short citation paths of average length 2 based on co-citation analysis. This analysis then suggests $\alpha = 0.5$.

4.3.2 *Usage logs and behavior analysis*

Huberman et al. [1998] studied the behavior of web surfers, even before the original paper on PageRank, and suggested a Markov model for surfing behavior. In contrast with the Brin and Page random surfer, Huberman et al. [1998] empirically measure the probability that surfers follow paths of length ℓ and then compute

$$\mathbf{n}_\ell = f_\ell \mathbf{P}^\ell \mathbf{n_0} \tag{4.5}$$

for the expected number of surfers on each page after ℓ transitions. They found that f_ℓ, the probability of following a path of length ℓ, is approximately an inverse Gaussian; see figure 4.3. This model is strongly related to the path damping models discussed next and in section 4.4.3. An earlier study showed that the average path length of users visiting a site decayed quickly, but did not match the decay to a distribution [Catledge and Pitkow, 1995]. Both of these studies focused on the browsing behavior at a single site and not across the web in general. Subsequently, many papers suggest measuring surfer behavior from usage logs to improve local site search [Wang, 2002; Xue et al., 2003; Farahat et al., 2006].

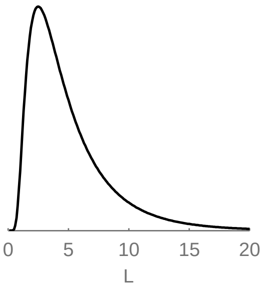


Figure 4.3 – Inverse Gaussian density. The inverse Gaussian distribution has a probability density $\rho(L) = \sqrt{\frac{\lambda}{2\pi L^3}} \exp\left[\frac{-\lambda(L-\mu)^2}{2\mu^2 L}\right]$ supported on $L = (0, \infty)$. The density plotted here uses $\mu = 5, \lambda = 10$.

In the context of web search, a recent study identified two types of surfers: navigators and explorers [White and Drucker, 2007]. Navigators proceed roughly linearly whereas explorers frequently branch their browsing patterns and revisit previous pages before going to new links. The first behavior corresponds to a larger value of α than the latter. This paper also contains an extensive review of relevant literature.

None of these studies directly measures α or the distribution A.

A recent patent from Yahoo! [Berkhin et al., 2008] describes a modification of the PageRank equations to build a "user-sensitive PageRank" system by incorporating observed page transitions and user segment modeling.[10] The key idea in the patent is to modify the Markov chain transition probabilities to give higher weight to transitions observed and change the teleportation

[8] Katz's model was $(\mathbf{I} - \alpha\mathbf{W}^T)\mathbf{k} = \alpha\mathbf{W}^T\mathbf{e}$ for an adjacency matrix $\mathbf{W}$.

[9] The authors employ graph theoretic techniques to examine the mass of PageRank in the largest strong component of the underlying graph. As $\alpha \to 1$, the mass in this strong component goes to zero if there are other strong components reachable from the largest strong component. This situation is undesirable because many important pages exist in the largest strong component. They argue that, consequently, α should be far from 1, and they suggest $\alpha = 1/2$ because $\alpha = 0$ gives an equally useless ranking.

[10] A user segment is a group of users related by a common factor. Age, sex, and interests are all possibilities.

vector in light of the start points of observed transitions. These weights depend on a user segment. They also recognize the inaccuracy of a single teleportation coefficient, but model separate teleportation coefficients to and from each page on the web. Our approach differs by modeling a random Markov chain and its associated random stationary distribution. The ideas in the patent often require smoothed estimates of probabilities from observed data. Using extensions of our ideas, we could replace some of these quantities with stochastic parameters and then apply our algorithms to generate truly random instances of these user-modified Markov chains.

4.3.3 *Path damping*

While working on the mathematics of RAPr, we discovered a strong relationship with path damping interpretations of the PageRank vector. Path damping models weight each path of length ℓ in the graph with a set of coefficients that sum to 1. Mathematically, they compute a ranking vector

$$\mathbf{r} = \sum_{\ell=0}^{\infty} \omega(\ell)\mathbf{P}^{\ell}\mathbf{v}, \tag{4.6}$$

where $\sum_{\ell=0}^{\infty} \omega(\ell) = 1$ [Boldi, 2005; Baeza-Yates et al., 2006]. As we show in section 4.4.3, the value $\mathrm{E}\left[\mathbf{x}(A)\right]$ corresponds to a particular choice of $\omega(\ell)$.

4.3.4 *Personalized PageRank*

A personalized PageRank vector is a PageRank vector targeted at a single person, or group of people [Page et al., 1999; Haveliwala, 2002; Jeh and Widom, 2003]. Consequently, the choice of α and $\mathbf{v}$ are more obvious in this case. Given these personalized PageRank vectors, a natural extension of our idea is to aggregate personalized PageRank vectors. One interpretation of RAPr is that it computes an aggregate *personalized* PageRank vector for all surfers. RAPr, however, currently constrains each personalized PageRank vector to use the same teleportation vector, $\mathbf{v}$.

4.3.5 *Spam ranking*

Zhang et al. [2004] investigates using the PageRank at different values of α to infer spam pages. Spam pages, they argue, ought to be sensitive to α. Their goal is to trap the surfer and boost their rank. Thus, changing α will reveal them. After computing PageRank at a few α's, they measure the correlation between the function $1/(1-\alpha)$ and the PageRank value on their small set of αs. This idea is related to the Gauss quadrature algorithm of section 4.6.4. In RAPr, the random variable A has an associated quadrature rule for its expectation that specifies the αs at which to compute the function. RAPr is also more general. It is not tied to just computing a spam correlation but produces a correlation between any group of pages as we illustrated in section 4.2.

4.4 THE RANDOM ALPHA PAGERANK MODEL

So far, we have discussed our vision for RAPr and the body of literature that surrounds our ideas. Now, RAPr is formally stated and analyzed.

Given a random variable A with finite moments distributed within the interval $[0, 1]$, the random alpha PageRank is the vector $\mathbf{x}(A)$ that satisfies

$$(\mathbf{I} - A\mathbf{P})\mathbf{x}(A) = (1 - A)\mathbf{v} \tag{4.7}$$

where $\mathbf{I}$, $\mathbf{P}$, and $\mathbf{v}$ are as in (2.5).[11] When we use this model, we often look at

$$\mathrm{E}\left[\mathbf{x}(A)\right] \qquad \text{and} \qquad \mathrm{Std}\left[\mathbf{x}(A)\right].$$

We address some theoretical implications of this model in the next few sections, and we defer the discussion of computation until section 4.6

Remark 8. *From this definition, we can immediately show that our model generalizes the TotalRank algorithm [Boldi, 2005], which produces a vector* $\mathbf{t}$ *defined as*

$$\mathbf{t} = \int_0^1 \mathbf{x}(\alpha)\, d\alpha.$$

If $A \sim U[0, 1]$ *in RAPr, then*

$$\mathrm{E}\left[\mathbf{x}(A)\right] = \int_0^1 \mathbf{x}(\alpha)\, d\alpha = \mathbf{t}.$$

A purported benefit of the TotalRank algorithm is that it eliminated picking an α in a PageRank computation. When compared to RAPr, however, it corresponds to a particular choice of the random variable A.

EXISTENCE It behooves us to check that the expectation of RAPr is well defined. The concern is that $\mathrm{E}\left[\mathbf{x}(A)\right] = \int_0^1 \mathbf{x}(\alpha)\rho(\alpha)\, d\alpha$ touches the value $\mathbf{x}(1) = 1$. Looking only at the linear system $(\mathbf{I} - \alpha\mathbf{P})\mathbf{x} = (1 - \alpha)\mathbf{v}$, we could conclude that $\mathbf{x}(1)$ does not exist because the matrix is singular when $\alpha = 1$. If $\mathbf{x}(1)$ is not defined, then the expectation of RAPr will not exist. However,

[11] Recall, $\mathbf{I}$ is the identity matrix, $\mathbf{P}$ is a column stochastic matrix ($\mathbf{e}^T\mathbf{P} = \mathbf{e}^T$), $\mathbf{v}$ is a non-negative probability vector ($\mathbf{e}^T\mathbf{v} = 1$).

readers who thought this must have skipped a section in chapter 2. There is no difficulty for our formulation of PageRank because $\alpha = 1$ corresponds with a removable singularity of the function.[12] Thus, we can extend the definition of $\mathbf{x}$ to $\alpha = 1$ with the limiting value. In contrast, the quantity $\int_0^1 \mathbf{y}(\alpha)\rho(\alpha)\, d\alpha$ does not exist for the PseudoRank vector $\mathbf{y}(\alpha)$.[13] This existence result is yet another reason that we prefer the PageRank definition to the PseudoRank definition.

4.4.1 *Choice of distribution*

The first order of business for RAPr is to choose the distribution of A. While choosing a distribution seems more difficult than picking a single value α, the right data makes it easy. The information for the empirical distribution of A is present in the logs from the surfer behavior studies discussed in section 4.3.2. This point is illustrated in section 4.5 where we take browsing logs and compute a distribution for α.

Picking A based on browsing behavior, however, is yet another choice. It seems correct and natural for the random surfer derivation of PageRank. When the PageRank or RAPr values are used in an application, the metrics of the application should drive the choice of α or A. We return to this point in section 4.8.4.

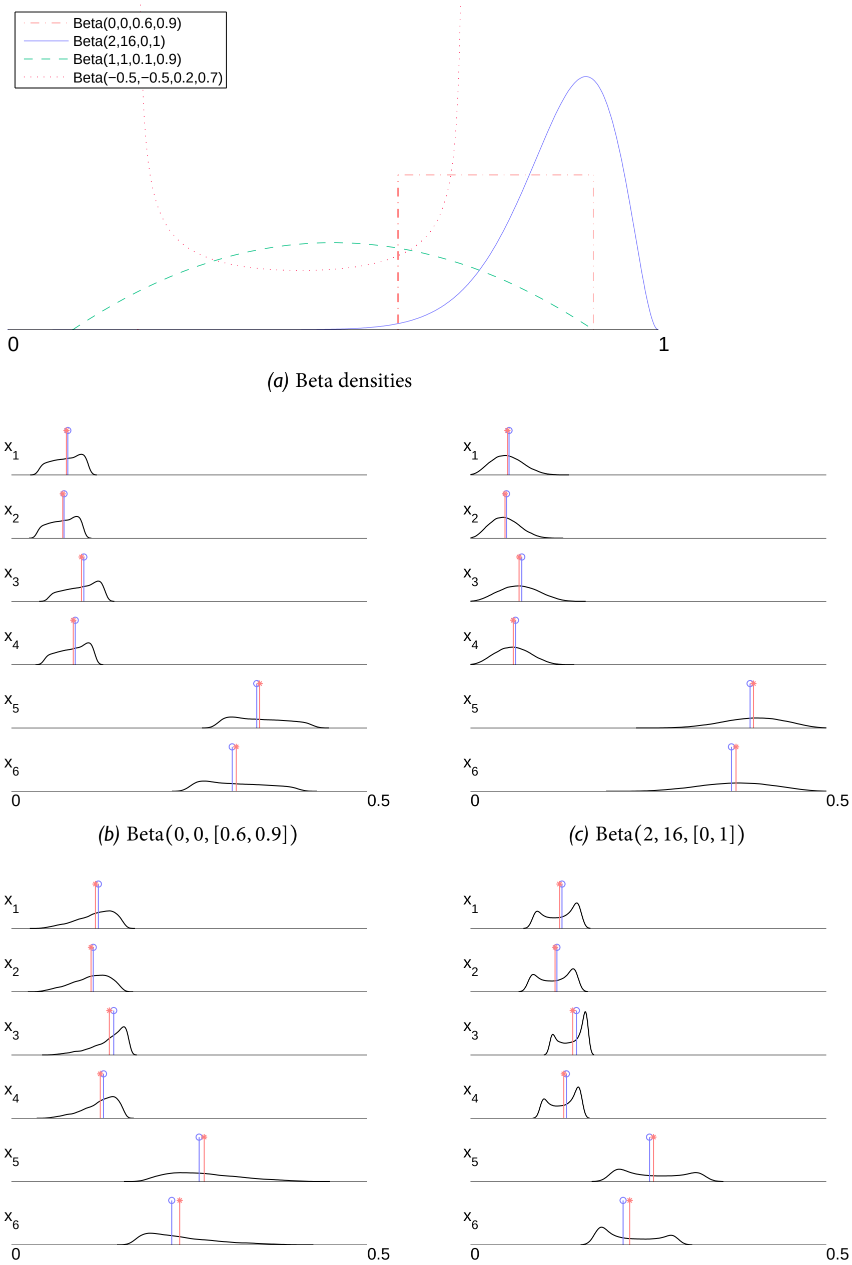
We assume that A has a continuous distribution over $[l, r]$ with $0 \le l < r \le 1$. Two distributions with bounded, continuous support are the uniform distribution and the Beta distribution. In fact, the uniform distribution is a special case of the Beta distribution and consequently, our "default" choice of A is a Beta random variable with distribution parameters a and b, and support $[l, r]$. To denote this, we write $A \sim \text{Beta}(a, b, [l, r])$. The probability density function for this random variable is

$$\rho(x) = \frac{1}{(r-l)^{a+b+1}} \frac{(x-l)^b (r-x)^a}{\text{Beta}(a+1, b+1)}. \tag{4.8}$$

It reduces to a uniform distribution when $a = b = 0$. Later, we will derive our algorithms in the most general settings possible, but all computations are done with some version of the Beta distribution. Section 4.5 presents an empirical distribution strikingly close to a Beta.

[12] Recall section 2.7. We showed that $\mathbf{x}(1)$ uniquely exists and equals $\mathbf{x}(1) = \mathbf{XYv}$ for $\mathbf{X}$ and $\mathbf{Y}$ based on the Jordan canonical form of $\mathbf{P}$.

[13] If the right-hand side vector in PseudoRank is $(1-\alpha)\mathbf{v}$, then $\mathbf{y}(1)$ is defined.



(a) Beta densities

(b) Beta$(0, 0, [0.6, 0.9])$

(c) Beta$(2, 16, [0, 1])$

(d) Beta$(1, 1, [0.1, 0.9])$

(e) Beta$(-0.5, -0.5, [0.2, 0.7])$

Figure 4.4 – Beta distributions and Random Alpha PageRank vectors. The four-parameter Beta distribution is quite flexible and exhibits a range of behaviors as a function of α, β, l, and r. The four density plots correspond to the graph from figure 4.2 with A drawn from the Beta distribution in the caption. When $\alpha, \beta < 0$, the resulting PageRank density functions are bimodal. (PageRank densities are computed with a kernel density estimator applied to 10,000 random samples.)

4.4.2 *Theoretical properties*

RAPr generalizes PageRank and all of the following theory reduces to known results about PageRank when A is a constant. We begin to discuss the theoretical properties of RAPr by computing a more tractable expression for the expectation of our random PageRanks.

Theorem 9. *If $A \sim Beta(a, b, [l, r])$ with $0 \le l < r \le 1$, then*

$$\mathrm{E}[\mathbf{x}(A)] = \sum_{\ell=0}^{\infty} \mathrm{E}\left[A^{\ell} - A^{\ell+1}\right] \mathbf{P}^{\ell}\mathbf{v}. \tag{4.9}$$

For PageRank, the theorem is the Neumann expansion $\mathbf{x}(\alpha) = \sum_{n=0}^{\infty}(\alpha^n - \alpha^{n+1})\mathbf{P}^n\mathbf{v}$ discussed in section 2.2.5.

Proof. From (4.7) we have

$$\mathbf{x}(A) = (1 - A)(\mathbf{I} - A\mathbf{P})^{-1}\mathbf{v}. \tag{4.10}$$

Because the spectral radius $\rho(A\mathbf{P}) < 1$ for any value of A in $[0, 1)$, we can expand $(\mathbf{I} - A\mathbf{P})^{-1}$ with its Neumann series [Meyer, 2000, page 618]:

$$\mathbf{x}(A) = (1 - A)\sum_{n=0}^{\infty} A^n\mathbf{P}^n\mathbf{v}. \tag{4.11}$$

Taking the expectation and rearranging gives

$$\mathrm{E}[\mathbf{x}(A)] = \mathrm{E}\left[\sum_{n=0}^{\infty}(A^n - A^{n+1})\mathbf{P}^n\mathbf{v}\right]. \tag{4.12}$$

To interchange the expectation and sum, note that $0 \le A^n\mathbf{P}^n\mathbf{v} \le 1$ for all n. Because the summands are non-negative, $(1 - A)A^n \ge$, Fubini's theorem[14] justifies this interchange. □

The previous theorem also holds when A is a constant between 0 and 1. Using this theorem, we can formally justify the claim that the expectation of RAPr is different from the PageRank vector computed with $\alpha = \mathrm{E}[A]$. The following pedagogic example restricts the claim to the case when $A \sim U[0, 1]$. Such a restriction allows us to use the expressions for the moments of A and compute the infinite sums exactly. Note that $\sum_{n=0}^{\infty} \mathrm{E}\left[A^n - A^{n+1}\right] = 1$ because the sums telescope.

Example 10. *Set* $\mathbf{P} = \begin{bmatrix} 0 & 1/2 & 1/2 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$, $\mathbf{v} = \begin{bmatrix} 1/3 & 1/3 & 1/3 \end{bmatrix}^T$. *Then*

$$\mathbf{P}^0\mathbf{v} = \mathbf{v} \quad \mathbf{P}^1\mathbf{v} = \begin{bmatrix} 0 & 1/6 & 5/6 \end{bmatrix}^T \quad \mathbf{P}^n = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T, n \ge 2. \tag{4.13}$$

[14] In general, Fubini's theorem states when we can exchange the order of integration. In probability terms, it governs when we can move an expectation operator inside an integral or infinite sum. For the case of an infinite sum, $\mathrm{E}\left[\sum_{i=1}^{\infty}\right] X_i$, we need to ensure that X_i is non-negative or that $\sum_{i=1}^{\infty} \mathrm{E}[|X_i|]$ converges to move the expectation inside.

To apply theorem 9, we need $\mathrm{E}[A^n]$. *If* $A \sim \mathit{Beta}(0,0,[0,1])$, *then A is uniform over* $[0,1]$ *and* $\mathrm{E}[A^n] = \frac{1}{n+1}$. *Finally,*

$$\begin{aligned}\mathrm{E}[\mathbf{x}(A)] &= \frac{1}{2}\mathbf{v} + \frac{1}{2}\begin{bmatrix}0 & 1/6 & 5/6\end{bmatrix}^T + \sum_{n=2}^{\infty}\left(\frac{1}{n+1} - \frac{1}{n+2}\right)\begin{bmatrix}0 & 0 & 1\end{bmatrix}^T \\ &= \begin{bmatrix}1/6 & 7/36 & 23/36\end{bmatrix}^T.\end{aligned} \tag{4.14}$$

For $\mathbf{x}(\mathrm{E}[A]) = \mathbf{x}(1/2)$, *we find*

$$\begin{aligned}\mathbf{x}(\mathrm{E}[A]) &= \frac{1}{2}\mathbf{v} + \frac{1}{4}\begin{bmatrix}0 & 1/6 & 5/6\end{bmatrix}^T + \sum_{n=2}^{\infty}\left(\frac{1}{2^n} - \frac{1}{2^{n+1}}\right)\begin{bmatrix}0 & 0 & 1\end{bmatrix}^T \\ &= \begin{bmatrix}1/6 & 5/24 & 5/8\end{bmatrix}^T.\end{aligned} \tag{4.15}$$

Thus, for this example, $\mathrm{E}[\mathbf{x}(A)] \neq \mathbf{x}(\mathrm{E}[A])$.

For this case, the RAPr solution satisfies $\mathbf{e}^T\left[\begin{smallmatrix}1/6 & 7/36 & 23/36\end{smallmatrix}\right] = 1$. This property is general and we next show that the vector $\mathrm{E}[\mathbf{x}(A)]$ is always a probability distribution.

Corollary 11. *If* $A \sim \mathit{Beta}(a,b,[l,r])$ *with* $0 \le l < r \le 1$ *and probability density function* ρ, *then* $\mathrm{E}[\mathbf{x}_i(A)] > 0$ *and* $\|\mathrm{E}[\mathbf{x}(A)]\| = 1$.

PageRank is defined as a probability vector, so this property does not change for RAPr.

Proof. First, $\mathrm{E}[\mathbf{x}_i(A)] \ge 0$ is because $0 \le A \le 1$ and $v_i \ge 0$. Then, we have

$$\|\mathrm{E}[\mathbf{x}(A)]\| = \mathbf{e}^T\int_0^1 \mathbf{x}(\alpha)\rho(\alpha)\,d\alpha = \int_0^1 \mathbf{e}^T\mathbf{x}(\alpha)\rho(\alpha)\,d\alpha = 1, \tag{4.16}$$

because $\mathbf{e}^T\mathbf{x} = 1$ for each α and $\int_0^1 \rho(\alpha)\,d\alpha = 1$. □

Finally, we show that for a certain class of pages, the expectation of RAPr is *equal* to PageRank with $\alpha = \mathrm{E}[A]$.

Theorem 12. *Let* $A \sim \mathit{Beta}(a,b,[l,r])$ *with* $0 \le l < r \le 1$. *If i is the index for a node with no in-links, then* $\mathrm{E}[\mathbf{x}_i(A)] = \mathbf{x}_i(\mathrm{E}[A])$ *and* $\mathrm{Std}[\mathbf{x}_i(A)] = v_i\,\mathrm{Std}[A]$.

When a state i in $\mathbf{P}$ has no in-transitions (in-links) then $x_i(\alpha) = (1-\alpha)v_i$ as well.

Proof. For a page with no in-links, $\mathbf{e}_i^T\mathbf{P}^n = 0$, $n > 0$, where $\mathbf{e}_i$ is the vector with a 1 in the ith component. Taking the Neumann series for $\mathbf{x}(A)$ gives

$$\mathbf{x}_i(A) = \mathbf{e}_i^T\sum_{j=0}^{\infty}(A^j - A^{j+1})\mathbf{P}^j\mathbf{v} = \mathbf{e}_i^T(A^0 - A^1)\mathbf{v} = (1-A)v_i. \tag{4.17}$$

Equality of the statistics follows from the linearity of the expectation operator. □

While theorem 12 yields one condition when the expectation is the same for the random and deterministic models, the result may not be useful. Given many of the standard corrections for dangling nodes (including the methods used in this paper, see section 2.2.1),[15] a graph with any dangling nodes will induce an effective graph where all nodes have an in-link.

4.4.3 *A path damping and browse path view*

Although we derived the RAPr model by replacing the deterministic coefficient α with a random variable A, the resulting model has strong connections with other generalizations of PageRank based on *path damping coefficients* [Baeza-Yates et al., 2006]. From the Neumann series for $\mathrm{E}\left[\mathbf{x}(A)\right]$, (4.9), the coefficient on the jth power of $\mathbf{P}$ is the weight placed on a path of length j in the Markov chain. Because these coefficients tend to decrease as j increases, they "damp" longer paths in the Markov chain. Figure 4.5 shows the path damping coefficients for the distributions from figure 4.4.

As shown in figure 4.5, the path damping view of RAPr provides interesting information about the impact of different distributions. For details on the algorithmic implications of the path damping view, see section 4.6.3.

In the full generalization of the path damping model [Baeza-Yates et al., 2006], we are free to choose the path damping coefficients to be any non-negative sequence with unit sum. One study suggests taking the coefficient on the path of length ℓ to be the empirical probability that surfers follow a path of length ℓ, or alternatively, an approximation from an inverse Gaussian distribution [Huberman et al., 1998]. Let's introduce a slightly different model based on this idea and demonstrate its relationship to RAPr.

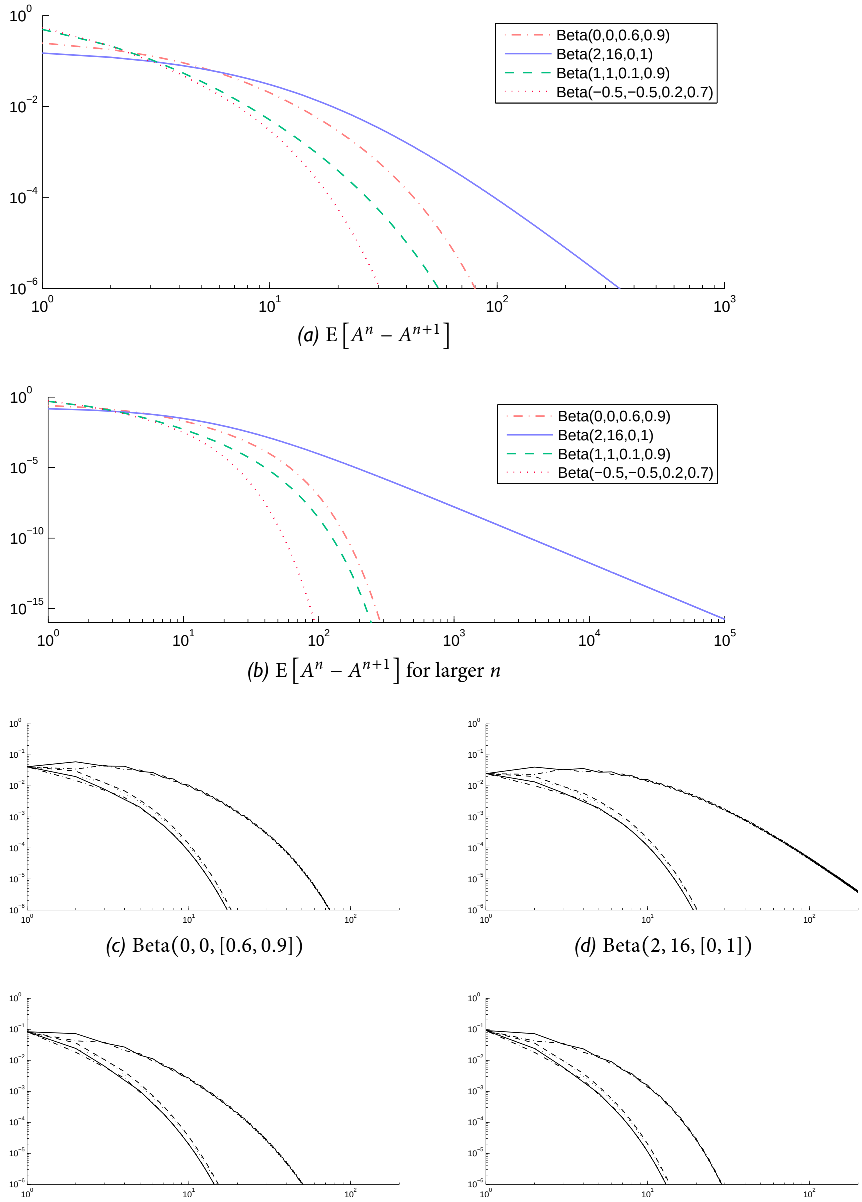
Let L be a non-negative integer random variable representing the length of a browsing path on the web. This L is a discrete random variable, not a continuous random variable like A. Using L in this model requires the probability operator, $\mathbb{P}\left[\cdot\right]$, because the expectation of a discrete random variable is only defined over the set of discrete values. For example,

$$\mathrm{E}\left[L\right] = \sum_{\ell=0}^{\infty} \ell \, \mathbb{P}\left[L = l\right]$$

is the expected, or average, length of a browsing path. In this model, a random surfer follows exactly L links on the web before stopping. Under L, surfers stop at $\mathbf{P}^L$ and can use $\mathrm{E}\left[\mathbf{P}^L\mathbf{v}\right]$ as another ranking vector,

$$\mathrm{E}\left[\mathbf{P}^L\mathbf{v}\right] = \sum_{\ell=0}^{\infty} \mathbb{P}\left[L = \ell\right] \mathbf{P}^\ell \mathbf{v}. \tag{4.18}$$

[15] In the most common case, $\mathbf{P} = \bar{\mathbf{P}} + (1/n)\mathbf{e}\mathbf{d}^T$ and every node has an in-link.



(a) $\mathrm{E}\left[A^n - A^{n+1}\right]$

(b) $\mathrm{E}\left[A^n - A^{n+1}\right]$ for larger n

(c) $\mathrm{Beta}(0, 0, [0.6, 0.9])$

(d) $\mathrm{Beta}(2, 16, [0, 1])$

(e) $\mathrm{Beta}(1, 1, [0.1, 0.9])$

(f) $\mathrm{Beta}(-0.5, -0.5, [0.2, 0.7])$

Figure 4.5 – Path damping coefficients for the Random Alpha PageRank model. The first two subfigures show the path damping coefficients for the distributions from figure 4.4 drawn with the same legend. The following figures have six lines showing $\mathrm{E}\left[A^n - A^{n+1}\right] \mathbf{P}^n\mathbf{v}$ for each of the six pages in the graph from figure 4.2. The two lines at the right of each of these plots correspond to pages 5 and 6. Using these plots, we see the covariance structure identified in section 4.2 in a different way. However, there is no hint of the bimodality for the $\mathrm{Beta}(-0.5, -0.5, [0.2, 0.7])$ distribution.

This equation gives the direct relationship between a path damping equation and RAPr. If we can construct A such that $\mathbb{P}[L = \ell] = \mathrm{E}\left[A^{\ell}\right] - \mathrm{E}\left[A^{\ell+1}\right]$, then we could establish a direct relationship between the models. Thus, we need to match moments

$$\begin{aligned} \mathbb{P}[L = 0] &= \mathrm{E}\left[A^0\right] - \mathrm{E}\left[A^1\right] \\ \mathbb{P}[L = 1] &= \mathrm{E}\left[A^1\right] - \mathrm{E}\left[A^2\right] \\ \mathbb{P}[L = 2] &= \mathrm{E}\left[A^2\right] - \mathrm{E}\left[A^3\right] \\ &\vdots \end{aligned} \implies \begin{aligned} \mathrm{E}\left[A^0\right] &= 1 \\ \mathrm{E}\left[A^1\right] &= \mathrm{E}\left[A^0\right] - \mathbb{P}[L = 0] \\ \mathrm{E}\left[A^2\right] &= \mathrm{E}\left[A^1\right] - \mathbb{P}[L = 1] \\ &\vdots \end{aligned} \tag{4.19}$$

Computing such a distribution of A is a special case of the Hausdorff moment problem [Talenti, 1987], which has a known characterization for a unique solution. We go no further than noting this equivalence.

4.5 EMPIRICAL DISTRIBUTION

As we argued in the previous sections, the RAPr model generalizes PageRank to multiple random surfers. Instead of picking a value of α to control when the random surfer teleports, RAPr forces us to pick a *distribution* for a random variable A that controls how likely surfers are to pick a value of α. This second task seems more problematic. For a natural choice of A, it is not. This natural choice is to pick A according to how surfers actually behave on the web.

Recall that a single value of α is the probability a user clicks a link on a web page. With a custom browser plug-in, you could compute your own value of α. It's a simple ratio

$$\alpha \approx \frac{\text{number of pages viewed after clicking a link}}{\text{total number of pages viewed}}.$$

With more browsing, and more information, the approximation to α grows more refined. If people tracked their own α, finding the empirical distribution for A would be just a matter of data collection.

Loosely speaking, browser toolbars collect precisely this type of information. That is, the Microsoft, Yahoo!, and Google browser toolbars—which users download and install into their browsers for a few improvements—collect this data and send it back to Microsoft, Yahoo!, and Google. (Of course, each company ensures that users provide explicit consent for transmitting the data.) Toolbar logs, then, have the information to compute A.

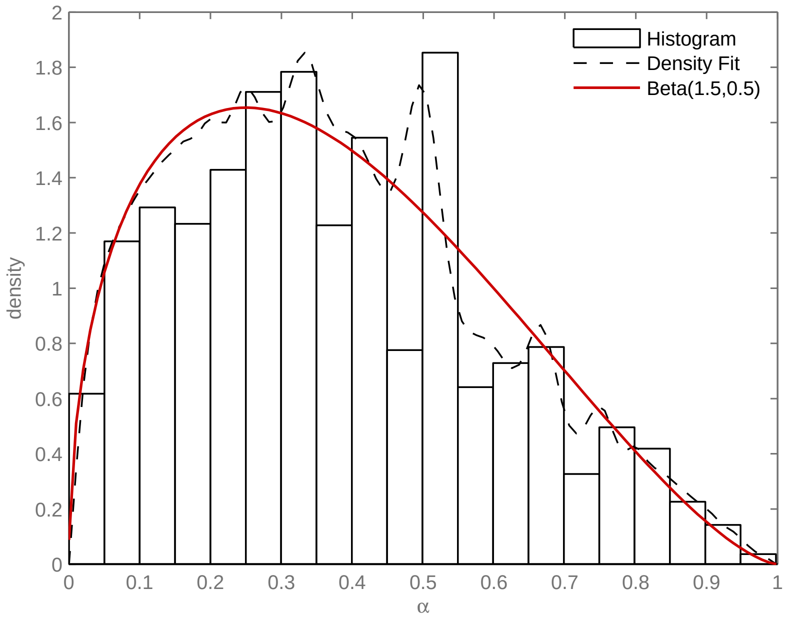


Figure 4.6 – Empirically measured teleportation coefficients. For two hours of toolbar logs, the black histogram represents the raw values of α recorded. The dashed line is a bounded kernel density fit to the data with a kernel width of 0.1. The blue line is a Beta density approximation to α. In the original data, rational values of α with small denominators are likely—and cause spikes in the histogram—because many individuals are only briefly observed.

Following our presentation on the initial RAPr model at the Workshop on Algorithms for the Web Graph, Abraham Flaxman and Asela Gunawardana provided a summary of these logs. They reported values of α from one million "users" on the web collected in a two hour window. From this data, the mean value of $\alpha = 0.375$. The data shows a good fit to a $\text{Beta}(1.5, 0.5, [0, 1])$ distribution (figure 4.6).

For the figure, the analysis used a kernel density estimator [Asmussen and Glynn, 2007] to generate an approximate probability distribution from the raw data. The density fit itself looks quite similar to a Beta distribution. A nonlinear least squares fit produces a $\text{Beta}(1.52, 0.53, [0, 1])$ distribution. Instead, a $\text{Beta}(1.5, 0.5, [0, 1])$ is more simple and matches the mean of the data.

For the estimate displayed in the figure, we dropped all values of α measured at 0 and 1. Both of these values are impossible and represent problems with the sampling procedure. In an ideal case, the data would be collected with pseudo-counts [Agresti, 2002], where we estimate

$$\alpha \approx \frac{\text{number of pages viewed after clicking a link} + 1}{\text{total number of pages viewed} + 2}.$$

Pseudo-counts correct for the two known, but unobserved, future actions. That is, a person will always click another link, so we add 1 to both totals.

Also, a person will always visit a page without clicking a link, and so we add another page to the total pages viewed. This adjustment fixes an important problem when the only observation of a person are page views with no clicks. The α is 0, which is incorrect. By assumption, everyone will click on a link at some point and the pseudo-counts adjust for these finite size results.

Two aspects of the Beta fit are surprising. First, web surfers do not often click links! The "mean" user clicks a link once for every three pages. Such behavior may suggest that *search* and *bookmarks* are the prevalent means of navigating the web. Second, the distribution and its fits extend to $\alpha = 1$. In all the nonlinear least squares variations, the right endpoint of each fit was 1. Thus, long browsing sessions are common on the web.

In closing, we repeat that A can be measured from data, and

$$A \sim \text{Beta}(1.5, 0.5, [0, 1])$$

is a reasonable approximation. While this analysis is preliminary, it supports a few surprising observations about random surfer browsing on the web.

4.6 ALGORITHMS

In this section we describe and compare three methods for computing the approximate statistics, $\text{E}[\mathbf{x}(A)]$ and $\text{Std}[\mathbf{x}(A)]$, of the RAPr model.

4.6.1 *PageRank*

One key component of these algorithms is a robust solver for a deterministic PageRank problem with $\alpha < 1$. For this task, we use two solvers: a direct method and an inner-outer method (chapter 5). The direct method uses the "backslash" solve in MATLAB. In versions R2007a and R2007b, this command calls the UMFPACK 5.0 library [Davis, 2004]. For a row sub-stochastic matrix,[16] we solve

$$(\mathbf{I} - \alpha \mathbf{P}^T)\mathbf{y} = \mathbf{v}, \qquad \mathbf{x}(\alpha) = \mathbf{y} / \|\mathbf{y}\| . \tag{4.20}$$

The inner-outer iteration requires only sub-stochastic matrix-vector products, which makes it a natural choice for data structure-free algorithms.[17] Program 8 is our implementation of the inner-outer method. Using the same code, the `inoutpr` function works with a native Matlab sparse matrix structure as well as a MATLAB wrapper around a BVGraph data structure [Boldi

[16] For computational efficiency, all the Matlab programming uses row sub-stochastic matrices; see section 2.4.2 for more information.

[17] Although using Gauss-Seidel iterations or a strong-component decomposition algorithm is typically faster, these algorithms require access to the graph as a structure and manipulate it.

and Vigna, 2004] through a custom version of *bvgraph* or *libbvg* software (section 6.5).

4.6.2 *Monte Carlo*

An enticingly straightforward method to compute the expectations, standard deviations, and density functions of RAPr is to use a Monte Carlo method. To wit, first generate N realizations of A from a chosen distribution, and then solve each resulting PageRank problem. With the N different realizations of $\mathbf{x}(\alpha_i)$, $i = 1, \ldots, N$, we can compute unbiased estimates for $\mathrm{E}\left[\mathbf{x}(A)\right]$ and $\mathrm{Std}\left[\mathbf{x}(A)\right]$ with the formulas

$$\begin{aligned} \mathrm{E}\left[\mathbf{x}(A)\right] &\approx \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}(\alpha_i) \equiv \hat{\mu}_{\mathbf{x}}, \\ \mathrm{Std}\left[\mathbf{x}(A)\right] &\approx \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(\mathbf{x}(A_i) - \hat{\mu}_{\mathbf{x}})^2} \end{aligned}$$

from Asmussen and Glynn [2007].

Unfortunately, as with any Monte Carlo method, these estimates converge as $1/\sqrt{N}$ [Asmussen and Glynn, 2007], which makes this approach prohibitively expensive for large graphs such as the web graph.

The real advantage of the Monte Carlo method is its beautiful simplicity. The following short code is our entire implementation of the Monte Carlo method, including a numerically stable method to update the running variance computation [Chan et al., 1983].

Program 5 – Computing RAPr with Monte Carlo. A Monte Carlo code in MATLAB to estimate the expectation and standard deviation of the RAPr model.

```
function [ex,stdx] = mcrapr(P,N,ba,bb,bl,br)
tol=1e-9; maxterms=500; n=size(P,1); v=1/n;
alphas = betarnd(bb+1,ba+1,N,1)*(br-bl) + bl;
ex=zeros(n,1); stdx=zeros(n,1);
for i=1:N
    % solve the PageRank system
    x = inoutpr(P,alphas(i),v,tol,2*ceil(log(tol)/log(alphas(i))));
    % update the running solution sum and variance sum formulas
    ex = ex+x; if i>1, stdx = stdx + (1./(i*(i-1))).*(i*x-ex).^2; end
end
ex = ex./N; stdx=sqrt(stdx./(N-1)); % compute the mean and std
```

4.6.3 *Path damping*

As discussed in sections 4.3.3 and 4.4.3, *path damping* algorithms for PageRank are not novel. RAPr simply provides a large set of functions that generate the path damping coefficients. In this section, we will discuss using these ideas to compute $\mathrm{E}[\mathbf{x}(A)]$ and $\mathrm{Std}[\mathbf{x}(A)]$.

Recall the Neumann series from theorem 9,

$$\mathrm{E}[\mathbf{x}(A)] = \sum_{\ell=0}^{\infty} \mathrm{E}\left[A^{\ell} - A^{\ell+1}\right] \mathbf{P}^{\ell}\mathbf{v}. \tag{4.21}$$

If we truncate this series to a finite value N, then an algorithm for $\mathrm{E}[\mathbf{x}(A)]$ immediately follows:

$$\mathrm{E}[\mathbf{x}(A)] \approx \mathbf{x}^{(N)} = \sum_{\ell=0}^{N} \mathrm{E}\left[A^{\ell} - A^{\ell+1}\right] \mathbf{P}^{\ell}\mathbf{v} + (1 - \sum_{\ell=0}^{N} \mathrm{E}\left[A^{\ell} - A^{\ell+1}\right])\mathbf{P}^{N+1}. \tag{4.22}$$

The final term in this summation ensures that $\mathbf{e}^T\mathbf{x}^{(N)} = 1$ for the path damping approximation.

To compute $\mathrm{Std}[\mathbf{x}(A)]$ using the path damping equations we compute $\mathrm{E}[\mathbf{x}(A) \bullet \mathbf{x}(A)]$ and then compute

$$\mathrm{Std}[\mathbf{x}(A)] = \sqrt{\mathrm{E}[\mathbf{x}(A) \bullet \mathbf{x}(A)] - (\mathrm{E}[\mathbf{x}(A)] \bullet \mathrm{E}[\mathbf{x}(A)])}.$$

Based on the Neumann expansion,

$$\mathrm{E}[\mathbf{x}(A) \bullet \mathbf{x}(A)] = \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} \mathrm{E}\left[A^{i+j} - 2A^{i+j+1} + A^{i+j+2}\right] (\mathbf{P}^{i}\mathbf{v}) \bullet (\mathbf{P}^{j}\mathbf{v}). \tag{4.23}$$

And again, we truncate this series to a common term in both i and j:

$$\mathrm{E}\left[\mathbf{x}(A)^2\right] \approx \mathbf{s}^{(N)} = \sum_{i,j}^{N} \mathrm{E}\left[A^{i+j} - 2A^{i+j+1} + A^{i+j+2}\right] (\mathbf{P}^{i}\mathbf{v}) \bullet (\mathbf{P}^{j}\mathbf{v}). \tag{4.24}$$

Note that we do not apply any correction to the sum to ensure a summation property of the solution as in the case for $\mathrm{E}[\mathbf{x}(A)]$.

Given the moments of the distribution A,

$$\mu_k(A) = \mathrm{E}\left[A^k\right], \quad 0 \le k \le 2N + 2, \tag{4.25}$$

the previous summation expressions become algorithms. As discussed in section 4.4.1 we only consider $A \sim \mathrm{Beta}(a, b, [l, r])$. For $A \sim \mathrm{Beta}(a, b, [0, 1])$, the values μ_k are known analytically [Zwillinger et al., 1996]:

$$\mu_0 = 1, \quad \mu_k = \frac{b+k}{a+b+k+1}\mu_{k-1} = \prod_{j=1}^{k} \frac{b+j}{a+b+j+1}, \quad k \geq 1. \tag{4.26}$$

To handle the general case, define

$$\hat{\mu}_j \equiv \mu_j(A) \text{ where } A \sim \mathrm{Beta}(a, b, [0, 1]). \tag{4.27}$$

For $A \sim \mathrm{Beta}(a, b, [l, r])$,

$$\begin{aligned} \mathrm{E}\left[A^k\right] &= \int_l^r \zeta^k \rho^{(l,r)}_{\mathrm{Beta}(a,b)}(\zeta)\, d\zeta = \int_0^1 ((r-l)\tau + l)^k \rho^{(0,1)}_{\mathrm{Beta}(a,b)}(\tau)\, d\tau \\ &= \sum_{j=0}^{k} \binom{k}{j} \hat{\mu}_j (r-l)^j l^{k-j} \end{aligned} \tag{4.28}$$

and we can compute the moments of $A \sim \mathrm{Beta}(a, b, [l, r])$ by scaling and shifting those of $A \sim \mathrm{Beta}(a, b, [0, 1])$. Program 6 gives a simple implementation of the path damping algorithms and an implementation of the recursion

$$\begin{aligned} \mu_k(A) &= \mu^{(0,k)} \\ \mu^{(i,j)} &= \sum_{m=i}^{j} \binom{j-i}{m-i} \hat{\mu}_m (r-l)^{m-i} l^{j-m} = (r-l)\mu^{(i,j-1)} + l\mu^{(i+1,j)} \end{aligned} \tag{4.29}$$

to compute the moments $\mu_k(A)$.[18]

4.6.4 *Gaussian quadrature*

Integration and interpolation are the fundamental concepts behind the class of uncertainty quantification techniques known as stochastic collocation [Xiu and Hesthaven, 2005], which were originally developed in the context of partial differential equation models with stochastic inputs. Much of the work in this context has focused on the problem of high-dimensional parameter spaces [Nobile et al., 2008], where multi-dimensional interpolation and integration can have a computational cost that increases exponentially with the dimension of the parameter space; one aspect of the so-called curse of dimensionality.

[18] The implementation is not straightforward, though it is correct. It follows from organizing the moments into a matrix

$$\begin{bmatrix} \mu^{(0,0)} & \mu^{(0,1)} & \dots & \mu^{(0,k)} \\ & \mu^{(1,1)} & \dots & \mu^{(1,k)} \\ & & \ddots & \vdots \\ & & & \mu^{(k,k)} \end{bmatrix}$$

and filling in the entries $\mu^{(0,1)}, \dots, \mu^{(0,k)}$ from the initially specified diagonal. At every step in the implementation, we compute a new diagonal.

Program 6 – Computing RAPr with path-damping. The first subfigure presents a compact algorithm to compute the moments of a Beta distribution. Next, we present our implementation of the path damping algorithms using the moments. Just as for the PageRank case, we perform our computations with `P` = $\mathbf{P}^T$ for efficiency.

(a) Moment computation

```
function m=beta_moments(N,a,b,l,r)
c = l; s = (r-l); m=zeros(N+1,1);            % c is the offset, s is the scale
uk=1; k=0; sk=1; m(1) = uk;                  % uk are the Beta(a,b,0,1) moments
for i=1:N, k = k+1; uk=s*uk*((b+k)/(a+b+k+1)); m(i+1) = uk; end
% form the shifted and scaled moments        % m are the Beta(a,b,l,r) moments
if c ≠ 0, for i=1:N, m(i+1:end) = c*m(i:(end-1)) + m((i+1):end); end, end
```

(b) Path damping computation

```
function [ex,stdx] = pdrapr(P,N,a,b,l,r)
tol=1e-9; maxterms=500; n=size(P,1); v=1/n;
ms = beta_moments(2*(maxterms+1),a,b,l,r);              % setup the moments
i=0; delta=2; ex=zeros(n,1); y = zeros(n,1) + v; s=0;   % setup vectors
while i<maxterms && ms(i+2)>tol
    Ptiv=y; ex = ex + (ms(i+1)-ms(i+2))*Ptiv;
    y = P’*(Ptiv); y = y + (1-norm(y,1)).*v; i=i+1; end % update P^i v
ex = ex + ms(i+2)*y;                                    % adjust with the last term
% compute stdx with same number of terms of sequence
nterms=i; ex2=zeros(n,1); Ptiv = zeros(n,1); Ptiv=Ptiv+v; Ptjv=Ptiv;
for i=0:nterms, for j=0:nterms
    ex2 = ex2 + (ms(i+j+1)-2*ms(i+j+2)+ms(i+j+3))*(Ptiv.*Ptjv);
    y = P’*(Ptjv); Ptjv = y + (1-norm(y,1)).*v;
  end                                                   % now update Ptiv and reset Ptjv
  y = P’*(Ptiv); Ptiv = y + (1-norm(y,1)).*v; Ptjv(:)=0; Ptjv=Ptjv+v;
end                                                     % finish by update ex2 to be stdx
stdx = sqrt(ex2-ex.^2);
```

Program 7 – Matlab code for computing RAPr using Gaussian quadrature. Using Gautschi’s OPQ codes, `r_jacobi01.m` and `gauss.m`, a Matlab quadrature implementation is quite easy.

```
function [ex,stdx] = gqrapr(P,N,a,b,l,r)
% first run these commands to get the OPQ codes
% urlwrite(’http://www.cs.purdue.edu/archives/2002/wxg/codes/gauss.m’,’gauss.m’)
% urlwrite(’http://www.cs.purdue.edu/archives/2002/wxg/codes/r_jacobi.m’,’r_jacobi.m’)
% urlwrite(’http://www.cs.purdue.edu/archives/2002/wxg/codes/r_jacobi01.m’,’r_jacobi01.m’)
tol=1e-9; maxit=1000; n=size(P,1); v=1/n;
ab=r_jacobi01(N,a,b); xw=gauss(N,ab); xw(:,2) = (1./beta(b+1,a+1))*xw(:,2);
xw(:,1) = (r-l).*xw(:,1)+l;         % generate the quadrature rule by scale and shift
ex = zeros(n,1); stdx = zeros(n,1); % initialize running sums
for i=1:N
    % solve the PageRank system
    x = inoutpr(P,xw(i,1),v,min(tol./xw(i,2),1e-2), ... % adjust tol and maxit
      2*ceil(log(min(tol./xw(i,2),1e-2))/log(xw(i,1))));% for mult by xw(i,2)
    ex = ex+xw(i,2).*x; stdx = stdx+xw(i,2).*(x.^2);
end
stdx = sqrt(stdx - ex.^2);          % convert to stdx
```

RAPr has only one random parameter $A \sim \text{Beta}(a, b, [l, r])$, so we can employ the one-dimensional interpolation and integration formulas to produce highly accurate statistics. In this section we discuss their application to RAPr.

For a modern reference on Gaussian quadrature, see Gautschi [2002]. In an N-point quadrature rule, we approximate

$$\int_l^r f(x)\, dw(x) \approx \sum_{i=1}^N f(z_i^G) w_i^G, \tag{4.30}$$

where z_i^G are the N nodes or points of a quadrature rule and w_i^G are the corresponding weights on those points. These nodes and weights are chosen to make the integration exact if f is a polynomial of degree less than $2N$, and there are efficient algorithms to compute these rules [Golub and Welsch, 1969]. Note that the quadrature rule changes if the integration endpoints change, or if the weight function w changes.

With the points and weights of the Gauss quadrature formula, we first solve N deterministic PageRank problems

$$(\mathbf{I} - z_i^G \mathbf{P})\mathbf{x}_i = (1 - z_i^G)\mathbf{v} \tag{4.31}$$

using methods described in section 4.6.1. Then we can compute statistics of RAPr with the quadrature formulas

$$\mathrm{E}\left[\mathbf{x}(A)\right] \approx \sum_{i=1}^N \mathbf{x}_i w_i^G, \quad \mathrm{Cov}[\mathbf{x}(A)] \approx \sum_{i=1}^N \mathbf{x}_i \mathbf{x}_i^T w_i^G - \left(\sum_{i=1}^N \mathbf{x}_i w_i^G\right)\left(\sum_{i=1}^N \mathbf{x}_i w_i^G\right)^T. \tag{4.32}$$

For the quadrature rule (4.30), the nodes z_i^G are known to lie on the interior of the integration region, $l < z_i^G < r$. Furthermore, the weights w_i^G are strictly positive. The first property is essential to using quadrature with PageRank when $r = 1$. It states that we do not have to compute a PageRank vector at $\alpha = 1$. Many other quadrature rules, such as Clenshaw-Curtis,[19] Gauss-Radau, or Gauss-Lobatto, all utilize a function value at one or both of the endpoints. For PageRank, computing the limit vector $\mathbf{x}(1)$ efficiently is still an open problem, and hence these alternatives are not appropriate.

As program 7 shows, implementing the Gaussian quadrature algorithm is easy using the OPQ routines [Gautschi, 2002]. In the code, we adjust the solution tolerance of the linear system based on the weights of the final quadrature summation. We call this a weighted tolerance τ.

[19] Fejér quadrature is a variant of Clenshaw-Curtis quadrature without the endpoints. It is less accurate than Gauss quadrature, but has nested point sets, which make it an attractive option in other settings.

4.7 algorithm analysis

In this section, we compare and analyze the algorithms using theoretical and numerical techniques. As with many such comparisons, the theoretical analysis does not treat every case, and the numerical comparisons are always limited to the chosen experiments. Nevertheless, the combined examination yields strong suggestions for the choice of algorithm and implementation when applied to RAPr. For a compact summary of the properties of the methods, see table 4.2. The twin objectives of the analysis are work and accuracy. Both are typically proportional to N, the number of terms used in the approximate statistics. A description of N for the methods is in the table.

Table 4.2 – Summary of convergence results for the RAPr model. A brief summary of our results about each method. A non-intrusive method only uses an existing PageRank solver. The Monte Carlo and Path Damping algorithms can be updated from N to $N+1$ with no more work than another iteration, whereas the Gaussian Quadrature routines produce different instances when N is incremented. For storage, `prsolve` is the storage required to solve a PageRank problem, and `P multiply` is the storage required for the matrix $\mathbf{P}$. The convergence results show how the norm of the error decays as a function of N, the intrinsic parameter of the method, and both a and r, the parameters from the Beta distribution.

(a) Algorithm Properties

Method	Non-Intrusive	Update	Storage
Monte Carlo	+	+	`prsolve` + 2 n-vectors
Path Damping	-	+	`P multiply` + 5 n-vectors
Gaussian Quadrature	+	-	`prsolve` + 2 n-vectors

(b) Convergence Analysis

Method	Conv.	Work Required	What is N?
Monte Carlo	$\frac{1}{\sqrt{N}}$	N PageRank systems	number of samples from A
Path Damping (without Std $[\mathbf{x}(A)]$)	$\frac{r^{N+2}}{N^{1+a}}$	$N+1$ matrix-vector products	terms of Neumann series
Gaussian Quadrature	r^{2N}	N PageRank systems	number of quadrature points

4.7.1 *Monte Carlo*

Monte Carlo is not a deterministic technique. In it, we have to solve N PageRank systems at random values of α. The question we address here regards the work that this procedure requires. Each PageRank problem is solved iteratively and matrix-vector multiples with $\mathbf{P}$ dominate the work. We cannot find a precise number of matrix-vector multiplies because the work varies between runs. Instead, we compute the expected (or average) number.

We begin our formal analysis by noting that the kth iterate from the power method on the PageRank system satisfies

$$\left\| \mathbf{x}^{(k)} - \mathbf{x}^{\star} \right\| \leq 2\varepsilon \tag{4.33}$$

when $k > \log(\varepsilon)/\log(\alpha)$ and $\mathbf{x}^{\star}$ is the exact solution; that is, it takes at most k iterations (matrix multiplications) for the power method to get 2ε accuracy.[20]

Using the bound (4.33), we can estimate the expected number of iterations in the Monte Carlo method given that we are taking N samples:

$$\mathrm{E}\left[M\right] = \sum_{i=1}^{N} \mathrm{E}\left[M_i\right] \leq N \underbrace{\int_l^r \frac{\log(\varepsilon)}{\log(\tau)} \rho(\tau)\, d\tau}_{\text{expected iterations for one sample}}, \tag{4.34}$$

where M is the total number of matrix-vector multiplies and M_i is the number of iterations in the ith random sample.

In the case when $\rho(\tau)$ corresponds to a Beta distribution over $(0,1)$ with integer $a > 0, b > 0$, we can solve the integral analytically. From Zwillinger et al. [1996, p. 401],

$$\int_0^1 \frac{x^p - x^q}{\log x} = \log(p+1) - \log(q+1), \quad p > -1, q > -1. \tag{4.35}$$

Now, recall and substitute the density function from (4.8)

$$\mathrm{E}\left[M\right] = N \frac{\log(\varepsilon)}{\mathrm{Beta}(a+1, b+1)} \int_0^1 \frac{\tau^b (1-\tau)^a}{\log \tau}\, d\tau. \tag{4.36}$$

To compute the integral, we subtract $0 = (1-1) = (1-\tau^0)^a$,[21] then expand both $(1-\tau)^a$ and $(1-\tau^0)^{-1}$ using binomial coefficients. Then we apply (4.35) with p or $q = 0$. Formally,

$$\begin{aligned} \int_0^1 \frac{\tau^b (1-\tau)^a}{\log \tau}\, d\tau &= \int_0^1 \frac{\tau^b (1-\tau)^a - (1-\tau^0)^a}{\log \tau}\, d\tau \\ &= \sum_{k=0}^{a} (-1)^k \binom{a}{k} \int_0^1 \frac{\tau^{k+b} - \tau^0}{\log \tau} \\ &= \sum_{k=0}^{a} (-1)^k \left[\binom{a}{k} \log(k+b+1) - \log 1 \right]. \end{aligned} \tag{4.37}$$

All of the previous work can be summarized in the following theorem.

[20] This result follows directly from lemma 3 with the initial error 2. Any two stochastic vectors have a 1-norm difference of at most 2, and so 2 is an upper bound on the initial error. Empirically, the inner-outer method uses fewer iterations than the power method, so we use the upper bound from the latter.

[21] For this problem, $1 - \tau^0$ is the right value of 0 to subtract for an easy computation.

Theorem 13. *If $A \sim Beta(a, b, [0, 1])$ with integers $a > 0$ and $b > 0$, then approximating $\mathrm{E}[\mathbf{x}(A)]$ with an N-sample Monte Carlo method takes*

$$N \frac{\log \varepsilon}{Beta(a+1, b+1)} \sum_{k=0}^{a} (-1)^k \binom{a}{k} \log(b+i+1) \tag{4.38}$$

matrix multiplications.

One problem with this theorem is that it does not handle $a = 0, b = 0$—the case when A is uniformly distributed. Computing this expectation exactly is impossible in this case because the indefinite integral $\int_0^1 \log^{-1}(\tau)\, d\tau$ does not converge. To handle this case, we would need to quantify the convergence of the power method when α is bigger than the second largest magnitude eigenvalue of $\mathbf{P}$. More precisely, $\mathbf{P}$ has many eigenvalues on the unit circle. Let λ_2 be the magnitude of the first eigenvalue inside the unit circle. We need the convergence of the power method when $\alpha > \lambda_2$ and that will depend more strongly on λ_2 than on α.

4.7.2 *Path damping*

When $A \sim \mathrm{Beta}(a, b, [0, r]), r \leq 1$, we can explicitly bound the convergence of the path damping algorithm for $\mathrm{E}[\mathbf{x}(A)]$. Recall the path damping approximation from (4.22):

$$\mathrm{E}[\mathbf{x}(A)] \approx \mathbf{x}^{(N)} = \sum_{\ell=0}^{N} \mathrm{E}\left[A^\ell - A^{\ell+1}\right] \mathbf{P}^\ell \mathbf{v} + (1 - \sum_{\ell=0}^{N} \mathrm{E}\left[A^\ell - A^{\ell+1}\right]) \mathbf{P}^{N+1} \mathbf{v}.$$

Note that

$$(1 - \sum_{\ell=0}^{N} \mathrm{E}\left[A^\ell - A^{\ell+1}\right]) = \mathrm{E}\left[A^{N+1}\right]. \tag{4.39}$$

Thus we have

$$\begin{aligned} \left\|\mathbf{x}^{(N)} - \mathbf{x}^\star\right\| &= \left\| \mathrm{E}\left[A^{N+2}\right] \mathbf{P}^{N+1} \mathbf{v} - \sum_{\ell=N+2}^{\infty} \mathrm{E}\left[A^\ell - A^{\ell+1}\right] \mathbf{P}^\ell \mathbf{v} \right\| \\ &\leq \mathrm{E}\left[A^{N+2}\right] + \mathbf{e}^T \sum_{\ell=N+2}^{\infty} \mathrm{E}\left[A^\ell - A^{\ell+1}\right] \mathbf{P}^\ell \mathbf{v} \\ &\leq 2\,\mathrm{E}\left[A^{N+2}\right]. \end{aligned}$$

We could have removed the final normalization term $\mathrm{E}\left[A^{N+1}\right] \mathbf{P}^{N+1} \mathbf{v}$ in the summation and bounded the result by $\mathrm{E}\left[A^{N+1}\right]$ instead. However, the iteration we outlined in the algorithms section gives us better performance in practice.[22]

[22] Give it a try yourself; maybe you will have a different experience.

For $r \le 1$, (4.29) gives

$$\begin{aligned}
\mathrm{E}\left[A^{N+2}\right] & \\
&= r^{N+2}\frac{\Gamma(b+N+3)\Gamma(a+b+2)}{\Gamma(b+1)\Gamma(a+b+N+4)} \\
&= r^{N+2}\frac{\Gamma(a+b+2)}{\Gamma(b+1)}\frac{1}{(b+N+3)(b+N+4)\cdots(a+b+N+3)} \\
&\le r^{N+2}\frac{\Gamma(a+b+2)}{\Gamma(b+1)}\frac{1}{(b+N+3)^{a+1}},
\end{aligned} \tag{4.40}$$

from which we conclude that the path damping algorithm for computing $\mathrm{E}\left[x(A)\right]$ converges like $\frac{r^{N+2}}{N^{a+1}}$.

4.7.3 *Error bounds on Gaussian quadrature*

Quadrature methods are extremely old tools and many excellent error analysis techniques exist. For example, Davis and Rabinowitz [1984] devotes an entire chapter to their study. Let $\mathbf{x}^{\mathrm{GQ}(N)}$ be the approximation to $\mathrm{E}\left[\mathbf{x}(A)\right]$ using an N-point quadrature rule. We can only achieve an error bound for any component and thus use the upper bound

$$\left\|\mathrm{E}\left[\mathbf{x}(A)\right] - \mathbf{x}^{\mathrm{GQ}(N)}\right\| \le n\left|\mathrm{E}\left[x_i(A)\right] - x_i^{\mathrm{GQ}(N)}\right|.$$

This bound is terrible for large n and we do not expect it to be tight. Instead, we focus on the error decay—how much the error drops when N.

Computing the bound on a component is involved, as the following theorem and proof demonstrate.

Theorem 14. *Let A be a random variable with finite moments and support $[0, r]$, where $r < 1$. The error in the Gauss quadrature approximation of $\mathrm{E}\left[\mathbf{x}(A)\right]$ is bounded above by*

$$\left|\mathrm{E}\left[x_i(A)\right] - x_i^{\mathrm{GQ}(N)}\right| \le \frac{32\omega r}{15(1-\rho^{-2})\rho^{2N+2}},$$

where N is the number of points in the Gauss quadrature rule,

$$\omega = \sqrt{1+\frac{1}{r}} \quad \textit{and} \quad \rho = \frac{1}{r} + \sqrt{\frac{1}{r^2} - 1}.$$

Proof. There are many statements for the error in Gauss quadrature and we begin with a modern statement from Trefethen [2008, theorem 4.5]. Consider $I = \int_{-1}^{1} f(x)\,dx$ for an analytic function f. Let I_N be the N-point Gauss quadrature approximation to I. Then

$$|I - I_N| \le \frac{64\omega}{15(1-(\rho_a+\rho_b)^{-2})(\rho_a+\rho_b)^{2N+N}}, \tag{4.41}$$

where $|f(z)| \le \omega$ for all z in the ellipse with foci ±1 and semi-major axis $\rho_a > 1$ and semi-minor axis ρ_b. Figure 4.7 illustrates the construction.

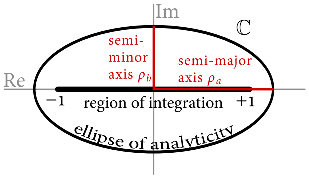


Figure 4.7 – The framework for Gauss quadrature error analysis. The ellipse of analyticity provides bounds on the error in a Gauss quadrature rule. Roughly, $|\text{error}| \le (\rho_a + \rho_B)^{2N}$.

Note that we are approximating the integral

$$\mathrm{E}\left[x_i(A)\right] = \int_0^r x_i(\alpha)\,d\alpha$$

with an N-point quadrature rule. Each PageRank component is a rational function, which is a special case of an analytic function. In the remainder of the proof, we go through the details of applying the bound from (4.41) precisely. First, we transform the problem to the integration region $[-1, 1]$ by a change of variables α to z. In this z-space, we build an ellipse in the complex plain where $x_i(z)$ is analytic. To study the function magnitude ω, we transform the ellipse back to α-space and examine the magnitude of PageRank as a function of α when α is complex.

Let

$$z = \frac{2\alpha}{r} - 1 \quad\Longleftrightarrow\quad \alpha = \frac{r}{2}(z+1)$$

be the change of variables between α and z. Consider $z = z_R + iZ_i$ for z in the ellipse with foci ±1. The ellipse satisfies

$$\frac{z_R^2}{\rho_a^2} + \frac{z_I^2}{\rho_b^2} = 1$$

and the constraint on the foci implies that $\rho_a^2 = 1 + \rho_b^2$. Both ρ_a and ρ_b live in z-space, so for

$$\rho_a = \frac{1}{r},$$

we consider an ellipse in α-space with a right end-point $r/2 + 1/2$—halfway between r and 1.[23] The function $x_i(\alpha(z))$ is analytic inside this ellipse. The right endpoint (in α-space) is less than 1 and $x_i(\alpha)$ is analytic for all complex

[23] This choice of ρ_a may not be optimal, but other choices increase the difficulty of the computations considerably. In particular, we tried using a right endpoint of $\gamma r + (1-\gamma)$, but could only compute the upper bound ω when $\gamma = 1/2$.

α with $|\alpha| < 1$. See Horn and Serra-Capizzano [2007] for the first study of PageRank with complex α. Thus,

$$\rho_a + \rho_b = \frac{1}{r} + \sqrt{\frac{1}{r^2} - 1}$$

slips into (4.41) for the PageRank case.

We have $\rho_a + \rho_b$; let's now find ω.

In α-space where $\alpha = \alpha_R + i\alpha_I$, the ellipse is

$$\frac{(r/2 - \alpha_R)^2}{(1/2)^2} + \frac{\alpha_I^2}{(\frac{1}{2}\sqrt{1-r^2})^2} = 1.$$

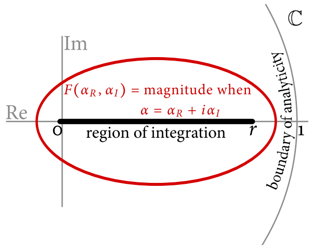

ℂ
Im
Re
$F(\alpha_R, \alpha_I)$ = magnitude when
$\alpha = \alpha_R + i\alpha_I$
0
region of integration
r
1
boundary of analyticity

Figure 4.8 – The Gauss quadrature error analysis applied to PageRank. When integrating $x_i(\alpha)$, we use the ellipse given in red to bound the error in a Gauss quadrature approximation. Note that $x_i(\alpha)$ is clearly analytic in this region as it's enclosed inside $|\alpha| < 1$.

This ellipse is centered at $r/2$ with semi-major axis length $1/2$, as illustrated in figure 4.8.

In (4.41), the value of ω is an upper bound on $f(z)$ inside the ellipse. Thus, we must bound the magnitude of PageRank components for a complex α. First,

$$\mathbf{x} = \alpha\mathbf{P}\mathbf{x} + (1-\alpha)\mathbf{v} \quad \text{gives} \quad \|\mathbf{x}\| \le |\alpha|\,\|\mathbf{x}\| + |1-\alpha|.$$

For complex α, this bound yields

$$\begin{aligned} |\mathbf{x}_i(\alpha)| \le \|\mathbf{x}(\alpha)\| &\le \frac{|1-\alpha|}{1-|\alpha|} \\ &= \frac{\sqrt{(1-\alpha_R)^2 + \alpha_I^2}}{1 - \sqrt{\alpha_R^2 + \alpha_I^2}} \\ &\equiv F(\alpha_R, \alpha_I). \end{aligned} \tag{4.42}$$

When $\alpha_I = 0$, this bound respects the property that $x_i(\alpha) \le 1$ for $0 \le \alpha_R < 1$. When $\alpha_I \ne 0$, the bound is considerably more interesting. In figure 4.9 at right, we see that as α_I increases, F increases. Analytically, we find that $\partial F/\partial\alpha_I > 0$ for $\alpha_I > 0$ and $\partial F/\partial\alpha_I < 0$ for $\alpha_I < 0$. Consequently, the maximum ω is going to occur on the boundary of the ellipse. In this case,

$$\alpha_I^2 = \frac{1}{4}(1-r^2)\left(1 - \frac{(r/2-\alpha_R)^2}{(1/2)^2}\right).$$

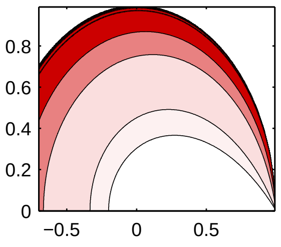


Figure 4.9 – PageRank magnitude for a complex damping parameter. In this contour plot, we show an upper bound on $\|\mathbf{x}(\alpha)\|$ when $\alpha \in \mathbb{C}$. Darker red indicates larger magnitude and white indicates a magnitude near 0. The magnitudes increase as α veers off the real line, or when the real component is negative.

Let $F_R(\alpha_R) = F(\alpha_R, \alpha_I(\alpha_R))$ be the value of F on the ellipse. The critical points of F are

$$\alpha_R = \frac{r^2-1}{2r}\ ; \ \frac{r^2-2r-3}{2r}\ ; \ \frac{r^2+2r-1}{2r}$$

with values

$$F_R(\cdot) = \sqrt{1 + \frac{1}{r} - r}\;;\; -i\sqrt{\frac{3}{r} - 1}\;;\; \sqrt{1 + \frac{1}{r}}.$$

Only

$$\alpha_R = \frac{r^2 + 2r - 1}{2r}$$

is inside the region of integration, and thus

$$\omega = \sqrt{1 + \frac{1}{r}}.$$

There is one more step:

$$\left|\mathrm{E}\left[x_i(A)\right] - x_i^{\mathrm{GQ}(N)}\right| \le \left|\frac{d\alpha}{dz}\right| \left|\mathrm{E}\left[x_i(z)\right] - x_i^{\mathrm{GQ}(N)}\right|.$$

The initial bound (4.41) now applies to the second expression with $\rho_a + \rho_b = \frac{1}{r} + \sqrt{\frac{1}{r^2} - 1}$ and $\omega = \sqrt{1 + \frac{1}{r}}$.[24] □

Thus, the quadrature codes converge to the exact solutions as $N \to \infty$ when $r < 1$. When $r = 1$, the story is much more complicated. Using Trefethen's bound on the convergence of quadrature, and bounds on analytic functions in the complex plane, it suffices to show that $\mathbf{x}(\alpha)$ is analytic in an ellipse that encloses the integration region $[0, 1]$. This follows because $\mathbf{x}(\alpha)$ is analytic at $\alpha = 1$ and has poles at $\frac{1}{\lambda_i}$ where λ_i is an eigenvalue of $\mathbf{P}$ that is different from 1. All of the other eigenvalues have $|\lambda_i| < 1$ and thus, we must be able to fit an ellipse (potentially a small ellipse) between these eigenvalues and the integration region $[0, 1]$. Figure 4.10 illustrates and shows a hypothetical ellipse with semi-major and semi-minor axes that sum to more than 1. This result gives us convergence when $r = 1$, but at an unknown rate.

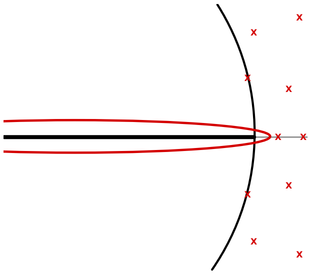
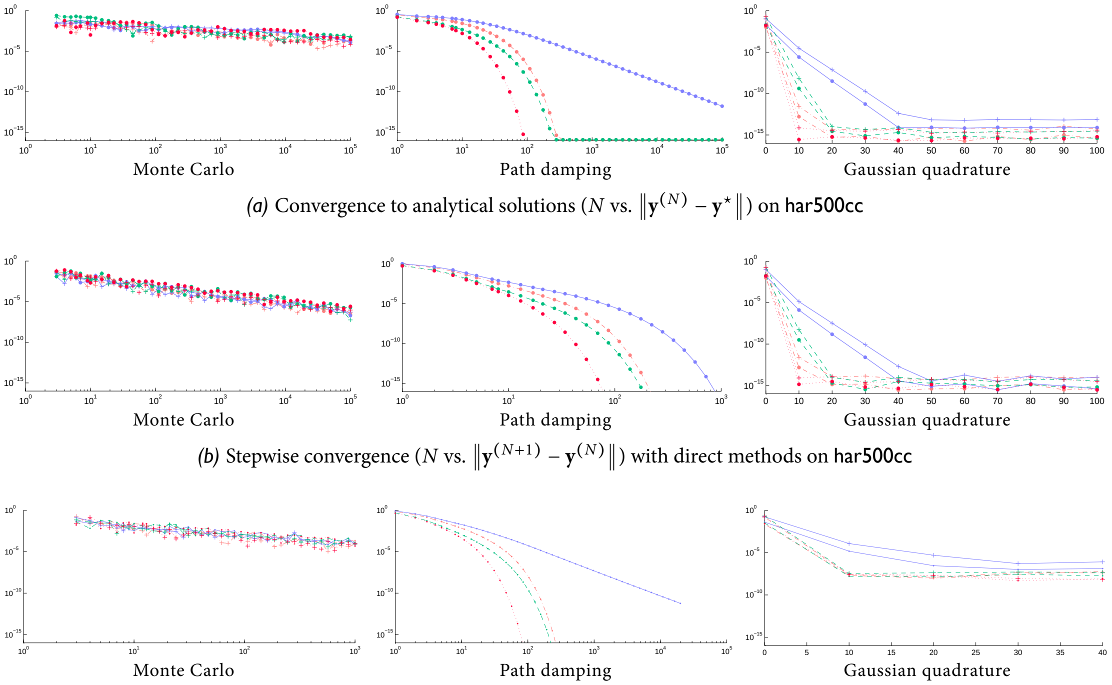


Figure 4.10 – Quadrature convergence with extreme endpoint. This figure illustrates the ellipse of analyticity with semi-major and semi-minor axis sum larger than 1 used to show that Gauss quadrature converges. The red x's are singularities of the PageRank function with $|\alpha| \ge 1$. The circle shows the boundary $|\alpha| = 1$.

4.7.4 *Implementation correctness and convergence*

In this section, we present empirical results pertaining to the accuracy and convergence of our implementations. This type of analysis is important because numerical experimentation allows us to explore broader ranges of parameter values than may be feasible in the theoretical analysis. Additionally, it provides strong evidence that we have correctly implemented all the algorithms in this chapter. To begin, we use three experiments to verify that our algorithms are convergent when implemented with and without approximate solutions of the linear algebra problems. Each of our algorithms has a

[24] We avoided much of the tedious algebra in this proof by valiantly employing the computer algebra packages Maple and Mathematica.



(a) Convergence to analytical solutions (N vs. $\|\mathbf{y}^{(N)} - \mathbf{y}^{\star}\|$) on har500cc

(b) Stepwise convergence (N vs. $\|\mathbf{y}^{(N+1)} - \mathbf{y}^{(N)}\|$) with direct methods on har500cc

(c) Stepwise convergence (N vs. $\|\mathbf{y}^{(N+1)} - \mathbf{y}^{(N)}\|$) with iterative methods on cnr-2000

Figure 4.11 – Convergence of algorithms for RAPr. All of our implementations converge with iterative methods and direct methods in a stepwise sense for $\mathbf{y}^{(N)} \approx \mathrm{E}[\mathbf{x}(A)]$ (dotted points) and $\mathbf{y}^{(N)} \approx \mathrm{Std}[\mathbf{x}(A)]$ ("+" points). Computing the standard deviation with path damping was too inefficient to include. The colors correspond to distributions from figure 4.4a.

parameter N that controls the degree of approximation. Theoretically, all the algorithms are convergent as $N \to \infty$.

We first test this convergence by comparing with a semi-analytical solution. Using the symbolic toolbox inside Matlab, we compute the PageRank vector as a rational function of α on the har500cc graph, a 335 node connected component.[25] Using Mathematica, we then numerically integrate (4.32) for the expectation and standard deviation in 32-digit arithmetic. This process resolves the "exact" solution when converted to a double precision number. Finally, we track convergence of each algorithm to these semi-analytical solutions in figure 4.11a. As the respective N increases, all methods demonstrate convergence to the exact solution. For the same graph, we also analyze stepwise convergence by tracking the 1-norm change when incrementing N to $N + 1$ (figure 4.11b). These results use a direct method to solve any linear system that arises. Finally, we replace har500cc with cnr-2000, a 325,557 node graph, and use the inner-outer iteration to solve the PageRank systems with a tolerance of 10^{-8}. In both of these cases, the algorithms are convergent.

[25] The symbolic expressions for even a single component of the PageRank vector as a function of α are incredible. See figure 2.5.

We now make a few additional observations:

- the Monte Carlo method has similar convergence behavior for all distributions and does not achieve better than typical accuracy for all tests;
- the $\text{Beta}(2, 16, [0, 1])$ problem (solid light blue line) requires the largest N for all methods except Monte Carlo;
- the accuracy of the standard deviation is less than the accuracy of the expectation; and
- using stepwise convergence as a proxy for analytical convergence in path damping can produce significant errors.

The last statement merits further comment. A simple calculation shows that stepwise convergence of the path damping expression is

$$\left\| \mathbf{x}_{PD}^{(N)} - \mathbf{x}_{PD}^{(N+1)} \right\| = \mathrm{E}\left[A^{N+2}\right] \left\| \mathbf{P}^{N+2}\mathbf{v} - \mathbf{P}^{N+1}\mathbf{v} \right\|, \tag{4.43}$$

which is how we compute the values for the figures. The theoretical bound is much weaker with $\left\|\mathbf{P}^{N+2} - \mathbf{P}^{N+1}\right\|$ replaced by the trivial value 2. When the vectors $\mathbf{P}^{N}\mathbf{v}$ reach a small value, stepwise convergence is no longer a good bound. Consequently, our final code for the path damping formulation uses $\mathrm{E}\left[A^{N+2}\right]$ to test convergence instead.

Next, we examine the runtime for these methods in the hard case of the $\text{Beta}(2, 16, [0, 1])$ distribution.[26] Figure 4.12 displays the values of figure 4.11c against the time they took to compute. Again, the standard deviation was not computed for the path damping algorithm. These timings include all computations of moments and eigenvalues for path damping and Gaussian quadrature.

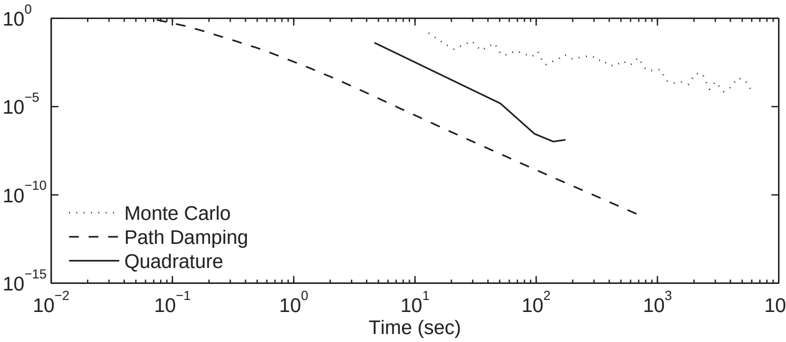


Figure 4.12 – Timing for the RAPr algorithms. The time required to compute the results from figure 4.11c for $\mathrm{E}\left[\mathbf{x}(A)\right]$, $A \sim \text{Beta}(2, 16, [0, 1])$.

[26] The case when $r = 1$ has the slowest convergence for all the methods.

Based on these experiments, we advise the following. Path damping is the algorithm of choice when $r \ll 1$ or the standard deviation is not required. Otherwise, the best method for computing both the expectation and standard deviation for reasonably accurate ($\approx 10^{-4}$–10^{-8}) solutions is Gaussian quadrature with about 33 points.[27]

4.8 APPLICATIONS

Thus far, we have theoretically examined the RAPr model, given algorithms to compute its statistics, and analyzed those algorithms; we have yet to address applications of this model. While the expected value of the RAPr model appears to order nodes like the deterministic PageRank vector at the expected α, the standard deviation vector orders nodes differently. We first demonstrate this behavior for a range of graphs and distributions of A. Then, we show similar observations on a large web graph and discuss the intersection similarity of the standard deviation vector for this graph. Next, we present an example of our model outside the web graph domain and observe that this ranking behavior of the standard deviation vector holds for a gene ranking application. Finally, we show that using the standard deviation information aids a spam classification task.

4.8.1 *PageRank*

To begin our empirical analysis of RAPr, we present table 4.3. For the four Beta distributions we have examined throughout this chapter, the table presents the 1-norm, Kendall's τ correlation coefficient, and a truncated-τ correlation coefficient between $\mathbf{x}(\mathrm{E}[A])$, $\mathrm{E}[\mathbf{x}(A)]$, and $\mathrm{Std}[\mathbf{x}(A)]$. The 1-norm difference is rescaled to be related to a correlation coefficient when applied to probability distribution vectors. The truncated-τ or τ_ε measure removes digits less than ε before computing τ. Formally,

$$\tau_\varepsilon(\mathbf{y}, \mathbf{z}) = \tau(\varepsilon \operatorname{round}(\mathbf{y}/\varepsilon), \varepsilon \operatorname{round}(\mathbf{z}/\varepsilon)), \tag{4.44}$$

where the "round function" rounds to the nearest integer. The τ_ε measure is motivated by inconsistencies with the τ measure and inaccurate computation [Boldi et al., 2007]. The expectation and standard deviation were computed with a 33-point quadrature rule[28] and each PageRank system solved to a weighted 10^{-9} tolerance (see section 4.6.4).

[27] Using 33 quadrature points may seem like a lot to those accustomed to integrating smooth functions. With PageRank, there is a singularity near the region of integration and we need to use many points.

[28] See note 27 for a comment about the number of points.

Table 4.3 – A comparison between PageRank and Random Alpha PageRank. The function $f(\mathbf{y},\mathbf{z}) = 1 - \|\mathbf{y}-\mathbf{z}\|$ shifts the difference in norm to $[-1,1]$; τ is Kendall's τ correlation coefficient; and τ_ε is τ with $\mathbf{y}$ and $\mathbf{z}$ truncated to 8 digits. Positive values of τ are shaded red whereas negative values of τ are shaded blue. Values near 0 have no shading and indicate places where the vectors are uncorrelated.

Graph	Beta				$\mathbf{y} = \mathbf{x}(\mathrm{E}[A]), \mathbf{z} = \mathrm{E}[\mathbf{x}(A)]$			$\mathbf{y} = \mathbf{x}(\mathrm{E}[A]), \mathbf{z} = \mathrm{Std}[\mathbf{x}(A)]$		
	a	b	l	r	$f(\mathbf{y},\mathbf{z})$	$\tau(\mathbf{y},\mathbf{z})$	$\tau_\varepsilon(\mathbf{y},\mathbf{z})$	$f(\mathbf{y},\mathbf{z})$	$\tau(\mathbf{y},\mathbf{z})$	$\tau_\varepsilon(\mathbf{y},\mathbf{z})$
uk-2006-host	0	0	0.6	0.9	0.972	0.995	0.995	0.173	0.200	0.196
	2	16	0	1	0.943	0.994	0.994	0.231	0.599	0.597
	1	1	0.1	0.9	0.963	0.984	0.983	0.229	-0.421	-0.418
	-0.5	-0.5	0.2	0.7	0.970	0.983	0.982	0.210	-0.457	-0.454
uk-2007-host	0	0	0.6	0.9	0.971	0.997	0.993	0.176	-0.071	-0.072
	2	16	0	1	0.944	0.996	0.995	0.232	0.498	0.455
	1	1	0.1	0.9	0.961	0.987	0.987	0.221	-0.578	-0.557
	-0.5	-0.5	0.2	0.7	0.969	0.986	0.975	0.201	-0.586	-0.563
nz2006	0	0	0.6	0.9	0.984	0.995	0.978	0.114	-0.546	-0.333
	2	16	0	1	0.976	0.996	0.966	0.135	0.027	-0.192
	1	1	0.1	0.9	0.975	0.981	0.980	0.143	-0.620	-0.506
	-0.5	-0.5	0.2	0.7	0.980	0.981	0.950	0.125	-0.614	-0.527
eu-2005	0	0	0.6	0.9	0.975	0.993	0.987	0.174	0.318	0.286
	2	16	0	1	0.952	0.992	0.982	0.214	0.517	0.524
	1	1	0.1	0.9	0.962	0.976	0.975	0.267	-0.536	-0.518
	-0.5	-0.5	0.2	0.7	0.968	0.975	0.974	0.251	-0.621	-0.604
us2004cc	0	0	0.6	0.9	0.971	0.989	0.990	0.173	0.179	0.177
	2	16	0	1	0.947	0.985	0.986	0.225	0.436	0.461
	1	1	0.1	0.9	0.960	0.969	0.973	0.247	-0.395	-0.364
	-0.5	-0.5	0.2	0.7	0.967	0.969	0.974	0.230	-0.489	-0.468
enwiki-2008	0	0	0.6	0.9	0.981	0.996	0.995	0.180	0.240	0.159
	2	16	0	1	0.975	0.995	0.994	0.189	0.381	0.184
	1	1	0.1	0.9	0.961	0.986	0.984	0.277	-0.444	-0.406
	-0.5	-0.5	0.2	0.7	0.966	0.986	0.984	0.262	-0.578	-0.222
indochina	0	0	0.6	0.9	0.975	0.993	0.968	0.165	0.189	0.229
	2	16	0	1	0.946	0.991	0.972	0.217	0.479	0.569
	1	1	0.1	0.9	0.966	0.974	0.958	0.250	-0.542	-0.284
	-0.5	-0.5	0.2	0.7	0.973	0.973	0.949	0.235	-0.613	-0.358
uk2005	0	0	0.6	0.9	0.985	0.997	0.903	0.110	-0.519	-0.199
	2	16	0	1	0.974	0.997	0.967	0.134	0.065	-0.034
	1	1	0.1	0.9	0.977	0.985	0.947	0.144	-0.596	-0.080
	-0.5	-0.5	0.2	0.7	0.981	0.984	0.916	0.128	-0.598	-0.137

From the table, we observe:

- the PageRank vector $\mathbf{x}(\mathrm{E}[A])$ and the expected value in the random model $\mathrm{E}[\mathbf{x}(A)]$ are numerically similar and induce similar orderings of the pages;
- the standard deviation vector $\mathrm{Std}[\mathbf{x}(A)]$ is neither numerically similar nor similar in either τ metric to $\mathbf{x}(\mathrm{E}[A])$;
- using τ_ε can give different results; and
- the behavior of the standard deviation vector is not consistent between graphs and distributions.

The first shaded column group of the table justifies the first statement. The marked reduction in shading in the second column group explains the second, and the seemingly random values in this column group justify the last statement. Interestingly, four graphs behave nearly the same: uk-2006-host, uk-2007-host, eu-2005, and us2004cc. With the exception of uk-2007-host, these graphs have the highest percentage of nodes in the largest strong component.

The graph uk2005 demonstrates the largest discrepancy between τ and τ_ε. This relatively large difference may signify that it differs characteristically from the other graphs. However, most of its standard deviation values are less than 10^{-8}, so truncating the τ metric with $\varepsilon = 10^{-8}$ may lose important information. Another explanation for the discrepancy is that more than half of the nodes in this graph have no links.

4.8.2 *PageRank on a large graph*

The graphs in the previous section are small compared with the size of the true web graph. Now we address computing the quantities on a graph with 78 million nodes and just under 3 billion edges: the uk-2006 web spam test graph [Castillo et al., 2006].[29] Our distributions of interest are $A_1 \sim \mathrm{Beta}(2, 16, [0, 1])$ and $A_2 \sim \mathrm{Beta}(1, 1, [0, 1])$. We chose the former because $\mathrm{E}[A_1] = 0.85$, the canonical value of α, and the latter because $\mathrm{E}[A_2] = 0.5$, a recently proposed alternative value of α. Both of these distributions have small a and support that extends all the way to 1. This makes computing the solution with path damping a difficult proposition, so we choose to use Gaussian quadrature. For A_1 we used a 25-point rule, and for A_2 we used a 10-point rule. The error bounds on quadrature state that these results may have

[29] Even this graph is tiny compared with the real web graph.

considerable error from the quadrature approximation. But for big problems, running hundreds of Gauss points is not feasible.[30]

While the MATLAB codes given throughout this chapter handle this graph through the *bvgraph* package, working in MATLAB is roughly half the speed of an optimized computation. Consequently, we used a C++ implementation of the inner-outer iteration to solve the PageRank systems and compute the aggregated solution using a *bvgraph* structure to hold the graph in memory [Boldi and Vigna, 2004].

The time required for our deterministic solves (tolerance 10^{-12}) was

$$\begin{array}{ll} \alpha = 0.85 & 204 \text{ minutes,} \\ \alpha = 0.5 & 51 \text{ minutes.} \end{array}$$

Computing the expectation and standard deviation in the RAPr model required

$$\begin{array}{ll} A_1 & 6199 \text{ minutes,} \\ A_2 & 1569 \text{ minutes.} \end{array}$$

Our codes solved each PageRank vector to a weighted tolerance of 10^{-12}. This accuracy is far more than required when given the intrinsic error in the quadrature approximation mentioned above. Nevertheless, we might as well get something accurate with these computations when we can.

To analyze the output, we use two schemes. First, we apply the truncated τ measure to the expectation and standard deviation vectors (table 4.4). The comparison shows that $\mathrm{E}[\mathbf{x}(A)] \approx \mathbf{x}(\mathrm{E}[A])$ in terms of ranking and that the standard deviation vectors behave differently under this measure. Interestingly, the standard deviation vector for A_2 appears to invert the orderings of all other measures and the magnitude of its anti-correlation is much stronger than for A_1.

Table 4.4 – PageRank vs. random alpha PageRank sensitivity on a big graph. The truncated τ values ($\tau_\varepsilon(\mathbf{y}, \mathbf{z})$ with $\varepsilon = 10^{-10}$) again show that the standard deviation vectors produce different rankings from the expectation vectors for the graph **uk-2006** with 77 million vertices and 2.2 billion edges. A_1 is a $\mathrm{Beta}(2, 16, [0, 1])$ random variable with statistics computed using a 25-point quadrature rule, and the parameter A_2 is a $\mathrm{Beta}(1, 1, [0, 1])$ random variable computed using a 10-point quadrature rule. The coloring is the same as in table 4.3.

	z					
y	$\mathbf{x}(0.85)$	$\mathbf{x}(0.95)$	$\mathrm{E}[\mathbf{x}(A_1)]$	$\mathrm{E}[\mathbf{x}(A_2)]$	$\mathrm{Std}[\mathbf{x}(A_1)]$	$\mathrm{Std}[\mathbf{x}(A_2)]$
$\mathbf{x}(0.5)$	0.850	0.765	0.845	*0.956*	0.412	*-0.538*
$\mathbf{x}(0.85)$		0.910	*0.967*	0.891	*0.294*	-0.675
$\mathbf{x}(0.95)$			0.916	0.808	0.219	-0.706
$\mathrm{E}[\mathbf{x}(A_1)]$				0.892	*0.287*	-0.675
$\mathrm{E}[\mathbf{x}(A_2)]$					0.378	*-0.577*

[30] In chapter 7, we discuss a few ideas to make the codes more scalable.

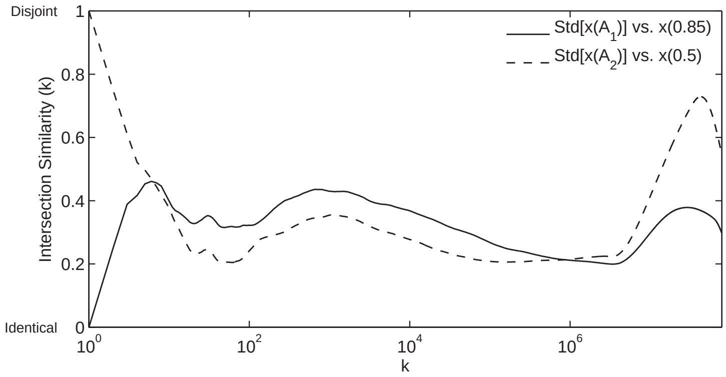


Figure 4.13 – Intersection similarity between PageRank and the RAPr model. The intersection similarity metric for the **uk-2006** graph shows that the standard deviation vector is unlike the PageRank vector under this measure. The computations were done for $A_1 \sim \text{Beta}(2, 16, [0, 1])$ with a 25-point quadrature rule and for $A_2 \sim \text{Beta}(1, 1, [0, 1])$ with a 10-point quadrature rule.

The second comparison metric is the intersection similarity metric [Boldi, 2005]. Given two ordered sequences of items $\mathcal{A}$ and $\mathcal{B}$, let $\mathcal{A}_k$ (resp. $\mathcal{B}_k$) be the top k items in $\mathcal{A}$ (resp. $\mathcal{B}$). Then

$$\text{isim}_k(\mathcal{A}, \mathcal{B}) = \frac{1}{k} \sum_{j=1}^{k} \frac{|\mathcal{A}_j \Delta \mathcal{B}_j|}{2j}, \tag{4.45}$$

where Δ is the symmetric difference operator between two sets. The intersection similarity is the average of the normalized symmetric differences for all top-j lists with $j \leq k$. If the two orderings are identical, then $\text{isim}_k = 0$ for all k. If the two sequences have disjoint items, then $\text{isim}_k = 1$. Figure 4.13 displays this value for the standard deviations vectors. For A_1, the intersection similarity hovers around 0.3 with increases at 10, 1,000 and 10,000,000 pages. In contrast, $\text{Std}[\mathbf{x}(A_2)]$ has a higher intersection similarity for the first 10^6 pages and orders the tail quite differently, resulting in a peak past 10^6 pages. This final peak is perhaps indicative of the negative τ correlation between $\text{Std}[\mathbf{x}(A_2)]$ and $\mathbf{x}(0.5)$.

These results support our claim that the standard deviation of RAPr reveals characteristically new information for the underlying graph.

4.8.3 *Gene regulatory networks*

Recently, many authors have used PageRank-type equations as measures on arbitrary graphs. Among these measures are GeneRank [Morrison et al., 2005] for identifying important genes in a regulatory network, ProteinRank [Freschi, 2007] for identifying important proteins, and IsoRank [Singh et al., 2007] for identifying important edges in a graph-isomorphism-like problem. We will demonstrate the results of RAPr on the GeneRank problem using the data published for that paper.

	r				
l	0.2	0.4	0.6	0.8	1.0
0.0	0.999	0.996	0.988	0.973	0.935
0.2		0.999	0.994	0.980	0.944
0.4			0.998	0.988	0.954
0.6				0.996	0.967
0.8					0.984

(a) Values of $\tau(\mathbf{x}(\mathrm{E}[A]), \mathrm{E}[\mathbf{x}(A)]), A \sim U(l, r)$

	r				
l	0.2	0.4	0.6	0.8	1.0
0.0	0.166	0.212	0.261	0.317	0.389
0.2		0.256	0.305	0.356	0.414
0.4			0.342	0.381	0.413
0.6				0.382	0.381
0.8					0.326

(b) Values of $\tau(\mathbf{x}(\mathrm{E}[A]), \mathrm{Std}[\mathbf{x}(A)]), A \sim \mathrm{U}(l, r)$

	b					
a	1	4	7	10	13	16
1	0.964	0.965	0.970	0.975	0.979	0.982
4	0.990	0.985	0.984	0.985	0.985	0.986
7	0.995	0.992	0.990	0.990	0.990	0.990
10	0.997	0.995	0.994	0.993	0.993	0.993
13	0.998	0.996	0.995	0.995	0.995	0.994
16	0.999	0.997	0.997	0.996	0.996	0.995

(c) Values of $\tau(\mathbf{x}(\mathrm{E}[A]), \mathrm{E}[\mathbf{x}(A)]), A \sim \mathrm{Beta}(a, b, 0, 1)$

	b					
a	1	4	7	10	13	16
1	0.378	0.410	0.386	0.362	0.344	0.331
4	0.263	0.362	0.395	0.399	0.392	0.383
7	0.217	0.305	0.355	0.382	0.392	0.394
10	0.194	0.268	0.319	0.352	0.373	0.385
13	0.180	0.244	0.291	0.326	0.350	0.367
16	0.170	0.226	0.269	0.303	0.329	0.349

(d) Values of $\tau(\mathbf{x}(\mathrm{E}[A]), \mathrm{Std}[\mathbf{x}(A)]), A \sim \mathrm{Beta}(a, b)$

Table 4.5 – RAPr vs. PageRank on the generank data. For the generank matrix, the Kendall-τ correlation coefficient shows that the PageRank and the expected PageRank order the genes similarly, whereas the standard deviation vector produces a different ordering under a wide range of parameters of the Beta distribution. The coloring is the same as in table 4.3.

In this context, we cannot interpret RAPr as representing a hypothetical random surfer. Instead, the GeneRank vector is used with a single choice of α to infer important genes. We propose using the standard deviation vector as another set of important genes, or as "confidence bounds" on the actual importance values for a gene. GeneRank uses an undirected graph of known relationships between genes instead of the directed web graph in PageRank and specifies a teleportation vector $\mathbf{v}$ based on the expression level for each gene in a micro-array experiment. In our experiments, we look at the τ correlation between $\mathbf{x}(\mathrm{E}[A])$, $\mathrm{E}[\mathbf{x}(A)]$, and $\mathrm{Std}[\mathbf{x}(A)]$ for a range of parameters when $A \sim \mathrm{Beta}(0, 0, [l, r])$ and when $A \sim \mathrm{Beta}(a, b, [0, 1])$. The expectation and standard deviation vectors are computed with a 50-point quadrature rule with a direct solution method.

Table 4.5 presents the τ correlations. We note a few interesting observations. Again, the τ difference between the expectation and PageRank vector is negligible, whereas the standard deviation vector does produce a value of τ much closer to 0. For all the tests, τ is positive and, as the mass of the Beta distribution shifts closer to 1, the τ values become larger. We hypothesize that these effects are due to the symmetric nature of the initial GeneRank graph, which has a stationary distribution proportional to the weighted degree of a node. From these experiments, we believe that looking at the standard deviation vector would be useful in this application.

4.8.4 *Spam classification*

Thus far, the evaluations of RAPr have been speculative. We've seen that the standard deviation vector *differs* from the standard PageRank vector. However, the proof is in the pudding and for RAPr, the pudding is spam.

Web spam occurs when a web site consists primarily of misleading content or links designed to draw visitors to generate ad revenue or inflate another site's importance. Web spam is distinguished by this artificiality. Identifying these sites is a growing problem and one technique is pure link analysis. Hypothetically, spam sites have dramatically different linking patterns than natural (non-spam) sites.

In Castillo et al. [2006] and Becchetti et al. [2008], the authors investigate identifying web spam purely from link analysis. They labeled around 7,500 hosts from the uk-2006 graph as follows.

Label	Train	Test
spam	674	1250
non-spam	4948	601
no label	5780	9551

The data have a training and test subset, although only the training subset is used in Becchetti et al. [2008] and in the following experiments. In the remainder of our own experiment, we continue following the methodology of Becchetti et al. [2008], and add the standard deviation vector from RAPr as an additional feature for a spam classification task. They released their data, which makes experimenting straightforward.

Figure 4.14 shows that the standard deviation information identifies some spam pages. In particular, a high standard deviation relative to PageRank (the right-hand side of the right figure) is a reasonably strong indicator. Ironically, a low standard deviation also appears to be an indicator.

Feature vectors for each host are included with the Becchetti et al. [2008] data. These features are numerical results that may have an impact on the "spaminess" of the pages on that web host and include TrustRank [Gyöngyi et al., 2004], PageRank, Truncated PageRank [Becchetti et al., 2008], amongst others. Thus, pure PageRank ideas are already included. To support our statement that the standard deviation of RAPr is different, then, we must be able to *improve* upon the performance with all these features present.

Although measures like PageRank, TrustRank, and RAPr produce one or two scores for each page, the previous study found that computing a few

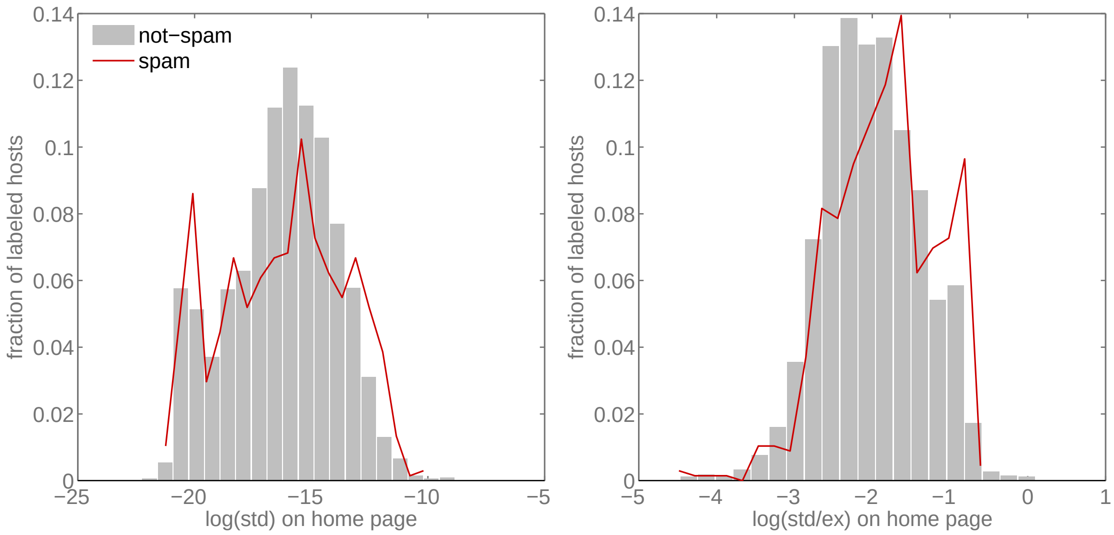


Figure 4.14 – Standard deviation and spam. The background histogram displays (log) standard deviation scores for non-spam hosts when $A \sim \text{Beta}(2, 16, [0, 1])$. The foreground (red) plot shows the same data for spam hosts. Each host is represented by its home page score and the statistics are computed with a 21-point quadrature rule. The second figure shows the same data for the (log) ratio of standard deviation over expectation.

statistics on these features aided the classification task. Thus, for RAPr on each host, we produce

- log of RAPr expectation
- log of (RAPr expectation / log of outdegree)
- log of (RAPr expectation / log of indegree)
- standard deviation of RAPr expectation on in-links
- log of (standard deviation of RAPr expectation on in-links / PageRank)
- log of RAPr standard deviation
- log of (RAPr standard deviation / log of outdegree)
- log of (RAPr standard deviation / log of indegree)
- standard deviation of standard deviation on in-links
- log of (standard deviation of RAPr standard deviation on in-links / PageRank)
- log of (standard deviation of RAPr / RAPr expectation)

where the RAPr scores are from the host home page, and the page with largest PageRank on the host. In total, we produce 22 features (= 11 from the list ×2 for the different host pages) from the RAPr statistics.

Hosts, with all of their features, are then input to a machine learning framework that attempts to learn a decision rule about spam based on these features.[31] Just like the original work, we use a Bagged J48 tree classifier in Weka [Witten and Frank, 2005] with 10 bags. Bagging a classifier produces a new classifier whose label is the concensus of a bag of independent classifiers. On the training data, we conducted 50 independent 10-fold cross-validation experiments to estimate the performance of the classifier, and table 4.6 displays the results. For each classifier, we show the

[31] Covering a full machine learning background is well outside the scope of this thesis.

Table 4.6 – Spam classification performance. Our performance baseline includes all the features from [Becchetti et al., 2008]. Each row represents adding features from either $RAPr$ or the derivative based on a particular Beta distribution or value of α. The results are averaged over 50 repetitions of 10-fold cross validation with a 10-bag J48 decision tree classifier.After adding features based on $RAPr$ and the derivative, we observe an improvement in the f-score. Consequently, these features uncover new information in the graph that is not expressed by PageRank.

	Precision	Recall	f-score	False Positive Ratio	False Negative Ratio
Baseline	0.694	0.558	0.618	0.034	0.442
Beta(1.5,0.5,0,0.99)	0.692	0.557	0.617	0.034	0.443
Beta(-0.5,-0.5,0.3,0.99)	0.698	0.564	0.624	0.033	0.436
Beta(0.5,1.5,0,0.99)	0.695	0.561	0.621	0.034	0.439
Beta(10,10,0.3,0.7)	0.690	0.560	0.620	0.034	0.442
Beta(1,1,0,1)	0.698	0.562	0.622	0.033	0.438
Beta(2,16,0,1)	0.699	0.562	0.623	0.033	0.438
Derivative (α = 0.75)	0.697	0.563	0.623	0.033	0.437
Derivative (α = 0.85)	0.697	0.561	0.622	0.033	0.439
Derivative (α = 0.95)	0.700	0.560	0.620	0.033	0.440

precision fraction of spam pages corrected labeled as spam;
recall fraction of total spam pages identified;
fscore harmonic mean of precision and recall;
false positive fraction of non-spam pages mislabeled as spam; and
false negative fraction of spam pages mislabeled as non-spam.

In the table we also add features based on the derivative. For the derivative features, we use the derivative instead of the standard deviation in the previous list.

Both the derivative and RAPr features improve the performance of the classifier! It is a small improvement, only a few tenths of a percent in both cases. Using features from the Beta$(-0.5, -0.5, [0, 3, 099])$ distribution, we obtain the best classification performance. In some sense, this distribution represents the least-likely surfer behavior. In contrast, the actual surfer behavior Beta$(1.5, 0.5, [0, 0.99])$ has the worst performance of all the experiments and fails to improve on the baseline. Unlike many of the other metrics investigated in the baseline performance, there is no tuning of the RAPr metrics for spam ranking. If we combined RAPr and TrustRank, for example, it may be possible to achieve even better performance.

SUMMARY

By incorporating information from multiple random surfers simultaneously, the RAPr model increases the flexibility of PageRank models considerably. We present theoretical results showing that it generalizes the properties of the PageRank vector. Whereas PageRank contains an oversight when applied to real-world surfer data, we show that web browsing logs contain the information to compute the multi-surfer distribution for RAPr. These logs show that users follow links with probability 0.375.

Next, we derive three algorithms to compute the expectation and standard deviation for the RAPr setup. Two of these algorithms just use PageRank solutions at multiple values of α. We present both theoretical and empirical error analysis for each algorithm. Thus, computing these quantities is not a problem.

Finally, our analysis of the expectation and standard deviation shows that the expectation is closely aligned to PageRank, but the standard deviation is not. This holds both for web search networks and gene identification networks. The RAPr statistics also improve a spam classification task.

Every block of stone has a statue inside it and it is the task of the sculptor to discover it.

—Michelangelo

5 AN INNER-OUTER ITERATION FOR PAGERANK

In the previous two chapters, we saw that computing both derivative of PageRank and the statistics of the RAPr model involve only solving PageRank problems. In this chapter, we develop an inner-outer iteration that solves PageRank as a series of PageRank problems with smaller values of α.

Recall that PageRank as a linear system is the vector $\mathbf{x}$ that satisfies

$$(\mathbf{I} - \alpha\mathbf{P})\mathbf{x} = (1 - \alpha)\mathbf{v}, \tag{5.1}$$

or equivalently

$$\mathbf{x} = \alpha\mathbf{P}\mathbf{x} + (1 - \alpha)\mathbf{v}. \tag{5.2}$$

In the Richardson iteration for PageRank, we convert this equation into a stationary iterative method by taking the previous left-hand side as the new iterate, that is

$$\mathbf{x}^{(k+1)} = \alpha\mathbf{P}\mathbf{x}^{(k)} + (1 - \alpha)\mathbf{v}. \tag{5.3}$$

Well known spectral and convergence properties of the PageRank problem show that it is *easier*[1] when α is closer to 0. Inspired by these results, we consider a stationary iteration given by the splitting

$$(\mathbf{I} - \beta\mathbf{P})\mathbf{x}^{(k+1)} = \underbrace{(\alpha - \beta)\mathbf{P}\mathbf{x}^{(k)} + (1 - \alpha)\mathbf{v}}_{\equiv\mathbf{f}^{(k)}} \tag{5.4}$$

with $0 < \beta < \alpha$. This expression defines the *outer iteration* in our new scheme and corresponds to a PageRank problem with β instead of α and a different right-hand side.[2]

To apply these iterations in practice, we use a Richardson iteration to solve the inner system

$$(\mathbf{I} - \beta\mathbf{P})\mathbf{x}^{(k+1)} = \mathbf{f}^{(k)} \tag{5.5}$$

[1] Easier in the sense that the condition number is smaller and the scheme in (5.3) converges linearly with rate α.

[2] Formally, it is the PageRank problem $(\mathbf{I} - \beta\mathbf{P})\mathbf{x}^{(k+1)} = (1 - \beta)\mathbf{z}$ for $\mathbf{z} = \frac{\alpha-\beta}{1-\beta}\mathbf{P}\mathbf{x}^{(k)} + \frac{1-\alpha}{1-\beta}\mathbf{v}$. Because $\mathbf{e}^T\mathbf{z} = 1$ and $v_i \geq 0$, it is a genuine PageRank problem.

yielding the *inner iteration*,

$$\mathbf{y}^{(0)} \equiv \mathbf{x}^{(k)}; \quad \mathbf{y}^{(j+1)} = \beta \mathbf{P} \mathbf{y}^{(j)} + \mathbf{f}^{(k)} \; j = 1, \ldots, \ell; \quad \mathbf{x}^{(k+1)} \equiv \mathbf{y}^{(\ell)}. \tag{5.6}$$

We discuss how to terminate this iteration below, and show that keeping ℓ small will accelerate convergence.

The above scheme was initially proposed by Gray et al. [2007] in a technical report. Subsequently, we have developed a new convergence analysis showing that the iteration always converges, a large-scale multi-core parallel implementation, a Gauss-Seidel variant, and an "inner-outer" preconditioner for PageRank with the BiCG-STAB solver [van der Vorst, 1992]. These additions are reported in Gleich et al. [to appear].

The goal of our inner-outer idea is a *low-memory*, *matrix-free*,[3] and *parameter-free* scheme to compute PageRank faster than the power method. Although we introduce the parameter β in the definition of the method, we show analytically and experimentally that $\beta = 0.5$ is an effective choice. The result is a PageRank scheme that takes merely three vectors of memory and consistently outperforms the power method by a substantial margin in time and matrix-vector products (see table 5.2, figure 1.4). Because the method is matrix-free, it is easy to parallelize, provided a parallel matrix-vector product exists. We present experiments demonstrating the parallel performance of the inner-outer algorithm on matrices with over 100,000,000 rows and 3.7 billion non-zeros. Similar ideas give both an inner-outer Gauss-Seidel method and a matrix-free preconditioner for the BiCG-STAB iteration. These contributions are discussed below, as well as a non-web application of PageRank with the IsoRank algorithm [Singh et al., 2007].

5.1 EXISTING PAGERANK ALGORITHMS

Improving the computation of PageRank is not a new problem, though we are not aware of any other PageRank specific method that simultaneously satisfies all three criteria of our inner-outer algorithm: *low-memory*, *matrix-free*, and *parameter-free*. Our algorithm converges much faster than the power method, which is the only previous algorithm that has these three properties. We briefly summarize the existing literature. When looking at the linear system from (5.1), Arasu et al. [2002] investigated the Gauss-Seidel method, which is not matrix-free. Later, McSherry [2005] suggested a modification of the Richardson iteration that exploits matrix structure. Some of the first improvements on the eigenvector problem were acceleration and

[3] A matrix-free method relies only on the availability of a matrix-vector product, and not the matrix itself.

extrapolation techniques [Kamvar et al., 2003, 2004], which are sensitive to the choice of parameter and often take considerable memory—approximately 6 vectors. Langville and Meyer [2006b] describe a method to update the stationary distribution of a Markov chain that depends on having the matrix structure in-hand. Both Gleich et al. [2004] and Del Corso et al. [2007] look at Krylov subspace methods for the linear system. While these methods can be matrix-free, both studies find preconditioners are often required for convergence. Golub and Greif [2006] propose a specialized Arnoldi method that is matrix-free, but not low-memory.

Another class of methods exploit properties of the graph structure to reduce the work involved in the computation [Eiron et al., 2004; Ipsen and Selee, 2007; Del Corso et al., 2005; Boldi and Vigna, 2004, 2005; Karande et al., 2009; Lin et al., 2009]. Because our method is matrix-free, for some of these methods we can integrate our inner-outer scheme with them to increase their effectiveness even further.

5.2 ALGORITHMS

We always start our algorithms with $\mathbf{x}^{(0)} = \mathbf{v}$, although other starting conditions are possible. To terminate the iterations, we use the 1-norm of the residuals of the outer system (5.1) and the inner system (5.5) as stopping criteria. For the outer iteration (5.4) we require[4]

$$\left\|(1-\alpha)\mathbf{v} - (\mathbf{I} - \alpha\mathbf{P})\mathbf{x}^{(k+1)}\right\| < \tau,$$

and for the inner iteration (5.6) we require

$$\left\|\mathbf{f} - (\mathbf{I} - \beta\mathbf{P})\mathbf{y}^{(j+1)}\right\| < \eta.$$

The resulting *inner-outer iteration*, based on the iterative formulas given in (5.4) and (5.6), is presented in algorithm 2. Lines 1 and 2 of algorithm 2 initialize $\mathbf{x} = \mathbf{v}$ and $\mathbf{y} = \mathbf{P}\mathbf{x}$. For the purpose of illustrating how the computation can be efficiently done, the roles of $\mathbf{x}$ and $\mathbf{y}$ are altered from the notation used in the text. Later, we show that $\beta = 0.5$ and $\eta = 10^{-2}$ are effective choices of these parameters for all graphs (assuming $\alpha \geq 0.85$).

The damping parameter α is assumed to be given as part of the model, and τ is a value typically provided by the user. *Thus, the challenge is to determine values of β and η that will accelerate the computation.*

[4] This quantity is both the residual of the linear system for PageRank and the change after a single power iteration.

Algorithm 2 – The basic inner-outer iteration.

```
Input: $\mathbf{P}, \mathbf{v}, \alpha, \tau, (\beta = 0.5, \eta = 10^{-2})$
Output: $\mathbf{x}$
$\mathbf{x} \leftarrow \mathbf{v}$
$\mathbf{y} \leftarrow \mathbf{P}\mathbf{x}$
while $\|\alpha\mathbf{y} + (1-\alpha)\mathbf{v} - \mathbf{x}\| \geq \tau$
    $\mathbf{f} \leftarrow (\alpha - \beta)\mathbf{y} + (1-\alpha)\mathbf{v}$
    repeat
        $\mathbf{x} \leftarrow \mathbf{f} + \beta\mathbf{y}$
        $\mathbf{y} \leftarrow \mathbf{P}\mathbf{x}$
    until $\|\mathbf{f} + \beta\mathbf{y} - \mathbf{x}\| < \eta$
end while
$\mathbf{x} \leftarrow \alpha\mathbf{y} + (1-\alpha)\mathbf{v}$
```

5.3 ALGORITHM DISCUSSION

Algorithm 2 has a number of properties worth noting. Consider the algorithm with $\beta = 0$. The inner loop will always exit after a single iteration because $\mathbf{x}$ is set to $\mathbf{f}$. In this case, $\mathbf{f} = \alpha\mathbf{y} + (1-\alpha)\mathbf{v}$, but $\mathbf{y} = \mathbf{P}\mathbf{x}$ from the previous iteration. Thus, the inner-outer iteration with $\beta = 0$ is *just the power method*!

Now, consider the case $\beta = \alpha$. We intuitively expect that this method reproduces the power method because the inner iteration with $\beta = \alpha$ is solved with a Richardson iteration, which is the same as the power method.[5] Here $\mathbf{f} = (1-\alpha)\mathbf{v}$ for every inner iteration and thus every inner iteration corresponds to a step of the power method. When $\|\mathbf{f} - \beta\mathbf{y} - \mathbf{x}\| < \eta$, we will do only one inner iteration for each outer iteration, but again, each step is just a step of the power method.

Thus, we expect no acceleration with $\beta = 0$ or $\beta = \alpha$.

We make one final observation about the basic inner-outer method. For each outer iteration, the first inner iteration is always a step of the power method. We use this fact repeatedly in the remainder of the chapter.

[5] Check section 2.4.2 for the formal equivalence.

5.4 CONVERGENCE

In the previous section, we showed that the algorithm corresponds to the power method when $\beta = 0$ or $\beta = \alpha$. Convergence of the power method implies that the inner-outer iteration also converges with these parameters. In this section, we analyze the convergence for general $0 < \beta < \alpha$.

We present the convergence analysis in two parts. First, we show that the outer iteration is a convergent scheme for the PageRank problem. Then we show that the outer scheme with an inner Richardson scheme will always converge on the PageRank problem.

Lemma 15. *Given* $0 < \alpha < 1$, *if the inner iterations are solved exactly, the scheme converges for any* $0 < \beta < \alpha$. *Furthermore,*

$$\left\|\mathbf{x}^{(k+1)} - \mathbf{x}\right\| \leq \frac{\alpha - \beta}{1 - \beta} \left\|\mathbf{x}^{(k)} - \mathbf{x}\right\|,$$

where $\frac{\alpha-\beta}{1-\beta}$ *indicates that the closer* β *is to* α, *the faster the outer iterations converge.*

Proof. Let $\mathbf{x}$ be the PageRank vector and $\mathbf{x}^{(k)}$ be the current iterate. The next iterate is

$$\mathbf{x}^{(k+1)} = (\mathbf{I} - \beta\mathbf{P})^{-1}((\alpha - \beta)\mathbf{P}\mathbf{x}^{(k)} + (1 - \alpha)\mathbf{v}), \tag{5.7}$$

and the solution is

$$\mathbf{x} = (\mathbf{I} - \beta\mathbf{P})^{-1}((\alpha - \beta)\mathbf{P}\mathbf{x} + (1 - \alpha)\mathbf{v}). \tag{5.8}$$

Subtracting these two expressions gives

$$\mathbf{x} - \mathbf{x}^{(k+1)} = (\alpha - \beta)(\mathbf{I} - \beta\mathbf{P})^{-1}\mathbf{P}(\mathbf{x} - \mathbf{x}^{(k)}). \tag{5.9}$$

The result now follows from applying 1-norms to both sides and using

$$\left\|(\mathbf{I} - \beta\mathbf{P})^{-1}\mathbf{P}\right\| \leq \frac{1}{1 - \beta}, \tag{5.10}$$

which comes from $\left\|(\mathbf{I} - \beta\mathbf{P})^{-1}\mathbf{P}\right\| \leq \left\|\sum_{\ell=0}^{\infty}(\beta\mathbf{P})^{\ell}\right\| \leq \frac{1}{1-\beta}$. □

The following argument summarizes the convergence theory for inexact inner iterations. First, we provide an expression for the error in the linear system during the inner iterations. Then we derive a monotonic bound on the error during the outer iterations.

Let

$$\mathbf{g}^{(j)} = \mathbf{y}^{(j)} - \mathbf{x}$$

be the error after the jth inner iteration.

Lemma 16. *In the inner-outer iteration with* $0 < \alpha < 1$ *and* $\beta < \alpha$*, when the inner iterations are solved using a Richardson iteration, after j inner iterations,*

$$\mathbf{g}^{(j)} = \mathbf{y}^{(j)} - \mathbf{x} = \left(\alpha\beta^{j-1}\mathbf{P}^{j} + \left(\frac{\alpha-\beta}{\beta}\right)\sum_{\ell=1}^{j-1}\beta^{\ell}\mathbf{P}^{\ell}\right)\mathbf{g}^{(0)} \qquad j \geq 1,$$

where $\mathbf{g}^{(0)} = \mathbf{y}^{(0)} - \mathbf{x} = \mathbf{x}^{(k)} - \mathbf{x}$ *is the error after the kth outer iteration.*

Proof. We proceed by induction. In the base case,

$$\mathbf{g}^{(1)} = \alpha\mathbf{P}\mathbf{g}^{(0)}$$

follows because one inner iteration is identical to an iteration of the power method.

Consider $\mathbf{g}^{(j+1)}$. First, expand

$$\begin{aligned}\mathbf{g}^{(j+1)} &= \beta\mathbf{P}\mathbf{y}^{(j)} + (\alpha-\beta)\mathbf{P}\mathbf{x}^{(k)} + (1-\alpha)\mathbf{v} - \mathbf{x}\\ &= \beta\mathbf{P}\mathbf{g}^{(j)} + (\alpha-\beta)\mathbf{P}\mathbf{g}^{(0)}.\end{aligned}$$

Now apply the induction hypothesis and simplify:

$$\begin{aligned}\mathbf{g}^{(j+1)} &= \beta\mathbf{P}\left(\left(\alpha\beta^{j-1}\mathbf{P}^{j} + \frac{\alpha-\beta}{\beta}\right)\sum_{\ell=1}^{j-1}\beta^{\ell}\mathbf{P}^{\ell}\right)\mathbf{g}^{(0)} + (\alpha-\beta)\mathbf{P}\mathbf{g}^{(0)}\\ &= \left(\alpha\beta^{j}\mathbf{P}^{j+1} + (\alpha-\beta)\mathbf{P} + (\alpha-\beta)\sum_{\ell=1}^{j-1}\beta^{\ell}\mathbf{P}^{\ell+1}\right)\mathbf{g}^{(0)}\\ &= \left(\alpha\beta^{j}\mathbf{P}^{j+1} + \left(\frac{\alpha-\beta}{\beta}\right)\sum_{\ell=1}^{j}\beta^{\ell}\mathbf{P}^{\ell}\right)\mathbf{g}^{(0)}.\end{aligned}$$

□

We know that $\left\|\mathbf{g}^{(1)}\right\| \leq \alpha \left\|\mathbf{g}^{(0)}\right\|$ because it's just a step of the power method, and the above equation says the same thing. The next iterate satisfies

$$\begin{aligned}
\left\|\mathbf{g}^{(2)}\right\| &= \left\|\alpha\beta\mathbf{P}^2\mathbf{g}^{(0)} + (\alpha-\beta)\mathbf{P}\mathbf{g}^{(0)}\right\| && (5.11)\\
&\leq \alpha\beta\left\|\mathbf{g}^{(0)}\right\| + (\alpha-\beta)\left\|\mathbf{g}^{(0)}\right\| && (5.12)\\
&= ((\alpha-1)\beta+\alpha)\left\|\mathbf{g}^{(0)}\right\| && (5.13)\\
&\leq \alpha\left\|\mathbf{g}^{(0)}\right\|, && (5.14)
\end{aligned}$$

and so we monotonically decrease a bound on the error for the first two iterations.

For any iterate, we have

$$\begin{aligned}
\left\|\mathbf{g}^{(j)}\right\| &= \left\|\alpha\beta^{j-1}\mathbf{P}^j\mathbf{g}^{(0)} + \left(\frac{\alpha-\beta}{\beta}\right)\sum_{\ell=1}^{j-1}\beta^\ell\mathbf{P}^\ell\mathbf{g}^{(0)}\right\| && (5.15)\\
&\leq \left\|\alpha\beta^{j-1}\mathbf{P}^j\mathbf{g}^{(0)}\right\| + \sum_{\ell=1}^{j-1}\left\|\frac{\alpha-\beta}{\beta}\beta^\ell\mathbf{P}^\ell\mathbf{g}^{(0)}\right\| && (5.16)\\
&\leq \alpha\beta^{j-1}\left\|\mathbf{g}^{(0)}\right\| + \sum_{\ell=1}^{j-1}\frac{\alpha-\beta}{\beta}\beta^\ell\left\|\mathbf{g}^{(0)}\right\| && (5.17)\\
&= \underbrace{\left(\alpha\beta^{j-1} + \left(\frac{\alpha-\beta}{\beta}\right)\sum_{\ell=1}^{j-1}\beta^\ell\right)}_{\equiv\kappa_j}\left\|\mathbf{g}^{(0)}\right\|. && (5.18)
\end{aligned}$$

We can rearrange κ_j to a more meaningful form using

$$\sum_{\ell=1}^{j-1}\beta^\ell = \beta\frac{1-\beta^{j-1}}{1-\beta}, \qquad (5.19)$$

which, after substitution into (5.18), yields

$$\begin{aligned}
\kappa_j &= \alpha\beta^{j-1} + \frac{(\alpha-\beta)(1-\beta^{j-1})}{1-\beta} && (5.20)\\
&= \frac{(\alpha-\beta)+(1-\alpha)\beta^j}{1-\beta}. && (5.21)
\end{aligned}$$

This bound is monotonically converging to $\frac{\alpha-\beta}{1-\beta}$ and we further have

$$\kappa_j \leq \alpha \qquad \text{for } j \geq 1. \qquad (5.22)$$

From this analysis, we cannot conclude that the error in the inner iteration is monotonically decreasing. It is, however, *bounded* by a monotonically decreasing function. At every outer iteration then, the inner Richardson iteration decreases the error by *at least* α. Thus, our formulation of the inner-outer algorithm will always converge regardless of the tolerance η, so long as we always do at least one inner-iteration. A mathematically precise statement of this argument follows.

Theorem 17. *Given* $0 < \alpha < 1$ *and* $0 < \beta < \alpha$, *if an inner-iteration is solved with j steps of a Richardson method then*

$$\left\|\mathbf{x}^{(k+1)} - \mathbf{x}\right\| \le \kappa_j \left\|\mathbf{x}^{(k)} - \mathbf{x}\right\| ,$$

with $\kappa_j = \frac{(\alpha-\beta)+(1-\alpha)\beta^j}{1-\beta}$. *Furthermore, if* $j \ge 1$ *for all inner iterations, then* $\kappa_j \le \alpha$ *and the inner-outer iteration always converges.*

This theorem places no restrictions on η and only requires that we perform at least one step of the inner iteration for each outer iteration. Algorithm 2 guarantees this property because the stopping condition is not checked until after the first iteration. Thus, that algorithm always converges on the PageRank problem.

5.5 EXTENSIONS

In this section, we derive a few improvements to algorithm 2.

5.5.1 *An inner-outer accelerated power method*

After a few outer iterations, the inner iterations of algorithm 2 converge rapidly. As previously mentioned, a single inner iteration is equivalent to a single step of the power method. Thus we can incorporate the following improvement to the power method: apply the inner-outer scheme, and once the inner iterations start converging quickly, switch back to the power method. After the switch, we eliminate the need to check the stopping criterion of the inner iteration and save touching one vector of memory. The change to algorithm 2 is simple. Add a line between 8-9 to check if the inner-iteration converged in less than i_m iterations, and if so, switch to the power method using the current iterate as the starting vector.

The modified scheme is presented in algorithm 3. The value of i_m is small and determines the point, in terms of inner iteration count, where we switch to the power method. If $i_m = 1$ then switching to the power method saves the need to compute **f** and check the stopping criterion in line 8 of algorithm 2. This optimization saves touching an extra vector in memory and a 1-norm computation. In our numerical experiments we adopt this version of the accelerated algorithm, i.e. we take $i_m = 1$. The "`power`$(\alpha\mathbf{y} + (1-\alpha)\mathbf{v})$" clause in line 9 means apply the power method with $\alpha\mathbf{y} + (1-\alpha)\mathbf{v}$ as an initial guess, until convergence.

Algorithm 3 – Inner-Outer power iterations.

```
Input: $\mathbf{P}, \alpha, \beta, \tau, \eta, \mathbf{v}, i_m$
Output: $\mathbf{x}$
1: $\mathbf{x} \leftarrow \mathbf{v}$
2: $\mathbf{y} \leftarrow \mathbf{P}\mathbf{x}$
3: while $\|\alpha\mathbf{y} + (1-\alpha)\mathbf{v} - \mathbf{x}\|_1 \geq \tau$
4:      $\mathbf{f} \leftarrow (\alpha - \beta)\mathbf{y} + (1-\alpha)\mathbf{v}$
5:      for $i = 1, \ldots$   repeat
6:          $\mathbf{x} \leftarrow \mathbf{f} + \beta\mathbf{y}$
7:          $\mathbf{y} \leftarrow \mathbf{P}\mathbf{x}$
8:      until $\|\mathbf{f} + \beta\mathbf{y} - \mathbf{x}\|_1 < \eta$
9:      if $i \leq i_m$, $\mathbf{x}$=power$(\alpha\mathbf{y} + (1-\alpha)\mathbf{v})$; return
10: end while
11: $\mathbf{x} \leftarrow \alpha\mathbf{y} + (1-\alpha)\mathbf{v}$
```

When $i_m = 1$, then algorithm 3 and algorithm 2 produce exactly the same iterates. Once the inner iteration of algorithm 2 converges in a single iteration, then it will always converge in a single iteration. We hope we aren't belaboring the point by reiterating that a single iteration of the inner iteration is precisely the power method.

Program 8 – The inner-outer iteration for PageRank. The input to the inner-outer code is `a` $= \alpha$, `v` $= \mathbf{v}$, `P` $= \mathbf{P}^T$, `tol` $= \varepsilon$, and `maxit`, an upper bound on the number of iterations. Our PageRank solvers are quite compact but work with $\mathbf{P}^T$ for performance reasons. On MATLAB R2007a and R2007b, the code uses only three vectors of storage.

```
function [x,flag,reshist]=inoutpr(P,a,v,tol,maxit)
b=0.5*(a≥0.6); itol=1e-2; n=size(P,1);
x=zeros(n,1)+v; y=P'*x;y=y+(sum(x)-sum(y))*v; nm=1; f=a*y;f=f+(1-a)*v;f=f-x;
dlta=norm(f,1);x=x-b*y;reshist=[dlta;zeros(maxit-1,1)];
while nm<maxit && dlta>tol
    f=(a-b)*y; f=f+(1-a)*v; x=x-f; % f=(a-b)*y+vt;x=x-b*y-f;
    d = norm(x,1); ii=0;
    while nm+ii<maxit && d>itol,
        x=b*y; x=x+f; y=P'*x; y=y+(sum(x)-sum(y))*v;ii=ii+1; % x=b*y+f;y=Pd'*x
        x=x-b*y; x=x-f; d=norm(x,1); end  % x=x-b*y-f;
    if ii<2, x=a*y; x=x+(1-a)*v; y=y-x; break; end % no mult => no hist updte
    x=x+f; f=a*y; f=f+(1-a)*v; f=f-x; f=f-b*y; dlta=norm(f,1); % ||a*y+(1-a)*v-x||_1
    reshist(nm+1:nm+ii)=dlta; nm=nm+ii; end
while nm<maxit && dlta>tol % run the power method until convergence
    y=x./norm(x,1); x=a*(P'*y);  w=1-sum(x); x=x+w*v;
    y=y-x; dlta=norm(y,1); nm=nm+1; reshist(nm)=dlta; end
x=x./norm(x,1); flag=dlta>tol; reshist=reshist(1:nm);
```

5.5.2 *Inner-Outer Gauss-Seidel iterations*

The performance of Gauss-Seidel applied to the PageRank linear system $(\mathbf{I} - \alpha\mathbf{P})\mathbf{x} = (1 - \alpha)\mathbf{v}$ is considered excellent, given its modest memory requirements. It often converges in roughly half the number of power method iterations. However, from a practical point of view, two pitfalls of the method are that it requires the matrix $\mathbf{P}$ by rows (i.e. the graph by in-edges) and it does not parallelize well.

We can accelerate Gauss-Seidel using inner-outer iterations, as follows. Our Gauss-Seidel codes implement the dangling correction to the matrix $\bar{\mathbf{P}}$ implicitly and match those in the *law* toolkit [Vigna et al., 2008]. To convert the inner-outer method to use Gauss-Seidel, we replace the inner Richardson iteration (5.6) with a Gauss-Seidel iteration. In our pseudocode, presented in algorithm 4, the `gssweep`$(\mathbf{x}, \mathbf{A}, \mathbf{b})$ function implements $\mathbf{x}^{(k+1)} = \mathbf{M}_{GS}^{-1}(\mathbf{N}_{GS}\mathbf{x}^{(k)} + \mathbf{b})$ in-place for a Gauss-Seidel splitting of the matrix $\mathbf{A}$.[6] The `gauss-seidel-pr` function in line 7 of the algorithm refers to the standard “Gauss-Seidel for PageRank” scheme described in section 2.4.3 starting with the current $\mathbf{x}$. This algorithm only requires two vectors of storage because $\mathbf{x}$ is always updated in-place. Details on the performance of this algorithm are provided in section 5.6.

Algorithm 4 – The inner-outer/Gauss-Seidel iteration.

```
Input: $\mathbf{P}, \alpha, \beta, \tau, \eta, \mathbf{v}$
Output: $\mathbf{x}$
1: $\mathbf{x} \leftarrow \mathbf{v}$
2: $\mathbf{y} \leftarrow \mathbf{P}\mathbf{x}$
3: while $\|\alpha\mathbf{y} + (1 - \alpha)\mathbf{v} - \mathbf{x}\|_1 \geq \tau$
4:     $\mathbf{f} \leftarrow (\alpha - \beta)\mathbf{y} + (1 - \alpha)\mathbf{v}$
5:     for $i = 1, 2, \ldots$
6:         $\mathbf{x}, \delta \leftarrow$ gssweep($\mathbf{x}, \mathbf{I} - \beta P, \mathbf{f}$) {$\delta = \|\mathbf{x}^{(i+1)} - \mathbf{x}^{(i)}\|_1$}
7:     until $\delta < \eta$
8:     if i=1, gauss-seidel-pr($\mathbf{x}, \mathbf{I} - \alpha P, (1 - \alpha)\mathbf{v}$); break
9:     $\mathbf{y} \leftarrow P\mathbf{x}$
10: end repeat
```

5.5.3 *Preconditioning for non-stationary schemes*

Thus far we have examined the inner-outer algorithm as a stationary iteration. In this section we switch our viewpoint and examine it as a preconditioner for a non-stationary iteration. As such, we will be mainly interested in how well the eigenvalues of the preconditioned matrix are clustered.

[6] This function is explicitly described in section 2.4.3.

Consider an approximation of $(\mathbf{I} - \beta\bar{\mathbf{P}})^{-1}$ as a preconditioner for the PageRank linear system $(\mathbf{I} - \alpha\bar{\mathbf{P}})$. Gleich et al. [2004] and Del Corso et al. [2007] examined the behavior of Krylov subspace methods on the system

$$(\mathbf{I} - \alpha\bar{\mathbf{P}})\mathbf{y} = (1 - \alpha)\mathbf{v}, \tag{5.23}$$

and concluded that, as expected in general, preconditioning is essential for the linear system formulation of the PageRank problem. Their preconditioners were incomplete factorizations or factorizations of diagonal blocks of the matrix, both of which modify the matrix data structure.

System (5.23) is different from system (5.1) because we use $\bar{\mathbf{P}}$ instead of $\mathbf{P}$. It corresponds to using PseudoRank instead of PageRank, but the solutions of the two systems are proportional: $\mathbf{x} = \mathbf{y}/\|\mathbf{y}\|_1$.[7]

We switch to the alternate formulation with $\bar{\mathbf{P}}$ for two primary reasons: it converged more often in our experience (we have no insight into why this occurs); and the behavior of Krylov methods has been empirically studied more often on this formulation of the system.

For β near α, the preconditioned system is as difficult to solve as the original linear system. We thus consider a Neumann series approximation, which is practically equivalent to a Richardson approach as in the basic inner-outer iteration:

$$(\mathbf{I} - \beta\bar{\mathbf{P}})^{-1} \approx \mathbf{I} + \beta\bar{\mathbf{P}} + (\beta\bar{\mathbf{P}})^2 + \cdots + (\beta\bar{\mathbf{P}})^m.$$

Since $\beta < 1$ and $\|\bar{\mathbf{P}}\|_1 \leq 1$, this approximation converges as $m \to \infty$, and it gives rise to an implementation that only uses matrix-vector multiplies with $\bar{\mathbf{P}}$ and avoids costly (or even impossible) modification of the matrix structure. Details on the performance of this approach are provided in section 5.6.

[7] See section 2.2.1 for the definition and equivalence between Pseudo-Rank and PageRank. Langville and Meyer [2006a, Theorem 7.2.1] also include a proof. The gist is that $\mathbf{P} = \bar{\mathbf{P}} + \mathbf{v}\mathbf{d}^T$ and so $(\mathbf{I} - \alpha\mathbf{P})\mathbf{x} = (\mathbf{I} - \alpha\bar{\mathbf{P}})\mathbf{x} + \gamma\mathbf{v} = (1 - \alpha)\mathbf{v}$. We can move $\gamma\mathbf{v}$ to the right-hand side and compute $\mathbf{y}$ with an unknown scale.

5.6 NUMERICAL RESULTS

Using the data enumerated in table 2.2, we evaluated these implementations in a wide range of experimental situations. The initial guess and teleportation distribution for every method is the uniform distribution, $\mathbf{x}^{(0)} = \mathbf{v} = (1/n)\mathbf{e}$, where $\mathbf{e}$ is a vector of all ones. We use this specific choice because it is commonly used. When we state a speedup, we use relative percentage gain

$$\frac{v_r - v_t}{v_r} \cdot 100\% \,, \tag{5.24}$$

where v_r is a reference performance and v_t is a test performance. The reference performance is either the power method or the Gauss-Seidel method. All solution vectors satisfy the residual tolerance

$$\left\| \alpha \mathbf{P}\mathbf{x}^{(k)} + (1-\alpha)\mathbf{v} - \mathbf{x}^{(k)} \right\| < \tau,$$

where τ is specified in the experiment description. Thus we are able to compare all methods directly in terms of total work and time. Parallel, large-scale (arabic-2005, sk-2005, and uk-2007), and C++ tests were run on an 8-core (4 dual-core chips) 2.8 GHz Opteron 8220 computer with 128 GB of RAM. MATLAB mex tests were run on a 4-core (2 dual-core chips) 3.0 GHz Intel Xeon 5160 computer with 16 GB of RAM.

Our codes are implemented in MATLAB and C++. We implement both single-core and multi-core algorithms in C++ using the BVGraph data structure for compressed web graphs [Boldi and Vigna, 2005].

5.6.1 *Inner-Outer parameters*

We first show the behavior of the inner-outer method for a range of graphs and parameters in figure 5.1. For β between 0 and 0.85 and $\eta = 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}$, we plot the number of matrix-vector multiplications (equivalent to the number of iterations) until the inner-outer iteration converged. First, note that all instances converged and thus there is no numerical violation of the convergence theory for inner-outer iteration (theorem 17). Next, when $\alpha = 0.85$, the inner-outer iteration with $\beta = 0.5$ and $\eta = 10^{-2}$ uses fewer multiplications (than the power method) on the in-2004 graph, but not on the www graph. When $\alpha = 0.99$, with these same β and η parameters, the inner-outer iteration uses roughly 30% fewer multiplications. Thus, the plots demonstrate that $\beta = 0.5$ and $\eta = 10^{-2}$ are effective choices. Based on these same plots, picking $\eta = 10^{-1}$ and $\beta = 0.5$ also seems a good choice. The

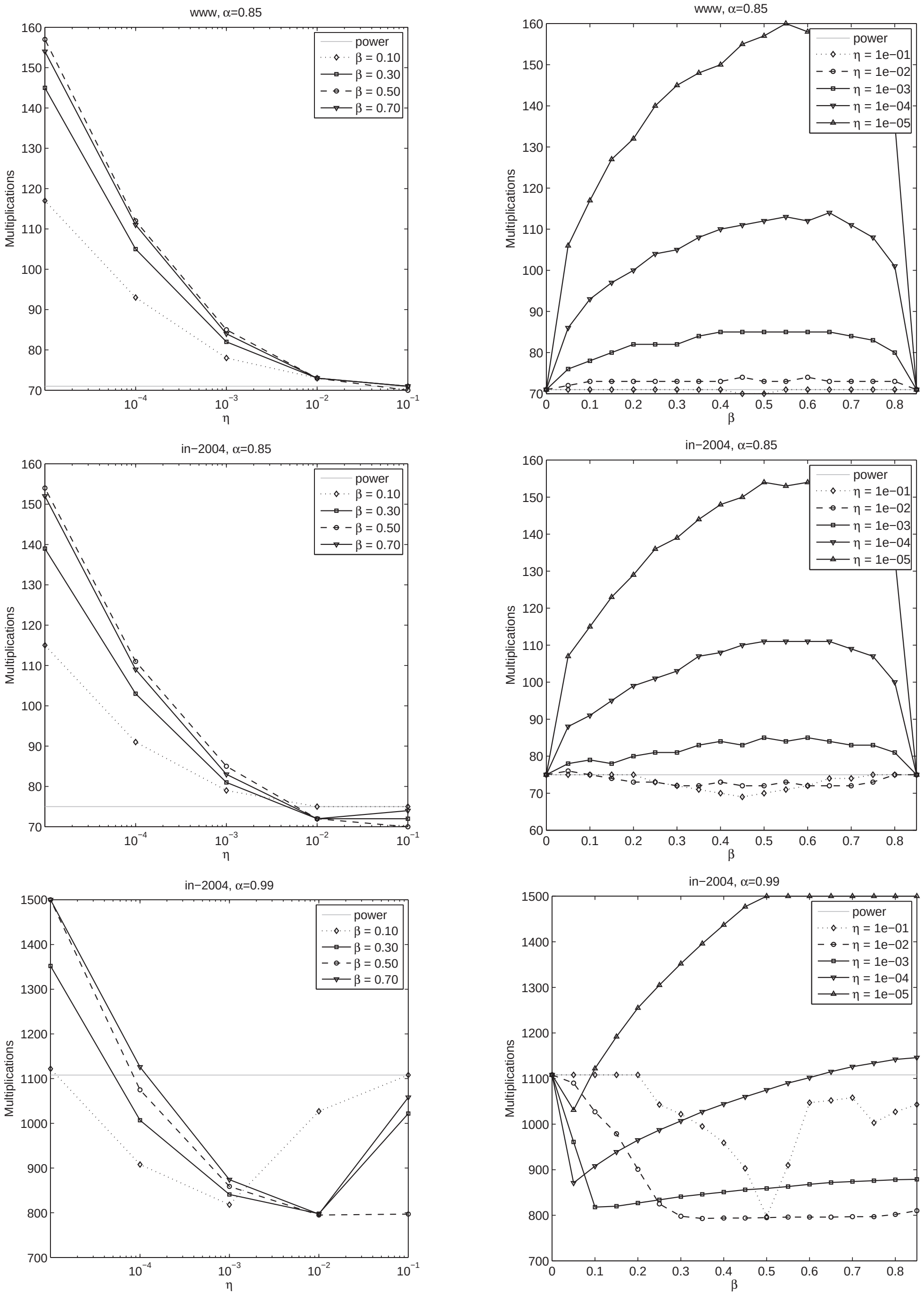


Figure 5.1 – Performance of the inner-outer iteration with varied β and η. We measure performance with matrix-vector products in this case. For the **www** and **in-2004** graphs, we fix β and vary η in the left figures, and we fix η and vary β in the right figures. The top rows use $\alpha = 0.85$ and the bottom row uses $\alpha = 0.99$. This plot suggests that $\beta = 0.5$ and $\eta = 10^{-2}$ is a good choice. The only graph that the inner-outer scheme does not accelerate with these parameters is the **www** graph with $\alpha = 0.85$.

results from **in-2004** with $\alpha = 0.99$ show that using $\eta = 10^{-1}$ may be sensitive to the value of β, whereas using $\eta = 10^{-2}$ reduces this sensitivity for a slight decrease in performance on the other graphs. Because our goal is robustness, we prefer the latter.

In the next investigation, we show the convergence of the iterates for the **wb-edu** graph in figure 5.2. For both values of α shown in the plots ($\alpha = 0.85$ and $\alpha = 0.99$), the inner-outer iteration uses fewer multiplications. The advantage is derived from accelerated convergence in all of the iterations. Recall that the inner-outer iteration *switches* back to running iterations of the power method after the inner-outer iterations converge instantly. For these cases, that occurred after roughly 10 and 20 outer iterations, respectively. Thus, this observation indicates that the convergence rate of the power method changes when used after the inner-outer iteration. We return to this point in section 5.6.6.

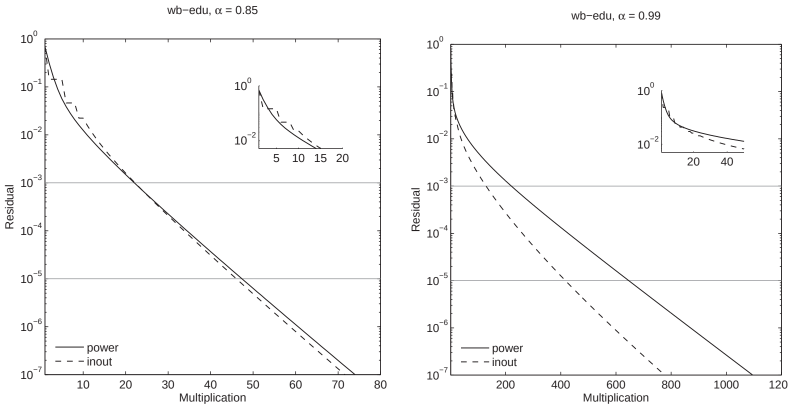


Figure 5.2 – Performance of the PageRank algorithm. The inner-outer method clearly outperforms the power method on **wb-edu** with $\alpha = 0.85$ on the left and $\alpha = 0.99$ on the right. The small inner figure shows the convergence in the first few iterations.

Table 5.1 shows that the inner iteration counts per outer iteration decrease monotonically down to a single iteration quite quickly. From the table we can see that it takes 24 inner iterations overall (within 9 outer iterations) until the inner iterates start converging immediately, at which point we switch to the power method.

	inner iterations for outer iteration									
graph	1	2	3	4	5	6	7	8	9	10
wb-edu	4	3	3	3	2	2	2	2	2	1
eu-2005	4	4	3	3	2	2	2	2	2	1

Table 5.1 – Inner iteration counts. Shown are the number of inner iterations for each outer iteration when $\alpha = 0.99$, $\beta = 0.5$, and $\eta = 10^{-2}$. Total iteration counts and CPU times for this example can be found in table 5.2.

Finally, we evaluate work and time for the inner-outer iteration on many graphs in table 5.2. The inner-outer scheme with $\beta = 0.5$ and $\eta = 10^{-2}$ is almost always faster in these experiments. From the wall-clock times reported, we can see that there is not any significant overhead. For loose outer tolerances (10^{-3}), the inner-outer iteration uses 45% fewer iterations than the power method and for tight outer tolerances (10^{-7}), it almost 30% fewer iterations. The results are consistently faster and involve no parameter search.

Table 5.2 – Performance of the inner-outer iteration on various graphs. Total number of matrix-vector products and wall-clock time required for convergence to three different outer tolerances τ, and the corresponding relative gains defined by (5.24). The parameters used here are $\alpha = 0.99$, $\beta = 0.5$, $\eta = 10^{-2}$. For the first five graphs we used the Matlab mex codes, and for the final three large graphs we used the C++ codes where the graph is stored by out-edges and the times refer to the performance of an 8-core parallel code.

tol.	graph	work	(mults.)		time	(secs.)	
		power	in/out	gain	power	in/out	gain
10^{-3}	ubc-cs-2006	226	141	37.6%	1.9	1.2	35.2%
	ubc	242	141	41.7%	13.6	8.3	38.4%
	in-2004	232	129	44.4%	51.1	30.4	40.5%
	eu-2005	149	150	-0.7%	26.9	28.3	-5.3%
	wb-edu	221	130	41.2%	291.2	184.6	36.6%
	arabic-2005	213	139	34.7%	779.2	502.5	35.5%
	sk-2005	156	144	7.7%	1718.2	1595.9	7.1%
	uk-2007	145	125	13.8%	2802.0	2359.3	15.8%
10^{-5}	ubc-cs-2006	574	432	24.7%	4.7	3.6	22.9%
	ubc	676	484	28.4%	37.7	27.8	26.2%
	in-2004	657	428	34.9%	144.3	97.5	32.4%
	eu-2005	499	476	4.6%	89.3	87.4	2.1%
	wb-edu	647	417	35.5%	850.6	572.0	32.8%
	arabic-2005	638	466	27.0%	2333.5	1670.0	28.4%
	sk-2005	523	460	12.0%	5729.0	5077.1	11.4%
	uk-2007	531	463	12.8%	10225.8	8661.9	15.3%
10^{-7}	ubc-cs-2006	986	815	17.3%	8.0	6.8	15.4%
	ubc	1121	856	23.6%	62.5	49.0	21.6%
	in-2004	1108	795	28.2%	243.1	179.8	26.0%
	eu-2005	896	814	9.2%	159.9	148.6	7.1%
	wb-edu	1096	777	29.1%	1442.9	1059.0	26.6%
	arabic-2005	1083	843	22.2%	3958.8	3012.9	23.9%
	sk-2005	951	828	12.9%	10393.3	9122.9	12.2%
	uk-2007	964	857	11.1%	18559.2	16016.7	13.7%

5.6.2 *Inner-Outer Gauss-Seidel*

We present our comparison for the Gauss-Seidel method in table 5.3. Rather than matrix-vector multiplications, the results are measured in sweeps through the matrix. The work for a single sweep is roughly equivalent to a single matrix-vector multiplication. For $\alpha = 0.99$, the inner-outer iteration only accelerates two of our smallest test graphs. Increasing α to 0.999 and using a strict τ shows that the inner-outer method also accelerates Gauss-Seidel-based codes.

We have not invested effort in optimizing the scheme in this case; our experiments are only intended to show that the inner-outer idea is promising in combination with other high-performance PageRank techniques. We believe that an analysis of the sort that we have performed for the Richardson iteration in the previous sections may point out a choice of parameters that could further improve convergence properties for the inner-outer scheme combined with Gauss-Seidel.

Table 5.3 – Performance of the Gauss-Seidel inner-outer iteration. Total number of Gauss-Seidel sweep iterations (equivalent in work to one matrix-vector multiply) and wall-clock time required for convergence, and the corresponding relative gains defined by (5.24). The parameters used here are $\beta = 0.5$ and $\eta = 10^{-2}$ and we used Matlab mex codes. The convergence tolerance was $\tau = 10^{-7}$.

α	graph	work	(sweeps.)		time	(secs.)	
		gs	in/out	gain	gs	in/out	gain
0.99	ubc-cs-2006	562	492	12.5%	2.9	2.7	7.0%
	ubc	566	503	11.1%	19.5	18.0	7.7%
	in-2004	473	469	0.8%	65.9	67.3	-2.2%
	eu-2005	439	462	-5.2%	56.6	60.4	-6.6%
	wb-edu	450	464	-3.1%	357.9	380.0	-6.2%
0.999	ubc-cs-2006	4430	3576	19.3%	19.8	16.8	14.9%
	ubc	4597	3646	20.7%	141.6	113.8	19.7%
	in-2004	3668	3147	14.2%	451.1	391.0	13.3%
	eu-2005	3197	3159	1.2%	354.8	352.4	0.7%
	wb-edu	3571	3139	12.1%	2532.5	2249.0	11.2%

5.6.3 *Parallel speedup*

Parallel PageRank algorithms take many forms [Gleich et al., 2004; Kollias et al., 2006; McSherry, 2005; Parreira et al., 2006]. Our implementation substitutes OpenMP shared memory operations for the linear algebra operations norm, axpy, and the matrix-vector multiply. We implemented two versions of the parallel code to manipulate the graph stored by out-edges (the natural order) or by in-edges (the Gauss-Seidel order).

Boldi and Vigna's bvgraph structure [Boldi and Vigna, 2004] efficiently iterates over the edges emanating from a vertex for a fixed ordering of the nodes. These could be either out- or in-edges, depending on how the graph is stored. To implement the parallel matrix-vector multiply for p processors, we make one pass through the file and store $p - 1$ locations in the structure that roughly divide the edges of the graph evenly between p processors. When the graph is stored by out-edges, the serial matrix-vector operation is

```
xi=x[i]/degree(i); for (j in edges of i) { y[j]+=xi; }
```

which writes to arbitrary and possibly overlapping locations in memory. In the OpenMP version, the update of $\mathbf{y}$ becomes the atomic operation

```
xi=x[i]/degree(i); for (j in edges of i) { atomic(y[j]+=xi); }.
```

When the graph is stored by in-edges, the original serial update works without modification as the processors never write to the same location in the vector $\mathbf{y}$.

We evaluated a few variants of the matrix-vector multiplication when the graph is stored by out-edges and found the atomic operation variant has similar performance to storing a separate $\mathbf{y}$ vector for each processor and aggregating the vectors at the end of the operation. Variants that replaced separate $\mathbf{y}$ vectors with separate hash tables were slower in our tests.

Figure 5.3 demonstrates the scalability of the codes when the graph is stored by out- and in-edges. In the figure, we distinguish between relative and true speedup. Relative speedup is the time on p processors compared with the time on 1 processor for the same algorithm. True speedup is the time

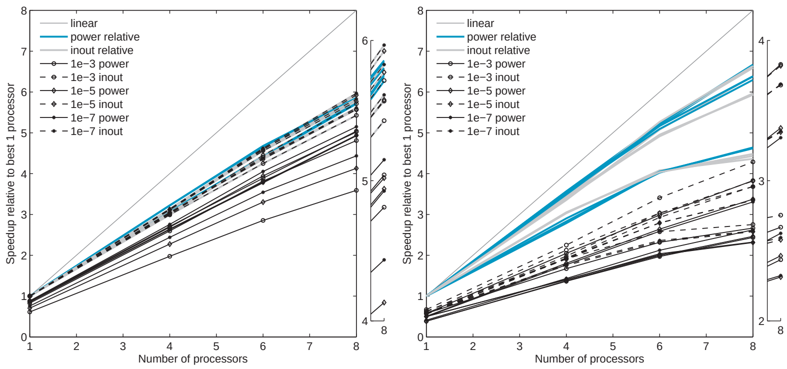


Figure 5.3 – Parallel performance of the inner-outer iteration. Parallel scalability for the three large graphs **arabic-2005**, **sk-2005**, and **uk-2007** with the matrix stored by out-edges (left) and in-edges (right). Light gray or teal lines are the relative speedup compared with the same algorithm on one processor. Black lines are the true speedup compared with the best single processor code. At the right of each sub-figure is a small enlargement of the 8 processor results. The parameters were $\alpha = 0.99, \beta = 0.5$, and $\eta = 10^{-2}$. In this plot, speedup is the ratio v_r/v_t.

on p processors compared with the best time on 1 processor. The higher relative speedup of methods based on the in-edges of the graph (6-7x) compared to out-edge methods (5-6x) demonstrates that in-edge algorithms are more scalable. In light of the atomic operation in out-edge methods, this parallelization difference is not surprising. No consistent differences appear when comparing the relative speedup of the inner-outer method and the power method. Consequently, we assume that these methods parallelize similarly and compare true speedup.

For an out-edge graph, the true speedup and relative speedup are similar. In contrast, the relative speedup for an in-edge graph is much higher than the true speedup. Gauss-Seidel causes this effect, since it is the fastest method on an in-edge graph and so the true speedup of most methods starts at around 0.5, a 50% slowdown. With one exception, the inner-outer methods (dashed lines) all demonstrate a higher true speedup than the power method.

5.6.4 *Preconditioning*

We evaluate $(\mathbf{I}-\beta\bar{\mathbf{P}})^{-1}$ or its Neumann series approximation as a preconditioner, see section 5.5.3, by examining eigenvalue clustering and matrix-vector multiplies.

To understand the preconditioner's effect on the convergence of Krylov subspace methods, we look at clustering of eigenvalues of the matrix $(\mathbf{I} - \beta\bar{\mathbf{P}})^{-1}(\mathbf{I} - \alpha\bar{\mathbf{P}})$. Let $\lambda_i,\ i = 1,\dots,n$ be the eigenvalues of $\bar{\mathbf{P}}$. If we solve the preconditioned iteration defined by $(\mathbf{I} - \beta\bar{\mathbf{P}})^{-1}$ exactly, then the eigenvalues of the matrix $\bar{\mathbf{P}}$ undergo a Möbius transform to the eigenvalues of the preconditioned system,

$$p(\lambda) = \frac{1-\alpha\lambda}{1-\beta\lambda}. \tag{5.25}$$

When we precondition with only a finite number of terms, then the modification of the eigenvalues is no longer a Möbius transform, but the polynomial

$$p_m(\lambda) = (1-\alpha\lambda)(1+\beta\lambda+\cdots+(\beta\lambda)^m). \tag{5.26}$$

Of course as $m \to \infty$, we recover the exact preconditioned system.

To illustrate the spectral properties of the preconditioned system, figure 5.4 shows the behavior of the preconditioned eigenvalues of two test matrices, for a variety of choices of β and m. In these examples, our preconditioner concentrates the eigenvalues around $\lambda = 1$. Solving the system exactly appears unnecessary as we see a strong concentration with even $m = 2$ or $m = 4$ terms of the Neumann series, for the values of β we tested.

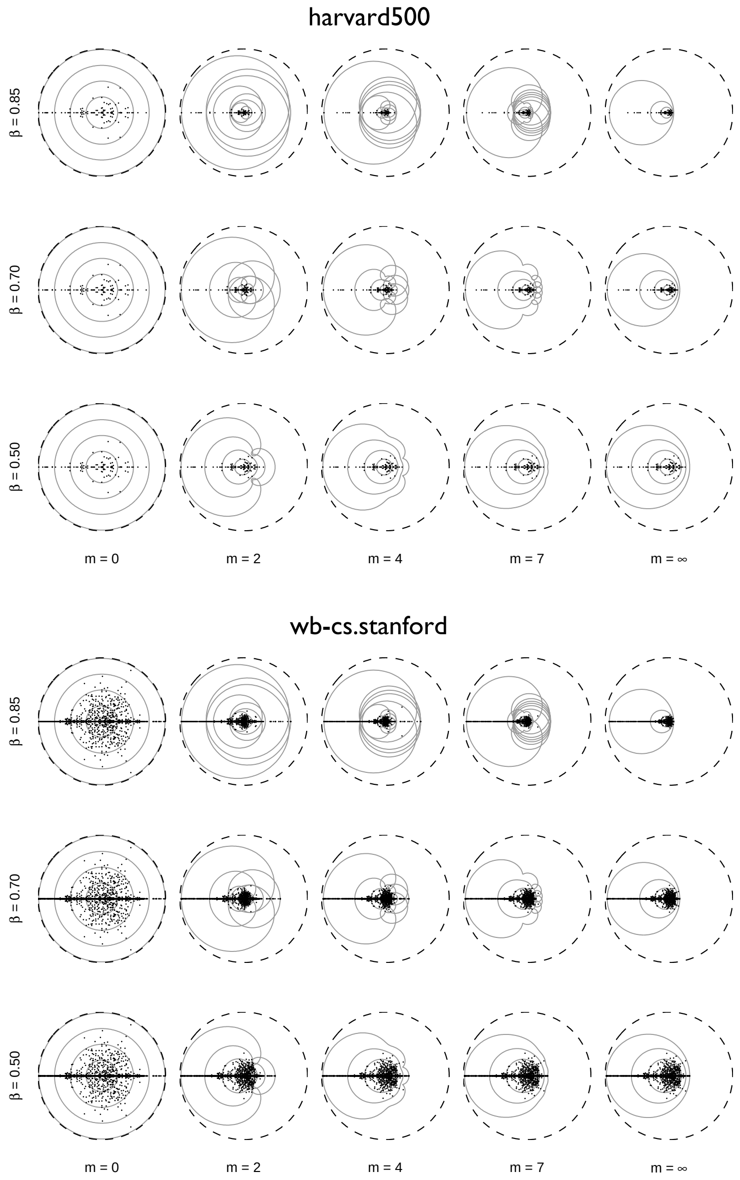


Figure 5.4 – Inner-Outer preconditioner spectrum. For the matrices harvard500 and wb-cs.stanford with $\alpha = 0.99$, this figure plots the eigenvalues of the preconditioned matrix $(\mathbf{I}-\beta\bar{\mathbf{P}})^{-1}(\mathbf{I}-\alpha\bar{\mathbf{P}})$ and approximations based on Neumann series. Each dashed circle encloses a circle of radius 1 in the complex plane centered at $\lambda = 1$, and hence the scale is the same in each small figure. Gray lines are contours of the function $p_m(\lambda)$ defined in (5.26), which is the identity matrix for $m = 0$ (i.e. no preconditioning) and the exact inverse when $m = \infty$. The dots are the eigenvalues of the preconditioned system, $p_m(\lambda_i)$. Interlacing contours of $p_m(\lambda)$ demonstrate that this function is not 1-1 for $\lambda : |1-\lambda| \leq 1$.

Gleich et al. [2004] and [Del Corso et al., 2007] explored the performance of preconditioned BiCG-STAB on the PageRank system. We have modified the Matlab implementation of BiCG-STAB to use the 1-norm of the residual as the stopping criterion.

To compare against the power-method and Gauss-Seidel, the normalized solution vectors $\mathbf{x} = \mathbf{y}/\|\mathbf{y}\|_1$ always satisfy

$$\left\|\alpha\mathbf{P}\mathbf{x}^{(k)} + (1-\alpha)\mathbf{v} - \mathbf{x}^{(k)}\right\| \le \tau.$$

This criterion is equivalent to taking one step of the power method and checking the difference in iterations. Consequently, all the solution vectors tested are at least as accurate as the vectors computed in the power method with tolerance τ. In practice, we did not test the previous solution criterion at every iteration and instead modified the Matlab BiCG-STAB function to terminate the computation when the 1-norm of the residual was less than $(\sqrt{1-\alpha})\tau$. Empirically, using $(\sqrt{1-\alpha})\tau$ as the tolerance for BiCG-STAB yielded $\mathbf{y}$'s that satisfied our actual tolerance criterion without a full rewrite of the residual computations.

BiCG-STAB diverges or breaks down on in-2004 without preconditioning for $\alpha = 0.99$, see table 5.4. This matches observations by Del Corso et al. [2007] that Krylov methods often have convergence difficulties. Adding the preconditioner with $m = 2$ and $\beta \ge 0.5$ avoids these break-down cases. The remainder of the table shows that preconditioning accelerates convergence in many cases for "reasonable" parameter choices. Among the methods discussed, BiCG-STAB with this preconditioner converges with the fewest matrix-vector multiplications on the in-2004 graph with $\alpha = 0.99$ and $\alpha = 0.999$. However, the cost per iteration is higher.

5.6.5 *Other applications*

The IsoRank algorithm [Singh et al., 2007] is a heuristic to solve the network alignment problem. Given two graphs, $\mathcal{A}$ and $\mathcal{B}$, the goal in the network alignment problem is to find a match for each vertex of graph $\mathcal{A}$ in $\mathcal{B}$ and vice-versa. The resulting matching should maximize the number of cases where i in $\mathcal{A}$ is mapped to j in B, i' in $\mathcal{A}$ is mapped to j' in $\mathcal{B}$, and both of the edges $(i, i') \in \mathcal{A}$ and $(j, j') \in \mathcal{B}$ exist. This objective alone is NP-hard. Often there are weights for possible matches, e.g. V_{ji} for i in $\mathcal{A}$ and j in $\mathcal{B}$, that should bias the results towards these matchings, and hence the objective also includes a term to maximize these weights.

Table 5.4 – Inner-Outer preconditioned BiCG-STAB performance. Matrix-vector products required for BiCG-STAB on in-2004, including preconditioning and residual computations, to converge on the system $(\mathbf{I} - \alpha\bar{\mathbf{P}})$ with preconditioner $\sum_{k=0}^{m}(\beta\mathbf{P})^k$. A dash indicates the method made progress but did not converge to a tolerance of $(\sqrt{1-\alpha})10^{-7}$ in the maximum number of iterations required for the power method (100 for $\alpha = 0.85$, ≈1500 for $\alpha = 0.99$, ≈15000 for $\alpha = 0.999$), and an × indicates the method diverged or broke-down. When $m = 0$, there is no preconditioning and the results are independent of β.

	α											
	0.85				0.99				0.999			
	β				β				β			
m	0.25	0.50	0.75	0.85	0.25	0.50	0.75	0.85	0.25	0.50	0.75	0.85
0	102	102	102	102	×	×	×	×	×	×	×	×
2	128	88	76	76	1140	672	508	500	×	6276	3972	2772
4	186	120	84	78	1584	786	438	414	×	5178	2358	2112
7	—	207	108	72	2565	1053	621	441	×	9567	2709	1449
25	—	—	—	81	—	—	1809	1026	—	20385	7911	2754

Let $\mathbf{P}$ and $\mathbf{Q}$ be the uniform random-walk transition matrices for $\mathcal{A}$ and $\mathcal{B}$, respectively. Also, let the weights in $\mathbf{V}$ be normalized so that $\mathbf{e}^T\mathbf{V}\mathbf{e} = 1$ and $V_{ij} \geq 0$. IsoRank uses the PageRank vector

$$\mathbf{x} = \alpha(\mathbf{P} \otimes \mathbf{Q})\mathbf{x} + (1-\alpha)\mathbf{v},$$

where the teleportation vector $\mathbf{v} = \text{vec}(\mathbf{V})$ encodes the weights and α indicates how much emphasis to place on matches using the weights information. Thus the IsoRank algorithm is a case when $\mathbf{v}$ is not uniform, and α has a more concrete meaning. For a protein matching problem, it is observed experimentally in Singh et al. [2007] that values of α between 0.7 and 0.95 yield good results.

We look at a case when $\mathcal{A}$ is the 2-core of the undirected graph of subject headings from the Library of Congress [Various, 2008] (abbreviated LCSH-2) and $\mathcal{B}$ is the 3-core of the undirected Wikipedia category structure [Various, 2007] (abbreviated WC-3). We previously used these datasets in analyzing the actual matches in a slightly different setting [Bayati et al., 2009]. The size of these datasets is reported in table 5.5. For this application, the weights come from a text-matching procedure on the labels of the two graphs.

Table 5.5 – IsoRank datasets.

dataset	size	non-zeros
LCSH-2	59,849	227,464
WC-3	70,509	403,960
Product Graph	4,219,893,141	91,886,357,440

In this experiment, we do not investigate all the issues involved in using a heuristic to an NP-hard problem and focus on the performance of the inner-outer algorithm in a non-Web ranking context. Without any parameter optimization (i.e., using $\beta = 0.5$ and $\eta = 10^{-2}$), the inner-outer scheme shows a significant performance advantage as demonstrated in table 5.6.

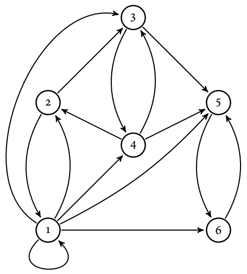
Inner-Outer	188 mat-vec	36.2 hours
Power	271 mat-vec	54.6 hours

Table 5.6 – Inner-Outer performance for IsoRank. The inner-outer iteration ($\beta = 0.5, \eta = 10^{-2}$) is also faster than the power method on IsoRank with $\alpha = 0.95, \tau = 10^{-7}$, and **v** sparse and non-uniform. The computations were done in pure MATLAB and the product graph is never explicitly formed.

5.6.6 *Inner-Outer acceleration*

We now compare our inner-outer method to the power method and attempt to explain *why* it converges faster. On with $\alpha = 0.99$, $\beta = 0.5$, and $\eta = 10^{-2}$, the inner-outer iteration takes 112 matrix-vector products to converge to an outer tolerance of 10^{-8}. In contrast, the power method takes 2,013 matrix-vector products—an incredible difference!

$\mathcal{G} =$



Understanding this performance requires that we investigate where the error occurs in the space of the eigenvalues of **P**.[8] For this small graph, the stochastic matrix **P** is diagonalizable with eigenvalues $1, -1, 0.83, -0.4 \pm 0.28i, 0.14$. In figure 5.5 we plot the projected error for a few eigenmodes in the solution, for each step of the power method and the inner-outer method. This figure shows that the power method spends most of those 2,000 iterations reducing the error in the eigenvector with eigenvalue -1. The inner-outer method dispatches with this eigenvector after 20 iterations and immediately before the switch to using only the power method. The remainder of the inner-outer iterations are consumed reducing the error in the eigenvector with eigenvalue 0.83, which is a much easier task.

In table 5.7 we illustrate the fact that convergence of the inner-outer iterations does not seem to depend on the eigenvalues of **P** that are on the unit circle, but rather on the magnitude of the largest non-dominant eigenvalue.

[8] Here, $\mathbf{P} = \begin{bmatrix} 1/6 & 1/2 & 0 & 0 & 0 & 0 \\ 1/6 & 0 & 0 & 1/3 & 0 & 0 \\ 1/6 & 1/2 & 0 & 1/3 & 0 & 0 \\ 1/6 & 0 & 1/2 & 0 & 0 & 0 \\ 1/6 & 0 & 1/2 & 1/3 & 0 & 1 \\ 1/6 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$.

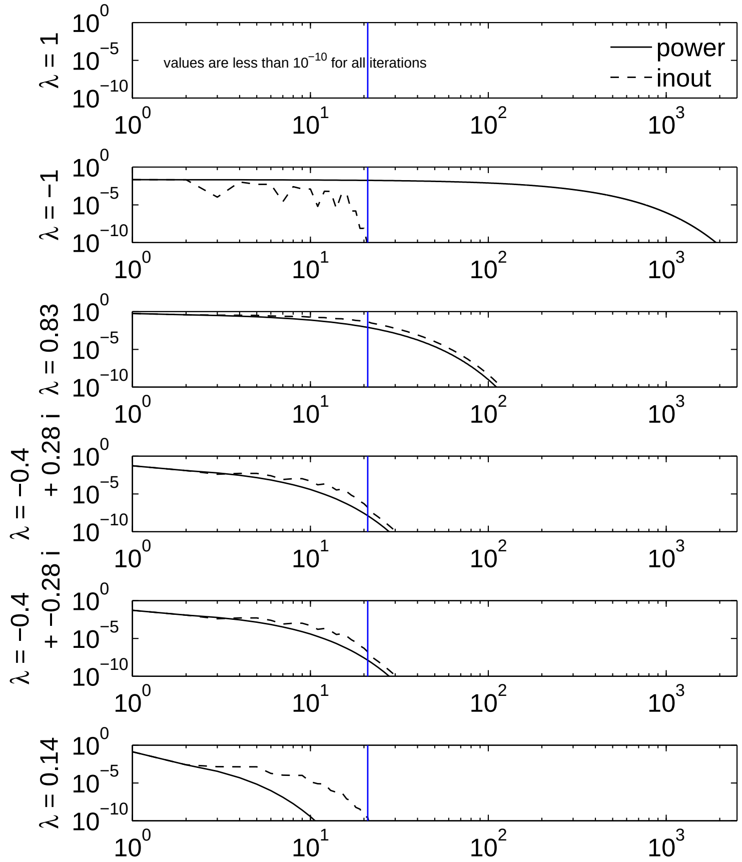


Figure 5.5 – Inner-Outer error analysis for a small graph. When $\alpha = 0.99, \beta = 0.5$, and $\eta = 10^{-2}$, the inner-outer iteration takes 112 matrix-vector multiplies to converge to an outer tolerance of 10^{-10} whereas the power method takes 2,013. The convergence of the power method is limited by the error in the eigenvector with $\lambda = -1$, but the inner-outer iteration quickly eliminates the error in this component. The lines in the middle show where the inner-outer iteration switched to the power method. (The graph is the same as figure 4.2 and at left.)

Table 5.7 – Convergence rates with inner-outer iterations. Convergence ratios from various graphs are $\epsilon_{k+1}/\epsilon_k$, where ϵ_k is the 1-norm change in $\mathbf{x}$ at the kth outer iteration of the inner-outer method. The algorithm switched to the power method after 10 iterations for all of these cases. The first eigenvalue off the unit-circle for small is $\lambda = 0.8257$; for ubc-cs-2006 it is $\lambda = 0.9998$. We observe that the convergence rate is approximately $\alpha\lambda$.

	α =0.85		α=0.99			
	Iteration					
graph	10	50	10	50	100	500
small	0.7019	0.8472	0.8174	0.8174	0.8174	0.99
ubc-cs-2006	0.8220	0.8420	0.9595	0.9801	0.9814	0.9868
in-2004	0.7913	0.8354	0.9343	0.9736	0.9803	0.9872
arabic-2005	0.7936	0.8359	0.9395	0.9746	0.9817	0.9875
sk-2005	0.7932	0.8386	0.9356	0.9773	0.9820	0.9872
uk-2007	0.7925	0.8358	0.9344	0.9743	0.9814	0.9881

This generalizes the results of the previous experiment. The ratios in the table are the change in iterates, not the error in the solution. This makes techniques based on these numbers amenable to computations. The table suggests a strategy to apply the inner-outer iterations dynamically. For the two smallest graphs we have computed all the eigenvalues, and observe the expected convergence based on λ until the round-off error inherent in the power iteration restores the component of error in the eigenmodes on the unit circle. We could build an algorithm to watch for an *increase* in the ratio, or watch for when the ratio gets sufficiently close to α, and then apply additional inner-outer iteration pairs to reduce the magnitude of these eigenvalues.

These experiments motivate the following conjecture.

Conjecture 18. *The asymptotic convergence rate of the inner-outer iteration is $\alpha\lambda_2$, where λ_2 is the eigenvalue inside the unit circle with largest magnitude.*

We offer the following support for the conjecture. Consider an eigenvalue of $\mathbf{P}$ on the unit circle: $\lambda = e^{i\theta}$. If the outer iterations are solved exactly, then we reduce the error in the associated eigenvector by $\frac{(\alpha-\beta)\lambda}{1-\beta\lambda}$. With $\beta = 0.5$, all eigenvalues different from one achieve some reduction and the largest reduction is for eigenvectors with eigenvalue -1. For the power method, none of these eigenvalues is reduced by more than α. Thus, the inner-outer method does reduce the error associated with eigenvalues on the unit circle.

SUMMARY

The inner-outer iteration solves a PageRank problem via a series of PageRank problems with smaller values of α. Outer PageRank problems are solved via an inner Richardson iteration. The subsequent algorithm always converges for PageRank (theorem 17). In addition, we combine the inner-outer idea with the power method (section 5.5.1), the Gauss-Seidel method (section 5.5.2), and the BiCG-STAB method (section 5.5.3). All of these ideas reduce the number of matrix-vector products (or an equivalent work metric) required to converge to a PageRank vector for $\alpha > 0.85$. We also analyze OpenMP shared memory parallelism and provide a conjecture about why the inner-outer iteration is faster (conjecture 18).

As a final note, the inner-outer algorithm is low-memory, easy to implement, and robust—it has become our PageRank solver of choice when Gauss-Seidel iterations are not practical.

If you build it, he will come.

—Field of Dreams

6 SOFTWARE

Over the course of this thesis, we developed a few software packages to aid our work. All are publicly available. This chapter briefly describes each package and its major architecture and design goals. A complete description of each package would be nearly as lengthy as the current document and needlessly tedious. Instead, we have editorialized and highlighted the most interesting or challenging pieces of each implementation.

One of the recurring themes in all the packages is implementing graph structures in Matlab. We discuss the setting for these idea in sections 6.1 and 6.2.

The MatlabBGL package (section 6.3) is an interface between the Boost graph library [Siek et al., 2001] and Matlab.[1] The interface is based on the native Matlab sparse matrix and provides a rich suite of graph algorithms that were previously unavailable in Matlab. Currently, the suite works on Windows, Mac OSX, and Linux in both 32-bit and 64-bit computing environments. It has been downloaded over 7,000 times and used in numerous publications.

Whereas MatlabBGL is a complicated set of C and C++ routines to extract the maximum performance for graph operations in Matlab, the `gaimc` routines (section 6.4) are written in pure Matlab m-code.[2] These algorithms are slower than their MatlabBGL counterparts, although, they are faster than the dictum "for-loops in Matlab are really slow" suggests. For a few algorithms, the `gaimc` code runs in twice the MatlabBGL time; for others, it runs in slightly more than four times the MatlabBGL time. The benefit of the library is that it is easy to add new algorithms with these performance characteristics. Adding algorithms to MatlabBGL is quite difficult.

In this thesis, many of the graphs explored are extremely large. When these are graphs created by crawling the WWW, they have some regularity that makes them highly compressible. Boldi and Vigna [2005] develop coding schemes to compress these graphs at under three bits per edge. Recent improvements in applying the same technique enhance compression to

[1] In the remainder of this chapter, we are going to remove the special styling on Matlab. Matlab is used as a proper noun throughout the remainder of the chapter instead of a product name.

[2] Matlab m-code is how we refer to Matlab's native language.

around one bit per edge [Boldi et al., 2009]. Our `libbvg` code (section 6.5) re-implements pieces of the `bvgraph` framework in portable C code to enable a Matlab wrapper and a shared memory parallel interface. Writing a true Python wrapper for the graphs should also be possible.

Many of the contributions of this thesis have already been published. Each publication has an accompanying software package providing all the source code and experiment scripts. These packages are briefly described in section 6.6.

6.1 ADJACENCY MATRICES

Throughout this chapter, we work with graphs. Each graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of two sets: $\mathcal{V} = \{1, \ldots, n\}$ is a set of vertices and $\mathcal{E}$ is a set of edges. Each edge $(u, v) = e \in \mathcal{E}$ is an ordered pair of vertices with $u \in \mathcal{V}, v \in \mathcal{V}$. Each vertex is already identified by a numeric index, and we identify graphs with their binary adjacency matrix:

$$\mathbf{A} = [A_{ij}] \qquad A_{ij} = \begin{cases} 1 & (i, j) \in \mathcal{E} \\ 0 & \text{otherwise.} \end{cases} \tag{6.1}$$

An undirected graph has both (i, j) and (j, i) in $\mathcal{E}$. Hence,

when $\mathcal{G}$ is undirected, then $\mathbf{A}$ is symmetric.

All of the following generalizations of the binary adjacency matrix maintain this property.

We handle graphs with weighted edges in two cases. In both cases, we consider the weights as elements from $\mathbb{R}$. Let $w(e)$ be a map from edges $e \in \mathcal{E}$ to weights in $\mathbb{R}$. When all the weights exclude the value 0, our first case, then the weighted adjacency matrix is

$$\mathbf{A} = [A_{ij}] \qquad A_{ij} = \begin{cases} w(e) & e = (i, j) \in \mathcal{E} \\ 0 & \text{otherwise.} \end{cases} \tag{6.2}$$

This approach obviously fails when the edge weights can include the value 0. In this second case, we store the graph as a pair of matrices: a value matrix **A** and a structure matrix **S**. This setup yields

$$\mathbf{A} = [A_{ij}] \qquad A_{ij} = \begin{cases} w(e) & e = (i,j) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases} \tag{6.3}$$

and

$$\mathbf{S} = [S_{ij}] \qquad S_{ij} = \begin{cases} 1 & (i,j) \in \mathcal{E} \\ 0 & \text{otherwise.} \end{cases} \tag{6.4}$$

By encoding the graph structure in **S** and leaving the values in **A**, we can distinguish where edges occur and their values. A similar encoding for structural and weight matrices is described in Latora and Marchiori [2001].

In the remainder of the chapter, the type of the adjacency matrix—whether it is binary, weighted, or paired—is determined by the type of graph. To summarize, when

$\mathcal{G}$ is unweighted, **A** is a binary adjacency matrix;
$\mathcal{G}$ has edge weights in $\mathbb{R} - \{0\}$, **A** is a weighted adjacency matrix; and
$\mathcal{G}$ has edge weights in $\mathbb{R}$, $(\mathbf{A}, \mathbf{S})$ is weighted adjacency matrix pair.

We do not consider multi-graphs.

With the setup of looking at graphs as adjacency matrices, we next discuss why these adjacency matrices are the natural graph data structures in the Matlab environment and briefly review what can be done with graphs in Matlab with only small snippets of code (section 6.2).

6.2 graphs in matlab

Although Matlab lacks an explicit set of graph algorithms, it does provide a rich set of techniques that are often effective surrogates. Given the representation of a graph as an adjacency matrix, and the efficient implementation of sparse matrices in Matlab [Gilbert et al., 1992], this is not greatly surprising. To be more concrete, many operations on graphs correspond precisely to an operation on the adjacency matrix. Operations on large, sparse graphs are efficient if the operation is also efficient on a sparse matrix. For example, computing in- and out-degrees is just an application of the sum command.

```
din  = sum(spones(A),1);  % compute in-degrees
dout = sum(spones(A),2);  % compute out-degrees
```

A more complete list of these equivalences is enumerated in Gilbert et al. [2008, Table 1]. Because Matlab already has efficient sparse matrices that easily handle millions of rows and columns with tens of millions of non-zeros on a single processor, re-using these sparse matrices as adjacency matrices of large graphs seems prudent.

Let's examine two more complicated examples. Clustering coefficients are measures of the local connectivity density around a vertex [Watts and Strogatz, 1998]. The key computation to compute clustering coefficients is counting the number of triangles around a vertex and dividing by the maximum possible triangle count.[3] On a symmetric adjacency matrix **A**, the following Matlab code returns the clustering coefficients.

```
A1=spones(A); A1=A1-diag(diag(A1));  % remove weights and self-loops
d=sum(A1,2); d(d<2)=2;               % avoid divide by zeros
cc=diag(A1^3)./(d.*(d-1));           % return clustering coefficients
```

Second, the dmperm function computes the Dulmage-Mendelsohn permutation by finding a maximum cardinality matching between the rows and columns when viewed as vertices of a bipartite graph. In the *meshpart* toolkit [Gilbert and Teng, 2002], this behavior is exploited to compute the strongly connected components of a graph.

While each of these approaches demonstates a particular algorithm or computation in terms of primitive Matlab operations, each approach is also inefficient compared to a straightforward implementation on a graph.[4] In the case of computing degrees, the in-degree counts are *stored* inside the Matlab sparse matrix type: we should not need to compute anything. In the case of clustering coefficients, we need only compute the diagonal elements of $\mathbf{A}^3$, and not the entire matrix. Finally, in the case of connected components, the

[3] For a vertex with degree d, it could form $d(d-1)$ with its neighbors.

[4] While these implementations are inefficient on a single processor, graph operations are notoriously difficult to parallelize. Gilbert et al. [2007] posits that these "inefficient" implementations are better for parallel graph computations.

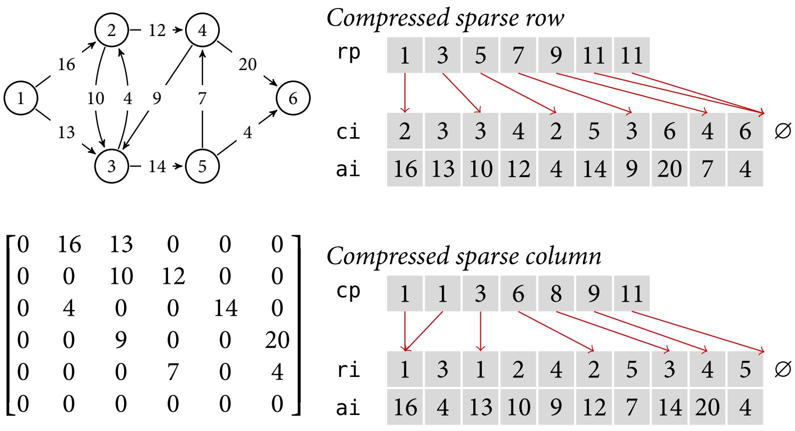
`dmperm` function solves a bipartite matching problem instead of a simple linear pass over the graph edges using a standard connected component algorithm.

Nonetheless, the issue with these computations is the algorithm implementation and not the data structure. Thus, much as others in the literature, we represent graphs as sparse matrices in Matlab. Let's review the sparse matrix data structure in Matlab.

6.2.1 *Sparse matrices in Matlab*

To store an $m \times n$ sparse matrix $\mathbf{M}$, Matlab uses compressed column format [Gilbert et al., 1992]. Matlab never stores a 0 value in a sparse matrix. It always "re-compresses" the data structure in these cases. If $\mathbf{M}$ is the adjacency matrix of a graph, then storing the matrix by columns corresponds to storing the graph as an in-edge list.

We briefly illustrate compressed row and column storage schemes in figure 6.1.



$$\begin{bmatrix} 0 & 16 & 13 & 0 & 0 & 0 \\ 0 & 0 & 10 & 12 & 0 & 0 \\ 0 & 4 & 0 & 0 & 14 & 0 \\ 0 & 0 & 9 & 0 & 0 & 20 \\ 0 & 0 & 0 & 7 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Compressed sparse row

rp	1	3	5	7	9	11	11				
ci	2	3	3	4	2	5	3	6	4	6	∅
ai	16	13	10	12	4	14	9	20	7	4	

Compressed sparse column

cp	1	1	3	6	8	9	11				
ri	1	3	1	2	4	2	5	3	4	5	∅
ai	16	4	13	10	9	12	7	14	20	4	

Figure 6.1 – Compressed row and column storage. At far left, we have a weighted, directed graph. Its weighted adjacency matrix lies below. At right are the compressed row and compressed column arrays for this graph and matrix. For sparse matrices, compressed row and column storage make it easy to access entries in rows and columns, respectively. Consider the 3rd entry in `rp`. It says to look at the 5th element in `ci` to find all the columns in the 3rd row of the matrix. The 5th and 6th elements of `ci` and `ai` tell us that row 3 has non-zeros in columns 2 and 5, with values 4 and 14. When the sparse matrix corresponds to the adjacency matrix of a graph, this corresponds to efficient access to the out-edges and in-edges of a vertex.

Most graph algorithms are designed to work with out-edge lists instead of in-edge lists.[5] Before running an algorithm, MatlabBGL explicitly transposes the graph so that Matlab's internal representation corresponds to storing out-edge lists. For algorithms on symmetric graphs, these transposes are not required.

The `mex` commands `mxGetPr`, `mxGetJc`, and `mxGetIr` retrieve pointers to Matlab's internal storage of the matrix without making a copy. These functions make it possible to access a sparse matrix efficiently without making a copy and are a requirement of our implementation.

[5] See section 6.3.1 for a discussion about the requirements for various graph algorithms.

Let us recap. Sparse matrices are the best way to store graphs in Matlab. They provide all the necessary pieces to integrate cleanly with "natural" Matlab syntax and allow us access to their internals to run algorithms efficiently.

6.2.2 *Other packages*

There are other graph packages for Matlab too. One of the first was the `meshpart` toolkit [Gilbert and Teng, 2002], which focuses on partitioning meshes. A more recent example is `Matgraph` [Scheinerman, 2009], which contains a rich set of graph constructors to create adjacency matrices for standard graphs. It also provides an interface to support graph properties, such as labels and weights. Various authors released individual graph theory functions on the Mathworks File Exchange [Various, 2009a, search for dijkstra]. For example, the Exchange contains more than three separate implementations of Dijkstra's shortest path algorithm.

A full comparison of these packages is beyond the scope of this chapter. The only package with the breadth of MatlabBGL is the bioinformatics graph package, which is also based on the Boost graph library. Among the native Matlab packages, only `meshpart` [Gilbert and Teng, 2002] handles large graphs well. Both MatlabBGL and `gaimc` are distinguished because they

- scale to large graphs;
- support *no-data-copy* paths when possible; and
- provide a suite of algorithms.

6.3 MATLABBGL

MatlabBGL is the first package we discuss. Its source code lives publicly on LaunchPad, `http://launchpad.net/matlab-bgl`.

As previously mentioned, MatlabBGL is a Matlab package for working with graphs. It uses the Boost graph library to implement the graph algorithms efficiently. MatlabBGL is designed to compute on large sparse graphs with hundreds of thousands of nodes. To do so, the library consists of "wrappers" for algorithms from the Boost graph library. Each wrapper is a `mex` function and it is callable directly from Matlab. The goal of the library was to introduce as little new material into Matlab as possible. To facilitate this, MatlabBGL does not introduce a new data structure and uses the Matlab sparse matrix type as the graph type directly.

For example,

```
n = 10;
A = spdiags([ones(n,1), zeros(n,1), ones(n,1)],[-1 0 1], n, n);
cc = clustering_coefficients(A)’ % transpose the output for display
comps = components(A)’           % transpose the output for display
```

constructs a 10-node line graph as a Matlab sparse matrix, computes clustering coefficients with the MatlabBGL `clustering_coefficients` function, and computes the index of a strongly connected component for each vertex. For both of these functions, the output consists of one number per vertex. For `clustering_coefficients`, it is the clustering coefficient of that vertex, and for `components`, it is the index of the strong component for that vertex. Examining the output

```
cc =
    0     0     0     0     0     0     0     0     0     0
comps =
    1     1     1     1     1     1     1     1     1     1
```

shows that the clustering coefficient for each node is 0, which is expected for the line graph `A`, and that each node is in the same connected component, which is also expected for a line graph.

To describe the library and the remainder of the implementation, we begin by reviewing the Boost graph library.

6.3.1 *The Boost graph library (BGL)*

The Boost graph library or BGL [Siek et al., 2001] is a large set of C++ codes that implement generic graph algorithms. The advantage of these generic graph algorithms is that they specify the algorithm independently of the data structure. This independence is not accomplished using interfaces or abstract classes, which are common in standard object oriented programming. Instead, the Boost graph library uses C++ templates and techniques from generic programming [Alexandrescu, 2001] to write their data-structure-free algorithms. These techniques, in theory, allows the compiler to view the algorithm and data structure simultaneously and optimize the entire package. In particular, the compiler can generate in-line optimizations between function calls.

In the Boost graph library, each algorithm places certain requirements on the C++ graph type. Concepts codify these requirements. In the BGL, the concepts largely express *mutability* (support for *changing* the graph during the algorithm) and *access* (support for querying the existing graph structure in various ways). Most of the algorithms wrapped in MatlabBGL do not change the graph; thus we focus on the access concepts.

For example, the `strong_components` BGL function requires a graph type that supports the *VertexListGraph* and *IncidenceGraph* concepts. These concepts allow the algorithm to iterate over all vertices in the graph and access the out-edges for an arbitrary vertex. This access to the graph suffices to implement Tarjan's algorithm for strongly connected components [Tarjan, 1972]. The other access concepts are

EdgeListGraph	iterate over the edges of the graph
BidirectionalGraph	access in-edges to an arbitrary vertex, and
AdjacencyGraph	access adjacent (out-edge connected) vertices to an arbitrary vertex.

Most algorithms are like `strong_components` and require only the *VertexListGraph* and *IncidenceGraph* concepts. Two notable exceptions are maximum flow (`push_relabel_max_flow`, `kolmogorov_max_flow`), and planar graph triangulation (`make_maximal_planar`). In this document, we focus on the common cases and leave the exceptions to the source code documentation and help files of MatlabBGL.

6.3.2 *Matlab and Boost graph library interface*

Before describing the interface, we reiterate one main point of MatlabBGL. To integrate cleanly with Matlab, MatlabBGL uses the Matlab sparse matrix type as the graph type. Thus, the goal of the interface between Matlab and the Boost graph library is to take a Matlab sparse matrix data structure and view it as a C++ type that implements the *VertexListGraph* and *IncidenceGraph* concepts. (Our actual implementation also supports the *EdgeListGraph* and *AdjacencyGraph* concepts.) With such an interface, we can access the majority of graph analysis algorithms in the Boost graph library. What is lost from the BGL is largely irrelevant to MatlabBGL. There is no need for the `copy_graph` function from Boost, for example.

Next, figure 6.2 shows the high level architecture of MatlabBGL. There are four main components: m-files, mex-files, libmbgl, and BGL functions. Let's illustrate a typical call to a MatlabBGL function: `dfs` for a depth-first search through the graph.

First, the `dfs.m` file (an M code) receives the sparse matrix representation of the graph and the identifier of a vertex that originates the search. It performs some basic parameter checking on the data, transposes the matrix to

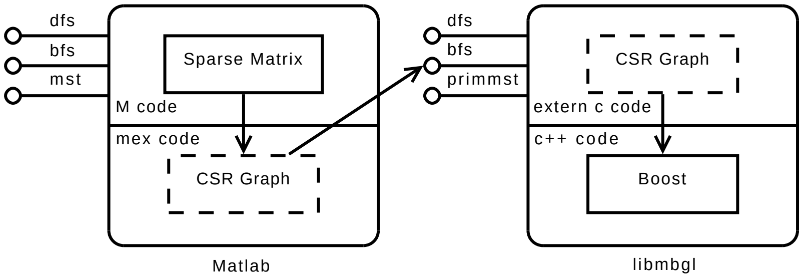


Figure 6.2 – MatlabBGL architecture. MatlabBGL consists of four components: m-files, mex-files, libmbgl, and Boost graph library functions. See the text for a description of how data flows through these components.

get the graph stored by out-edges in the Matlab data structure, and forwards the information to the `dfs_mex.c` mex-file. By providing an optional argument to the function, both the check and the transpose can be eliminated for the fastest performance. The mex-file extracts the compressed sparse column arrays for the sparse matrix, which corresponds to a compressed out-edge list representation of the graph, and sends the information to the libmbgl function `depth_first_search`. The libmbgl functions implement wrappers around Boost functions on compressed sparse row arrays and expose them via a C calling convention. This library is further described in section 6.3.3. For the `depth_first_search` function, the wrapper takes the compressed sparse row arrays and instantiates a `csr_graph` type that implements the *VertexListGraph*, *IncidenceGraph*, *EdgeListGraph*, and *AdjacencyGraph* concepts directly on the compressed sparse row arrays. With the `csr_graph` object, the libmbgl wrapper calls a Boost graph library function.

Throughout this entire process, the only *copy* of the data occurs when the initial sparse matrix is transposed to store the data by out-edges (rows) instead of in-edges (columns).[6]

Thus far, the interface between the libraries is only complicated by the layers of abstraction. Although maintaining three layers (m-files, mex-files, and libmbgl) may seem unnecessary, it simplifies calling conventions across multiple platforms. The m-files call mex-files, which Matlab always supports. The mex-files call functions in libmbgl with a C calling convention, which is also extremely portable. And the C functions interface with the Boost graph library. We discuss other reasons to keep libmbgl separate from the mex files in the next section.

The most complicated piece of the interface is the `csr_graph` object. This object is itself a generic object because it must support both 64-bit and 32-bit index types on the compressed sparse row arrays. Furthermore, it supports

[6] Some Boost graph functions make a copy of the graph inside the algorithm. We can do nothing about these copies without modifying the BGL itself.

an optimization for algorithms that do not require edge weights. We will not delve into its implementation here as the arcana of implementing a Boost graph concept are best left to the Boost graph documentation.[7]

COPY-FREE? A key feature of MatlabBGL is that it provides large-scale graph algorithms in Matlab by avoiding copies. However, the first step in almost every algorithm is to *copy* and transpose the sparse matrix. For true large-scale computations, users must take care to ensure that the no-copy computation paths within MatlabBGL are used. Most MatlabBGL functions support these paths and exceptional cases are all documented.[8]

6.3.3 *libmbgl: A compressed sparse row interface*

Within MatlabBGL, libmbgl is the component that translates a compressed-sparse row graph representation into a BGL graph type and makes the BGL function call. As figure 6.2 reflects, this library is completely independent of the Matlab components. One of the reasons for this separation is that libmbgl provides a fast interface to the Boost graph library from any application that can link against a set of C functions. We are aware of at least one manuscript where this lower level interface was used [Karci, 2008].

Furthermore, libmbgl opens the possibility to reuse a piece of MatlabBGL with open source scientific computing packages such as SciLab and Octave. If these libraries provide access to a compressed row representation, then libmbgl provides the necessary link. MatlabBGL is mostly usable "as-is" in the Octave program because they implement the Matlab mex interface and support a rough equivalent to the Matlab language.

In summary, we consider libmbgl an integral, but independent, piece of MatlabBGL and plan to keep it that way.

6.3.4 *Return types*

Thus far, we've seen how MatlabBGL takes a sparse matrix and translates it to a function call in the Boost graph library. Next, the Boost graph library generally computes a result as part of this function call. We need to represent these results in Matlab.

Recall that one of the design philosophies of MatlabBGL is to add as little to Matlab as possible. With this constraint, picking amongst options for the return types is straightforward. Algorithms that compute a metric on a vertex, such as `strong_components` or `clustering_coefficients`, already support writing data to a standard `int*` or `double*` array. The wrappers in libmbgl

[7] It is interesting to note that the first version of MatlabBGL with an entirely correct Graph concept was the third iteration of the library. Earlier revisions did not implement an obscure property that was eventually used in the `dominator_tree` algorithm.

[8] Presently, the exceptional cases are the maximum flow routines and the planar graph triangulation function.

return these data to the mex functions, which proceed to store them in Matlab arrays.[9]

For measures on edges, such as biconnected_components, we often support two different return value mechanisms. When the values are all non-zero, MatlabBGL will return a sparse matrix with the edge information. When the values on an edge may include zero, MatlabBGL optionally returns the edges and values explicitly as a set of "(source, destination, value)" triplets.[10]

Graph outputs are returned as sparse adjacency matrices.

Finally, when the output is a rooted tree, we store it as a single array where the ith entry gives the parent of vertex i in the tree. Matlab already uses this storage format for elimination trees of sparse matrix algorithms (see treeplot). Weighted trees, from prim_minimum_spanning_tree for example, are returned as triplet arrays or sparse matrices.

This section concludes our description of the common Matlab to Boost graph library interface that exists in MatlabBGL. The next few subsections discuss some extensions of this standard interface to take advantage of more of the Boost graph library.

6.3.5 *An in-place class for visitors*

The Boost graph library implements "visitors" to make its algorithms flexible. Visitors are small classes whose methods are triggered by events in an algorithm. They enable reuse for an implementation of a simple graph algorithm in a more complicated algorithm. One example is the strong_components function. Using a visitor, it reuses the existing depth_first_search implementation. An alternative use of the visitor pattern is to collect additional information about an algorithm while it is running. One example of this feature is recording the first visit "time" for a vertex in a breadth-first search. Another example is stopping a search algorithm when it reaches a target vertex.

Inside the BGL, these visitors are C++ classes that the algorithm calls at various points. In MatlabBGL, we pass in a set of function handles to receive the calls instead. Historically, Matlab functions would not have been able to be full featured visitors in the BGL as they could not store information within a function call like a C++ class. (We did not consider using global variables for this task an acceptable solution.) From Matlab version 7, however, Matlab supports functions that maintain state using nested functions and closures. These tools are the keys to making visitors useful in MatlabBGL.[11]

We explain this by way of a small example.

[9] Because libmbgl requires the array storage to be allocated before the function call, the data are stored directly into the Matlab arrays when possible.

[10] The libmbgl code always returns triplet arrays and users get the option of which form to view the triplet arrays in the MatlabBGL calls

[11] From Matlab 2008a, the new class semantics support yet another way of constructing visitors in Matlab.

Consider this function

```
function fh=generate_counter()
  value = 0;
  function curval=counter(k)
    curval = value;
    value = value+k;
  end
  fh=@counter;
end
```

With this function stored in `generate_counter.m`, then

```
c = generate_counter()
c(5)
c(1)
c2 = generate_counter()
c2(1)
c2(1)
```

produces the following output:

```
ans =
      0
ans =
      5
ans =
      0
ans =
      1
```

Using this technique, we first design an "in-place" library for Matlab that allows a matrix or vector to be passed by reference and not by value.[12] The three classes of the in-place library are

1. `ipdouble`,
2. `ipint32`, and
3. `inplace`.

Each class constructor takes an existing array and converts it to a type that has pass-by-reference semantics.

Without pass-by-reference, the following function is useless:

```
function add_one_inplace(a)
a(1) = a(1) + 1;
```

When `a` is an in-place array from our library, then it becomes useful. Consider this next example.

```
n = 1000000;
ipa = ipdouble(ones(n,1)); % turn the array of all ones into
add_one_inplace(ipa);      % a pass by reference type
ipa(1)
```

```
ans =
     2
```

[12] Matlab implements a pass-by-value scheme with a copy-on-write optimization.

Using in-place variables allows us to write visitors easily in MatlabBGL. Let us now address the types of visitors in the BGL.

Algorithms in the BGL define two types of visitors: vertex visitors and edge visitors. A common vertex visitor is the `examine_vertex` function. For the *BFSVisitor* concept, the Boost graph library defines the following prototype for that visitor:

```
void visitor::examine_vertex(Vertex u, Graph& g)
```

The corresponding MatlabBGL function takes only one argument, the vertex index. Here, we illustrate this visitor type by example as Matlab does not have function prototypes:

```
visitor.examine_vertex = @(u) fprintf('called examine_vertex(%i)!\n', u);
```

In MatlabBGL, the other *vertex* visitors follow this same pattern: the only argument is the index of the vertex.

Edge visitors in Boost provide a BGL edge type:

```
void visitor::examine_edge(Edge e, Graph& g)
```

MatlabBGL translates the edge type into an edge index, a source vertex, and a target vertex, which are provided to the visitor function (again illustrated with an example visitor):

```
visitor.examine_edge = @(ei,u,v) fprintf('called examine_edge(%i,%i)!\n', u);
```

A considerable issue with visitors is that they call back to Matlab from a mex file. This operation is not efficient and visitors are not appropriate for large-scale computations. Nevertheless, they are an important part of the Boost graph library and belong in MatlabBGL.

We end this section with two examples that show how visitors are used in the library.

WATCHING AN ALGORITHM WITH A VISITOR In this example, we write a simple visitor that outputs an algorithm's behavior. The algorithm we examine is `dijkstra_sp`. To examine the runtime behavior we use a visitor that outputs a string every time a function is called. The `dijkstra_sp` function supports the following visitors:

- `initialize_vertex(u)`
- `discover_vertex(u)`
- `examine_vertex(u)`
- `examine_edge(ei,u,v)`
- `edge_relaxed(ei,u,v)`

- `edge_not_relaxed(ei,u,v)`
- `finish_vertex(u)`

Rather than implementing 7 functions ourselves, we define two helper functions. There is one helper that returns a vertex visitor and one helper than returns an edge visitor:

```
vertex_vis_print_func = @(str) @(u) ...
      fprintf('%s called on %s\n', str, char(labels{u}));
edge_vis_print_func = @(str) @(ei,u,v) ...
      fprintf('%s called on (%s,%s)\n', str, char(labels{u}), char(labels{v}));
```

We are almost done. We just have to setup the visitor structure to pass to the `dijkstra_sp` call:

```
vis = struct();
vis.initialize_vertex = vertex_vis_print_func('initialize_vertex');
vis.discover_vertex = vertex_vis_print_func('discover_vertex');
vis.examine_vertex = vertex_vis_print_func('examine_vertex');
vis.finish_vertex = vertex_vis_print_func('finish_vertex');
vis.examine_edge = edge_vis_print_func('examine_edge');
vis.edge_relaxed = edge_vis_print_func('edge_relaxed');
vis.edge_not_relaxed = edge_vis_print_func('edge_not_relaxed');
```

With the visitor set up, all that remains is to call the function:

```
dijkstra_sp(A,1,struct('visitor', vis));
```

```
discover_vertex called on s
examine_vertex called on s
examine_edge called on (s,u)
edge_relaxed called on (s,u)
discover_vertex called on u
...
```

STOPPING AN ALGORITHM EARLY At any point, if a visitor function returns a zero value, the algorithm halts. This behavior may be desirable. Consider the following example with `astar_search`:

```
load graphs/bgl_cities.mat
goal = 11; % Binghamton
start = 9; % Buffalo
% Use the Euclidean distance to the goal as the heuristic
h = @(u) norm(xy(u,:) - xy(goal,:));
% Setup a routine to stop when we find the goal
ev = @(u) (u ≠ goal);
[d pred f] = astar_search(A, start, h, ...
    struct('visitor', struct('examine_vertex', ev)));
```

The `examine_vertex` function returns 0 when the vertex is a targeted vertex. In this example, we want to find the shortest path between Binghamton and Buffalo. Once we find a shortest path to Buffalo, we can halt the algorithm!

6.3.6 *Handling zero edge weights*

Matlab sparse matrices only store non-zero values. Because the structure of the Matlab sparse matrix is used to infer the edges of an underlying graph, MatlabBGL cannot distinguish between a 0-weight edge and an absent edge.

To fix this problem, codes that accept a weighted graph allow the user to specify a vector of edge weights in an optional parameter. The easiest way to use this optional parameter is using a helper function called `edge_weight_vector`. Given a weighted matrix pair $(\mathbf{A}, \mathbf{S})$, this function provides the appropriate optional argument when the graph has zero edge weights.

Internally, this function computes the index of all the non-zero elements of **A** when using all of the structural non-zeros of **S**. An example helps to clarify what happens.

We wish to create an undirected cycle where the weight of every edge is 0, except for a single edge with weight 1.

```
n=8;                                    % 8 vertices,
E = [1:n 2:n 1; 2:n 1 1:n]';            % edge list for a cycle
w = [1 zeros(1,n-1) 1 zeros(1,n-1)]';   % weight of each edge
A = sparse(E(:,1), E(:,2), w, n, n);    % create a weighted sparse matrix
As = sparse(E(:,1), E(:,2), true, n, n); % create a structural sparse matrix
ws = edge_weight_vector(As,A)'
```

The output reveals the single non-zero edge:

```
ws =
  Columns 1 through 11
     1     0     1     0     0     0     0     0     0     0     0
  Columns 12 through 16
     0     0     0     0     0
```

The edge weight vector ws also satisfies the following property:

```
[i j] = find(As);
Aw = sparse(j,i,ws,size(As,1),size(As,2));
isequal(Aw,A)
```

```
ans =
     1
```

In other words, the order of the edge weight vector corresponds to the order of non-zeros in the transposed matrix. The function has to return weights in the order of the transposed matrix because MatlabBGL transposes the sparse matrix on a standard function call.

Using the vector is easy. The trivial usage

```
[d pred] = shortest_paths(A,1);
d(2)
```

gives the wrong answer:

```
 ans =
      1
```

Instead, use

```
[d pred] = shortest_paths(As,1,struct('edge_weight',ws));
d(2)
```

to find the correct distance using all the zero-weight edges:

```
 ans =
      0
```

6.3.7 *Usage*

MatlabBGL has been used in over 10 publications (that we are aware of) and downloaded over 7,000 times. We highlight three of these publications. In Honey et al. [2007], the authors utilize the shortest path computations to analyze the functional connectivity of a macaque neocortex. Next, Lin et al. [2008] use the connected component, betweenness centrality, and cluster coefficients functions to study a network of co-authorship relationships in abstracts presented at the Society for Neuroscience. Third, and finally, Raman and Chandra [2008] use the shortest path routine to analyze a subset of the interactome network of *M. tuberculosis*.

MatlabBGL continues to be a highly downloaded file from the Matlab central file exchange and we hope to continue improving it in the future.

6.4 GAIMC

The `gaimc` library implements a similar set of graph algorithms. It is written entirely in Matlab and the project is maintained at GitHub, http://github.com/dgleich/gaimc.

It follows the motivation of MatlabBGL closely and also uses the Matlab sparse matrix as its graph type. In the following example, we generate a random weighted graph and compute a minimum spanning tree:

```
randn('state',0); rand('state',0);
A=abs(sprandsym(10,0.2)); % symmetric random graph
T = mst_prim(A);
sum(sum(T))/2             % output the MST weight
```

Table 6.1 – Algorithms in `gaimc`. For each function in `gaimc`, we list the common algorithm name and the source for the implementation.

Function	Algorithm	Source
`bipartite_matching.m`	max-weight bipartite matching	Papadimitriou and Steiglitz [1998]
`clustercoeffs.m`	clustering coefficients	Watts and Strogatz [1998]
`corenums.m`	core numbers	Batagelj and Zaversnik [2003]
`dijkstra.m`	Dijkstra's single source shortest path	Cormen et al. [2001]
`dirclustercoeffs.m`	directed clustering coefficients	Fagiolo [2007]
`floydwarshall.m`	Floyd-Warshall's all-pairs shortest paths	Cormen et al. [2001]
`mst_prim.m`	Prim's minimum spanning tree	Cormen et al. [2001]
`scomponents.m`	Strongly-connected components	Tarjan [1972]

If we run this code with *`gaimc`*, it produces the output:

```
ans =
    3.9531
```

In comparison with MatlabBGL, the *`gaimc`* library is simple. Each algorithm is a single Matlab m-file that contains a single function. This conveys some advantages. MatlabBGL, while portable to 32-bit and 64-bit Matlab on Windows, Mac OSX, and Linux, is difficult to compile and maintain on all platforms. The *`gaimc`* library is trivial to port between all of these platforms.

However, folklore about Matlab performance claims that "loops" are slow. After The Mathworks introduced the just-in-time compiler in Matlab 7 (R14), loops are no longer slow—although our experience is that extracting good performance can be difficult for complicated functions.

In *`gaimc`*, we did our best to extract performance for each graph algorithm enumerated in table 6.1. Each algorithm is serial and operates on a set of compressed sparse row arrays. Using this approach, we obtained performance that is 2-4 times slower than MatlabBGL (see section 6.4.3).[13]

Let us begin describing the library by analyzing a necessary tool for many graph algorithms: a heap.

[13] In Matlab R2008b, it seems that the just-in-time compiler yields worse performance than in Matlab R2007b. Instead of a slowdown of 2-4 times, we found a slowdown of 3-8 times with respect to MatlabBGL!

6.4.1 *A Matlab heap structure*

To implement both Dijkstra's shortest path algorithm and Prim's minimum spanning tree algorithm we need a means to store and access vertices, in sorted order, based on a constantly changing set of values. A heap is one data structure that meets these requirements [Cormen et al., 2001]. In this section, we discuss a Matlab implementation of a heap.

The following implementation is inspired by Kahaner [1980].[14] From a data structure perspective, a heap is a binary tree where smaller elements are parents of larger elements. It supports the following operations:

INSERT add an element to the heap;

POP remove the element from the heap with the smallest value; and

UPDATE change the value of an element in the heap.

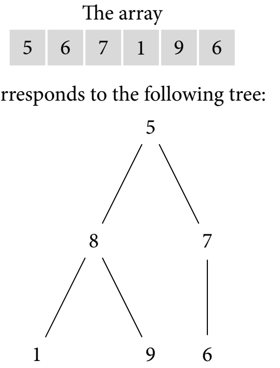


Figure 6.3 – Binary trees as arrays.

Matlab specializes in arrays (or vectors), and a common way to store a binary tree in an array is to associate the tree node of index j with a left child of index $2j$ and a right child of index $2j + 1$. See figure 6.3 for an example.

The data structure for our Matlab heap will consist of four arrays and one number.

T the heap tree. The array T stores the identifiers of the items in the heap. That is, $T(i)$ is the id of the element in tree node i and $T(1)$ is the id of the element at the root of the heap tree.

D the data store. The array T stores ids of elements in D so that $D(T(i))$ is the actual item for tree node i.[15] The size of D must be the maximum number of items ever added to the heap.[16]

L a look up table. The size of L is the maximum id of any item added to the heap. For id i, $L(i)$ is the tree node index of i in T, and $T(L(i)) = i$.

V the value array. The current value associated with id i is given by $V(i)$. This means that $D(i)$ and $V(i)$ are the item and its value, respectively.

N the current size of the heap.

When we use this heap structure to store vertices of a graph, there is no need to maintain the data array D. Each vertex is just a unique numeric identifier for the compressed sparse row arrays that `gaimc` uses. With D, $T(\cdot)$ contains the index of an element in D. When we store vertices in the heap, each vertex already has a unique identifier—its index—and the array D is unnecessary.

[14] More generally speaking, algorithms written in Fortran 77 are excellent candidates for the Matlab just-in-time compiler.

[15] For items without natural ids, ids can be uniquely assigned based on how many items have already been added to the heap. In this case, $D(i)$ contains the ith item added to the heap.

[16] An alternative is to grow the heap by reallocating the arrays if additional items must be added.

Let us now provide implementations for each of the heap operations on these arrays in the case that items are vertices of a graph.

INSERT An insert operation adds a vertex with id `v` and value `val` to the heap arrays at the bottom of the heap tree. It then moves the item up the heap until the parent element is smaller than the inserted element.

```
function [n,T,L,V]=heapinsert(v,val,n,T,L,V)
V(v) = val;
% insert item v (v is the unique id)
n=n+1;                         % increment n
T(n)=v; L(v)=n;                % add v to the arrays
% move the element up the heap
j=n;
while 1
  if j==1, return; end         % element at the top of the heap
  tj=T(j);
  j2=floor(j/2);               % parent element
  tj2=T(j2);
  if V(tj2)<V(tj), return;     % parent is smaller
  else                         % parent is larger, so swap
    T(j2)=tj; L(tj)=j2;
    T(j)=tj2; L(tj2)=j;
    j=j2;
  end
end
```

POP A pop operation removes the item with smallest value, which resides in $T(1)$. It then promotes the last element of the heap to this position and moves the item down the heap to restore the heap property that the parent node is always smaller than both children.

```
function [v,n,T,L,V] = heappop(n,T,L,V)
if n==0, return; end           % check simple case, else delete T(1)
v=T(1);                        % remove the first (smallest) item
T(1)=T(n);                     % promote the last element
T(n)=v;                        % place a dummy value
L(v)=0;                        % and ensure lookups fail
n=n-1;                         % reduce the size
k=1;                           % move the new first element down the heap
while 1
    i=2*k; kt=T(k);
    if i>n, return; end          % end of heap
    if i==n,                     % only one child, skip choice between children
    else
      % pick the smallest child
      lc=T(i); rc=T(i+1);
      if V(rc)<V(lc), i=i+1; end % right child is smaller
    end
    if V(kt)<V(T(i)), return;    % k is smaller than both children, so end
    else
      T(k)=T(i); L(T(i))=k;      % swap
      T(i)=kt; L(kt)=i;
      k=i;
    end
end
```

UPDATE An update operation restores the heap property after changing the value associated with vertex v. If the value of the element is increased, then we must move the element down the tree. If the value of the element is decreased, then we must move the element up the tree.

```
function [n,T,L,V] = heapupdate(v,n,T,L,V)
% V(v) changed, so we need to update the heap
if L(v)==0, return; end        % v isn’t in the heap
% move the element down in the heap
k=L(v);
while 1
    i=2*k; kt=T(k);
    if i>n, break; end           % end of heap
    if i==n                      % only one child, skip choice between children
    else
      % pick the smallest child
      lc=T(i); rc=T(i+1);
      if V(rc)<V(lc), i=i+1; end % right child is smaller
    end
    if V(kt)<V(T(i)), break;     % k is smaller than both children, so end
    else
      T(k)=T(i); L(T(i))=k;      % swap
      T(i)=kt; L(kt)=i;
      k=i;
    end
end
% move the element up the heap if necessary
j=k;
while 1
  if j==1, return; end         % element at the top of the heap
  tj=T(j);
  j2=floor(j/2);               % parent element
  tj2=T(j2);
  if V(tj2)<V(tj), return;     % parent is smaller
  else                         % parent is larger, so swap
    T(j2)=tj; L(tj)=j2;
    T(j)=tj2; L(tj2)=j;
    j=j2;
  end
end
```

To design an algorithm using the heap, we often write an algorithm using the heap functions as subroutines. But then to make the final implementation efficient, we cut-and-paste the heap subroutines “in-line.” The next section makes it clear why all the operations cannot reside in separate functions.

6.4.2 *Matlab performance optimizations: Dijkstra’s algorithm*

We now discuss performance optimizations for the implementation of Dijkstra’s single-source shortest path algorithm. This algorithm returns the shortest path distance between a source vertex and every other reachable vertex in the graph. One required assumption is that the edge distances (or weights) are non-negative. The algorithm explores the graph in a breadth-first manner based on the current shortest distance from the source. To pick the next vertex to explore, we need to find the closest vertex to the source, add or update the distance to all their neighbors to the current set of known vertices, and repeat. Finding the next vertex to explore based on the current distances can be done with the heap data structure in the previous section.

Using the heap, Dijkstra’s algorithm is only a few lines of code.

```
function [d pred]=dijkstra_slow(A,u)
At = A’; n = size(At,1); % transpose for row access
d=Inf*ones(n,1); T=zeros(n,1); L=zeros(n,1); pred=zeros(1,n); % allocate heap
n=1; T(n)=u; L(u)=n; % n is now the size of the heap
% enter the main dijkstra loop
d(u) = 0;
while n>0
    [v,n,T,L,d] = heappop(n,T,L,d);   % find the closest vertex
    % for each vertex adjacent to v, relax it
    [si sj sv]=find(At(:,v));
    for ei=1:length(si)               % ei is the edge index
        w=si(ei); ew=sv(ei);          % w is the target, ew is the edge weight
        % relax edge (v,w,ew)
        if d(w)>d(v)+ew
            d(w)=d(v)+ew; pred(w)=v;
            % check if w is in the heap, insert if not, else update
            if L(w)==0, [n,T,L,d] = heapinsert(w,d(w),n,T,L,d);
            else [n,T,L,d] = heapupdate(w,n,T,L,d);
            end
        end
    end
end
```

Just like the previous description, we find the closest vertex and compute the distance to each neighbor. We then update the heap and continue.

This function has respectable performance:

```
load_gaimc_graph(’cs-stanford’) % 9914 by 9914 with 36854 edges (directed)
A = sprand(A);
rand(’state’,0); vs = ceil(size(A,1)*rand(100,1));
tic, for i=1:100, [d pred] = dijkstra_slow(A,i); end, toc
```

```
Elapsed time is 10.748844 seconds.
```

If we remove the function calls for the heap and place the function code in-line,[17] then the performance improves considerably.

```
Elapsed time is 3.459063 seconds.
```

[17] We have omitted the source code for this case because it gets rather verbose with all the heap manipulations.

The critical graph operation in Dijkstra's algorithm is accessing the out-edges of an arbitrary vertex. The approach used in the `dijkstra_slow` implementation above is to transpose the sparse matrix and work with the Matlab matrix structure directly. For each out-edges query, this approach involves extracting a column and using the `find` function to get the non-zeros.[18] An alternative is to convert the sparse matrix into a compressed sparse row structure on every call. Using this approach, we obtain even better performance.

```
Elapsed time is 1.052805 seconds.
```

In *gaimc*, then, we first convert any Matlab sparse matrix input to a compressed sparse row data structure for the graph operations. A support function `sparse_to_csr` performs this simple conversion. All *gaimc* functions also accept a structure with the compressed sparse row arrays pre-computed.

Another advantage of using our own compressed sparse row structure occurs when the graph has edge weights of 0. The Matlab sparse matrix structure removes such entries [Gilbert et al., 1992].

Based on this analysis, we offer some advice to optimize Matlab functions:

- do not use function calls
- evaluate performance of built-in Matlab functions
- question performance assumptions.

While just-in-time compiled Matlab code may not be as fast as optimized C or C++ code, it need not be slow.

6.4.3 *Performance*

In the previous sections, we have reviewed how to implement a heap structure in Matlab, and optimized the performance of an algorithm using the heap. In this section, we compare the performance of *gaimc* to MatlabBGL. Writing native Matlab code does not offer the same performance as native C or C++ code and we evaluate *gaimc* by how much slower the routines are compared with their C++ Boost graph library counterparts. In our tests, we plot the *slowdown* ratio

$$s = \frac{t_{\mathit{gaimc}}}{t_{\mathrm{MatlabBGL}}},$$

where t_{gaimc} is the time of an operation in *gaimc* and $t_{\mathrm{MatlabBGL}}$ is the time of an operation in MatlabBGL. Thus, a slowdown ratio around 1 implies that the routines took roughly the same amount of time.

We evaluate the performance of 6 functions:

1. depth first search (`dfs`),
2. strong components (`scomponents`),

[18] Recall that Matlab stores sparse matrices using the *compressed column format* [Gilbert et al., 1992] and thus we'd need to extract columns for efficiency.

3. Dijkstra's single-source shortest paths (`dijkstra`),
4. directed clustering coefficients (`dirclustercoeffs`),
5. Prim's minimum spanning tree (`prim_mst`), and
6. clustering coefficients (`clustercoeffs`),

in two cases:

- standard Matlab sparse matrix input (standard), and
- "pre-converted" input (fast).

In MatlabBGL and *gaimc*, the input must be transposed or converted to compressed-sparse row arrays, respectively. In the second case, we show the performance with inputs that are already converted, which corresponds to a faster function call.

We evaluate each function on either a small set of sample graphs (`dfs` and `dijkstra`) or a set of synthetic graphs (`scomponents`, `dirclustercoeffs`, `prim_mst`, and `clustercoeffs`). For each function, we call it once to ensure that the Matlab just-in-time compiler has the current version compiled. The two search functions that begin with a source vertex—`dfs` and `dijkstra`—are called on each of the graphs listed in table 6.2 with 100 random starting vertices, and every test is repeated 30 times. The functions `scomponents` and `dirclustercoeffs` are evaluated on 30 instances of random directed graphs with 25 edges per row and 10, 100, 5000, 10000, and 50000 vertices. The function `clustercoeffs` is evaluated similarly, but with random symmetric graphs instead. Finally, the minimum spanning tree function is evaluated on 30 instances of a random symmetric graph with average degree 25 and 100, 5000, and 10000 vertices. The aggregated results of all these tests are shown in figure 6.4.

Graph	Verts.	Edges
allsp1	5	9
clr24-1	9	14
wb-cs.stan	9914	36584
minnesota	2642	3303
tapir	1024	2846

Table 6.2 – `gaimc` evaluation graphs.

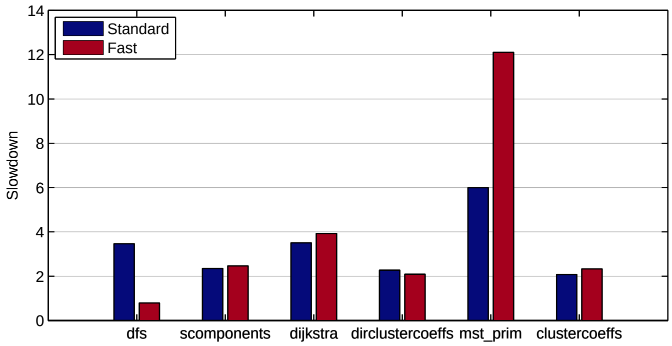


Figure 6.4 – Performance of the `gaimc` library. An experimental comparison of the performance of the *gaimc* library to MatlabBGL shows that many functions in *gaimc* take only twice as much time as their MatlabBGL counterparts. The difference between the standard and fast operations is that fast operations eliminate any data translations and measure pure algorithm speed. Standard calls in these libraries involve some data translation, which is included in the time for the standard operations.

With the exception of `mst_prim`, the *gaimc* functions are roughly 2-4 times slower than their MatlabBGL counterparts. At the moment, we don't understand why the `dfs` function is faster in *gaimc* or why the `mst_prim` routine has dramatically different performance. Exploring these differences is a task for the future.

6.5 LIBBVG AND BVGRAPH'S IN MATLAB

The final software package that we discuss in this chapter is *libbvg* and its Matlab counterpart *bvgraph*. All the source code and examples for these paired packages are online at the LaunchPad open-source hosting repository, `https://launchpad.net/libbvg`.[19]

Both of these packages work with web graphs, which are graphs formed by hyper-linking relationships on the world wide web. These graphs are extremely large—the complete network has over one trillion nodes [Alpert and Hajaj, 2008]—and subsets often have more than one hundred million nodes. Although gigantic, the network and its subsets have considerable structure.

Both *libbvg* and *bvgraph* provide an interface for web graphs compressed in the Boldi-Vigna (BV) scheme [Boldi and Vigna, 2005]. With this scheme, web graphs often use fewer than *three bits* per link. Standard graph storage techniques need more than *four bytes* per link.[20] The Boldi-Vigna compression scheme exploits two empirically observed properties of web graphs to obtain such remarkable compression rates. First, when the URLs of each node are ordered lexicographically, then the link distances ($|i - j|$ where i and j are the indices of the URLs of the link) follow a power-law. Boldi and Vigna designed a special coding scheme to compress these power-law distributed numbers. Second, many URLs within a site repeat the same links. Allowing nodes to "copy" links from previous nodes (within a limited window of previous nodes) allows them to compress these structures. Together, these techniques are incredibly effective and stable—they have compressed web graphs collected between 2001 and 2007 at nearly the same rate (three bits per edge).

The only problem with their compression scheme is that it makes random access to the out-edges of a node less efficient than sequential (or streaming) access. This occurs because the desired out-edges may reside in a node that copies its links from a previous node, which also copies its links from a previous node, and so on. To combat the "infinite" copying, the implementation imposes an optional limit on the maximum copy depth.

[19] We anticipate migrating them to the github system soon.

[20] This estimate assumes that graphs are stored with compressed sparse row arrays with 32-bit indices (four bytes) and without any compression.

Efficient access to large graphs is a powerful experimental tool for algorithms like PageRank. In the remainder of this section, we describe the two libraries and how they let us compute on enormous graphs with limited resources.

6.5.1 *libbvg*

In *libbvg*, we provide a C99 interface for web graphs compressed in the Boldi-Vigna scheme. The library enables both in-core and out-of-core access to these graphs and intends to mirror the Java WebGraph framework [Vigna, 2008].

The following program uses the library to extract the degree for each node in a graph and store them in the array `degs`:

```
bvgraph g = {0};
bvgraph_load(&g, graphfilename, strlen(graphfilename), -1); /* load out-of-core */
bvgraph_iterator iter; unsigned int deg;
double *degs = malloc(sizeof(double)*g.n);
int rval = bvgraph_nonzero_iterator(&g, &iter);
for (; bvgraph_iterator_valid(&iter); bvgraph_iterator_next(&iter)) {
  bvgraph_iterator_outedges(&iter, NULL, &deg); *(degs++) = (double)deg;
}
bvgraph_iterator_free(&iter);
```

The two types `bvgraph` and `bvgraph_iterator` encompass most of the functionality of the library. The `bvgraph` type stores the data about the graph and the `bvgraph_iterator` type sequentially accesses vertices and their edges.

The current *libbvg* does not yet support accessing vertices in the graph randomly. Some of the support for random access exists in an unfinished `bvgraph_random_iterator` structure.

There is little left to the library beyond the two types above. Primarily, this follows because access to out-edges is sufficient to compute PageRank and other simple problems on the graph. In the next section, we discuss an optimized implementation of the power method for PageRank using these data structures.

parallel extensions Before getting to the power method, let us mention that *libbvg* has a few extensions to work with the `bvgraph` data structures in a multi-core environment. The idea is that each core will have its own computation thread and we can assign a contiguous set of nodes to that core. In the implementation, we assign a special nonzero iterator to each thread. These special iterators store their state so that they can begin iterating in the middle of the graph, which eliminates the problem with links copying their predecessors.[21]

[21] See section 5.6.3 for more information about the performance of this technique with PageRank.

6.5.2 *The power method in libbvg*

In program 9, we present an implementation of the power method in *`libbvg`*. Many of the vector operations in the standard power method are gone. This particular code replaces them with a series of combined operations. So that we can understand how the optimizations work, we begin by repeating a simple implementation of the power method from program 2. We stripped the implementation down to the pure computational pieces (and removed all the convenience features).

```
function [x flag reshist]=powerpr(P,a,v,tol,maxit,verbose)
n=size(P,1);
x=zeros(n,1)+v; delta=2; iter=0;
while iter<maxit && delta>tol
    y = a∗(P’∗x);
    w = 1-csum(y);
    y = y + w∗v;
    delta = normdiff(x,y);
    x = y./csum(y);
end
```

In program 9, the vectors `x` and `y` must be initialized before the `bvpr` function, and thus the first few lines of the Matlab implementation are irrelevant. The `y = a∗(P’∗x)` work is handled by the `mult` function. In addition to computing `y`, this function also computes $\mathbf{e}^T\mathbf{y}$ and $\mathbf{d}^T\mathbf{x}$—both using compensated summation. Consequently, once the function returns, line 53 of `bvpr` computes `w` without any additional work. The next line performs the shift `y = y + w∗v` and simultaneously computes `csum(y)`. Line 56 normalizes `y`, computes the difference `normdiff(x,y)`, and resets `x`. Line 57 extracts the compensated sum for `delta` and line 58 swaps the interpretation of `x` and `y`. Note that, before the swap, `y` held the newly normalized iterate and `x` was reset to 0. Thus, the program is ready for another iteration after the swap.

In total, this approach utilizes two passes of the memory of `x` and `y`: the first in line 54 and the second in line 56. In contrast, the Matlab code uses 5 passes over the memory of `x` and `y`. Vectorized computations, such as the Matlab code, sometimes sacrifice some performance.[22]

6.5.3 *bvgraph*

The last section discussed a case when using a high-level language like Matlab incurred a possible performance penalty. Nevertheless, using Matlab often increases productivity in other aspects of work. Our final package is *`bvgraph`*, which is a Matlab class to work with *`libbvg`*.

[22] Furthermore, we also apply multi-threading to program 9 to take advantage of working on a multi-core system.

Program 9 – An optimized power method.

```
/* define macros for compensated summation */
#define CSUM(x,y,t,z) { t=y[0]; z=(x)+y[1]; y[0]=t+z; y[1]=(t-y[0])+z; }
#define FCSUM(y) (y[0]+y[1])

/** Compute a matrix vector product with a substochastic bvgraph structure
 *
 * y = y + alpha*P’*x where P = D^{-1} A for the adjacency matrix A given
 * by a bvgraph structure.
 *
 * g: the bvgraph; x: the right hand vector; y: the result vector (see above)
 * alpha: the value of alpha (see above);
 * sum_aPtx: the value e^T (alpha*P’*x); sum_dtx: the value d^T x
 */
int mult(bvgraph *g, double *x, double *y, double alpha,
            double *sum_aPtx, double *sum_dtx) {
    using namespace std;
    bvgraph_iterator git; int *links; unsigned int i, d;
    bvgraph_nonzero_iterator(g, &git);               /* omit error checking */
    double id=0.0; double sumy[2]={0},t,z; double dtx[2]={0};
    for (; bvgraph_iterator_valid(&git); bvgraph_iterator_next(&git)) {
        bvgraph_iterator_outedges(&git, &links, &d); /* get outlinks */
        if (d > 0) { id = 1.0/(double)d; } else { CSUM(x[git.curr],dtx,t,z); }
        for (i = 0; i < d; i++) {
            y[links[i]] += alpha*x[git.curr]*id;
            CSUM(alpha*x[git.curr]*id,sumy,t,z); /* update the running sum */
        }
    }
    if (sum_aPtx) { *sum_aPtx = FCSUM(sumy); }
    if (sum_dtx) { *sum_dtx = FCSUM(dtx); }
    bvgraph_iterator_free(&git);
    return (0);
}

/** Solve PageRank using the power method with uniform teleportation
 *
 * For the strongly preferential model of PageRank with uniform
 * teleportation, this algorithm computes a vector x such that
 * x \approx alpha (P + dv’)’*x + (1-alpha) ve’*x
 *
 * g: the bvgraph; alpha: the value of alpha in the computation;
 * tol: the stopping tolerance; maxiter: the maximum number of iterations
 * x: a vector length n initialized to 1/g.n in each component
 * y: a vector initialized to 0 in each component
 * return: the pointer to the vector (x or y) that contains the solution
 */
double* bvpr(bvgraph* g, double alpha, double tol, int maxit, double *x, double *y)
 {
    size_t n = (size_t)g->n, n1;
    int iter = 0; double delta = 2, sumy, dtx, ny, *xi, *yi, t, z;
    simple_time_clock(&start);
    while (delta > tol && iter++ < maxit) {
        if (mult(g, x, y, alpha, &sumy, &dtx)) { return (NULL); }
        double w = (alpha*dtx+(1-alpha))/(double)n,nys[2]={0}; delta=0.0;
        n1=n;yi=y; while (n1-->0) { (*yi)+=w; CSUM(fabs(*yi++),nys,t,z); }
        n1=n;yi=y;xi=x;ny=FCSUM(nys);nys[0]=0.;nys[1]=0.; /* compute norm y */
        while (n1-->0) {(*yi)/=ny; CSUM(fabs(*yi-*xi),nys,t,z); *xi=0.; xi++; yi++;}
        delta=FCSUM(nys); /* get the current change */
        { double *temp; temp = x; x = y; y = temp; } /* swap */
    }
    return x;
}
```

The bvgraph class (in the *bvgraph* package) creates a Matlab object that presents a *libbvg* bvgraph type as an adjacency matrix. For example,

```
G = bvgraph('wb-cs.stanford');
d = sum(G,2);
D = diag(G);
y = G*rand(n,1);
```

loads the file wb-cs.stanford.graph into memory with *libbvg* and then computes the degrees of each node by summing a row of the "matrix"; extracts the diagonal entries of the "matrix"; and performs a "matrix"-vector product. The "matrix" in this case is the adjacency matrix of the **wb-cs.stan** graph. With the bvgraph class, this graph always stays compressed and can reside in main memory (RAM) for faster access, or be streamed from disk when memory is tight.

The focus of this thesis is PageRank, and we are willing to spend a bit more time to make PageRank work efficiently. Using just the bvgraph class, unfortunately, makes PageRank inefficient. Consider the matrix-vector product with $\mathbf{P}$ given only the adjacency matrix G:

```
d = sum(G,2); id = full(spfun(@(x) 1./x, sparse(d)));
y = G'*(x.*id);
```

This computation requires storing an extra vector id. For a graph with 10^8 nodes (such as **webbase-2001**), an extra vector takes nearly one gigabyte of memory. For many machines, storing an extra vector of memory may not be possible. To address this problem, we introduce the ssbvgraph class:

```
P = ssbvgraph(G);
y = P'*x;
```

The ssbvgraph class is a *sub-stochastic* bvgraph, because it computes a sub-stochastic matrix-vector product. It computes this product implicitly by taking advantage of the same implicit normalization that program 9 used. Using this class, we can substitute a ssbvgraph object for a sub-stochastic matrix in many of the Matlab codes for PageRank given throughout this thesis.[23]

Together, these classes make it easy to work with huge graphs in Matlab. They introduce minimal memory overhead, and integrate nicely with standard Matlab matrix operations.

We now end our discussion of the three software packages composed for this thesis.

[23] The ssbvgraph class works with both program 2 (the power method), program 5 (Monte Carlo RAPr), program 6 (path damping RAPr), program 7 (quadrature RAPr), and and program 8 (the inner-outer iteration). It does not work with any of the Gauss-Seidel routines.

6.6 PUBLICATION PACKAGES

Britian's Royal Society is the first group of academics to gather regularly and discuss what we now call science. A key feature of their gatherings was the presentation of experiments. Experimental evidence and demonstration were critical aspects of this group. It is disturbing how common the lack of experimental demonstration has become. Papers have experimental results, but public demonstration of these results is rare. Some scientists release their experimental codes, others do not.

This situation is unacceptable.

Computational results are some of the easiest to publicly demonstrate and share. Within this thesis, the majority of the experiments work with roughly 200 lines of code. This code is part of the thesis, incorporated into the text, and publicly released. Furthermore, the experimental scripts for the thesis are also publicly available. Anyone with access to similar computational resources and a bit of patience has the capability of reproducing our results.

Furthermore, a similar public disclosure holds for the publications supporting this thesis. We now describe the three *publication* packages.

RAPR Computing the Random Alpha PageRank vectors involves merely a few lines of Matlab codes, as program 5, program 6, and program 7 show. In the RAPr[24] package, we provide these codes along with all of the experimental routines. These routines reproduce all the figures in chapter 4 and even include other experiments that are not incorporated into the text. Within these codes, we also provide the C++ routines to compute the RAPr statistics using *`libbvg`*.

INNOUT We decided to pun on the name of the popular California burger chain "In-n-out" with the name of the package corresponding to the inner-outer iteration.[25] Just like the RAPrpackage, the *`innout`* package contains all the Matlab codes, C and C++ programs, and experimental drivers to reproduce the figures in chapter 5.

THESIS A forthcoming package includes all the codes generated for extra figures in this thesis.[26] Some of these programs use the RAPr and *`innout`* packages to derive new results. One example is the list of pages in Wikipedia with largest derivative from table 3.3. Another example is the derivative code itself.[27]

We hope that releases like these become standard in the future.

[24] Available online: `http://stanford.edu/~dgleich/publications/2008/rapr`.

[25] Available online: `http://stanford.edu/~dgleich/publications/2009/innout`.

[26] Available online: `http://stanford.edu/~dgleich/publications/2009/thesis`.

[27] While there was never a prior derivative package, our derivative routine was released in an older Matlab PageRank package.

REPRODUCIBLE RESEARCH There is some cause for hope. Jon Claerbout at Stanford has long advocated *reproducible research* for scientific computations [Schwab et al., 2000]. Others have adopted his philosophy [Buckheit and Donoho, 1995; Donoho et al., 2009]. Indeed, an entire issue of the Computing in Science and Engineering journal was recently devoted to this topic [Fomel and Claerbout, 2009]. The software packages of this thesis fit the general motivation of reproducible research, although some of the specifics differ.

SUMMARY

Three software packages support the computations in this thesis.

MATLABBGL This library brings the power of the Boost graph library to Matlab. It uses Matlab's own sparse matrix type as the adjacency matrix of a graph and supports a wide range of graph analysis algorithms.

GAIMC In contrast with MatlabBGL, the `gaimc` package is small and compact, but adds graph algorithms to Matlab in the same way. It is designed for cases when performance is not paramount.

LIBBVG/BVGRAPH These paired libraries make it easy to work with extremely large web graphs in C and Matlab. Internally, they use graphs compressed in the Boldi-Vigna scheme [Boldi and Vigna, 2005], and keep them compressed throughout the computations.

Finally, we briefly highlight three publication packages that make all the results of two publications and this thesis completely reproducible.

That it should come to this!

—Hamlet

7 CONCLUSION

At the outset of this thesis, we embarked on an exploration of α. Recall the setting. PageRank is a technique to rank the nodes of any graph by their importance. All too often, people introduce PageRank with the idea that "important nodes" connect to other "important nodes." Such a definition suggests an importance vector $\mathbf{x}$ that satisfies

$$\mathbf{P}\mathbf{x} = \mathbf{x},$$

where $\mathbf{P}$ is a column stochastic matrix describing a flow of importance. Typically, importance flows uniformly along the edges of the graph. But, these introductions ignore α, and it is α that distinguishes PageRank! PageRank needs α because $\mathbf{P}\mathbf{x} = \mathbf{x}$ creates a model where the importance scores, or ranks of the nodes, are not well defined.

Instead, better definitions of PageRank begin with α. We suggest a few possibilities. First, "important pages" *probably* connect to other "important pages." The value of α arises immediately to quantify the term *probably* in this definition. Another possibility is to begin outright with

α: a parameter between 0 and 1 to reduce the flow of influence in a graph.

Or, perhaps

α: the probability that a random surfer in the web follows a link.

This last definition ties PageRank too closely to web search, however. Beginning with all of these definitions quickly leads to PageRank itself:

$$(\mathbf{I} - \alpha\mathbf{P})\mathbf{x} = (1 - \alpha)\mathbf{v}.$$

In the first two definitions, $\mathbf{v}$ is any intrinsic measure of node importance. Without other information, a uniform choice is entirely appropriate as the choice of intrinsic importance. In all definitions though, importance flows as $\alpha\mathbf{P}$—and that is the key to PageRank.

Given any definition of PageRank, it must involve α, and it is surprising how little attention α received in early discussions. The first five years of PageRank research largely ignored the impact of α. That finally changed. It almost seemed as if α were "in the air," to borrow a phrase from my late adviser. A stream of papers between 2003 and 2007 attacked α directly. These attacks established that PageRank is a rational function α [Boldi et al., 2005], examined the parametric structure of the so-called Google matrix definition of PageRank [Serra-Capizzano, 2005], and even proposed a theoretically motivated choice of α [Avrachenkov et al., 2007]—among other contributions, of course.

This thesis fits the canon of that attack. We focus on the interaction between α and PageRank from the perspective of sensitivity analysis in three ways.

THE DERIVATIVE First, PageRank is a rational function of α. and a simple measure of the stability or sensitivity of such a function is the derivative. PageRank's derivative with respect to α satisfies

$$(\mathbf{I} - \alpha\mathbf{P})\mathbf{x}'(\alpha) = \mathbf{P}\mathbf{x}(\alpha) - \mathbf{v}.$$

Chapter 3 begins with this derivative. We provide two theoretical contributions: a discussion of mathematical formulations of the PageRank derivative (section 3.1) and a theorem relating a first order Taylor step along the derivative to another PageRank vector (theorem 7). Furthermore, we introduce a new algorithm to compute the derivative using any existing techniques to compute PageRank (section 3.2).

RANDOM ALPHA Second, the random surfer model for PageRank on web pages suggests that the value of α ought to reflect the probability of *real people* following links when browsing the web. Chapter 4 embraces this view and follows it to its logical conclusion. Because PageRank is a nonlinear function of α, the PageRank vector with any aggregate probability α is incorrect and we need to regard α as a random variable distributed according to *all* the probabilities of following a link. In order to explore the resulting model computationally (section 4.6), we employ techniques from the uncertainty quantification community, which were developed to solve partial differential equations models with random variables. When α is random, the PageRank vector itself is also random, and the standard deviation of the random variables in the PageRank vector produces a new sensitivity measure for PageRank (section 4.2). We also present an empirically measured distribution of α values for around 2,000,000 people (section 4.5) and show that the standard deviation vector aids a spam classification task (section 4.8.4). This chapter contains rigorous error analysis and convergence theory for all of the algorithmic techniques (section 4.7).

PAGERANK SOLVERS Third, the problem of computing a PageRank vector for a particular value of α can be formulated as computing a sequence of PageRank vectors with a β smaller than α. To wit, PageRank with $0 < \beta < \alpha$ solves PageRank at any desired α. The resulting inner-outer iteration for PageRank is discussed in chapter 5. Sensitivity does not arise directly in these ideas. Rather, the interplay of α and β in an efficient computation depends on many ideas related to the structure of the PageRank function itself, which inevitably leads to some sensitivity interpretations. The inner-outer computation is also one of the most efficient PageRank solvers, and we use it throughout the preceding chapters to compute PageRank vectors for the other sensitivity analyses.

Three conclusions from this thesis are:

α matters, but don't pick it;
everything is just PageRank, so make it fast; and
don't ignore sensitivity, it could help.

In all of the experiments, we pick a value of α. Just as illustrated in the introduction (chapter 1), changing α produces a different PageRank vector. Nothing in this thesis solves this problem. We offer an alternative, however.

Don't pick α.
Pick a distribution for a random α instead.

Choosing an appropriate distribution does not change the sensitivity because changing the distribution affects the new random PageRank vector, too. Sometimes—like in web search—there is a natural distribution to use. Otherwise, consider a uniform distribution. Even though the distribution may not be perfect, the distribution produces a useful sensitivity measure: the standard deviation.

For both sensitivity analyses, the derivative and the standard deviation, the key computational technique was PageRank itself! Using both models reduces to solving a few PageRank problems and then deriving the results from these PageRank vectors. The inner-outer iteration even solves PageRank using PageRank, albeit with a smaller value of α. These results seem like a fluke. But they show the remarkable flexibility of PageRank as a function of α. The devil's advocate will argue that such results are expected from an exploration of sensitivity analysis. This objection is unfounded. Consider the conversion from the derivative into PageRank problems. It is surprising that this conversion is possible because it depends crucially on the structure of the PageRank problem. Investing the effort in a speedy PageRank solver enables these, and other, experiments.

The sensitivity measures helped the spam classification task. Nothing in the design of these measures is tuned to spam identification. This suggests that using the sensitivity vectors in other applications may produce similar improvement. Thus, do not ignore sensitivity.

7.1 DISCUSSION

Will this thesis matter? Predicting the future is a difficult problem best avoided in this case. Instead, let us critically address a few points raised by this thesis.

- Is PageRank research still useful?
- Is picking a distribution for α really helpful?
- Why use such strict tolerances in your computations?
- What about ties in the PageRank vector?

We address each question in order.

7.1.1 *Is PageRank research still useful?*

The death of PageRank has been forecast since 2003 [Zawodny, 2003]. Zawodny claims that the success of PageRank necessarily induces its future failure. Because PageRank utilizes the link structure of the web, it originally produced useful information for web ranking. But, the impact of PageRank on web search caused people to change their link structures to manipulate PageRank. Thus, links on the web will become less reliable over time.

It is now 2009, and Google still uses PageRank [Cutts, 2009]. Rumors about its death are greatly exaggerated, apparently.

In fact, Cutts [2009] discusses a critical change in the PageRank formulation used by Google. The change is that they no longer construct a $0, 1$ sub-stochastic matrix from the link structure, but construct a general sub-stochastic matrix instead.[1] This change shows that PageRank is still useful to Google, and thus research on it matters.

On the other hand, Najork et al. [2007] claimed that PageRank is one of the least effective measures in a machine learning framework for web search.

[1] Formally, for the new sub-stochastic matrix $\bar{\mathbf{P}}$ we find that $\mathbf{e}^T\bar{\mathbf{P}}$ can be any number between 0 and 1. These column-sums need not be 0 or 1 as in the formulation in chapter 2.

It is worse, incredibly, than a measure based on in-degree. They conjecture that this phenomenon arises from link manipulation to enhance PageRank. Thus, it would seem that they empirically confirm Zawodny's claim. If this is true, why would Google still use PageRank?

We believe that the resolution relies in *how* Google uses PageRank. Recently, Becchetti et al. [2008] show that metrics derived from PageRank are helpful in identifying spam pages. PageRank also helps web crawling operations [Lee et al., 2008]. Google supposedly uses PageRank to influence the crawling rate as well [Cutts, 2006]. These two tasks are fundamentally different from determining the order of web search results.

PageRank *is still useful*.

7.1.2 *Is picking a distribution for α really helpful?*

In chapter 4, we argue that the RAPr model suggests an obvious choice for α and the distribution of α: use the values obtained by studying surfers. Critics will object: these choices may not yield the best results for web search or spam detection. We agree, and our web spam analysis supports this objection.

Our point is that, regardless of the application, picking a distribution tends to yield more information about the graph. Some of the information is correlated with PageRank (the expectation) and some appears uncorrelated (the standard deviation). Picking a distribution gives us more flexibility to obtain a "best" vector. For our spam example, the best results occurred with a distribution that looks nothing like the empirically measured distribution.

In terms of sensitivity, picking a distribution and using the standard deviation seems superior to using a sensitivity measure from the derivative. Values of the derivative are difficult to interpret, whereas the standard deviation sensitivity values have a natural probabilistic interpretation.

Our advice: Just pick a distribution.

7.1.3 *Why use such a strict tolerance in your computation?*

In all the PageRank computations throughout the thesis, we computed PageRank vectors with a strict tolerance, typically tighter than 10^{-10}. These vectors are needlessly accurate. Many applications use PageRank vectors with loose tolerances around 10^{-4} [Kamvar et al., 2003].

We felt that computing PageRank accurately was necessary to distinguish between the effects of our new sensitivity measures and the effects due to inaccurate computations. Many of the PageRank values are small and a few are quite large. Distinguishing differences among the small values implies we should use a strict tolerance.

Also, the real answer to the tolerance question is: because we can. The graphs studied in this thesis are small compared with industrial web graphs. In this sense, the difference in computation time for extra accuracy is meaningless. There are no application requirements we have to meet, so why not get extra accuracy?

7.1.4 *What about ties in the PageRank vector?*

At various points in this thesis, we illustrate a PageRank vector with an ordered list. For the nodes with highest PageRank, showing an ordered list is okay because the top few PageRank values are clearly separated from each other. The remainder of the PageRank vector, however, is often riddled with tied values. These ties are identical floating-point numbers and not just values within the machine precision tolerance. Exact floating-point ties occur when two pages have identical in-links, the value of $\mathbf{v}$ is the same on both pages, and the PageRank solver is invariant to permutations.[2] While we do not attempt to quantify the total number of ties—they do seem to be common.

This affects the results here in two ways. First, the Kendall-τ computation requires the order of the nodes. We used a version of τ that incorporates tied values, however. Second, the intersection similarity measure also uses a ranked order. This computation may change in the presence of ties. We only expect a slight change as the measure itself is considerably less sensitive to tied values. This follows because of the smoothing effect in the running average nature of the metric.

In short, ties are a problem with some PageRank computations, but we do not expect them to alter the results of this thesis in any meaningful way.

7.2 FUTURE WORK

We are almost done. There are some small loose ends (dangling nodes?) to address (connect?). Each is an idea to improve a piece of the results.

7.2.1 *RAPr speed*

Although Gauss quadrature is the fastest method to compute the expectation and standard deviation in the RAPr model, it is slow. We need to solve many PageRank problems at values of α that are close to 1. These vectors take considerable computation time.

[2] It is worth noting that Gauss-Seidel algorithms are not invariant to permutations. This may suggest that they are less reliable for ranking purposes.

Gauss-Turán quadrature [Gautschi, 2004] uses derivatives in a quadrature rule. If we use $2s$ derivatives, then we get a rule with degree of exactness $(2s+2)n-1$. The error estimates extend to $O(\rho^{-(2s+1)n})$ accuracy for certain weight functions [Milovanović and Spalević, 2003]. In other words, it satisfies all the properties of the Gauss quadrature rule, but uses derivatives instead of extra nodes. Computing PageRank derivatives can be done at the *same* value of α. Switching to this new quadrature rule seems a promising idea to speed the RAPr computations.

One concern is that computing the nodes and weights for the Gauss-Turán quadrature rule involves solving a set of nonlinear equations. Algorithms exist to compute the weights. A brief test using the `turan.m` function from Gautschi [2002] showed convergence problems for all but the most trivial rules. For example, the Newton iteration did not converge for more than three quadrature points. This suggests that high-precision numerical codes using Mathematica may be required. One possible implementation is in the "OrthogonalPolynomials" Mathematica package [Cvetković and Milovanović, 2004] but it does not appear to be publicly available.

It seems that using these rules first requires improving their computation.

7.2.2 *Random modifications to other methods*

PageRank has a beautiful interpretation with a random parameter, but there are many other numerical methods to rank web pages. HITS [Kleinberg, 1999] and SALSA [Lempel and Moran, 2000] are two examples. Both of these methods are parameter free, and there is no analog of α to convert to a random value. We did not conduct an exhaustive survey to locate methods that would benefit from such a treatment, but there are many variations of PageRank—e.g. TrustRank [Gyöngyi et al., 2004], N-step PageRank [Zhang et al., 2007]—and some should benefit.

At least one algorithm for learning a function on a graph includes a regularization parameter that is similar to PageRank [Zhou et al., 2005]. Computing a function with error bounds seems an exciting possibility for our methods.

Finally, there are many mathematical models that describe a process on a network (e.g. [Atkinson, 2009]). Using random parameters should provide new insights into analyses from these models.

7.2.3 *Tensor problems*

Tensors are generalizations of indexed data, e.g. a matrix, with more than two indices. Modern tensor theory and applications of tensor analysis are quickly evolving fields, see Kolda and Bader [2009] for a recent survey. One application synthesizing tensors and PageRank is the sports ranking problem from Govan et al. [2008]. They form a linear combination of multiple frames of a tensor:

$$(\mathbf{I} - \alpha_1 \mathbf{P_1} - \alpha_2 \mathbf{P_2} - \cdots - \alpha_k \mathbf{P_k})\mathbf{x} = (1 - \textstyle\sum \alpha_j)\mathbf{v}.$$

Using ideas from Constantine [2009], we could analyze this equation as a multi-variate parameterized matrix problem. Similar techniques will apply to other types of "tensor-frame" problems.

We would have liked to experiment with these ideas, but there is only so much time. Perhaps we will be able to explore these techniques in the future.

7.2.4 *gaimc performance*

We have not discussed the software chapter at all in the current chapter. That is not to imply that it is not important, but rather the conclusions of the thesis are about PageRank and not software.

We would like to conclude that using the MATLAB code in *gaimc* is nearly as fast as using the complicated MatlabBGL package. Our current performance results indicate that this occurs for a few of the functions, but not for others.

The next step on the way to this conclusion is to track down these performance discrepancies and understand why they are unavoidable or produce a solution.

7.2.5 *Inner-Outer extensions*

We ended the chapter on the inner-outer iteration with a conjecture about its asymptotic performance. Namely, the asymptotic convergence rate of the inner-outer algorithm seems to be $\alpha\lambda_2$, where λ_2 is the first eigenvalue with $|\lambda_2| < 1$, even though it runs iterations of the power method, which converge at rate α. As written, the conjecture is false.[3] For many large graphs, it appears to be a close approximation of what happens. We hope to formalize this conjecture and establish when and why it holds.

Rigorously proving such a conjecture is incredibly important because we are not aware of any other iterative algorithm that always computes PageRank faster than the power method with similar memory requirements. With the inner-outer iteration, it seems possible.

[3] Using $\mathbf{P} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$ produces a counterexample.

7.2.6 *PageRank as a function of a matrix*

It would be nice to be able to compare the PageRank derivative and the RAPr models directly. Each comes from a totally different approach, but perhaps there are some similarities. As luck would have it, there is a framework where we can simultaneously look at both models: PageRank as a function of a matrix.

Recall that PageRank is a vector rational function of α. A rational function is a special case of an analytic function and analytic functions of scalars have equivalent matrix functions. Formally, if $f(x)$ is an analytic function of x, then $f(\mathbf{A})$ is well defined for any square matrix $\mathbf{A}$. Thus, we propose the PageRank function of a matrix:

$$\mathbf{x}(\mathbf{A}).$$

When $\mathbf{A} = \left[\begin{smallmatrix} \alpha & 1 \\ 0 & \alpha \end{smallmatrix}\right]$ then $\mathbf{x}(\mathbf{A})$ contains both the PageRank vector and its derivative. When $\mathbf{A} = \mathbf{J}$, the Jacobi matrix for the orthogonal polynomials on the distribution of a random variable A, then $\mathbf{A}$ computes $\mathrm{E}\left[\mathbf{x}(A)\right]$!

While both vectors fit into this model, it does not seem possible to compare them further.

Nevertheless, the PageRank function of a matrix is a tantalizing generalization of the PageRank problem. It is somewhat of an aesthetic generalization because we have no compelling uses for it; although, the equivalence between the linear system formulation of PageRank and the eigensystem and Markov chain interpretation finally disappears. There are many aspects of this thesis that included results for both the PageRank linear system, which tended to be easy to derive, and the PageRank eigensystem, which tended to be difficult to derive. It is refreshing to conclude there are some differences between them.

SUMMARY

PageRank—with its elegant simplicity and enthralling complexity—captured the hearts and minds of scientists and researchers over the past decade. They have formalized, quantified, extended, reinterpreted, and improved many aspects of the PageRank problem and algorithms. With this thesis, we join this global enterprise with our contributions: (i) some new understanding of the PageRank derivative; (ii) a new generalization of PageRank; and (iii) a faster PageRank algorithm.

Research is never perfect and the discussion in this chapter (section 7.1) addresses some aspects of these contributions that are not ideal.

But the excitement of research is all the work that is yet come.

> *Now this is not the end.*
> *It is not even the beginning*
> *of the end. But it is, perhaps,*
> *the end of the beginning.*
>
> —Winston Churchill

BIBLIOGRAPHY

[Agresti, 2002] A. Agresti. *Categorical Data Analysis*, John Wiley and Sons, 2002. Cited on page 79.

[Alexandrescu, 2001] A. Alexandrescu. *Modern C++ Design: Generic Programming and Design Patterns Applied*, Addison-Wesley Professional, 2001. Cited on page 135.

[Alpert and Hajaj, 2008] J. Alpert and N. Hajaj. *We knew the web was big ....* Official Google Blog. Available online `http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html`, 2008. Accessed on July 11, 2009. Cited on pages 22 and 152.

[Andersen et al., 2006] R. Andersen, F. Chung, and K. Lang. *Local graph partitioning using PageRank vectors*. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*. 2006. Cited on pages 7, 13, 32, and 68.

[Arasu et al., 2002] A. Arasu, J. Novak, A. Tomkins, and J. Tomlin. *PageRank computation and the structure of the web: Experiments and algorithms*. In *Proceedings of the 11th international conference on the World Wide Web*. 2002. Poster session. Cited on pages 28 and 106.

[Asmussen and Glynn, 2007] S. Asmussen and P. W. Glynn. *Stochastic Simulation: Algorithms and Analysis*, Springer, 2007. Cited on pages 79 and 81.

[Atkinson, 2009] M. P. Atkinson. *Mathematical Models of Terror Interdiction*. Ph.D. thesis, Stanford University, 2009. Cited on page 165.

[Avrachenkov et al., 2007] K. Avrachenkov, N. Litvak, and K. S. Pham. *Distribution of PageRank mass among principle components of the web*. In *Proceedings of the 5th Workshop on Algorithms and Models for the Web Graph (WAW2007)*, pp. 16–28. 2007. `doi:10.1007/978-3-540-77004-6_2`. Cited on pages 34, 63, 69, and 160.

[Baeza-Yates et al., 2006] R. Baeza-Yates, P. Boldi, and C. Castillo. *Generalizing PageRank: Damping functions for link-based ranking algorithms*. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR2006)*, pp. 308–315. 2006. `doi:10.1145/1148170.1148225`. Cited on pages 70 and 76.

[Baeza-Yates and Ribeiro-Neto, 1999] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*, Addison Wesley, 1999. Cited on page 1.

[Barabási et al., 1999a] A.-L. Barabási, H. Jeong, and R. Albert. *Diameter of the world wide web*. Nature, 401, pp. 130–131, 1999a. `doi:10.1038/43601`. Cited on page 45.

[Barabási et al., 1999b] ———. *The www network*. Accessed online via `http://www.nd.edu/~networks/resources.htm` in May 2009, 1999b. Cited on page 45.

[Batagelj and Zaversnik, 2003] V. Batagelj and M. Zaversnik. *An O(m) algorithm for cores decomposition of networks*. arXiv, 2003. `arXiv:cs/0310049v1`. Cited on page 145.

[Bayati et al., 2009] M. Bayati, M. Gerritsen, D. F. Gleich, A. Saberi, and Y. Wang. *Algorithms for large, sparse network alignment problems*. arXiv, 0907.3338, p. Online, 2009. `arXiv:0907.3338`. Cited on page 125.

[Becchetti et al., 2008] L. Becchetti, C. Castillo, D. Donato, R. Baeza-Yates, and S. Leonardi. *Link analysis for web spam detection*. ACM Trans. Web, 2 (1), pp. 1–42, 2008. `doi:10.1145/1326561.1326563`. Cited on pages 101, 103, and 163.

[Berkhin, 2005] P. Berkhin. *A survey on PageRank computing*. Internet Mathematics, 2 (1), pp. 73–120, 2005. Cited on pages 47 and 53.

[Berkhin et al., 2008] P. Berkhin, U. M. Fayyad, P. Raghavan, and A. Tomkins. *User-sensitive PageRank*. United States Patent Application 20080010281, 2008. Cited on page 69.

[Berman and Plemmons, 1994] A. Berman and R. J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences*, SIAM, 1994. Cited on page 16.

[Bianchini et al., 2005] M. Bianchini, M. Gori, and F. Scarselli. *Inside PageRank*. ACM Transactions on Internet Technologies, 5 (1), pp. 92–128, 2005. `doi:10.1145/1052934.1052938`. Cited on page 32.

[Boldi, 2005] P. Boldi. *TotalRank: Ranking without damping*. In *Poster Proceedings of the 14th international conference on the World Wide Web (WWW2005)*, pp. 898–899. 2005. Cited on pages 70, 71, and 99.

[Boldi et al., 2004] P. Boldi, B. Codenotti, M. Santini, and S. Vigna. *UbiCrawler: A scalable fully distributed web crawler*. Software: Practice & Experience, 34 (8), pp. 711–726, 2004. `doi:10.1002/spe.587`. Cited on pages 40 and 45.

[Boldi et al., 2007] P. Boldi, R. Posenato, M. Santini, and S. Vigna. *Traps and pitfalls of topic-biased PageRank*. In *WAW2006, Fourth International Workshop on Algorithms and Models for the Web-Graph*, pp. 107–116. 2007. `doi:10.1007/978-3-540-78808-9_10`. Cited on pages 18, 54, and 95.

[Boldi et al., 2005] P. Boldi, M. Santini, and S. Vigna. *PageRank as a function of the damping factor*. In *Proceedings of the 14th international conference on the World Wide Web (WWW2005)*. 2005. `doi:10.1145/1060745.1060827`. Cited on pages 33, 34, 47, 52, and 160.

[Boldi et al., 2009] ———. *Permuting web graphs*. In *WAW '09: Proceedings of the 6th International Workshop on Algorithms and Models for the Web-Graph*, pp. 116–126. 2009. `doi:10.1007/978-3-540-95995-3_10`. Cited on page 130.

[Boldi and Vigna, 2004] P. Boldi and S. Vigna. *The Webgraph Framework I: Compression techniques*. In *Proceedings of the 13th international conference on the World Wide Web*, pp. 595–602. 2004. `doi:10.1145/988672.988752`. Cited on pages 80, 98, 107, and 121.

[Boldi and Vigna, 2005] ———. *Codes for the world wide web*. Internet Mathematics, 2 (4), pp. 407–429, 2005. Cited on pages 40, 45, 107, 116, 129, 152, and 158.

[Brandes and Erlebach, 2005] U. Brandes and T. Erlebach, editors. *Network Analysis: Methodological Foundations*, Springer, 2005. `doi:10.1007/b106453`. Cited on pages 4 and 173.

[Brauer, 1952] A. Brauer. *Limits for the characteristic roots of a matrix. IV: Applications to stochastic matrices*. Duke Mathematics Journal, 19 (1), pp. 75–91, 1952. `doi:10.1215/S0012-7094-52-01910-8`. Cited on page 38.

[Buckheit and Donoho, 1995] J. B. Buckheit and D. L. Donoho. *WaveLab and reproducible research*. In *Wavelets and statistics*, pp. 55–81. Springer-Verlag, 1995. Cited on page 158.

[Castillo et al., 2006] C. Castillo, D. Donato, L. Becchetti, P. Boldi, S. Leonardi, M. Santini, and S. Vigna. *A reference collection for web spam*. SIGIR Forum, 40 (2), pp. 11–24, 2006. `doi:10.1145/1189702.1189703`. Cited on pages 45, 97, and 101.

[Catledge and Pitkow, 1995] L. D. Catledge and J. E. Pitkow. *Characterizing browsing strategies in the world-wide web*. Computer Networks and ISDN Systems, 27 (6), pp. 1065–1073, 1995. `doi:10.1016/0169-7552(95)00043-7`. Cited on page 69.

[Chan et al., 1983] T. F. Chan, G. H. Golub, and R. J. LeVeque. *Algorithms for computing the sample variance: Analysis and recommendations*. The American Statistician, 37 (3), pp. 242–247, 1983. `doi:10.2307/2683386`. Cited on page 81.

[Chen et al., 2007] P. Chen, H. Xie, S. Maslov, and S. Redner. *Finding scientific gems with Google's PageRank algorithm*. Journal of Informetrics, 1 (1), pp. 8–15, 2007. `doi:10.1016/j.joi.2006.06.001`. Cited on pages 63 and 69.

[Chien et al., 2004] S. Chien, C. Dwork, R. Kumar, D. R. Simon, and D. Sivakumar. *Link evolution: Analysis and algorithms*. Internet Mathematics, 1 (3), pp. 277–304, 2004. Cited on page 30.

[Cho and Schonfeld, 2007] J. Cho and U. Schonfeld. *RankMass crawler: a crawler with high personalized PageRank coverage guarantee*. In *VLDB 2007: Proceedings of the 33rd international conference on very large data bases*, pp. 375–386. 2007. Cited on page 41.

[Constantine, 2009] P. G. Constantine. *Spectral methods for parameterized matrix equations*. Ph.D. thesis, Stanford University, 2009. Cited on pages 65 and 166.

[Constantine and Gleich, 2007] P. G. Constantine and D. F. Gleich. *Using polynomial chaos to compute the influence of multiple random surfers in the PageRank model*. In *Proceedings of the 5th Workshop on Algorithms and Models for the Web Graph (WAW2007)*, pp. 82–95. 2007. `doi:10.1007/978-3-540-77004-6_7`. Cited on pages 10 and 64.

[Constantine et al., 2009] P. G. Constantine, D. F. Gleich, and G. Iaccarino. *Spectral methods for parameterized matrix equations*. arXiv, 2009. `arXiv:0904.2040`. Cited on page 64.

[Cormen et al., 2001] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*, MIT Press, Cambridge, Massachusetts, second edition, 2001. Cited on pages 145 and 146.

[Cuil, 2009] Cuil. *Cuil – FAQs (item 10).* Available online `http://www.cuil.com/info/faqs/`, 2009. Accessed on July 11, 2009. Cited on page 22.

[Cutts, 2006] M. Cutts. *Q&A march 27, 2006*. Matt Cutts: Gadgets, Google, and SEO. Available online `http://www.mattcutts.com/blog/q-a-thread-march-27-2006/`, 2006. Cited on page 163.

[Cutts, 2009] ———. *PageRank scupting*. Matt Cutts: Gadgets, Google, and SEO blog. Available online `http://www.mattcutts.com/blog/pagerank-sculpting/`, 2009. Cited on page 162.

[Cvetković and Milovanović, 2004] A. S. Cvetković and G. V. Milovanović. *The Mathematica package 'OrthogonalPolynomials'*. Facta universitatis, 19, pp. 17–36, 2004. Cited on page 165.

[Davis and Rabinowitz, 1984] P. J. Davis and P. Rabinowitz. *Methods of Numerical Integration*, Academic Press, New York, 2nd edition, 1984. Cited on page 89.

[Davis, 2007] T. Davis. *University of Florida sparse matrix collection*. `http://www.cise.ufl.edu/research/sparse/matrices/`, NA Digest, vol. 92, no. 42, October 16, 1994; NA Digest, vol. 96, no. 28, July 23, 1996; and NA Digest, vol. 97, no. 23, June 7, 1997, 2007. Cited on page 45.

[Davis, 2004] T. A. Davis. *Algorithm 832: UMFPACK v4.3—an unsymmetric-pattern multifrontal method*. ACM Trans. Math. Softw., 30 (2), pp. 196–199, 2004. `doi:10.1145/992200.992206`. Cited on page 80.

[de Kerchove et al., 2008] C. de Kerchove, L. Ninove, and P. van Dooren. *Maximizing PageRank via outlinks*. Linear Algebra and its Applications, 429 (5-6), pp. 1254–1276, 2008. Special Issue devoted to selected papers presented at the 13th Conference of the International Linear Algebra Society. `doi:10.1016/j.laa.2008.01.023`. Cited on page 30.

[Dean, 2009] J. Dean. *Challenges in building large-scale information retrieval systems*. Presentation at WSDM2009, 2009. Accessed online at `http://videolectures.net/wsdm09_dean_cblirs/`. Cited on page 1.

[Del Corso et al., 2005] G. M. Del Corso, A. Gullí, and F. Romani. *Fast PageRank computation via a sparse linear system*. Internet Mathematics, 2 (3), pp. 251–273, 2005. Cited on pages 28 and 107.

[Del Corso et al., 2007] ———. *Comparison of Krylov subspace methods on the PageRank problem*. J. Comput. Appl. Math., 210 (1-2), pp. 159–166, 2007. `doi:10.1016/j.cam.2006.10.080`. Cited on pages 107, 115, and 124.

[Donoho et al., 2009] D. L. Donoho, A. Maleki, I. U. Rahman, M. Shahram, and V. Stodden. *Reproducible research in computational harmonic analysis*. Computing in Science and Engineering, 11 (1), pp. 8–18, 2009. `doi:http://doi.ieeecomputersociety.org/10.1109/MCSE.2009.15`. Cited on page 158.

[Eiron et al., 2004] N. Eiron, K. S. McCurley, and J. A. Tomlin. *Ranking the web frontier*. In *Proceedings of the 13th international conference on the World Wide Web (WWW2004)*, pp. 309–318. 2004. `doi:10.1145/988672.988714`. Cited on page 107.

[Eldén, 2004] L. Eldén. *A note on the eigenvalues of the Google matrix*. arXiv, 2004. `arXiv:math/0401177`. Cited on pages 21 and 38.

[Fagiolo, 2007] G. Fagiolo. *Clustering in complex directed networks*. Physical Review E (Statistical, Nonlinear, and Soft Matter Physics), 76 (2), 026107, 2007. `doi:10.1103/PhysRevE.76.026107`. Cited on page 145.

[Farahat et al., 2006] A. Farahat, T. LoFaro, J. C. Miller, G. Rae, and L. A. Ward. *Authority rankings from HITS, PageRank, and SALSA: Existence, uniqueness, and effect of initialization*. SIAM Journal on Scientific Computing, 27 (4), pp. 1181–1201, 2006. `doi:10.1137/S1064827502412875`. Cited on page 69.

[Fomel and Claerbout, 2009] S. Fomel and J. F. Claerbout. *Guest editors' introduction: Reproducible research*. Computing in Science and Engineering, 11 (1), pp. 5–7, 2009. `doi:10.1109/MCSE.2009.14`. Cited on page 158.

[Freschi, 2007] V. Freschi. *Protein function prediction from interaction networks using a random walk ranking algorithm*. In *Proceedings of the 7th IEEE International Conference on Bioinformatics and Bioengineering (BIBE 2007)*, pp. 42–48. 2007. `doi:10.1109/BIBE.2007.4375543`. Cited on pages 7 and 99.

[Gautschi, 2002] W. Gautschi. *OPQ: A Matlab suite of programs for generating orthogonal polynomials and related quadrature rules*. `http://www.cs.purdue.edu/archives/2002/wxg/codes/OPQ.html`, 2002. Accessed in January 2008. Cited on pages 85 and 165.

[Gautschi, 2004] ———. *Orthogonal Polynomials: Computation and Approximation*, Oxford University Press, 2004. Cited on page 165.

[Gilbert et al., 1992] J. R. Gilbert, C. Moler, and R. Schreiber. *Sparse matrices in Matlab: Design and implementation*. SIAM Journal on Matrix Analysis and Applications, 13 (1), pp. 333–356, 1992. `doi:10.1137/0613024`. Cited on pages 132, 133, and 150.

[Gilbert et al., 2007] J. R. Gilbert, S. Reinhardt, and V. B. Shah. *High performance graph algorithms from parallel sparse matrices*. In *PARA 2006, 8th International Workshop in Applied Parallel Computing and State of the Art in Scientific Computing*, pp. 260–269. 2007. Cited on page 132.

[Gilbert et al., 2008] ———. *A unified framework for numerical and combinatorial computing*. Computing in Science and Engineering, 10 (2), pp. 20–25, 2008. doi:10.1109/MCSE.2008.45. Cited on page 132.

[Gilbert and Teng, 2002] J. R. Gilbert and S.-H. Teng. *Matlab Mesh Partitioning and Graph Separator Toolbox*, 2002. Http://www.cerfacs.fr/6-26298-Matlab-Mesh-Partitioning-and-Graph-Separator-Toolbox.php. Cited on pages 132 and 134.

[Gleich et al., 2007] D. F. Gleich, P. Glynn, G. H. Golub, and C. Greif. *Three results on the PageRank vector: eigenstructure, sensitivity, and the derivative*. In *Web Information Retrieval and Linear Algebra Algorithms*. 2007. Cited on page 10.

[Gleich et al., to appear] D. F. Gleich, A. P. Gray, C. Greif, and T. Laure. *An inner-outer iteration for PageRank*. SIAM Journal of Scientific Computing, to appear. Cited on page 106.

[Gleich and Polito, 2007] D. F. Gleich and M. Polito. *Approximating personalized PageRank with minimal use of webgraph data*. Internet Mathematics, 3 (3), pp. 257–294, 2007. Cited on pages 32 and 41.

[Gleich and Zhukov, 2005] D. F. Gleich and L. Zhukov. *Scalable computing with power-law graphs: Experience with parallel PageRank*. In *SuperComputing 2005*. 2005. Poster. Cited on page 51.

[Gleich et al., 2004] D. F. Gleich, L. Zhukov, and P. Berkhin. *Fast parallel PageRank: A linear system approach*. Technical Report YRL-2004-038, Yahoo! Research Labs, 2004. Cited on pages 107, 115, 120, and 122.

[Golub and Greif, 2006] G. Golub and C. Greif. *An Arnoldi-type algorithm for computing PageRank*. BIT Numerical Mathematics, 46 (4), pp. 759–771, 2006. doi:10.1007/s10543-006-0091-y. Cited on pages 34, 47, 50, 53, 68, and 107.

[Golub and van Loan, 1996] G. H. Golub and C. F. van Loan. *Matrix Computations*, The Johns Hopkins University Press, third edition, 1996. Cited on pages 19, 22, and 25.

[Golub and Welsch, 1969] G. H. Golub and J. H. Welsch. *Calculation of Gauss quadrature rules*. Mathematics of Computation, 23 (106), pp. 221–230, 1969. Cited on page 85.

[Govan et al., 2008] A. Y. Govan, C. D. Meyer, and R. Albright. *Generalizing Google's PageRank to rank national football league teams*. In *SAS Global Forum 2008*. 2008. Cited on pages 7 and 166.

[Gray et al., 2007] A. Gray, C. Greif, and T. Lau. *An inner/outer stationary iteration for computing PageRank*. CPSC Technical Report TR-2007-27, University of British Columbia, 2007. Cited on pages 9, 10, 45, and 106.

[Grinstead and Snell, 1997] C. M. Grinstead and J. L. Snell. *Introduction to Probability*, American Mathematical Society, 1997. Cited on page 65.

[Gyöngyi et al., 2004] Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen. *Combating web spam with TrustRank*. In *Proceedings of the 30th International Very Large Database Conference*. 2004. Cited on pages 13, 19, 101, and 165.

[Haveliwala, 2002] T. H. Haveliwala. *Topic-sensitive PageRank*. In *Proceedings of the 11th international conference on the World Wide Web*. 2002. Cited on pages 32 and 70.

[Higham, 2005] D. J. Higham. *Google PageRank as mean playing time for pinball on the reverse web*. Applied Mathematics Letters, 18 (12), pp. 1359 – 1362, 2005. doi:10.1016/j.aml.2005.02.020. Cited on page 6.

[Higham, 2002] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*, SIAM, 2002. Cited on page 24.

[Hirai et al., 2000] J. Hirai, S. Raghavan, H. Garcia-Molina, and A. Paepcke. *Webbase: a repository of web pages*. Computer Networks, 33 (1-6), pp. 277–293, 2000. doi:10.1016/S1389-1286(00)00063-3. Cited on pages 40 and 45.

[Honey et al., 2007] C. J. Honey, R. Kötter, M. Breakspear, and O. Sporns. *Network structure of cerebral cortex shapes functional connectivity on multiple time scales*. Proceedings of the National Academy of Sciences, 104 (24), pp. 10240–10245, 2007. doi:10.1073/pnas.0701519104. Cited on page 144.

[Horn and Johnson, 1991] R. A. Horn and C. R. Johnson. *Topics in Matrix Analysis*, Cambridge University Press, Cambridge, UK, 1991. Cited on page 14.

[Horn and Serra-Capizzano, 2007] R. A. Horn and S. Serra-Capizzano. *A general setting for the parametric Google matrix*. Internet Mathematics, 3 (4), pp. 385–411, 2007. Cited on page 91.

[Huberman et al., 1998] B. A. Huberman, P. L. T. Pirolli, J. E. Pitkow, and R. M. Lukose. *Strong regularities in World Wide Web surfing*. Science, 280 (5360), pp. 95–97, 1998. Cited on pages 64, 69, and 76.

[Ipsen and Selee, 2007] I. C. F. Ipsen and T. M. Selee. *PageRank computation, with special attention to dangling nodes*. SIAM J. Matrix Anal. Appl., 29 (4), pp. 1281–1296, 2007. doi:10.1137/060664331. Cited on page 107.

[Jeh and Widom, 2003] G. Jeh and J. Widom. *Scaling personalized web search*. In *Proceedings of the 12th international conference on the World Wide Web*, pp. 271–279. 2003. doi:10.1145/775152.775191. Cited on pages 32 and 70.

[Kahaner, 1980] D. K. Kahaner. *Algorithm 561: Fortran implementation of heap programs for efficient table maintenance*. ACM Trans. Math. Softw., 6 (3), pp. 444–449, 1980. doi:10.1145/355900.355918. Cited on page 146.

[Kamvar et al., 2004] S. D. Kamvar, T. H. Haveliwala, and G. H. Golub. *Adaptive methods for the computation of PageRank*. Linear Algebra Appl., 386, pp. 51–65, 2004. doi:10.1016/j.laa.2003.12.008. Cited on page 107.

[Kamvar et al., 2003] S. D. Kamvar, T. H. Haveliwala, C. D. Manning, and G. H. Golub. *Extrapolation methods for accelerating PageRank computations*. In *Proceedings of the 12th international conference on the World Wide Web*, pp. 261–270. 2003. doi:10.1145/775152.775190. Cited on pages 26, 68, 107, and 163.

[Karande et al., 2009] C. Karande, K. Chellapilla, and R. Andersen. *Speeding up algorithms on compressed web graphs*. In *WSDM '09: Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pp. 272–281. 2009. doi:10.1145/1498759.1498836. Cited on page 107.

[Karci, 2008] M. H. Karci. *Higher order levelable MRF Energy Minimization via Graph Cuts*. Ph.D. thesis, Middle East Technical University, 2008. Cited on page 138.

[Katz, 1953] L. Katz. *A new status index derived from sociometric analysis*. Psychometrika, 18 (1), pp. 39–43, 1953. Cited on pages 21, 63, and 68.

[Keener, 1993] J. P. Keener. *The Perron-Frobenius theorem and the ranking of football teams*. SIAM Review, 35 (1), pp. 80–93, 1993. doi:10.1137/1035004. Cited on page 7.

[Kleinberg, 1999] J. M. Kleinberg. *Authoritative sources in a hyperlinked environment*. Journal of the ACM, 46 (5), pp. 604–632, 1999. Cited on page 165.

[Kolda and Bader, 2009] T. G. Kolda and B. W. Bader. *Tensor decompositions and applications*. SIAM Review, 51 (3), pp. 455–500, 2009. doi:10.1137/07070111X. Cited on page 166.

[Kollias et al., 2006] G. Kollias, E. Gallopoulos, and D. B. Szyld. *Asynchronous iterative computations with web information retrieval structures: The PageRank case*. In *Parallel Computing: Current and Future Issues of High-End Computing (Proceedings of the International Conference Parco05)*, pp. 309–316. 2006. Cited on page 120.

[Koschützki et al., 2005] D. Koschützki, K. A. Lehmann, L. Peeters, S. Richter, D. Tenfelde-Podehl, , and O. Zlotowski. *Centrality Indicies*, chapter 3, pp. 16–61. Volume 3418 of [Brandes and Erlebach, 2005], 2005. doi:10.1007/b106453. Cited on page 4.

[Langville, 2009] A. N. Langville. *Sports ranking*, 2009. From a draft of a forthcoming book. Cited on page 7.

[Langville and Meyer, 2006a] A. N. Langville and C. D. Meyer. *Google's PageRank and Beyond: The Science of Search Engine Rankings*, Princeton University Press, 2006a. Cited on pages 1, 13, 21, 22, 32, 49, 51, 57, and 115.

[Langville and Meyer, 2006b] ———. *Updating Markov chains with an eye on Google's PageRank*. SIAM J. Matrix Anal. Appl., 27 (4), pp. 968–987, 2006b. doi:10.1137/040619028. Cited on page 107.

[Latora and Marchiori, 2001] V. Latora and M. Marchiori. *Efficient behavior of small world networks*. Physical Review Letters, 87 (19), p. 198701, 2001. doi:10.1103/PhysRevLett.87.198701. Cited on page 131.

[Lee et al., 2008] H.-T. Lee, D. Leonard, X. Wang, and D. Loguinov. *IRLbot: scaling to 6 billion pages and beyond*. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pp. 427–436. 2008. doi:10.1145/1367497.1367556. Cited on page 163.

[Lempel and Moran, 2000] R. Lempel and S. Moran. *The stochastic appraoch for link-structure analysis (SALSA) and the TKC effect.* In *The Ninth International World Wide Web Conference*, pp. 387–401. 2000. doi:10.1016/S1389-1286(00)00034-7. Cited on page 165.

[Lin et al., 2008] J. M. Lin, J. W. Bohland, P. Andrews, G. A. P. C. Burns, C. B. Allen, and P. P. Mitra. *An analysis of the abstracts presented at the annual meetings of the society for neuroscience from 2001 to 2006*. PLoS ONE, 3 (4), p. e2052, 2008. doi:10.1371/journal.pone.0002052. Cited on page 144.

[Lin et al., 2009] Y. Lin, X. Shi, and Y. Wei. *On computing PageRank via lumping the Google matrix*. Journal of Computational and Applied Mathematics, 224 (2), pp. 702–708, 2009. doi:10.1016/j.cam.2008.06.003. Cited on page 107.

[Manning et al., 2008] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*, Cambridge University Press, 2008. Cited on page 1.

[McSherry, 2005] F. McSherry. *A uniform approach to accelerated PageRank computation*. In *Proceedings of the 14th international conference on the World Wide Web*, pp. 575–582. 2005. doi:10.1145/1060745.1060829. Cited on pages 19, 51, 106, and 120.

[Meyer, 2000] C. D. Meyer. *Matrix analysis and applied linear algebra*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000. Cited on pages 14, 33, 35, and 74.

[Milovanović and Spalević, 2003] G. V. Milovanović and M. M. Spalević. *Error bounds for Gauss-Turán quadrature formulae of analytic functions*. Mathematics of Computation, 72, pp. 1855–1872, 2003. doi:10.1090/S0025-5718-03-01544-8. Cited on page 165.

[Morrison et al., 2005] J. L. Morrison, R. Breitling, D. J. Higham, and D. R. Gilbert. *GeneRank: using search engine technology for the analysis of microarray experiments*. BMC Bioinformatics, 6 (1), p. 233, 2005. doi:10.1186/1471-2105-6-233. Cited on pages 7, 13, 45, 68, and 99.

[Najork et al., 2007] M. A. Najork, H. Zaragoza, and M. J. Taylor. *HITS on the web: how does it compare?* In *Proceedings of the 30th annual international ACM SIGIR conference on Research and Development in information retrieval (SIGIR2007)*, pp. 471–478. 2007. doi:10.1145/1277741.1277823. Cited on pages 63, 68, and 162.

[Nobile et al., 2008] F. Nobile, R. Tempone, and C. Webster. *A sparse grid stochastic collocation method for partial differential equations with random input data*. SIAM Journal on Numerical Analysis, 46 (5), pp. 2309–2345, 2008. doi:10.1137/060663660. Cited on page 83.

[Page et al., 1999] L. Page, S. Brin, R. Motwani, and T. Winograd. *The PageRank citation ranking: Bringing order to the web*. Technical Report 1999-66, Stanford University, 1999. Cited on pages 1, 13, 26, 32, 63, and 70.

[Papadimitriou and Steiglitz, 1998] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*, Dover Publications, 1998. Cited on page 145.

[Parreira et al., 2006] J. X. Parreira, D. Donato, S. Michel, and G. Weikum. *Efficient and decentralized PageRank approximation in a peer-to-peer web search network*. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pp. 415–426. 2006. Cited on page 120.

[Raman and Chandra, 2008] K. Raman and N. Chandra. *Mycobacterium tuberculosis interactome analysis unravels potential pathways to drug resistance.* BMC Microbiology, 8, p. 234, 2008. doi:10.1186/1471-2180-8-234. Cited on page 144.

[Saad, 2003] Y. Saad. *Iterative Methods for Sparse Linear Systems*, Society for Industrial and Applied Mathematics, Philadelphia, 2nd edition, 2003. Cited on page 30.

[Scheinerman, 2009] E. Scheinerman. *Matgraph: A graph theory toolbox for Matlab*. http://www.ams.jhu.edu/~ers/matgraph/, 2009. Cited on page 134.

[Schwab et al., 2000] M. Schwab, M. Karrenbach, and J. Claerbout. *Making scientific computations reproducible*. Comput. Sci. Eng., 2 (6), pp. 61–67, 2000. doi:10.1109/5992.881708. Cited on page 158.

[Serra-Capizzano, 2005] S. Serra-Capizzano. *Jordan canonical form of the Google matrix: A potential contribution to the PageRank computation*. SIAM Journal on Matrix Analysis and Applications, 27 (2), pp. 305–312, 2005. doi:10.1137/S0895479804441407. Cited on pages 13, 36, and 160.

[Shepelyansky and Zhirov, 2009] D. L. Shepelyansky and O. V. Zhirov. *Google matrix and dynamical attractors*. CoRR, abs/0905.4162, 2009. arXiv:abs/0905.4162. Cited on page 13.

[Siek et al., 2001] J. G. Siek, L.-Q. Lee, and A. Lumsdaine. *The Boost Graph Library: User Guide and Reference Manual*, Addison-Wesley Professional, 2001. Cited on pages 129 and 135.

[Singh et al., 2007] R. Singh, J. Xu, and B. Berger. *Pairwise global alignment of protein interaction networks by matching neighborhood topology*. In *Proceedings of the 11th Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, pp. 16–31. 2007. doi:10.1007/978-3-540-71681-5_2. Cited on pages 8, 68, 99, 106, 124, and 125.

[Talenti, 1987] G. Talenti. *Recovering a function from a finite number of moments*. Inverse Problems, 3 (3), pp. 501–517, 1987. doi:10.1088/0266-5611/3/3/016. Cited on page 78.

[Tarjan, 1972] R. Tarjan. *Depth-first search and linear graph algorithms*. SIAM Journal of Computing, 1 (2), pp. 146–160, 1972. doi:10.1137/0201010. Cited on pages 136 and 145.

[Thelwall, 2003] M. Thelwall. *A free database of university web links: Data collection issues*. International Journal of Scientometrics, Informetrics and Bibliometrics, 6/7 (1), 2003. Cited on page 45.

[Trefethen, 2008] L. N. Trefethen. *Is Gauss quadrature better than Clenshaw-Curtis?* SIAM Review, 50 (1), pp. 67–87, 2008. doi:10.1137/060659831. Cited on page 90.

[Trefethen and Embree, 2005] L. N. Trefethen and M. Embree. *Spectra and Pseudospectra: The Behavior of Nonnormal Matrices and Operators*, Princeton University Press, Princeton, New Jersey, 2005. Cited on page 14.

[van der Vorst, 1992] H. A. van der Vorst. *Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*. SIAM Journal on Scientific and Statistical Computing, 13 (2), pp. 631–644, 1992. doi:10.1137/0913035. Cited on page 106.

[Varga, 1962] R. S. Varga. *Matrix Iterative Analysis*, Prentice Hall, 1962. Cited on pages 26 and 28.

[Various, 2007] Various. *Wikipedia XML database dump from April 2, 2007*. Accessed from http://en.wikipedia.org/wiki/Wikipedia:Database_download, 2007. Cited on page 125.

[Various, 2008] ———. *The Library of Congress subject headings*. http://lcsh.info recently moved to http://id.loc.gov, 2008. Removed from the web in early 2009, accessed in late 2008. Cited on page 125.

[Various, 2009a] ———. *Matlab central file exchange*. Available online http://www.mathworks.com/matlabcentral/fileexchange/, 2009a. Cited on page 134.

[Various, 2009b] ———. *Wikipedia XML database dump from March 6, 2009*. Accessed from http://en.wikipedia.org/wiki/Wikipedia:Database_download, 2009b. Cited on page 3.

[Vigna, 2005] S. Vigna. *TruRank: Taking PageRank to the limit*. In *WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web*, pp. 976–977. 2005. doi:10.1145/1062745.1062826. Cited on page 21.

[Vigna, 2008] ———. *Webgraph 2.1*. http://webgraph.dsi.unimi.it/, 2008. Cited on page 153.

[Vigna et al., 2008] S. Vigna, R. Posenato, M. Santini, and S. Vigna. *LAW 1.3.1: Library of algorithms for the webgraph*. http://law.dsi.unimi.it/software/docs/, 2008. Cited on pages 25, 29, and 114.

[Wang, 2002] M. Wang. *A significant improvement to clever algorithm in hyperlinked environment*. In *Proceedings of the 11th international conference on the World Wide Web (WWW2002)*. 2002. Cited on page 69.

[Watts and Strogatz, 1998] D. J. Watts and S. H. Strogatz. *Collective dynamics of "small-world" networks*. Nature, 393 (6684), pp. 440–442, 1998. doi:10.1038/30918. Cited on pages 132 and 145.

[White and Drucker, 2007] R. W. White and S. M. Drucker. *Investigating behavioral variability in web search*. In *Proceedings of the 16th international conference on the World Wide Web (WWW2007)*, pp. 21–30. 2007. doi:10.1145/1242572.1242576. Cited on pages 64 and 69.

[Wills and Ipsen, 2009] R. S. Wills and I. C. F. Ipsen. *Ordinal ranking for Google's PageRank*. SIAM Journal on Matrix Analysis and Applications, 30, pp. 1677–1696, 2009. doi:10.1137/070698129. Cited on pages 25 and 27.

[Witten and Frank, 2005] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*, Morgan Kaufmann, 2005. Cited on page 102.

[Witten et al., 1999] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*, Morgan Kaufmann, San Francisco, 2nd edition, 1999. Cited on page 1.

[Xiu and Hesthaven, 2005] D. Xiu and J. Hesthaven. *High order collocation methods for the differential equation with random inputs*. SIAM J. Sci. Comput., 27 (3), pp. 1118–1139, 2005. doi:10.1137/040615201. Cited on page 83.

[Xue et al., 2003] G.-R. Xue, H.-J. Zeng, Z. Chen, W.-Y. Ma, H. Zhang, and C.-J. Lu. *User access pattern enhanced small web search*. In *Poster Proceedings of the 12th international conference on the World Wide Web (WWW2003)*. 2003. Cited on page 69.

[Zawodny, 2003] J. Zawodny. *PageRank is dead*. Blog. Available online `http://jeremy.zawodny.com/blog/archives/000751.htm`, 2003. Cited on page 162.

[Zhang et al., 2004] H. Zhang, A. Goel, R. Govindan, K. Mason, and B. Van Roy. *Making eigenvector-based reputation systems robust to collusion*. In *Proceedings of the third Workshop on Web Graphs (WAW)*, pp. 92–104. 2004. `doi:10.1007/b101552`. Cited on pages 30 and 70.

[Zhang et al., 2007] L. Zhang, T. Qin, T.-Y. Liu, Y. Bao, and H. Li. *N-step pagerank for web search*. In *Proceedings of the 29th European Conference on Information Retreival Research (ECIR2007)*, pp. 653–660. 2007. `doi:10.1007/978-3-540-71496-5_63`. Cited on page 165.

[Zheng et al., 2008] Z. Zheng, H. Zha, and G. SungEun. *Query-level learning to rank using isotonic regression*. In *46th Annual Allerton Conference on Communication, Control and Computing*, pp. 1108–1115. 2008. `doi:10.1109/ALLERTON.2008.4797684`. Cited on page 8.

[Zhou et al., 2005] D. Zhou, J. Huang, and B. Schölkopf. *Learning from labeled and unlabeled data on a directed graph*. In *ICML '05: Proceedings of the 22nd International Conference on Machine Learning*, pp. 1036–1043. 2005. `doi:10.1145/1102351.1102482`. Cited on pages 32, 68, and 165.

[Zhu and Hayes, 2009] Y.-K. Zhu and W. B. Hayes. *Correct rounding and a hybrid approach to exact floating-point summation*. SIAM Journal on Scientific Computing, 31 (4), pp. 2981–3001, 2009. `doi:10.1137/070710020`. Cited on page 25.

[Zwillinger et al., 1996] D. Zwillinger, S. G. Krantz, and K. H. Rosen, editors. *Standard Mathematical Tables and Formulae*, CRC Press, 30th edition, 1996. Cited on pages 83 and 87.