Scalable Algorithms for Data and Network Analysis

Shang-Hua Teng
Computer Science and Mathematics
University of Southern California
shanghua.teng@gmail.com

Full text available at: http://dx.doi.org/10.1561/0400000051
Editorial Scope

Topics

Foundations and Trends® in Theoretical Computer Science publishes surveys and tutorials on the foundations of computer science. The scope of the series is broad. Articles in this series focus on mathematical approaches to topics revolving around the theme of efficiency in computing. The list of topics below is meant to illustrate some of the coverage, and is not intended to be an exhaustive list.

- Algorithmic game theory
- Computational algebra
- Computational aspects of combinatorics and graph theory
- Computational aspects of communication
- Computational biology
- Computational complexity
- Computational geometry
- Computational learning
- Computational Models and Complexity
- Computational Number Theory
- Cryptography and information security
- Data structures
- Database theory
- Design and analysis of algorithms
- Distributed computing
- Information retrieval
- Operations research
- Parallel algorithms
- Quantum computation
- Randomness in computation

Information for Librarians

Foundations and Trends® in Theoretical Computer Science, 2016, Volume 12, 4 issues. ISSN paper version 1551-305X. ISSN online version 1551-3068. Also available as a combined paper and online subscription.
Scalable Algorithms for Data and Network Analysis

Shang-Hua Teng
Computer Science and Mathematics
University of Southern California
shanghua.teng@gmail.com
## Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>Scalable Algorithms</td>
<td>8</td>
</tr>
<tr>
<td>1.1</td>
<td>Challenges of Massive Data</td>
<td>8</td>
</tr>
<tr>
<td>1.2</td>
<td>The Scalability of Algorithms</td>
<td>10</td>
</tr>
<tr>
<td>1.3</td>
<td>Complexity Class S</td>
<td>12</td>
</tr>
<tr>
<td>1.4</td>
<td>Scalable Reduction and Algorithmic Primitives</td>
<td>14</td>
</tr>
<tr>
<td>1.5</td>
<td>Article Organization</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>Networks and Data</td>
<td>24</td>
</tr>
<tr>
<td>2.1</td>
<td>Weighted Graphs and Affinity Networks</td>
<td>24</td>
</tr>
<tr>
<td>2.2</td>
<td>Possible Sources of Affinities</td>
<td>26</td>
</tr>
<tr>
<td>2.3</td>
<td>Beyond Graph Models for Social/Information Networks</td>
<td>27</td>
</tr>
<tr>
<td>2.4</td>
<td>Basic Problems in Data and Network Analysis</td>
<td>33</td>
</tr>
<tr>
<td>2.5</td>
<td>Sparse Networks and Sparse Matrices</td>
<td>39</td>
</tr>
<tr>
<td>3</td>
<td>Significant Nodes: Sampling - Making Data Smaller</td>
<td>42</td>
</tr>
<tr>
<td>3.1</td>
<td>Personalized PageRank Matrix</td>
<td>45</td>
</tr>
<tr>
<td>3.2</td>
<td>Multi-Precision Annealing for Significant PageRank</td>
<td>47</td>
</tr>
<tr>
<td>3.3</td>
<td>Local Approximation of Personalized PageRank</td>
<td>48</td>
</tr>
<tr>
<td>3.4</td>
<td>Multi-Precision Sampling</td>
<td>50</td>
</tr>
<tr>
<td>3.5</td>
<td>Significant-PageRank Identification</td>
<td>62</td>
</tr>
</tbody>
</table>
### 4 Clustering: Local Exploration of Networks

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Local Algorithms for Network Analysis</td>
<td>67</td>
</tr>
<tr>
<td>4.2 Local Clustering and Random Walks</td>
<td>72</td>
</tr>
<tr>
<td>4.3 Performance Analysis of Local Clustering</td>
<td>77</td>
</tr>
<tr>
<td>4.4 Scalable Local Computation of Personalized PageRank</td>
<td>81</td>
</tr>
<tr>
<td>4.5 Discussions: Local Exploration of Networks</td>
<td>87</td>
</tr>
<tr>
<td>4.6 Interplay Between Dynamic Processes and Networks</td>
<td>90</td>
</tr>
<tr>
<td>4.7 Cheeger’s Inequality and its Parameterization</td>
<td>98</td>
</tr>
</tbody>
</table>

### 5 Partitioning: Geometric Techniques for Data Analysis

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Centerpoints and Regression Depth</td>
<td>106</td>
</tr>
<tr>
<td>5.2 Scalable Algorithms for Centerpoints</td>
<td>108</td>
</tr>
<tr>
<td>5.3 Geometric Separators</td>
<td>115</td>
</tr>
<tr>
<td>5.4 Dimension Reduction: Random vs Spectral</td>
<td>122</td>
</tr>
<tr>
<td>5.5 Scalable Geometric Divide-and-Conquer</td>
<td>124</td>
</tr>
<tr>
<td>5.6 Graph Partitioning: Vertex and Edge Separators</td>
<td>127</td>
</tr>
<tr>
<td>5.7 Multiway Partition of Network and Geometric Data</td>
<td>133</td>
</tr>
<tr>
<td>5.8 Spectral Graph Partitioning: The Geometry of a Graph</td>
<td>135</td>
</tr>
</tbody>
</table>

### 6 Sparsification: Making Networks Simpler

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 Spectral Similarity of Graphs</td>
<td>142</td>
</tr>
<tr>
<td>6.2 Some Basic Properties of Spectrally Similar Networks</td>
<td>144</td>
</tr>
<tr>
<td>6.3 Spectral Graph Sparsification</td>
<td>146</td>
</tr>
<tr>
<td>6.4 Graph Inequalities and Low-Stretch Spanning Trees</td>
<td>149</td>
</tr>
<tr>
<td>6.5 Edge Centrality, Sampling, and Spectral Approximation</td>
<td>155</td>
</tr>
<tr>
<td>6.6 Scalable Dense-Matrix Computation via Sparsification</td>
<td>159</td>
</tr>
<tr>
<td>6.7 PageRank Completion of Networks</td>
<td>161</td>
</tr>
</tbody>
</table>

### 7 Electrical Flows: Laplacian Paradigm for Network Analysis

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1 SDD Primitive and Its Scalability</td>
<td>166</td>
</tr>
<tr>
<td>7.2 Electrical Flow and Laplacian Linear Systems</td>
<td>167</td>
</tr>
<tr>
<td>7.3 Spectral Approximation and Spectral Partitioning</td>
<td>172</td>
</tr>
<tr>
<td>7.4 Learning from Labeled Network Data</td>
<td>174</td>
</tr>
<tr>
<td>7.5 Sampling From Gaussian Markov Random Fields</td>
<td>176</td>
</tr>
<tr>
<td>7.6 Scalable Newton’s Method via Spectral Sparsification</td>
<td>180</td>
</tr>
<tr>
<td>7.7 Laplacian Paradigm</td>
<td>190</td>
</tr>
</tbody>
</table>
# 8 Remarks and Discussions

8.1 Beyond Graph-Based Network Models .......................... 199
8.2 Discussions: a Family of Scaling-Invariant Clusterability ...... 217
8.3 Data Clustering: Intuition versus Ground Truth ............... 227
8.4 Behaviors of Algorithms: Beyond Worst-Case Analysis ...... 236
8.5 Final Remarks .......................................................... 244

# Acknowledgements

# References

Full text available at: http://dx.doi.org/10.1561/0400000051
Abstract

In the age of Big Data, efficient algorithms are now in higher demand more than ever before. While Big Data takes us into the asymptotic world envisioned by our pioneers, it also challenges the classical notion of efficient algorithms: Algorithms that used to be considered efficient, according to polynomial-time characterization, may no longer be adequate for solving today’s problems. It is not just desirable, but essential, that efficient algorithms should be scalable. In other words, their complexity should be nearly linear or sub-linear with respect to the problem size. Thus, scalability, not just polynomial-time computability, should be elevated as the central complexity notion for characterizing efficient computation.

In this tutorial, I will survey a family of algorithmic techniques for the design of provably-good scalable algorithms. These techniques include local network exploration, advanced sampling, sparsification, and geometric partitioning. They also include spectral graph-theoretical methods, such as those used for computing electrical flows and sampling from Gaussian Markov random fields. These methods exemplify the fusion of combinatorial, numerical, and statistical thinking in network analysis. I will illustrate the use of these techniques by a few basic problems that are fundamental in network analysis, particularly for the identification of significant nodes and coherent clusters/communities in social and information networks. I also take this opportunity to discuss some frameworks beyond graph-theoretical models for studying conceptual questions to understand multifaceted network data that arise in social influence, network dynamics, and Internet economics.

In 1997, I attended an invited talk given by Shafi Goldwasser at the 38th Annual Symposium on Foundations of Computer Science. It was a very special talk. The title of her talk, printed in the conference program, “New Directions in Cryptography: Twenty Some Years Later,” was modest. However, the talk was beautiful and poetic. In particular, the talk’s subtitle, “Cryptography and Complexity Theory: a Match Made in Heaven,” has stayed with me after all these years.

The rise of the Internet, digital media, and social networks has introduced another wonderful match in the world of computing. The match between Big Data and Scalable Computing may not be as poetic as the match between Cryptography and Complexity Theory: Big Data is messier than cryptography and scalable computing uses more heuristics than complexity theory. Nevertheless, this match — although practical — is no less important: “Big Data and Scalable Computing: a Pragmatic Match Made on Earth.”

Reasons to Write and People to Thank

I would like start by thanking Madhu Sudan and James Finlay for inviting me to write a survey for Foundations and Trends in Theoretical Computer Science, and for their patience, support, and guidance during this long process.

When Madhu and James first reached out to me in the February of 2012 to write a survey on graph sparsification, I was noncommittal and
used my busy schedule as the chair of a large department as my excuse. When they came back to me in the Fall of 2012 — knowing that I had successfully become a former chair — I did not reply until the June of 2013, when I had received the confirmation that the USC Daycare finally accepted my 8 month-old daughter Sonia off the wait-list. But during the span of these 16 months, many things happened that were relevant to their initial invitation.

- Nisheeth Vishnoi completed a wonderful and comprehensive survey, titled, $Lx = b$ [344], that appeared in the May issue of Foundations and Trends in Theoretical Computer Science.


- Several exciting new results emerged in spectral graph theory that were enabled or inspired by spectral sparsification and scalable Laplacian solvers.

These developments had reduced the need for another longer survey solely devoted to (spectral) sparsification. But I got unexpected encouragement to write a survey from researchers outside my usual theory community. Yan Liu, my machine learning/data mining colleague at USC, invited me to present my work on spectral graph theory and network analysis at the 2012 SIAM Data Mining Conference. I gave a talk titled, “Algorithmic Primitives for Network Analysis: Through the Lens of the Laplacian Paradigm,” based on my joint work with Dan Spielman. Although the talk was a typical theoretical computer science talk, I was excited by the reception that I received from the Big Data experts: Huan Liu, Joydeep Ghosh, Vipin Kumar, Christos Faloutsos, and particularly Yan Liu, who strongly encouraged me to write a survey on these scalable algorithmic techniques for readers beyond theoretical computer science.

It was a tall order! But given this potential interest from the Big Data community, I reconnected with Madhu and James and proposed
a tutorial to further expand my talk. My goal was to survey some basic theoretical developments (on scalable algorithms) and their techniques that might be useful for practical data/network analysis. The plan was to select a collection of fundamental and illustrative topics of potential practical relevances. I naturally favor problems and algorithms whose rigorous mathematical analysis are clean enough to present for researchers outside theory. Towards this end — in the survey — I selectively encapsulate some “heavy duty” mathematical materials, state them as theorems without proofs, and only expose the relevant essentials aiming to make the survey readable without losing its rigor.

Here, I would like to thank Yan Liu for initiating all this and her valuable feedback on the draft. I thank Madhu and James again for supporting this changed plan and their advice on how to proceed with this writing. I thank Amy Schroeder of the USC Viterbi Engineering Writing Program for editing this monograph, and the anonymous referee and my Ph.D. student Yu Cheng for valuable feedback. I thank Dan Spielman and all my collaborators who have directly contributed to this survey: To Nina Amenta, Reid Andersen, Nina Balcan, Joshua Batson, Marshall Bern, Christian Borgs, Michael Baur, Mark Braverman, Jennifer Chayes, Paul Christiano, Wei Chen, Xi Chen, Dehua Cheng, Yu Cheng, Sin-Wing Cheng, Ken Clarkson, David Eppstein, Tamal Dey, John Dunagan, Herbert Edelsbrunner, Matthias Eichstaedt, Michael Elkin, Yuval Emek, Michael Facello, Alan Frieze, Daniel Ford, John Gilbert, Rumi Ghosh, John Hopcroft, Kamal Jain, Jon Kelner, Marco Kiwi, James Lee, Tobin Lehman, Kristina Lerman, Xiangyang Li, Yan Liu, Qi Lu, Aleksander Mądry, Adrian Marple, Gary Miller, Vahab Mirrokni, Richard Peng, Greg Price, Heiko Röglin, Horst Simon, Nikhil Srivastava, Carl Sturtivant, Dafna Talmor, Bill Thurston, Steve Vavasis, Konstantin Voevodski, Noel Walkington, Yu Xia, and Xiaoran Yan, thank you!

**Theory and Practice**

While I believe in the importance of provably-good algorithms in data and network analysis, my own experiences at Intel, NASA, and Aka-
mai have also taught me the limitation of “provably-good algorithms.” The gap between proof-based theory and relevance-based practice is beyond the limitation of the worst-case or average-case analyses. The theory-practice gap exists also because essentially all measures of qualities — from the centrality of a node in a network, to the similarity between two datasets/networks, to the coherence of a network cluster and community — may have their limitations.

Many conceptual questions that arise in modeling real-world data are fundamentally challenging.

Thus, as much as I would like — in this survey — to make an algorithmic connection between theory and practice in the area of data and network analysis, I would also like readers to approach this survey with an open mind: Theory is usually my guide for understanding practical problems, but theoretical thinking has too often been the obstacle that makes me struggle in connecting with practice. On the few occasions that theoretical thinking provided me with the insight to make a connection, I was elated. But indeed, I usually found myself unable to balance the connection between theory and practice, and then decided to do theory for its mathematical beauty.

For example, in theoretical computer science, we have also encountered the notion of clusterability in various settings, including VLSI layout, parallel processing, network clustering, community identification, and more generally, the design of divide-and-conquer algorithms for matrix/graph problems. However, as illustrated throughout this survey, I remain unsure about how to evaluate and validate the relevance of various mathematical notions of clusterability, particularly the conductance or cut-ratio measures that I have been studying for more than a decade. We use these partition/clusterability measures because they appear to be reasonable, and because, using them, we have obtained mathematical proofs that are aligned with our intuition.

My Ph.D. advisor Gary Miller once said to me, “coping with the uncertainty between theory and practice gives rise to plausible and sometimes good research questions, but questioning the certainty can lead to excellent questions.” So I do think it is more than reasonable to question and challenge every notion one chooses.
Preface

Discrete vs Continuous and Beyond

I would like to conclude the preface by remarking that several subjects of this survey are at the intersection between combinatorial optimization and numerical analysis. Thus, I think they serve as good examples of the interplay between combinatorial thinking and numerical thinking [330]. While it is more conventional to view many network analysis problems as graph-theoretical problems, it can often be constructive to view them as numerical, statistical, or game-theoretical subjects as well. Network data is richer than its graph representation, and network science is beyond graph theory.

Numerical thinking is also more than numerical analysis — it is a creative process of discovering useful numerical connections that may not be apparent [330]. For example, in the 70s, Hall [162], Donath and Hoffman [115, 116], and Fiedler [133, 134] made insightful connection between graphs and matrices — beyond just the matrix representation of graphs — which set the stage for spectral graph theory. The field of algorithm design has benefited greatly from the deep connection between graph properties (such as connectivity, conductance, and mixing time) and algebraic properties (spectral bounds of Laplacian/adjacency/random-walk matrices [82]). Over the last decade, scientists have made even broader and deeper connections between numerical solutions and network solutions [211, 73], between numerical representations and digital representations [110, 108], between

---

1To the memory of Miroslav Fiedler (1926 – 2015): It is difficult to overstate the impact of Fiedler’s work to spectral graph theory. His paper, “Algebraic Connectivity of Graphs,” [133] established a far-reaching connection between graph theory and linear algebra. Fiedler’s spectral theorem, together with Koebe’s disk-packing characterization of planar graphs [213] (see Theorem 5.34), Sperner’s lemma for Brouwer’s fixed-point theorem [310], and Cheeger’s inequality [82] (see Theorem 4.2), are my favorite mathematical results — they beautifully connect continuous mathematics with discrete mathematics. I have always cherished my only meeting with Fiedler. After my talk, “The Laplacian Paradigm: Emerging Algorithms for Massive Graphs,” [329] at 7th Annual Conference on Theory and Applications of Models of Computation, Jaroslav Nešetřil (my former officemate at Microsoft Research Redmond) thoughtfully invited my wife Diana (a US historian) and I to a dinner with him and Fiedler. I still vividly remembered that special evening in the beautiful Prague on June 10, 2010. At age eighty four, Miroslav was charming and talkative, not just about mathematics but also about history.
numerical methods and statistical methods \cite{111,118}, between numerical concepts and complexity concepts \cite{314}, and between numerical formulations and privacy formulations \cite{121}. Numerical analysis has played an increasing role in data analysis through dimension reduction \cite{111} and in machine learning through optimization.

In the examples of this survey, the Laplacian paradigm \cite{319} has not only used numerical concepts such as preconditioning to model network similarity and graph sparsification, but also used combinatorial tools to build scalable solvers for linear systems \cite{318,317,319}, Gaussian sampling \cite{86}, and geometric median \cite{98}. Scalable techniques for PageRank approximation \cite{65} have also led to algorithmic breakthroughs in influence maximization \cite{64,325,324} and game-theoretical centrality formulation \cite{84}. These results illustrate the rich connection between network sciences and numerical analysis.

A Quick Guide of the Survey

The survey begins with two background chapters. Chapter 1 discusses scalability measures for evaluating algorithm efficiency. It also introduces the basic characterizations of scalable algorithms, which are the central subjects of this survey. Chapter 2 reviews mathematical models for specifying networks, and highlights a few basic problems in data and network analysis. We will use these problems as examples to illustrate the design and analysis of scalable algorithms in the technical chapters that follow the background chapters. In Section 1.5 I will give a more detailed outline of these technical chapters.

Acknowledgements

The writing of this article is supported in part by a Simons Investigator Award from the Simons Foundation and the NSF grant CCF-1111270.

Shang-Hua Teng
Los Angeles
In light of the explosive growth in the amount of available data and the diversity of computing applications, efficient algorithms are in higher demand now more than ever before.

1.1 Challenges of Massive Data

Half a century ago, the pioneers of Theoretical Computer Science began to use asymptotic analysis as the framework for complexity theory and algorithm analysis [167]. At the time, computers could only solve small-scale problems by today’s standards. For example, linear programs of fifty variables, linear systems with a hundred variables, or graphs with a thousand vertices, were considered large scale. In those days, the world’s most powerful computers (e.g., the IBM main frames) were far less capable than the iPhones of today. Even though these pioneers were mindful that constant factors did matter for practical computing, they used asymptotic notations — such as Big-O — to define complexity classes such as $P$ and $NP$ to characterize efficient algorithms and intractable problems [122, 100, 233, 189, 141]. Asymptotic complexity simplifies analyses and crucially puts the focus on the order of the
1.1. Challenges of Massive Data

leading complexity terms of algorithms [103, 209, 307]. However, with respect to the size of practical inputs at the time, the asymptotic world seemed to belong to the distant future.

Our pioneers persisted because they had a vision that one day computational problems would be massive, and therefore, the rate of complexity growth in relation to the growth of the inputs would be essential for characterizing the efficiency of an algorithm as well as the computational difficulty of a problem.

*The asymptotic world has arrived with the rise of the Internet!*

Today, the Web has grown into a massive graph with trillions of nodes; social networks and social media have generated an unbounded amount of digital records; smart phones have produced billions of images and videos; and tasks of science and engineering simulation have created equations and mathematical programs that involve hundreds of millions of variables. Even our computing devices have grown rapidly in size and complexity: In the middle 90s, the Intel Pentium processor had 2 millions transistors; today’s PCs contain more than a billion.

While *Big Data* has taken algorithm design into the asymptotic world envisioned by our pioneers, the explosive growth of problem size has also significantly challenged the classical notion of *efficient algorithms*, particularly the use of *polynomial time* as the characterization of *efficient computation* [122]: Algorithms that used to be considered efficient (in the classification according to P) — such as a neat $O(n^2)$-time or $O(n^{1.5})$-time algorithm — may no longer be adequate for solving today’s problems.

*Therefore, more than ever before, it is not just desirable, but essential, that efficient algorithms should be scalable. In other words, their complexity should be nearly linear or sub-linear with respect to the problem size. Thus, scalability — not just polynomial-time computability — should be elevated as the central complexity notion for characterizing efficient computation.*
1.2 The Scalability of Algorithms

The scalability of an algorithm measures the growth of its complexity — such as the running time — in relation to the growth of the problem size. It measures the capacity of an algorithm to handle big inputs.

Suppose $\Pi$ is a computational problem whose input domain is $\Omega$. For each instance $x \in \Omega$, let $\text{size}(x)$ denote the size of input $x$. The input domain $\Omega$ can be viewed as the union of a collection of subdomains $\{\ldots, \Omega_n, \ldots\}$, where $\Omega_n$ denotes the subset of $\Omega$ with input size $n$.

Now, suppose $A$ is an algorithm for solving $\Pi$. For $x \in \Omega$, let $T_A(x)$ denote the time complexity for running $A(x)$. Instead of directly using instance-based complexity $T_A(x)$ to measure the performance of algorithm $A$ for solving $x$, we consider the following related quality measure:

**Definition 1.1 (Instance-Based Scalability).** The scalability of an algorithm $A$ for solving an instance $x \in \Omega$ is given by:

$$\text{scalability}(A, x) = \frac{T_A(x)}{\text{size}(x)}$$

We now summarize the instance-based scalability of algorithm $A$ over all instances in $\Omega_n$ as

$$\text{scalability}_A(n) := \sup_{x \in \Omega_n} \text{scalability}(A, x) = \sup_{x \in \Omega_n} \frac{T_A(x)}{\text{size}(x)}.$$

Then, scalability$_A(n)$ is a function that measures the growth of the complexity of $A$ in relation to the growth of the problem. Let $T_A(n) = \sup_{x \in \Omega_n} T_A(x)$ denote the (worst-case) complexity of algorithm $A$ on inputs of size $n$. Note that:

$$\text{scalability}_A(n) = \frac{T_A(n)}{n}.$$

Thus, $A$ is a polynomial-time algorithm if $\text{scalability}_A(n)$ is polynomial in $n$. However, the scalability measure puts the focus on scalable algorithms:

---

1. See Appendix A.1.1 for a review of the basic types of computational problems.
2. We may also use other beyond worst-case formulae for performance summarization [314]. For more discussion, see Section 8.4.
1.2. The Scalability of Algorithms

**Definition 1.2 (Scalable Algorithms).** An algorithm $A$ is scalable if there exists a constant $c > 0$ such that:

$$\text{scalability}_A(n) = O(\log^c n).$$

In the special case when $c = 0$, we say $A$ is linearly-scalable. We say algorithm $A$ is super-scalable, if scalability$_A(n) = o(1)$. Super-scalable algorithms have a complexity that is sub-linear in problem size. Thus, necessarily, these algorithms must find solutions without examining the entire input data set. Sampling and local data/network exploration are two basic tools for designing super-scalable algorithms, and will be the subjects of Chapters 3 and 4.

**Remark:** One may say that scalability$_A(n)$ encodes no more information than $T_A(n)$ about algorithm $A$, because scalability$_A(n) = T_A(n)/n$. For example, $A$ is scalable if and only if $T_A(n)$ is nearly linear in $n$ as referred to in [313]. However, the former identifies scalability as an essential concept for the characterization of efficient algorithms.

The scalability measure puts the emphasis not on polynomial-time algorithms, but on scalable algorithms. It highlights the exponential gap between $O(\log^c n)$ and $n$, and between scalable algorithms and quadratic-time algorithms.

To capture the essence of scalable algorithms, throughout the article, we will adopt the following commonly-used asymptotic notation.

**Definition 1.3 ($\tilde{O}$-Notation).** For a given function $g(n)$, we denote by $\tilde{O}(g(n))$ the set of functions:

$$\tilde{O}(g(n)) = \{ f(n) : \exists \text{ constant } c > 0, \text{ such that } f(n) = O(g(n) \log^c g(n)) \}$$

For any positive integer $n$, we also use $\tilde{O}_n(1)$ to denote:

$$\{ f(n) : \exists \text{ constant } c > 0, \text{ such that } f(n) = O(\log^c n) \}.$$
In other words, $\tilde{O}$ is a variant of the asymptotic $O$-notation that hides additional poly-logarithmic factors. For example, $n \log^3 n = \tilde{O}(n)$. With this notation, an algorithm is scalable if its scalability measure on an input of size $n$ is $O_n(1)$. In other words, its complexity on an input of size $n$ is $\tilde{O}(n)$.

**Remarks:** The scalability analysis of algorithms is not unique to Big Data. For example, in parallel processing [71, 223], the notion of scalability is used to measure the efficiency of a parallel algorithm in utilizing parallel machines: Let $T_A(p, n)$ denote the parallel complexity of an algorithm $A$ on a machine with $p$ processors. So, the speed-up of this parallel algorithm with respect to the sequential one is $\frac{T_A(n)}{T_A(p, n)}$.

Then, $\frac{T_A(n)}{T_A(p, n)} \cdot \frac{1}{p}$ measures the ratio of the achievable speedup to maximum-possible speedup by running $A$ on $p$ processors. In this context, a parallel algorithm $A$ is linearly-scalable if this ratio is bounded from below by a constant. Although the focus of scalability analysis of parallel algorithms is different from the scalability analysis of sequential algorithms, we can draw on insights from previous studies.

### 1.3 Complexity Class $S$

A basic step in algorithm analysis and complexity theory is to characterize the family of problems that have efficient algorithmic solutions. In the world of *Big Data*, instead of using the traditional polynomial-time as the criterion for efficient algorithms, we require that efficient algorithms must be scalable.

**Definition 1.4 (Complexity Class $S$).** We denote by $S$ the set of computational problems that can be solved by a deterministic scalable algorithm.

In other words, a computational problem $\Pi$ is in class $S$ if there exists an algorithm $A$ that solves $\Pi$ with scalability $A(n) = \tilde{O}(1)$.

Complexity class $S$ is analogous to the traditional complexity classes $P$ and $\mathbb{F}P$. In complexity theory, one usually classifies computational problems into three basic types [307, 103]: decision problems, search
1.3. Complexity Class $S$

problems, and optimization problems (see Appendix A.1.1 for a quick review). Formalism is needed to precisely define complexity classes in order to address the subtlety among different types of computational problems. For example, to use polynomial time as the benchmark for efficient computation, the class $P$ is usually reserved only for decision problems that can be solved in polynomial time without randomization \[307\]. The search version of complexity class $P$ is known as $FP$, i.e., the class of functions that can be deterministically computed in polynomial time. In this article, we intend to be less formal in this regard, so that the focus will be on the notion of scalability rather than the difference between decision, search, and optimization problems.

**Definition 1.5 (Complexity Class RS).** We denote by $RS$ the set of computational problems that can be solved by a randomized scalable algorithm.

Randomization also introduces its own subtlety, which has given rise to classes such as $BPP$, $RP$, and $ZPP$ for decision problems. For example, $ZPP$ denotes the set of problems that can be solved by a Las Vegas algorithm with an expected polynomial runtime, which makes no errors in all instances. $BPP$ and $RP$ relax the latter condition by allowing the polynomial-time randomized algorithms to make bounded errors. Algorithms for $RP$ are allowed to make errors only for YES instances, but algorithms for $BPP$ are allowed to make errors for both YES and NO instances.

Again, with regard to randomization, we intend to be less formal as well so the focus will be on the notion of scalability rather than different types of errors.

**Remarks:** In computational complexity theory, one has to first define a computational model in order to define a complexity class. Commonly used models are Turing machines and random-access machines (RAM). It’s well known that complexity classes, such as $P$ and $FP$, are essentially robust with respect to these models.

The scalable classes $S$ and $RS$ can be formally defined according to these computational models. Their robustnesses with respect to computational models require further investigation, but are outside the scope of
Scalable Algorithms

this article. As universal as these computational models are, data and network analysis programs on Turing machines or abstract random-access machines could be cumbersome.

However, as Sipser said [307], “Real computers are quite complicated — too much so to allow us to set up a manageable mathematical theory of them directly.” In the world of Big Data, massive networks, and large-scale optimization, without getting bogged down with details, I encourage readers to think about the real RAM model of Blum, Shub, and Smale [54], but with unit-cost computation of basic rational operations over reals at a given machine precision $\epsilon_{\text{machine}}$.

1.4 Scalable Reduction and Algorithmic Primitives

Algorithm design for scalable computing is like building a software library. Once we develop a new scalable algorithm, we can add it to our scalable library, and use it as a subroutine to design the next wave of scalable algorithms.

At the heart of this perspective is the notion of scalable reduction.

---

**Definition 1.6 (Scalable Reduction).** A computational problem $\Pi$ (over domain $\Omega$) is $S$-reducible to another computational problem $\Pi'$ (over domain $\Omega'$), denoted by $\Pi \leq_S \Pi'$, if the following is true: Given any solver $B$ for $\Pi'$, there exists an algorithm $A$ for solving $\Pi$ such that for every instance $x \in \Omega$, $A(x)$ takes $\tilde{O}(\text{size}(x))$ steps including (i) generating a collection of instances $y_1, ..., y_{L(x)} \in \Omega'$ with:

$$\sum_{i=1}^{L(x)} \text{size}(y_i) = \tilde{O}(\text{size}(x))$$

and (ii) making calls to $B$ on these instances.

In other words, $\Pi \leq_S \Pi'$ if $\Pi$ has a scalable Turing-Cook-reduction to $\Pi'$. In this definition, $A$ is assumed to be a deterministic algorithm. We say $\Pi \leq_{RS} \Pi'$ if we use a randomized algorithm $A$ in the definition above. Directly from the definition, we have:
1.4. **Scalable Reduction and Algorithmic Primitives**

**Proposition 1.7.** (1) If \( \Pi \leq S \Pi' \) and \( \Pi' \leq S \Pi'' \) imply \( \Pi \leq S \Pi'' \). (2) \( \Pi \leq_{RS} \Pi' \) and \( \Pi' \leq_{RS} \Pi'' \) imply \( \Pi \leq_{RS} \Pi'' \).

**Proposition 1.8.** (1) If \( \Pi \leq S \Pi' \) and \( \Pi' \in S \), then \( \Pi \in S \). (2) If \( \Pi \leq_{RS} \Pi' \) and \( \Pi' \in RS \), then \( \Pi \in RS \).

The field of computing has produced a number of remarkable scalable algorithms in various applicational domains. The following are a few examples:

- **Basic Algorithms**: FFT, merge sort, median selection, Huffman codes
- **Graph Algorithms**: minimum (maximum) spanning trees, shortest path trees, breadth-first search, depth-first search, connected components, strongly connected components, planar separators
- **Optimization Algorithms**: linear programming in constant dimensions
- **Probabilistic Algorithms**: VC-dimension based sampling, quick sort
- **Data structures**: many wonderful data structures
- **Numerical and Geometric Algorithms**: the multigrid method, the fast multipole method, 2D Delaunay triangulations and Voronoi diagrams, 3D convex hulls, quadtrees and its fixed dimensional extensions, \( \epsilon \)-nets, nearest neighbors, and geometric separators in any fixed dimensions

More recently, *property testing* [154, 288], a subfield of theoretical computer science — inspired by PAC learning [340], holographic proofs
Scalable Algorithms

... has led to thriving developments of sub-linear-time algorithms [291]. These results, together with the work of Vapnik-Chervonenkis [342] and Johnson-Lindenstrauss [180], have demonstrated the power of sampling in scalable algorithm design. On the practical side, a rich body of scalable algorithms has been developed in fields of network science [184, 183, 230, 357, 354, 349, 64, 325, 324, 84], machine learning [182, 218, 143], and numerical computing [68, 190].

1.5 Article Organization

In this article, we will survey scalable algorithmic techniques, particularly those based on rapid progress in spectral graph theory, numerical analysis, probabilistic methods, and computational geometry. Many of these techniques are simple on their own, but together they form a powerful toolkit for designing scalable algorithms. After a brief review of network models in Chapter 2, we will proceed with the technical chapters of this article as follows:

In Chapter 3, “Significant Nodes: Sampling — Making Data Smaller,” we will start by focusing on the smallest structures in big networks — nodes. We will discuss results from [65] that incorporate efficient local network exploration methods into advanced sampling schemes.

Both sampling and local network exploration are widely used techniques for designing efficient algorithms. Here, the combination of the two leads to the first super-scalable algorithm for identifying all nodes with significant PageRank values in any network [65]4. Leaving the details of local network exploration for the next chapter, this short chapter will focus on an annealing approach to construct robust PageRank estimators. This approach uses a multi-precision sampling scheme to

4This result was covered by Richard Lipton and Kenneth Regan, under title “Shifts In Algorithm Design,” on their popular blog Gödel’s Lost Letter and P=NP (https://rjlipton.wordpress.com/2014/07/21/shifts-in-algorithm-design/). The surprising conclusion that one can in fact identify all nodes with significant PageRank values without examining the entire network also landed this result on the list of “Top ten algorithms preprints of 2012” by David Eppstein (http://11011110.livejournal.com/260838.html).
1.5. Article Organization

guide a local statistics-gathering algorithm. We will show that this annealing method can build a robust PageRank estimator by visiting only a sub-linear number of nodes in the network. The reverse-structural techniques of this algorithm have been used in subsequent scalable algorithms for computing other network centrality measures [84], and for influence maximization [64, 325, 324].

Keywords: Markov processes; PageRank; personalized PageRank; PageRank matrix; multi-precision sampling; multi-precision annealing; Riemann estimator.

In Chapter 4, “Clustering: Local Exploration of Networks,” we will turn our attention to slightly larger structures in networks — clusters. We will survey the basic framework of local network exploration algorithms introduced in [318]. These algorithms conduct locally expandable searches of networks: Starting from a small set of input nodes of a network, local algorithms iteratively expands its knowledge about the “hidden” network by only exploring the neighborhood structures of the explored nodes.

We first discuss a family of provably-good local clustering algorithms [318, 22, 93, 24, 23]. We will then analyze two scalable algorithms for personalized-PageRank approximation. These local algorithms highlight the usefulness of graph-theoretical concepts, such as random walks, personalized PageRank, and conductance, for analyzing network structures, both mathematically and algorithmically. We then conclude the section with a study of the interplay between network structures and dynamic processes (such as random walks, social influence, or information propagation). In particular, we will discuss the framework introduced in [146] for quantifying the impact of this interplay on the clusterability of subsets in the network, and prove a parameterized Cheeger’s inequality [82].

This chapter, together with Section 5.8 Chapter 6 and Chapter 7, also provide us with a quick tour through spectral graph theory.

5 Several of these local clustering algorithms have been implemented by Konstantin Voevodski of Google (http://gaussian.bu.edu/lpcf.html) and used in the context of protein network analysis.
Keywords: clusterability measures; conductance; Laplacian matrix; Cheeger’s inequality; sweep with eigenvectors; local network-analysis algorithms; local clustering; power methods; random-walk sampling; interplay between dynamic processes and networks; spectral graph theory.

In Chapter 5, “Partitioning: Geometric Techniques For Data Analysis,” we will focus on networks defined by geometric data. We illustrate the power of geometric techniques — such as spatial decomposition and divide-and-conquer — in scalable data analysis. The geometric structures, such as nearest neighborhood graphs, also offer potentially useful measures of clusterability and cluster stability [38], which have both structural and algorithmic consequences. We will discuss a family of geometric partitioning techniques [6] and apply them to the computation of nearest neighborhood graphs and geometric clusters. These techniques lead to powerful scalable geometric divide-and-conquer schemes [252, 253]. They also provide a beautiful bridge between spectral graph theory and network analysis, involving a popular spectral partitioning method [315].

Keywords: nearest neighborhood graphs; geometric graphs; geometric partitioning; spectral partitioning; separator theorems; centerpoints; evolutionary algorithms, conformal maps; geometric divide-and-conquer; VC dimensions; random projection; spectral projection.

In Chapter 6, “Spectral Similarity: Sparsification — Making Networks Simpler,” we will focus on what is means to say “one network is similar to another network.” We will address three basic questions:

- **Conceptual Question:** How should we measure the similarity between two given networks?

- **Mathematical Question:** Does every graph have a “similar” sparse graph?

Many algorithms discussed in this chapter have been implemented [148] and are available in MESHPART, a Matlab mesh partitioning and graph separator toolbox (http://www.cerfacs.fr/algor/Softs/MESHPART/).
1.5. Article Organization

- **Algorithmic Question**: Is there a scalable algorithm for constructing a good sparsifier?

This is my favorite subject in spectral graph theory.

**Keywords**: spectral similarity; cut-similarity; effective resistances; spectral sparsification; low-stretch spanning tree; matrix sampling; conjugate gradient; PageRank completion of networks.

In Chapter 7, “Electrical Flows: Laplacian Paradigm for Network Analysis,” we survey a family of scalable algorithmic toolkits. At the heart of these toolkits is a scalable solver for Laplacian linear systems and electrical flows [313, 318, 317, 319]. This scalable Laplacian solver has initiated and enabled many new scalable algorithms for spectral approximation [319], geometric and statistical approximation [98], graph embedding [335], machine learning [358], numerical methods, random-walk approximation, and Gaussian sampling in graphical models [86].

These applications illustrate a powerful, general algorithmic framework — called the Laplacian paradigm — for network analysis. In this framework, we attempt to reduce an optimization problem to one or more linear algebraic or spectral graph-theoretical problems that can be solved efficiently by the Laplacian solver or primitives from this scalable family. In addition to the applications above, this framework has also led to several recent breakthrough results — including scalable algorithms for max-flow/min-cut approximation [303, 197, 275, 243, 282] and sparse Newton’s method [86] — that have gone beyond the original scalable Laplacian linear solvers [319, 215, 200, 97, 217].

These success stories point to an exciting future for scalable algorithm design in data and network analysis. I hope that continuing advancements will help enrich our scalable algorithmic library, and inspire new efficient solutions for data and network analysis.

**Keywords**: SDD primitive; electrical flows; Laplacian linear systems; spectral approximation; graph embedding; machine learning; Gaussian sampling; Gaussian Markov random fields; random walk sparsification; sparse Newton’s method; Laplacian paradigm.
In Chapter 8 “Remarks and Discussions,” we conclude this article with a few “inconclusive” remarks. The inconclusiveness reflects the conceptual challenges that we usually face in data and network analysis. We also discuss a few frameworks beyond the commonly-used graph-theoretical network models to address fundamental conceptual questions in data analysis and network science.

**Keywords:** centrality; clusterability; k-means methods; multifaceted network data; beyond graph-based network models; incentive networks; interplay between influence processes and social networks; social choice theory; axiomatization; game theory; cooperative games; Shapley value; behaviors of algorithms; beyond worst-case analysis.

### A.1 Appendix to the Chapter

#### A.1.1 Basic Types of Computational Problems

In complexity theory, one usually classifies basic computational problems into three types [307, 103]:

**Decision Problems:** A decision problem concerns the membership of a language \( L \subseteq \{0, 1\}^* \): Given an input string \( x \in \{0, 1\}^* \), one is asked to determine if \( x \in L \). The output of a decision problem has constant complexity as the answer is either YES or NO (or DON’T KNOW, when randomization is used). This family of problems is commonly used to capture the computational challenge in deciding whether or not an input problem has a feasible solution.

**Search Problems (or function problems):** A search problem typically works with a binary relation, \( R \subseteq \{0, 1\}^* \times \{0, 1\}^* \): Given an input \( x \), one is asked to determine if there exists \( y \) such that \((x, y) \in R\), and furthermore, when such a \( y \) exists, one must also produce an element from solution\( R(x) = \{y \in \{0, 1\}^*|(x, y) \in R\} \). A search problem has two basic size measures: the size of the input \( x \) and the size of an output \( y \). Thus, the complexity for solving a search problem is measured by a function in terms of either or both of these sizes. If the scalability of an algorithm \( A \) (for a search problem \( \Pi \)) is bounded by a poly-logarithmic...
function with respect to the size of the output it produces, then we say that $A$ is *output-scalable*.

**Optimization Problems:** In a basic constrained optimization problem, we have a single utility/cost function $u$, whose value depends on multiple decision parameters $(x_1, x_2, ..., x_n)$. Each $x_i$ has its own domain $x_i \in \Omega^{(i)}$, and the feasible region is given by a global constraint:

$$(x_1, x_2, ..., x_n) \in C.$$ 

The optimization problem could either be a maximization or a minimization problem:

$$\text{optimize } u(x_1, x_2, ..., x_n)$$
$$\text{subject to } (x_1, x_2, ..., x_n) \in C \quad \text{and} \quad x_i \in \Omega^{(i)}, \forall i.$$ 

An input instance of an optimization problem is given by a representation of $(u, C, \Omega^{(1)} \times \cdots \times \Omega^{(n)})$. Like search problems, the complexity of an optimization problem (or an optimization algorithm) can be measured in terms of either or both input and output sizes.

**Beyond Decision, Search, and Optimization:** Other types of computational problems exist. For a binary relation $R$, for example, when given an input $x$, the *counting problem* aims to determine the number of solutions in $|\text{solution}_R(x)|$, the *enumeration problem* needs to identify all members in $\text{solution}_R(x)$, while the *sampling problem* generates an element from $\text{solution}_R(x)$, chosen according to uniform or a given distribution over $\text{solution}_R(x)$. The *multi-objective optimization problem* captures the potential trade-offs among several — possibly competing — objective functions in *multiple-criteria* decision-making:

$$\text{optimize } u_1(x_1, x_2, ..., x_n), ..., u_k(x_1, x_2, ..., x_n)$$
$$\text{subject to } (x_1, x_2, ..., x_n) \in C \quad \text{and} \quad x_i \in \Omega^{(i)}, \forall i.$$ 

A basic solution concept for multi-objective optimization is the *Pareto set*, which contains every feasible solution not strictly dominated by other feasible solutions.

The *game-theoretical problem* captures possible compromises among multiple decision makers in strategic decision-making, where each decision maker has his/her own utility function and can only determine
his/her own subset of decision parameters. The basic solution concept in game theory is the Nash Equilibrium\cite{259,258}. Schematically, there are $n$ players. The $i^{th}$ player has utility function $u_i$, and controls only input parameter $x_i \in \Omega^{(i)}$. These players have to jointly set their decision parameters in order to satisfy a global constraint: $(x_1, ..., x_n) \in C$. Each player’s utility usually depends on all decision parameters. The domain $\Omega^{(i)}$ is referred to as the strategy space for player $i$. The simplest example of a game is a two-player matrix game\cite{256,259,258}.

We can formulate the computation of Pareto points and Nash equilibria as search problems\cite{104,268,269,272,287}. However, we can capture more complex real-world phenomena using multi-objective optimization and game theory than with search and optimization\cite{271,269,270}.

A.1.2 Convention for Basic Notation

In this article, we will largely follow the conventions below for mathematical notation\cite{314}:

- **Lower-case English and Greek letters**:
  - Scalar constants, variables, and functions
  - Vertices in a graph

- **Upper-case English and Greek letters**:
  - Sets and graphs
  - Distributions and events
  - Constants (occasionally)

- **Lower-case English and Greek letters in bold font**:
  - Vectors
  
  If an $n$-dimensional vector is denoted by $v$, then we assume $v = (v_1, v_2, ..., v_n)^T$, where $T$ denotes the transpose operator. We use $v_i$ or $v[i]$ to denote the $i^{th}$ entry of $v$. We always assume $v$ is a column vector. Thus, its transpose, $v^T$, is a row vector.
1.5. Article Organization

- Permutation (mostly in Greek letters)
  If \( \pi \) denotes a permutation of \( n \) elements, then we use \( \pi_i \) or \( \pi[i] \) to denote its \( i^{th} \) element.

- Upper-case English and Greek letters in bold font:
  - Matrices
    If an \( m \times n \)-matrix vector is denoted by \( M \), then we use \( m_{i,j} \) or \( M[i,j] \) to denote its \((i,j)^{th}\) entry.

- Special matrices and vectors:
  - \( I_n \) denotes the identity matrix in \( n \) dimensions. When it is clear from the context, we use \( I \) to denote the identity matrix of the assumed dimensions.
  - \( 1 \) and \( 0 \), respectively, denote the vectors of all 1s or 0s in the assumed dimensions.
  - For \( v \in [n] \), \( 1_v \) denotes the \( n \)-place vector that has 1 at entry \( v \) and 0 everywhere else.
  - For \( S \subset [n] \), \( 1_S \) denotes the \( n \)-place vector that has 1 at entries in \( S \) and 0 everywhere else.

- \([n]\) or \([1: n]\) denotes the set of integers between 1 and \( n \). More generally, for integers \( a \leq b \), \([a : b]\) denotes the set of integers between \( a \) and \( b \).

- Matrix entry-wise inequality
  - For two \( m \times n \) matrices \( \mathbf{A} \) and \( \mathbf{B} \), and parameters \( \epsilon > 0 \) and \( c > 0 \), we use \( \mathbf{A} \preceq c \cdot \mathbf{B} + \epsilon \) to denote \( a_{i,j} \leq c \cdot b_{i,j} + \epsilon \), \( \forall i \in [m], \forall j \in [n] \).

- Respectively, \( \log \) and \( \ln \) denote the logarithm base 2 and the natural logarithm.

- The indicator random variable for an event \( A \) is \( \mathbf{I}[A] \) or \( [A] \).

- Respectively, \( \Pr_D[A] \) and \( \mathbb{E}_D[X] \) denote the probability of event \( A \) and the expectation of variable \( X \), over a distribution \( D \).
References


[22] R. Andersen, F. Chung, and K. Lang. Using PageRank to locally par-

[23] R. Andersen, S. O. Gharan, Y. Peres, and L. Trevisan. Almost opti-
mal local graph clustering using evolving sets. *Journal of the ACM*,
63:Article No. 15, 2015.

sets. In *Proceedings of the 41st Annual ACM Symposium on Theory of

[25] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approx-
imate nearest neighbor in high dimensions. *Communications of the

some deep representations. In *Proceedings of the International Confer-

[27] S. Arora, R. Ge, Y. Halpern, D. M. Mimno, A. Moitra, D. Sontag,
Y. Wu, and M. Zhu. A practical algorithm for topic modeling with
provable guarantees. In *Proceedings of the International Conference on

[28] S. Arora, R. Ge, T. Ma, and A. Moitra. Simple, efficient, and neural

[29] S. Arora, R. Ge, A. Moitra, and S. Sachdeva. Provable ICA with un-
known Gaussian noise, and implications for Gaussian mixtures and au-


verification and the hardness of approximation problems. *Journal of

Political Economy*, 58, 1950.


[34] D. Arthur, B. Manthey, and H. Röglin. Smoothed analysis of the k-


References


References


References


References


References


References

References


References


References


References


References


References


References


References


Full text available at: http://dx.doi.org/10.1561/0400000051