ON GRAPH ISOMORPHISM AND THE PAGERANK ALGORITHM

DISSERTATION

Christopher J. Augeri

AFIT/DCS/ENG/08-08

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

The views expressed in this dissertation are those of the author and do not reflect the official policy or position of the United States Air Force, the Department of Defense, or the United States Government.

AFIT/DCS/ENG/08-08

ON GRAPH ISOMORPHISM AND THE PAGERANK ALGORITHM

DISSERTATION

Presented to the Faculty

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

Christopher J. Augeri, A.A.S., B.G.S., M.S.

September 2008

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

AFIT/DCS/ENG/08-08

ON GRAPH ISOMORPHISM AND THE PAGERANK ALGORITHM

Christopher J. Augeri, A.A.S., B.G.S., M.S.

Approved:

Barry E. Mullins, Ph.D., P.E. (chairman)	12 Aug 08 Date
Rusty O. Baldwin, Ph.D., P.E. (member)	12 Aug 08 Date
Dursun A. Bulutoglu, Ph.D. (member)	7/31/2008 Date
Lt Col Leemon C. Baird III, Ph.D. (member)	31 Jul 08 Date

Accepted:

M. U. Thomas Dean, Graduate School of Engineering and Management Air Force Institute of Technology	14 Aug 08 Date

AFIT/DCS/ENG/08-08

Abstract

A graph is a key construct for expressing relationships among objects, such as the radio connectivity between nodes contained in an unmanned vehicle swarm. The study of such networks may include ranking nodes based on importance, for example, by applying the PageRank algorithm used in some search engines to order their query responses. The PageRank values correspond to a unique eigenvector typically computed by applying the power method, an iterative technique based on matrix multiplication.

The first new result described herein is a lower bound on the execution time of the PageRank algorithm that is derived by applying standard assumptions to the scaling value and numerical precision used to determine the PageRank vector. The lower bound on the PageRank algorithm's execution time also equals the time needed to compute the coarsest equitable partition, where that partition is the basis of all other results described herein.

The second result establishes that nodes contained in the same block of a coarsest equitable partition must yield equal PageRank values. The third result is an algorithm that eliminates differences in the PageRank values of nodes contained in the same block if the PageRank values are computed using finite-precision arithmetic. The fourth result is an algorithm that reduces the time needed to find the PageRank vector by eliminating certain dot products when any block in the partition contains multiple vertices. The fifth result is an algorithm that further reduces the time required to obtain the PageRank vector of such graphs by applying the quotient matrix induced by the coarsest equitable partition. Each algorithm's complexity is derived with respect to the number of blocks contained in the coarsest equitable partition and compared to the PageRank algorithm's complexity.

These results further existing research in several ways. For instance, the practical lower bound on the PageRank algorithm's execution time was previously only suggested using experimental results. The proof showing vertices contained in the same block of the coarsest equitable partition have equal PageRank values is based on relating dot products and Weisfeiler-Lehman stabilization, which is a much different approach than applied in an existing proof. The existing proof was also extended to show the quotient matrix could be used to reduce the PageRank algorithm's execution time. However, its authors did not develop an algorithm or analyze its execution time bounds. Finally, these results motivate several avenues of future research related to graph isomorphism and linear algebra.

Acknowledgments

A toast, to

my former colleagues in the Department of Computer Science at the United States Air Force Academy, for affording me the opportunity to pursue doctoral studies.

the Air Force Communications Agency (AFCA), for funding our books, travels, and resources—I hope I have returned your support in kind.

Dr. Barry Mullins, for your encouragement and feedback as my research advisor, and most significantly, providing the latitude I needed to complete this research.

Lt Col Leemon Baird, for mentoring me as a computer scientist, and particularly, the many discussions on complexity theory.

Dr. Dursun Bulutoglu, for showing me the beauty of linear algebra, loaning some key texts, and for hosting extended discussions on graph isomorphism.

Dr. Rusty Baldwin, for the various suggestions on the doctoral candidacy process and for catching a key omission during the dissertation review.

Dr. Mike Temple, for being my minor advisor and providing several suggestions throughout the dissertation process.

Dr. Michael Grimaila, for being the dean's representative to my committee and his thought-provoking questions.

Dr. Barry Mullins, Dr. Dursun Bulutoglu, Dr. Mike Temple, Dr. Andrew Terzuoli, Dr. Ken Hopkinson, Dr. Mark Oxley, and Maj Scott Graham, for being my instructors.

Kevin Morris and Greg Brault, for your continued collaboration, even as you have moved on to other duties.

Victor Hubenko, Kevin Cousin, Andy Leinart, Kevin Gilbert, Chris Mann, and all of my fellow graduate students, for your continued encouragement and friendship.

Janice Jones, Charlie Powers, David Doak, and everyone supporting the students at AFIT, thank you for your time and energy as we pursue our research.

the reader, may you enjoy reading this dissertation as much as I enjoyed preparing its contents for your consumption.

my family, for your love, patience, and support as I pursued this dream. I love you and look forward to spending more time with you.

All the best,

Chris Augeri

Table of Contents

Abstract iv

Acknowledgments vi

Table of Contents vii

List of Figures ix

List of Tables xi

List of Theorems xiii

List of Symbols xiv

I. Introduction 1
- 1.1. The PageRank Vector 1
- 1.2. Research Motivation 2
- 1.3. Problem Statement 4
- 1.4. Research Goals 5
- 1.5. Assumptions 7
- 1.6. Overview 8

II. Background 9
- 2.1. Ordering Nodes in Sensor Networks and Unmanned Vehicle Swarms 9
- 2.2. Deciding Isomorphism: A Classic Graph-Theoretic Problem 12
  - 2.2.1. The Relationship to Canonical Vertex Ordering 12
  - 2.2.2. Graph Isomorphism Applications 13
  - 2.2.3. A Formal Definition 14
  - 2.2.4. Canonical Isomorphs 17
- 2.3. Vertex Partitions 19
  - 2.3.1. The Degree Partition 22
  - 2.3.2. The Equitable Partitions 24
  - 2.3.3. The Coarsest Equitable Partition 26
  - 2.3.4. The Orbit Partition 36
  - 2.3.5. Induced Quotient Graphs and Matrices 45
- 2.4. A Brief Interlude: Eigen Decomposition Applications in Graph Theory 51
- 2.5. The PageRank Algorithm 52
  - 2.5.1. Computing the PageRank Perturbation 55
  - 2.5.2. Computing the PageRank Vector 60
  - 2.5.3. PageRank: An Algorithm for Ranking Vertices 62
- 2.6. Observations about Equitable Vertices and PageRank Values 68
- 2.7. Known Results 70
- 2.8. Summary 71

III. Establishing Equitable Equivalency 72
3.1. Overview 72
3.2. Lower Bound on the Expected Number of Power Method Iterations 73
3.3. Motivating Equitable Dot Products and PageRank Values 74
3.3.1. From Weisfeiler-Lehman Stabilization to Iterated Dot Products 75
3.3.2. Finding the Coarsest Equitable Partition By Iterated Dot Products 80
3.4. Relating Equitable Dot Products and PageRank Values 82
3.4.1. Equitable Dot Products 82
3.4.2. Equitable PageRank Values 84
3.4.3. Additional Equitable Relationships 85
3.4.4. Complexity Analysis 86

IV. Reducing Equitable Differences and Dot Products 88
4.1. Overview 88
4.2. Eliminating Equitable PageRank Differences 89
4.2.1. Numerical Differences and Equitable Vertices 89
4.2.2. AverageRank: An Algorithm for Eliminating Equitable Differences 91
4.2.3. Complexity Analysis 92
4.3. Eliminating Equitable PageRank Dot Products 93
4.3.1. Excess Dot Products and Equitable Vertices 93
4.3.2. ProductRank: An Algorithm for Eliminating Equitable Dot Products 95
4.3.3. Complexity Analysis 97
4.3.4. Algorithm Applicability 100

V. Lifting PageRank Values 101
5.1. Overview 101
5.2. Quotient Computations 102
5.3. Lifting the House Graph's PageRank Vector 105
5.4. QuotientRank: An Algorithm for Lifting PageRank Vectors 108
5.5. Complexity Analysis 110
5.6. A QuotientRank Example 114
5.7. QuotientRank Applicability 117

VI. Conclusions and Future Research 120
6.1. Conclusions 120
6.1.1. Complexity Bounds 120
6.1.2. Obtaining Canonical Vertex Orderings from the PageRank Vector 121
6.1.3. Relating the PageRank Vector and the Coarsest Equitable Partition 122
6.2. Future Work 124
6.2.1. Implementation Improvements 124
6.2.2. Other Linear Algebra Applications 125
6.2.3. *k*-D Weisfeiler-Lehman Stabilization 126
6.2.4. Open Call for Parallel Software that Decides Graph Isomorphism 127
6.3. Summary 129

Bibliography 130

List of Figures

Figure Page

1. Mansion Graph [CDR07] 2
2. House Graph [Gol80, Gol04] 3
3. House Graph's 3-Block Coarsest Equitable Partition, $\left[\{c,d\},\{a\},\{b,e\}\right]$ 4
4. House Graph and Its Induced Quotient Graph 6
5. Non-Connected Graph 7
6. Two Graph Isomorphs: The Triangle and Twisted Rope 12
7. Two Chemical Isomers: $C_2H_2F_2$ 13
8. Two Isomorphs: The Square and Hourglass 14
9. Canonical Isomorph's Permutation Triangle 18
10. Vertex Partition Tree of an Arbitrary 3-Vertex Graph 21
11. Two Graphs Yielding Different Degree Sequences 22
12. Devil's Pair for the Sorted Degree Sequence 23
13. Exploring Equitable Partitions 25
14. Coarsest Equitable Partitions and Canonical Orderings 26
15. Method 1: Finding the House Graph's Coarsest Equitable Partition 28
16. Method 2: Fast Algorithm for Finding Equitable Partitions [PaT87, KrS98] 30
17. Method 2: Finding the House Graph's Coarsest Equitable Partition 31
18. Method 3: 1-D Weisfeiler-Lehman Stabilization 32
19. Method 3: 1-D Weisfeiler-Lehman Stabilization Example 33
20. Method 3: Finding the House Graph's Coarsest Equitable Partition 33
21. Isomorph and Automorph of the Square 36
22. House Graph's Coarsest Equitable and Orbit Partition, $\left[\{c,d\},\{a\},\{b,e\}\right]$ 37
23. Cuneane Graph's Distinct Coarsest Equitable and Orbit Partitions 37
24. House Graph's Coarsest Equitable Partition, $\left[\{c,d\},\{a\},\{b,e\}\right]$ 38
25. Refining to Equitable Partitions after Vertex Individualization 39
26. House Graph's Vertex Partition Tree (Equitable Partitions Boxed) 40
27. House Graph 41
28. Easy, Medium, and Hard: Discrete, Non-Discrete and Unit Partitions 44
29. House Graph's Induced Quotient Graph 46
30. 3-D Buckyball Drawing Based on Its Signless Laplacian's Eigenvectors 51
31. Mansion Graph 52
32. House Graph 53
33. PageRank: An Algorithm for Ordering Vertices [PBM+98] 63
34. Paw Graph [Wes01] 64
35. Applying the PageRank Perturbation to the Paw Graph, $\alpha = 0.85$ 64
36. Paw Graph's PageRank Vector, $\alpha = 0.85$ 66
37. Paw Graph's PageRank Ordering, $\alpha = 0.85$ 66
38. Cuneane Graph's Coarsest Equitable and Orbit Partitions 67

39. Coarsest Equitable Partitions of the House and Octahedron Graphs 68
40. Two Graphs Yielding a Non-Discrete Coarsest Equitable Partition.....69
41. 1-D Weisfeiler-Lehman Stabilization Using Primes and Dot Products.....81
42. 1-D Weisfeiler-Lehman Stabilization Using Matrices.....82
43. Graph Yielding Different Coarsest Equitable and Orbit Partitions 85
44. 9-Vertex Tree: A Graph Yielding a 4-Block Equitable Partition 89
45. AverageRank: An Algorithm for Ensuring Equitable PageRank Values 91
46. House Graph and Its 3-Block Coarsest Equitable Partition.....94
47. ProductRank: An Algorithm for Eliminating Equitable Dot Products 96
48. 4×4 Grid: A Graph Yielding a 3-Block Equitable Partition.....99
49. Pseudo-Benzene: A Graph Yielding a 2-Block Equitable Partition [StT99] 101
50. House Graph's 3-Block Coarsest Equitable Partition.....105
51. QuotientRank: An Algorithm for Lifting PageRank Vectors.....109
52. Pseudo-Benzene Graph and Its PageRank-Induced Quotient Graph 114
53. 8×3 Grid: A Graph Yielding a Non-Discrete Coarsest Equitable Partition.....119
54. Canonical PageRank Orderings of the Mansion and House Graphs 121

List of Tables

Table Page

1. Mansion Graph's PageRank Vector 2
2. House Graph's PageRank Vector 3
3. Two Adjacency Matrix Isomorphs: The Triangle and Twisted Rope 12
4. Two Isomorphs: The Adjacency Matrices of the Square and Hourglass 15
5. Identity Matrix and Permutation Matrices for $\phi = [2,1,3,4]$ 15
6. Establishing $\mathbf{A}_2 = \mathbf{P} \cdot \mathbf{A}_1 \cdot \mathbf{P}^T$ and $\mathbf{A}_1 \cong \mathbf{A}_2$ 15
7. Computing the Inverse Permutation 16
8. Computing Inverse Permutation Matrices 16
9. Three Isomorphs of the House Graph's Adjacency Matrix 17
10. Method 2: Finding the House Graph's Coarsest Equitable Partition 31
11. The House Graph's Adjacency and Sorted Degree Matrices 34
12. Method 3: 1-D Weisfeiler-Lehman Stabilization of the House Graph 34
13. Isomorph and Automorph of the Square 36
14. Partial Permutations of the House Graph's Adjacency Matrix 41
15. House Graph Isomorphs 42
16. House Graph's Induced Quotient Matrix 46
17. House Graph's and Its Induced Quotient Graph's Eigenvalues 47
18. Block Matrix, **B**, of the House Graph's Coarsest Equitable Partition 48
19. Block Matrix, **N**, of the House Graph's Coarsest Equitable Partition 48
20. Lifting a Dominant Eigenvector of the House Graph 49
21. Mansion Graph's PageRank Vector 52
22. House Graph's PageRank Vector 53
23. House Graph's Adjacency Matrix, **A** 55
24. House Graph's Degree Matrix and Degree Matrix Inverse 55
25. A Row-Stochastic Matrix, $\sum \mathbf{S}(i,:) = 1$ 56
26. A Column-Stochastic Matrix, $\sum \mathbf{S}(:,j) = 1$ 56
27. House Graph's PageRank Matrix, **S**, $\alpha = 0.85$ 57
28. Paw Graph's Adjacency and PageRank Matrices, $\alpha = 0.85$ 65
29. Paw Graph's PageRank Vector, $\alpha = 0.85$ 65
30. Iterated Dot Products of the House Graph's Adjacency Matrix 75
31. Iterated Prime Dot Products of the House Graph's Adjacency Matrix 77
32. Constructing the First Prime Diagonal Matrix 78
33. First Prime Dot Product Iteration 78
34. Two Equal Dot Products, $\mathbf{x} \cdot \mathbf{z}^T = \mathbf{y} \cdot \mathbf{z}^T$ 79
35. A 9-Vertex Tree's Adjacency Matrix 89
36. A 9-Vertex Tree's PageRank Vector, $\alpha = 0.85$ 90
37. House Graph's Adjacency and Degree Matrix 94
38. House Graph's Stochastic PageRank Matrix, **S**, $\alpha = 0.85$ 94
39. Initial PageRank Power Method Iterations of the House Graph 94

40. 4×4 Grid Graph's 3-Block Coarsest Equitable Partition 99
41. Characteristic Block Matrix, **B** 105
42. Characteristic Block Matrix Products.....105
43. House Graph's Adjacency Matrix, **A**.....106
44. House Graph's PageRank Matrix, **S**, $\alpha = 0.85$ 106
45. Quotient Matrix, **Q**, of the House Graph's PageRank Matrix, **S**, $\alpha = 0.85$ 106
46. Eigenvalues of the PageRank and Quotient Matrices.....107
47. Dominant Eigenvectors of the PageRank and Quotient Matrices 107
48. A 2×2 PageRank-Induced Quotient Matrix, **Q**.....114
49. Lifting the PageRank Vector from the Dominant Eigenvector.....115
50. Theoretical Iterations: $n = 12,\ b = 2,\ \tau = 2^{-52} \ \rightarrow\ t = 222,\ r = 64$ 116
51. Observed Iterations: $n = 12,\ b = 2,\ \tau = 2^{-52} \ \rightarrow\ t = 60,\ r = 62$ 116

List of Theorems

Theorem Page

1. PageRank: Practical Lower Bound on Power Method Iterations 73
2. Equitable Dot Products 82
3. Equitable PageRank Values 84
4. Dominant Eigenvalue of Equitable Quotient Matrix of a PageRank Matrix.....103
5. Lifting a PageRank Vector from an Equitable Quotient Matrix 104
6. QuotientRank: Practical Lower Bound on Power Method Iterations.....112
7. QuotientRank: Upper Bound on Power Method Iterations 112

Symbol	Meaning
$G=(V,E)$	simple graph, G, with a vertex set, V, and an edge set, E
V	vertex set, $V=\{v_1,v_2,\ldots,v_{n-1},v_n\}$, $1\le i\le n$
E	edge set, $E=\{e_1,e_2,\ldots,e_{m-1},e_m\}$, $m\le n\cdot(n-1)/2$
v_i	arbitrary, but specific, vertex of a vertex set, V
e_i, $e_i=\{v_j,v_k\}$	arbitrary, but specific, edge of an edge set, E
$\deg(v_i)$	degree of vertex (number of incident edges)
$B=[b_1,b_2,\ldots,b_k]$	disjoint vertex partition, $V=b_1\dot\cup b_2\dot\cup\cdots+\dot\cup b_k$, $b_i\cap b_j=\varnothing$
$\mathbf{x}^{n,1}$	$n\times 1$ vector (lower-case)
$\mathbf{M}^{n,n}$	$n\times n$ matrix (upper-case)
$\mathbf{A}$	adjacency matrix
$\mathbf{D}$	diagonal matrix, (cf. diag below)
$\mathbf{I}$	identity matrix
$\mathbf{P}$	permutation matrix
$\mathbf{1}$, $\mathbf{J}$	matrix whose entries equal one
$\mathbf{0}$, $\mathbf{Z}$	matrix whose entries equal zero
$\mathbf{M}_{i,j}$ or $\mathbf{M}(i,j)$	matrix element (i^{th} row, j^{th} column)
$\mathbf{M}^T$	matrix transpose
$\mathbf{M}_1\cdot\mathbf{M}_2$ or $\mathbf{M}_1\times\mathbf{M}_2$	matrix multiplication (dot product)
$\lambda\ (\Lambda)$	eigenvalue vector (matrix)
$\mathbf{X}$	eigenvectors
$\lVert\mathbf{A}\rVert_p$	vector (matrix) norm, $1\le p\le\infty$
$\mathbf{D}=\mathrm{diag}_i(\mathbf{d})$	construct matrix with $\mathbf{d}$ on i^{th} diagonal, $-(n-1)\le i\le n$
$\mathbf{d}=\mathrm{diag}_i^{-1}(\mathbf{D})$	extract i^{th} diagonal, $-(n-1)\le i\le n$
P	deterministic polynomial complexity
NP	non-deterministic polynomial complexity
$\Omega(n)$	lower bound
$\mathrm{O}(n)$	upper bound
$\Theta(n)$	exact bound
ϕ	permutation, e.g., $\phi=[4,3,5,2,1]\to[d,c,e,b,a]$
$\cong$	isomorphism, e.g., $G_1\cong G_2$ or $\mathbf{A}_1\cong\mathbf{A}_2$
$v_i\to v_j$	permutation mapping of an arbitrary vertex
ω	canonical isomorph, e.g., G_ω or $\mathbf{A}_\omega$

ON GRAPH ISOMORPHISM AND THE PAGERANK ALGORITHM

I. Introduction

1.1. The PageRank Vector

A graph is a useful construct for expressing relationships between a set of objects, such as the radio connectivity among nodes in an unmanned aerial vehicle (UAV) swarm. Some analysis tools use a measure of node centrality to rank a graph's vertices, such as a UAV swarm's nodes, by their relative importance. For instance, the PageRank algorithm is used in certain search engines to rank query responses [PBM+98].

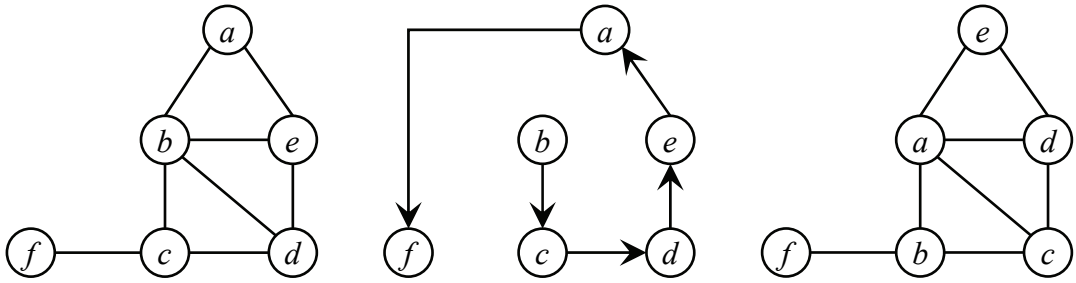
The algorithm produces a unique eigenvector, the PageRank vector, whose entries equal the probability each node is visited by an object that randomly traverses the graph. The PageRank algorithm uses the PageRank vector, which is guaranteed to exist, to order a graph's vertices, such as a UAV swarm's nodes, according to their probability of being randomly visited. The results described in Chapters 3–5 reduce the time needed to obtain a swarm's PageRank vector if the swarm contains nodes having equal PageRank values.

In the context of UAV swarms, the PageRank vector can be used to identify where to inject a message to ensure it is efficiently disseminated among the swarm's nodes by a rumor-routing protocol. In social network analysis, a PageRank vector can be used to find group members that are useful for efficiently spreading (mis)information. The PageRank vector can also be used to find roadblock locations for capturing fleeing suspects. A more sedate application determines which road intersections to avoid during a city's rush hour. In general, the PageRank vector, which is computed by the PageRank algorithm, is useful for analyzing the probable behavior of some object traversing a network's nodes, such as some (mis)information distributed by a rumor-routing protocol in a UAV swarm.

1.2. Research Motivation

The PageRank vector often canonically orders the graph's vertices. For example, the mansion graph shown in Figure 1(a) [CDR07] yields the PageRank vector shown in Table 1(a). Sorting the entries of this vector in descending order yields the vector listed in Table 1(b). The vertex ordering induced by the sorted vector is illustrated in Figure 1(b).

The PageRank vector is unique up to graph isomorphism, where graphs are said to be isomorphs if their edges define equivalent relationships on their vertices. Since entries in the mansion graph's PageRank vector are distinct, all mansion graph isomorphs induce the same vertex ordering, i.e., the canonical ordering shown in Table 1(b). For example, the isomorph shown in Figure 1(c) yields the sorted PageRank vector listed in Table 1(c), which is equivalent to the ordering listed in Table 1(b) and illustrated in Figure 1(b).



(a) Mansion Graph (b) PageRank Order (c) Another Isomorph

Figure 1. Mansion Graph [CDR07]

Table 1. Mansion Graph's PageRank Vector

(a) PageRank Vector

a	0.126
b	0.236
c	0.195
d	0.182
e	0.180
f	0.080

(b) PageRank Order

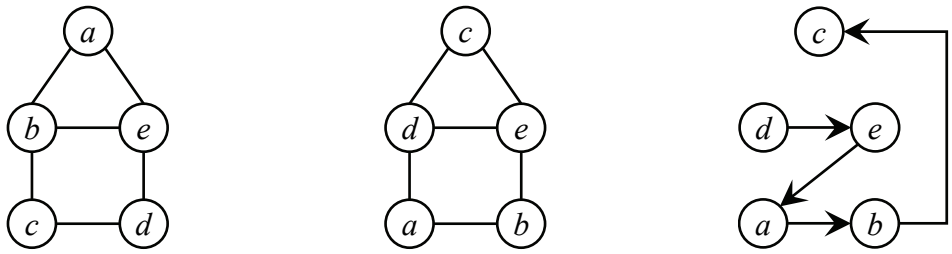
0.236	*b*
0.195	*c*
0.182	*d*
0.180	*e*
0.126	*a*
0.080	*f*

(c) Isomorph's Order

0.236	*a*
0.195	*b*
0.182	*c*
0.180	*d*
0.126	*e*
0.080	*f*

However, a PageRank vector may contain duplicate entries, which cannot induce a canonical ordering. The issue can be resolved in some applications, e.g., search engines, by sorting on other keys, such as the data's original location. A second method is to order vertices of a canonical isomorph according to its PageRank vector. For instance, *nauty* is often used to determine such a canonical isomorph [McK81, McK04].

For example, the house graph shown in Figure 2(a) [Gol80, Gol04], which yields the PageRank vector listed in Table 2(a). The house graph's canonical isomorph produced by *nauty* is listed in Figure 2(b) and yields the PageRank vector listed in Table 2(b). The vertex mapping, $i \rightarrow j \Leftrightarrow r \rightarrow s$ denotes vertex v_i, labeled *r*, is mapped to the vertex v_j, labeled *s*. The canonical isomorph induces the canonical ordering listed in Table 2(c) and depicted in Figure 2(c). The tie in PageRank value between vertices *b* and *e*, or similarly, vertices *c* and *d*, are broken by their relative order in the canonical isomorph.



(a) House Graph (b) Canonical Isomorph (c) PageRank Order

Figure 2. House Graph [Gol80, Gol04]

Table 2. House Graph's PageRank Vector

(a) House Graph

a	0.168
b	0.244
c	0.172
d	0.172
e	0.244

(b) Canonical Isomorph

$1 \rightarrow 3 \Leftrightarrow a \rightarrow c$	0.168
$2 \rightarrow 4 \Leftrightarrow b \rightarrow d$	0.244
$3 \rightarrow 1 \Leftrightarrow c \rightarrow a$	0.172
$4 \rightarrow 2 \Leftrightarrow d \rightarrow b$	0.172
$5 \rightarrow 5 \Leftrightarrow e \rightarrow e$	0.244

(c) PageRank Order

0.244	d
0.244	e
0.172	a
0.172	b
0.168	c

1.3. Problem Statement

The PageRank vector of many graphs, such as the mansion graph, contains unique entries and induces a canonical vertex ordering. In contrast, the PageRank vector of some graphs, such as the house graph, contains duplicate entries and cannot induce a canonical vertex ordering. However, graphs containing vertices that yield the same PageRank value suggest several methods of improving the PageRank algorithm's performance.

One avenue to improvement is yielded by the graph's coarsest equitable partition, an invariant often used in applications that find canonical isomorphs, e.g., *nauty*. Vertices contained in the same block are adjacency-wise equivalent, or equitable, with respect to the vertices in other blocks and more importantly, appear to yield equal PageRank values. For example, the house graph's coarsest equitable partition is $\left[\{c,d\},\{a\},\{b,e\}\right]$, which is depicted using distinct shapes in Figure 3. The PageRank values yielded by the vertices contained in each block are $\left[0.172, 0.168, 0.244\right]$, respectively, corresponding to Table 2.

The first task is to show a relationship exists between PageRank values of vertices contained in the same block. If a relationship exists, an algorithm must be developed that ensures vertices contained in the same block, e.g., *b* and *e*, have equal PageRank values, within an arbitrary finite precision. The last task is to construct algorithms that reduce the execution time needed to compute the PageRank vector of such graphs.

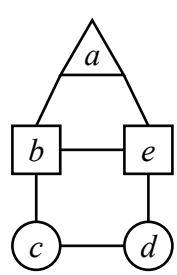


Figure 3. House Graph's 3-Block Coarsest Equitable Partition, $\left[\{c,d\},\{a\},\{b,e\}\right]$

1.4. Research Goals

The first goal is to obtain a lower bound on the PageRank algorithm's complexity by applying recent work that determined its upper bound [HaK03]. The second goal is to establish a relationship between the coarsest equitable partition and the PageRank vector. The key insight is obtained by constructing a modified dot product process that performs 1-D Weisfeiler-Lehman stabilization, which yields the coarsest equitable partition. Since PageRank values can be obtained by applying the power method, which is an iterated dot product process, vertices contained in the same block of the coarsest equitable partition must yield equal PageRank values. Another proof of the relationship between the coarsest equitable partition and PageRank vectors was developed by Boldi *et al*. [BLS+06].

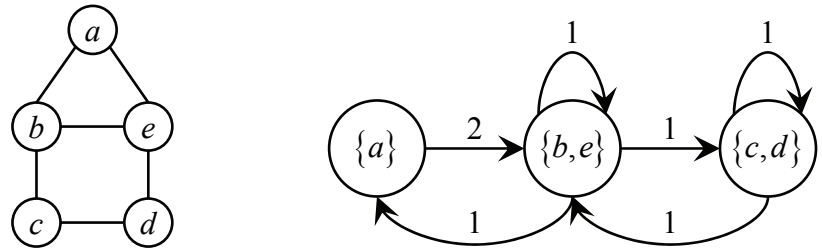
The third goal is to exploit the relationship between the PageRank vector and the coarsest equitable partition to eliminate differences between PageRank values of vertices contained in the same block of the coarsest equitable partition, where such differences may occur if PageRank values are computed using finite numerical precision. The fourth, and most important, goal is to reduce the time needed to obtain the PageRank vector. The third and fourth goals are achieved by the three algorithms described in Chapters 4 and 5.

The first algorithm described in Chapter 4, AverageRank, sets the PageRank value of every vertex to the average PageRank value of the vertices in its corresponding block. The second algorithm, ProductRank, reduces the time needed to find the PageRank vector by only computing a subset of the dot products defined in each power method iteration and ensures each block's vertices have equal PageRank values. Hence, the ProductRank algorithm supersedes the AverageRank algorithm, since it computes the PageRank vector more efficiently if the graph's coarsest equitable partition is non-discrete.

The third and most notable algorithm, QuotientRank, is described in Chapter 5. This algorithm obtains the PageRank vector by obtaining the dominant eigenvector of the quotient graph induced by the coarsest equitable partition. The quotient graph's vertices correspond to a partition's blocks and the edge weights denote the number of edges from any source block vertex to each destination block's vertices [God93, GoR01, McK04].

For example, the house graph depicted in Figure 4(a) yields the coarsest equitable partition, $\left[\{a\},\{c,d\},\{b,e\}\right]$, which induces the quotient graph illustrated in Figure 4(b). Since vertex *a* is connected to vertices *b* and *e*, an edge of weight '2' proceeds from block $\{a\}$ to block $\{b,e\}$. Similarly, vertices *b* and *e* are linked, as are vertices *c* and *d*, thus, a loop of weight '1' is attached to blocks $\{b,e\}$ and $\{c,d\}$. Every vertex in block $\{b,e\}$ is connected to a vertex in block $\{c,d\}$, thus, an edge of weight '1' links these blocks. This quotient graph yields the same PageRank ordering illustrated in Table 2.

Boldi *et al*. showed a quotient graph can reduce the time required to compute the PageRank vector, but did not construct or analyze any such method [BLS+06]. Another proof of that result is developed in Section 5.2. The remainder of Chapter 5 describes and analyzes the QuotientRank algorithm, which uses a certain quotient matrix to reduce the time needed to compute the PageRank vector.



(a) House Graph (b) Induced Quotient Graph

Figure 4. House Graph and Its Induced Quotient Graph

1.5. Assumptions

A common assumption, which is applied herein, is that each input graph is simple. That is, an undirected edge links two distinct vertices, and each vertex pair is linked by at most a single edge. For example, the house graph shown in Figure 4(a) is a simple graph. Conversely, the quotient graph induced by the graph's coarsest equitable partition is often a weighted directed graph, as shown in Figure 4(b). The simple graph assumption is only applied for convenience—the results described herein can be generalized to other classes of graphs, such as directed graphs and weighted graphs.

Another conventional assumption used herein is that an input graphs is connected. Thus, each vertex can be reached from any other vertex by traversing one or more edges. For example, the house graph is connected, since non-adjacent vertices can be reached by simply traversing two or more edges, e.g., $a \to e \to d$ or $b \to c \to d$. Conversely, the graph depicted in Figure 5 is not connected, since it contains two connected components, a triangle and a square.

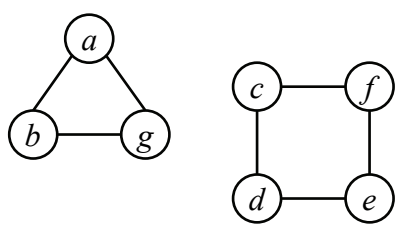


Figure 5. Non-Connected Graph

If a graph contains n vertices, any permutation applied to the graph must contain n distinct elements, e.g., $V = \{a,b,c,d,e\}$ and $\phi = [3,2,1,4,5]$. Finally, any phrases such as "up to a permutation" or "up to isomorphism" means that the particular graph property is similarly permuted by a vertex permutation. For example, a PageRank vector is one such property, as reflected by the house graph isomorphs shown in Figure 2 and Table 2.

1.6. Overview

The constructs presented in this chapter, such as the coarsest equitable partition and the quotient graph, are formally defined in Chapter 2, along with concepts such as the orbit partition. The next few sections describe the PageRank algorithm and its use of the power method. Chapter 2's last section describes some related results, such as the work of Boldi *et al*., who independently established the relationship between the graph's coarsest equitable partition and PageRank vector by constructing an alternate proof [BLS+06].

Chapter 3 begins by deriving a practical lower bound on the execution time of the PageRank algorithm. The remainder is devoted to establishing a relationship between the coarsest equitable partition and the PageRank vector. This relationship yields two simple methods described in Chapter 4 that improve the PageRank algorithm's performance. The first, the AverageRank algorithm, eliminates PageRank value differences among vertices in the same block. The second, the ProductRank algorithm, decreases the time required to obtain a PageRank vector by computing a single PageRank value for each block. A more dramatic reduction is yielded by the QuotientRank algorithm described in Chapter 5, which lifts a PageRank vector from the quotient matrix and further extends [BLS+06].

As summarized in Chapter 6, these algorithms improve the PageRank algorithm's performance if a graph's coarsest equitable partition contains at least one block composed of multiple vertices. One section in Chapter 6 explores potential future research avenues. Some of these avenues include generalizing the proof relating a graph's coarsest equitable partition and PageRank vector, constructing a graph library based on that generalization, and developing functions that improve the performance of additional linear algebra tasks using the techniques described in Chapters 3–5.

II. Background

2.1. Ordering Nodes in Sensor Networks and Unmanned Vehicle Swarms

A distributed sensor network (DSN) is a heterogeneous set of nodes for collecting data in a specified environment. Many proposed DSN applications, such as an earthquake detection network, may interact with infrastructure systems, e.g., moving elevators to the first floor [ElE04]. DSNs currently monitor glaciers [Gui05], animal species [SOP+04], and road traffic conditions [Tra07]. Some military applications include monitoring soldier health using (small!) ingested sensors or sensors sewn into uniforms [TBH+04], locating snipers using either fixed or helmet-mounted sensors [LNV+05, DBC+98] and detecting radiation [BMT+04].

An unmanned aerial vehicle (UAV) swarm is a heterogeneous network of mobile vehicles whose sensors can collect multi-dimensional data [IyB05, ZhG04]. UAV swarms may collaborate, e.g., by cooperatively searching a geographic area for objects matching some search criteria [MMP+06, Mor06, PBO03]. Related efforts include automated aerial refueling [Spi06], attacking targets using munitions deployed from an unmanned combat aerial vehicle (UCAV) [Cla00, JAU, OSD05, USA05], and decreasing payload weight by using more advanced sensors [Rig03].

A natural problem to consider involves ordering nodes based on some measure of relative importance. An attacker who knows the $1^{st}, 2^{nd}, \ldots, n^{th}$ most critical network node knows where to expend the most resources. Some related problems include finding nodes to facilitate spreading malicious data (in social networks, diseases or rumors), computing an optimal node polling order (traveling salesman), adding network services (upgrading), or sequencing nodes for transmission (logistics planning).

As described in Section 1.2, an ideal node ordering is canonical and independent of the node input order. Alternatively stated, if the graphs of two networks are isomorphs, the networks ideally yield the same $1^{st}, 2^{nd}, \ldots, n^{th}$ canonical node ordering. For example, a simple heuristic orders nodes based on their number of neighbors. The node linked to the greatest number of nodes is first, then the node linked to the second-most number of nodes, and so forth. However, since every node may have the same number of neighbors, such a node ordering is typically not canonical. Similarly, sorting the nodes based on their relative geographic positions also may not define a canonical ordering. Thus, more robust methods of ordering nodes are typically needed to obtain a canonical node ordering.

One robust method of ordering a network's nodes based on relative importance is the PageRank algorithm used in some search engines to order query responses, e.g., the web pages matching a user's search criteria [PBM+98]. The PageRank algorithm perturbs the adjacency matrix corresponding to the original network such that the resulting matrix specifies the probability of visiting each node from any other node. The perturbed matrix satisfies the Perron-Frobenius theorem's conditions. Therefore, the matrix yields a unique dominant eigenvector whose entries correspond to the probability of visiting each node.

The PageRank algorithm orders the nodes, e.g., the query responses, based on this eigenvector's entries, which correspond to the stationary distribution of the Markov chain defined by the perturbed adjacency matrix. Thus, the PageRank algorithm finds a unique vector, the PageRank vector, where a node's PageRank value also determines its position in the node ordering. This ordering corresponds to the order nodes are likely to be visited by an object that randomly selects the next node to visit, e.g., a user who randomly surfs web pages or message distributed using a rumor-routing protocol.

In the context of an unmanned aerial vehicle (UAV) swarm, the PageRank vector can be used to determine where to inject a message in the swarm to ensure the message is quickly transmitted to each node by a rumor-routing protocol. For instance, the PageRank vector can identify which group members are likely to disseminate (mis)information most efficiently. It similarly can identify good roadblock locations to capture fleeing suspects. A more sedate application determines intersections to avoid during rush hour. In general, the PageRank algorithm is applicable to any scenario that requires assessing the probable behavior of an object traversing the nodes in some network.

Certain networks, however, yield a PageRank vector containing duplicate entries, which cannot induce a canonical vertex ordering. This occurs most often in networks that are not randomly constructed, i.e., networks having some pattern of regularity among its links between nodes. The issue can be resolved in some applications, e.g., search engines, by sorting on additional keys, such as web page addresses. A second method, described in Section 1.2, is to apply an application that produces canonical isomorphs, e.g., *nauty*, and order the canonical isomorph's nodes using its PageRank vector [McK81, McK04].

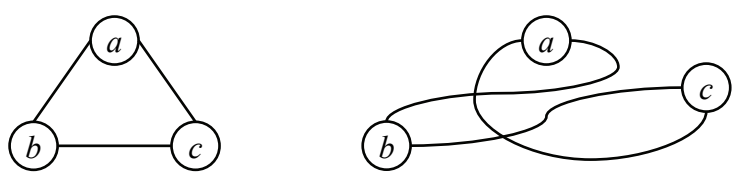
Networks containing nodes that yield equal PageRank values define an interesting duality. First, such networks motivate using application, such as *nauty*, to find a canonical PageRank vector. Second, if a non-canonical PageRank vector suffices, which is often the case, such graphs also simultaneously suggest methods for decreasing the time needed to obtain the PageRank vector. These latter methods also eliminate some numerical errors in the PageRank vector. The three algorithms constructed in Chapters 4 and 5 leverage these ideas to improve the PageRank algorithm's performance if a network contains nodes that must have equal PageRank values.

2.2. Deciding Isomorphism: A Classic Graph-Theoretic Problem

2.2.1. The Relationship to Canonical Vertex Ordering

The results described herein are obtained by applying tools used in algorithms that decide graph isomorphism by finding a canonical vertex order, which induces a canonical isomorph. Deciding graph isomorphism involves deciding if the edge sets, E_1 and E_2, of two graphs, G_1 and G_2, define equivalent links on their respective vertex sets, V_1 and V_2. For example, Figure 6 depicts two isomorphs of the triangle graph. Since the triangle is a *complete graph*, where each pair of vertices is connected, every off-diagonal entry in the corresponding adjacency matrices equals '1', as shown in Table 3.

A naive method of deciding graph isomorphism is to compare each permutation of G_1 with G_2. For example, a triangle has $n! = 3!$ permutations, where $n = |V|$, and the six vertex permutations are $\Phi = \{[abc],[acb],[bac],[bca],[cab],[cba]\}$. All permutations of G_1 equal G_2, since triangles have one unique isomorph. The number of permutations, $|\Phi|$, grows exponentially in proportion to $|V|$, thus, this approach is generally intractable.



(a) G_1, The Triangle (b) G_2, Twisted Rope

Figure 6. Two Graph Isomorphs: The Triangle and Twisted Rope

Table 3. Two Adjacency Matrix Isomorphs: The Triangle and Twisted Rope

(a) $\mathbf{A}_1$, The Triangle

	a	b	c
a	0	1	1
b	1	0	1
c	1	1	0

(b) $\mathbf{A}_2$, Twisted Rope

	a	b	c
a	0	1	1
b	1	0	1
c	1	1	0

2.2.2. Graph Isomorphism Applications

Exposure to the graph isomorphism problem, denoted GI, may infect researchers with the isomorphism "disease" [ReC77, Gat79]. The problem is important, since it is not known if GI is decidable in deterministic polynomial time on arbitrary graphs. However, GI is decidable in polynomial time for some graphs, e.g., trees [KrS98]. If GI defines a new complexity class, it would imply deterministic polynomial time problems, denoted P, are a proper subset of non-deterministic polynomial time problems, denoted NP. Hence, if GI defines a new complexity class, it also implies $P \neq NP$, a significant result currently worth $1,000,000 to its discoverer(s) [CMI00].

A classic application of algorithms that decide GI is identifying chemical isomers, or compounds sharing the same formula but having different atomic structures [Fau98]. For example, two isomers of $C_2H_2F_2$ are shown in Figure 7, which contain two atoms of carbon (C), fluorine (F), and hydrogen (H), and the edges denote chemical bonds [NIS]. Isomers are often stored in some canonical representation to simplify their comparison.

Another application finds a subgraph within some larger graph, e.g., finding small circuits in large circuits [OEG+93]. Algorithms capable of deciding isomorphism can be used in optical character recognition (OCR) [WaG04], to compare files [Car03, BeC06], or to analyze social networks patterns, such as enemy communication routes [GCM06]. A novel application involves guiding a UAV to replace sensor network nodes [CHP+04].

(a) 1,1-Difluoroethene (b) 1,2-Difluoroethene

Figure 7. Two Chemical Isomers: $C_2H_2F_2$

2.2.3. A Formal Definition

A pair of graphs, G_1 and G_2, are isomorphs if and only if a permutation, ϕ, exists satisfying (1), where for each edge in E_1, an equivalent edge exists in E_2. For example, applying the permutation, $\phi = [2,1,3,4]$, to the square's vertices illustrated in Figure 8(a) confirms the square is an isomorph of the hourglass shown in Figure 8(b). In other words, each graph contains four vertices, where each vertex is adjacent with two other vertices, such that the edges define a cycle graph of length four, denoted C_4.

$$G_1 \cong G_2 \quad \leftrightarrow \quad \begin{array}{c} \exists \phi(V_1) = V_2 \text{ s.t.} \\ \forall e_i = \{v_r, v_s\} \in E_1 \ \wedge \ v_r \in V_1 \ \wedge \ v_s \in V_1, \\ \exists e_j = \{\phi(v_r), \phi(v_s)\} \in E_2 \ \wedge \ \phi(v_r) \in V_2 \ \wedge \ \phi(v_s) \in V_2 \end{array} \tag{1}$$

(a) G_1, The Square (b) G_2, The Hourglass

Figure 8. Two Isomorphs: The Square and Hourglass

Equivalently, two adjacency matrices, $\mathbf{A}_1$ and $\mathbf{A}_2$, are isomorphs if and only if a permutation matrix, **P**, exists satisfying (2), where $\mathbf{P}^T$ denotes the matrix transpose, and '$\cdot$' denotes matrix multiplication. A permutation matrix, **P**, is a row permutation, ϕ, of the identity matrix, **I**, a square matrix whose diagonal entries equal one that otherwise equals zero. Thus, the row permutation orders **I**'s rows based on the given permutation vector, ϕ. The transpose of the permutation matrix, **P**, denoted $\mathbf{P}^T$, is equivalent to applying ϕ as a column permutation to **I**, i.e., ordering **I**'s columns with respect to ϕ.

$$\mathbf{A}_1 \cong \mathbf{A}_2 \quad \leftrightarrow \quad \exists \mathbf{P} \text{ s.t. } \mathbf{A}_2 = \mathbf{P} \cdot \mathbf{A}_1 \cdot \mathbf{P}^T \tag{2}$$

For example, the adjacency matrices of the square and hourglass graphs depicted in Figure 8 are listed in Table 4, where $\mathbf{A}_1 \neq \mathbf{A}_2$. The identity matrix, denoted $\mathbf{I}^4$, is listed in Table 5(a). The permutation matrix, $\mathbf{P}$, and its associated transpose, $\mathbf{P}^T$, listed in Tables 5(b) and (c), are obtained by permuting rows and columns of $\mathbf{I}^4$ with respect to the permutation, $\phi = [2,1,3,4]$. Multiplying $\mathbf{A}_1$ by $\mathbf{P}$ yields the matrix listed in Table 6(a) and multiplying the result by $\mathbf{P}^T$ yields the matrix listed in Table 6(b). Since the matrices shown in Tables 4(b) and 6(b) are equal, i.e., since $\mathbf{A}_2 = \mathbf{P} \cdot \mathbf{A}_1 \cdot \mathbf{P}^{\mathbf{T}}$, $\mathbf{A}_1 \cong \mathbf{A}_2$.

Table 4. Two Isomorphs: The Adjacency Matrices of the Square and Hourglass

(a) $\mathbf{A}_1$, The Square

	a	*b*	*c*	*d*
a	0	1	0	1
b	1	0	1	0
c	0	1	0	1
d	1	0	1	0

(b) $\mathbf{A}_2$, The Hourglass

	a	*b*	*c*	*d*
a	0	1	1	0
b	1	0	0	1
c	1	0	0	1
d	0	1	1	0

Table 5. Identity Matrix and Permutation Matrices for $\phi = [2,1,3,4]$

(a) $\mathbf{I}^4$, Identity Matrix

	1	2	3	4
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

(b) Rows, $\mathbf{P} = \mathbf{I}^4_{[2,1,3,4],:}$

	a	*b*	*c*	*d*
b	0	1	0	0
a	1	0	0	0
c	0	0	1	0
d	0	0	0	1

(c) Columns, $\mathbf{P}^T = \mathbf{I}^4_{:,[2,1,3,4]}$

	b	*a*	*c*	*d*
a	0	1	0	0
b	1	0	0	0
c	0	0	1	0
d	0	0	0	1

Table 6. Establishing $\mathbf{A}_2 = \mathbf{P} \cdot \mathbf{A}_1 \cdot \mathbf{P}^T$ and $\mathbf{A}_1 \cong \mathbf{A}_2$

(a) $\mathbf{P} \cdot \mathbf{A}_1$

	a	*b*	*c*	*d*
b	1	0	1	0
a	0	1	0	1
c	0	1	0	1
d	1	0	1	0

(b) $\mathbf{A}_2 = \mathbf{P} \cdot \mathbf{A}_1 \cdot \mathbf{P}^{\mathbf{T}}$

	b	*a*	*c*	*d*
b	0	1	1	0
a	1	0	0	1
c	1	0	0	1
d	0	1	1	0

A permutation, ϕ, has a unique inverse, denoted ϕ^{-1}, where $\phi_i^{-1} = j$ if and only if $\phi_j = i$, i.e., ϕ_i^{-1}, equals the position, j, of element i in ϕ. For example, for $\phi = [3,4,2,1,5]$, '1' is in position four, thus, $\phi_1^{-1} = 4$ and $\phi^{-1} = [4,3,1,2,5]$. The composition of ϕ and ϕ^{-1} is commutative and yields the identity permutation, hence, $\phi(\phi^{-1}) = \phi^{-1}(\phi) = [1,2,\ldots,n]$. One method of obtaining ϕ^{-1} is to augment ϕ with the identity permutation, $[1,2,\ldots,n]$, and sort entries in ϕ, yielding ϕ, as shown in Table 7, in $\Theta(n \cdot \log n)$ time [Knu97].

Table 7. Computing the Inverse Permutation

(a) $\mathbf{M} = [\phi, \phi_{\mathbf{i}}]$

2	1	5	3	4
1	2	3	4	5

(b) $\mathbf{T} = \text{lex_sort_columns}(\mathbf{M})$

1	2	3	4	5
2	1	4	5	3

(c) $\phi^{-1} = \mathbf{T}_{:,2}$

2	1	4	5	3

Similarly, $\mathbf{P}_r$ and $\mathbf{P}_c$ are pair-wise inverses: finding $\mathbf{P}_r^T$ and $\mathbf{P}_c^T$ is equivalent to swapping ϕ's entries and positions to obtain ϕ^{-1}. Thus, permuting the rows of $\mathbf{I}^n$ by ϕ is equivalent to permuting columns by ϕ^{-1}, and vice versa. Thus, $\mathbf{I}_{\phi,:} = \mathbf{I}_{:,\phi^{-1}}$ and $\mathbf{I}_{:,\phi} = \mathbf{I}_{\phi^{-1},:}$. For example, in Table 8, $\mathbf{P}_r = \mathbf{I}^5_{[5,2,1,3,4],:} = \mathbf{I}^5_{:,[3,2,4,5,1]} = \mathbf{P}_c^T$. Thus, ϕ^{-1} can be determined by enumerating the column or row heading of each '1' contained in $\mathbf{P}_r$ or $\mathbf{P}_c$. Finally, since $\mathbf{P}^{-1} = \mathbf{P}^T$, if $\mathbf{A}_1 \cong \mathbf{A}_2$, then $\mathbf{A}_1 = \mathbf{P} \cdot \mathbf{A}_2 \cdot \mathbf{P}^T = \mathbf{P}^T \cdot \mathbf{A}_1 \cdot \mathbf{P} = \mathbf{A}_2$.

Table 8. Computing Inverse Permutation Matrices

(a) $\mathbf{P}_r = \mathbf{I}^5_{[5,2,1,3,4],:} = \mathbf{I}^5_{:,[3,2,4,5,1]} = \mathbf{P}_c^T$

	1	2	3	4	5
5	0	0	0	0	1
2	0	1	0	0	0
1	1	0	0	0	0
3	0	0	1	0	0
4	0	0	0	1	0

(b) $\mathbf{P}_c = \mathbf{I}^5_{:,[5,2,1,3,4]} = \mathbf{I}^5_{[3,2,4,5,1],:} = \mathbf{P}_r^T$

	5	2	1	3	4
1	0	0	1	0	0
2	0	1	0	0	0
3	0	0	0	1	0
4	0	0	0	0	1
5	1	0	0	0	0

2.2.4. Canonical Isomorphs

A useful method of deciding graph isomorphism involves computing a canonical isomorph of each graph, since isomorphs yield identical canonical isomorphs [McK81]. For example, one method of assessing if the words, *logarithm* and *algorithm*, are relative permutations is to search for letters of *logarithm* in *algorithm* by searching for an '*l*' in *algorithm*, then an '*o*', a '*g*', and so forth, in $\Theta\left(n^2\right)$ time. A faster method compares the sorted letters in each word in $\Theta\left(n \cdot \log n\right)$ time. In fact, *logarithm* and *algorithm* yield the same canonical isomorph, *aghilmort*.

Similarly, concatenating entries by column in the upper-right triangle of a graph's adjacency matrix, **A**, yields a binary value, denoted $\text{num}\left(\mathbf{A}\right)$ [KrS98]. For example, the adjacency matrices listed in Tables 9(a) and (b) are from distinct house graph isomorphs. The shaded upper-right triangle in each matrix necessarily also must yield distinct values, $\text{num}\left(\mathbf{A}_1\right) = 1010011101_2 = 669_{10}$ and $\text{num}\left(\mathbf{A}_2\right) = 1100110101_2 = 821_{10}$, respectively. The *minimum canonical isomorph* (MCI), denoted $\mathbf{A}_\omega$, is the isomorph yielding the smallest binary number with respect to all isomorphs. The house graph's MCI, given its 60 unique isomorphs, is listed in Table 9(c), where $\text{num}\left(\mathbf{A}_\omega\right) = 0011011101_2 = 221_{10}$.

Table 9. Three Isomorphs of the House Graph's Adjacency Matrix

(a) $\mathbf{A}_1$

	a	*b*	*c*	*d*	*e*
a	0	1	0	0	1
b	1	0	1	0	1
c	0	1	0	1	0
d	0	0	1	0	1
e	1	1	0	1	0

(b) $\mathbf{A}_2$

	a	*b*	*c*	*d*	*e*
a	0	1	1	0	0
b	1	0	0	1	1
c	1	0	0	1	0
d	0	1	1	0	1
e	0	1	0	1	0

(c) $\mathbf{A}_\omega$, MCI

	a	*b*	*c*	*d*	*e*
a	0	0	0	1	1
b	0	0	1	0	1
c	0	1	0	1	0
d	1	0	1	0	1
e	1	1	0	1	0

The canonical isomorph, $\mathbf{A}_\omega$, of two graphs can be obtained separately, and then compared in $\Theta\left(n^2\right)$ time, e.g., in a chemical isomer database. If the graphs are large, a hash value of the canonical isomorphs, e.g., MD5, can be used to determine if two graphs could be isomorphs [GHK+03]. Synonyms for the graph's canonical isomorph include its *certificate*, *lexicographic leader*, or *signature* [Rea72, BaL83].

Formally, two isomorphs, $\left(\mathbf{A}_1\right)_\omega$ and $\left(\mathbf{A}_2\right)_\omega$, and their respective permutations, $\left(\mathbf{P}_1\right)_\omega$ and $\left(\mathbf{P}_2\right)_\omega$, to some canonical isomorph, $\mathbf{A}_\omega$, share the relationships illustrated by the *permutation triangle* shown in Figure 9. The solid lines denote explicit relationships, $\left(\mathbf{A}_1\right)_\omega \cong \mathbf{A}_\omega$ and $\left(\mathbf{A}_2\right)_\omega \cong \mathbf{A}_\omega$, yielded by $\left(\mathbf{P}_1\right)_\omega$ and $\left(\mathbf{P}_2\right)_\omega$, respectively. The dashed line denotes the implicit relationship, $\left(\mathbf{A}_1\right)_\omega \cong \left(\mathbf{A}_2\right)_\omega$, yielded by $\left(\mathbf{A}_1\right)_\omega \cong \mathbf{A}_\omega \cong \left(\mathbf{A}_2\right)_\omega$.

There are other canonical isomorphs exist, like the maximum canonical isomorph, but the canonical isomorph of choice is the MCI, if only for standardization [KrS98]. The MCI has many other useful properties. For example, the graph's MCI also identifies the graph's maximum independent set (MIS), or equivalently, the maximum clique (MC) of the graph's complement [BaL83, KrS98].

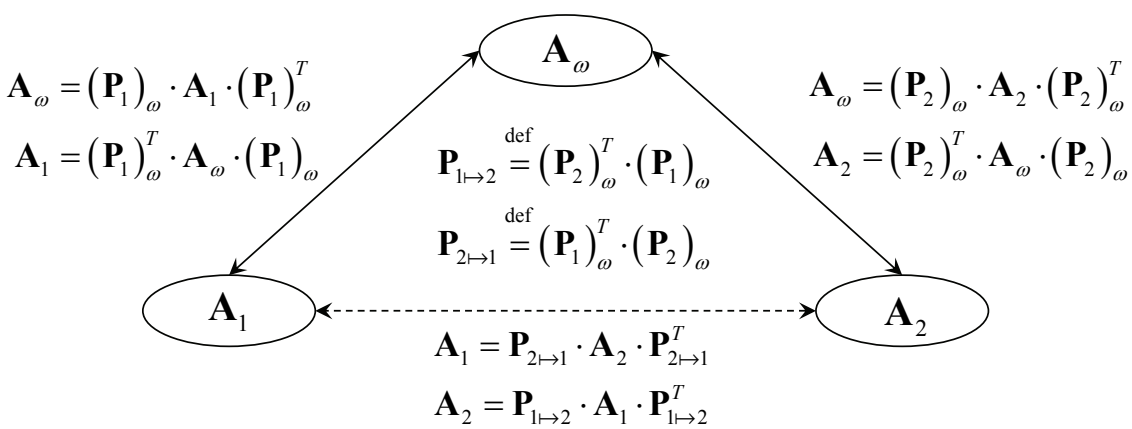


Figure 9. Canonical Isomorph's Permutation Triangle

2.3. Vertex Partitions

A common method for finding a graph's canonical isomorph is based on pruning a search tree using the process described in Section 2.3.4. However, it is first necessary to define tools used in that search process, namely, certain specific disjoint vertex partitions.

Vertex partitions can be constructed in many ways, where such partitions are used to prune the solution space of various problems, e.g., determining a canonical isomorph. For instance, a readily obtained vertex partition is the degree partition, in which vertices are grouped based on their number of immediate neighbors. For some graphs, the degree partition is an equitable partition, where vertices contained in each block have the same number of neighbors with respect to each of the partition's blocks. Although the degree partition is often not equitable, it is often used as the seed partition to initialize the search for more refined vertex partitions, such as the coarsest equitable partition.

If vertices are initially placed in the same block, the coarsest equitable partition is the most refined partition that can be obtained if the neighbors of every vertex are known. The coarsest equitable partition of some graphs corresponds to their orbit partition, which can be obtained using a pruning process similar to that used to find canonical isomorphs. However, the coarsest equitable partition can be computed in deterministic polynomial time whereas finding the orbit partition may require exponential time.

Significant attention is devoted to how a coarsest equitable partition is obtained to facilitate relating that partition to the dot product and PageRank vector. The relationships are established in Chapter 3 by relating the process of multiplying entries in a matrix row to sorting a row's entries. The coarsest equitable partition and its induced quotient graph are leveraged in Chapters 4 and 5 to improve the PageRank algorithm's performance.

Given an arbitrary graph, $G=(V,E)$, a partition, B, is a set of blocks containing vertices in V such that the union of the blocks equals V. Thus, if $V=\{v_1,v_2,\ldots,v_n\}$, then $B=\{b_1,b_2,\ldots,b_k\}$, where $b_i \subseteq V$ and $\bigcup_{i=1}^{m} b_i = V$. A partition is *proper* if block pairs are disjoint, i.e., $b_i \cap b_j = \varnothing, i \neq j$, where a disjoint union is denoted $V = b_1 \dot{\cup} b_2 \dot{\cup} \cdots \dot{\cup} b_{k=|B|}$.

A *unit partition* contains one that equals V, where $B=\{V\}$. Given a partition, B_i, a *refinement* is an operation yielding some partition, B_{i+1}, where each block contained in a partition, B_{i+1} is a subset of some block in B_i. For example, if $B_i=\{\{a,b\},\{c,d\}\}$, then $B_{i+1}=\{\{a,b\},\{c\},\{d\}\}$ is a refinement of B_i. However, $B_{i+1}=\{\{a,b,c\},\{d\}\}$ cannot be a refinement of B_i, since block $\{a,b,c\}$ is not a subset of block $\{a,b\}$ or block $\{c,d\}$.

An *ordered partition*, denoted $B=[b_1,b_2,\ldots,b_k]$, induces a block order such that the relative order of each block's vertices is maintained in any future partition refinement. For example, if $B_i=[\{a,b\},\{c,d\}]$, $B_{i+1}=[\{a,b\},\{d\},\{c\}]$ maintains the relative block order with respect to block B_i. However, $B_{i+1}=[\{d\},\{c\},\{a,b\}]$ does not maintain the block order with respect to block B_i, since block $\{a,b\}$ is located before block $\{c,d\}$ in block B_i. Each partition is assumed to be proper, i.e., disjoint, and ordered.

A *discrete partition* is maximally refined, where every block contains one vertex, e.g., $B_i=[\{d\},\{c\},\{a\},\{b\}]$. A discrete partition induces a vertex permutation that also induces isomorph of the input graph. Finding canonical discrete partitions is a key goal in many applications that produce canonical graph isomorphs, e.g., *nauty* [McK81].

The maximal set of ordered vertex partitions yields a partition tree. The root node is the coarsest possible partition, the unit partition, where each descendant is a refinement of its ancestor partition. Partitions along a path in this search tree retain vertices in their block-relative order. Every leaf node is an ordered discrete partition, where the number of leaf partitions corresponds to the number of vertex permutations, $|V|!$.

For example, the unit partition of a 3-vertex graph is $[\{a,b,c\}]$, the root node of the corresponding partition tree shown in Figure 10. The root node's descendants are the smallest possible refinements of the graph's unit partition, where the blocks are ordered by the number of vertices contained in each block. The leaves are the discrete partitions corresponding to the six possible vertex permutations of a 3-vertex graph. For instance, $[\{a\},\{b\},\{c\}]$ and $[\{c\},\{b\},\{a\}]$ correspond to the permutations, $[1,2,3]$ and $[3,2,1]$, respectively. The performance yielded by *nauty*, and most methods, that find a canonical isomorph, is obtained by dramatically pruning the number of nodes in the vertex partition tree using information yielded by the graph's edges.

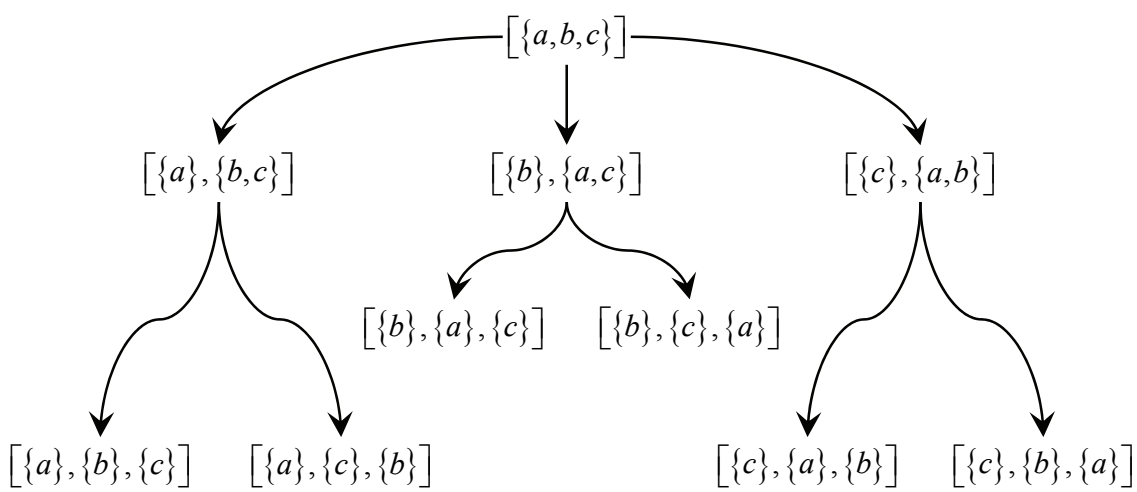


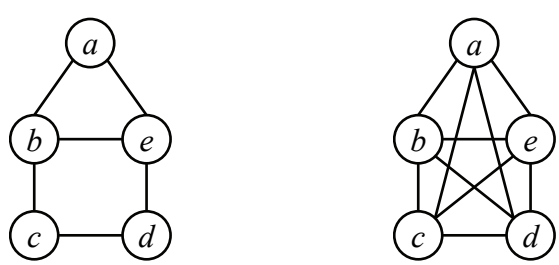
Figure 10. Vertex Partition Tree of an Arbitrary 3-Vertex Graph

2.3.1. The Degree Partition

There are many methods exist for pruning an arbitrary graph's partition tree. One method is to group vertices by their degrees, or number of neighbors, denoted $\deg(v_r)$, where $v_r \in V$, $e_i = \{v_r, v_s\} \in E$, and $\deg(v_r) = |v_r \cap e_i|, i \le |E|$. The degree sequence is an example of a graph invariant, i.e., isomorphs have the same degree sequence. Similarly, isomorphs contain an equal number of vertices and edges, i.e., $|V_1| = |V_2|$ and $|E_1| = |E_2|$.

Thus, one method of checking if two graphs may be isomorphs is to compare the number of vertices and edges each graph contains and their degree sequences. Formally, an *invariant* is a property, ψ, isomorphs must share, for instance, $|V_1| = |V_2|$ and $|E_1| = |E_2|$ if $G_1 \cong G_2$. Conversely, a complete, or sufficient, invariant is an invariant that completely decides graph isomorphism, such as a canonical isomorph.

The degree sequence is often used to determine two graphs cannot be isomorphs. For example, the house graph and the complete graph, K_5, shown in Figure 11 cannot be isomorphs, since they yield two different degree sequences, $[2,2,2,3,3]$ and $[4,4,4,4,4]$, respectively. The house graph and K_5 similarly have distinct degree partitions, which are $[\{a,c,d\},\{b,e\}]$ and $[\{a,b,c,d,e\}]$, respectively.

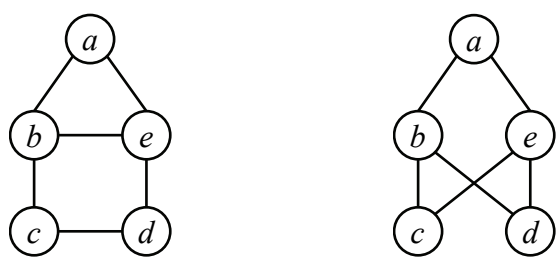


(a) G_1, House Graph (b) $G_2 = K_5$

Figure 11. Two Graphs Yielding Different Degree Sequences

However, certain non-isomorphic graphs yield identical sorted degree sequences. For example, the house graph and the complete bipartite graph, $K_{2,3}$, shown in Figure 12 yield the same sorted degree sequence, $[2,2,2,3,3]$, which yield the degree partitions, $[\{a,c,d\},\{b,e\}]$ and $[\{c,d,e\},\{a,b\}]$, respectively. However, simple visual inspection reveals these two graphs are not isomorphs. Two non-isomorphic graphs having the same value with respect to an arbitrary invariant are said to be a *devil's pair* [Ros00]. Thus, the house graph and $K_{2,3}$ are a devil's pair with respect to the sorted degree sequence.

Degree sequences have a more elementary shortcoming. Ideally, a graph invariant uniquely labels each vertex and induces a canonical vertex ordering. However, if $|V| \geq 2$, it is impossible to uniquely label the vertices using the degree sequence, since at least two vertices have equal degrees. First, a connected graph on $n \geq 2$ vertices has a degree range of $[1, n-1]$. By the pigeonhole principle, at least two vertices must have the same degree, since the range only contains $n-1$ distinct values, whereas the graph contains n vertices. Therefore, at least one block in the degree partition contains two or more vertices, which precludes obtaining a discrete partition that induces a canonical vertex ordering. A similar approach generalizes this proof to disconnected graphs.



(a) G_1, House Graph (b) $G_2 = K_{2,3}$

Figure 12. Devil's Pair for the Sorted Degree Sequence

2.3.2. The Equitable Partitions

An *equitable partition* is an extension of the degree partition, since every vertex contained in the same block of an equitable partition yields the same degree. However, an equitable partition adds the criterion that vertices in the same block have an equal number of neighbors with respect to any block in the partition, including their own block. Thus, an equitable partition extends the concept of vertex degrees to block degrees, where given an equitable partition, *B*, and an arbitrary pair of blocks, $b_i, b_j \in S$, each vertex contained in b_i has an equal number of neighbors contained in b_j, and vice versa.

Given an arbitrary graph, $G = (V, E)$, a partition, *B*, is determined to be equitable by applying the vertex neighborhood, denoted $N(v)$, where $v \in V$ and $|N(v)| = \deg(v)$. The *neighborhood* of an arbitrary vertex, *v*, is its set of adjacent vertices, where [KrS98]

$$N(v) = \{u \in V : \{u, v\} \in E\}. \tag{3}$$

Therefore, an arbitrary partition, $B = [b_1, b_2, \ldots, b_k], 1 \le k \le n$, where $n = |V|$, is equitable with respect to *G*, if for all *i* and *j*, $1 \le i, j \le k$, and for all $u, v \in b_i$,

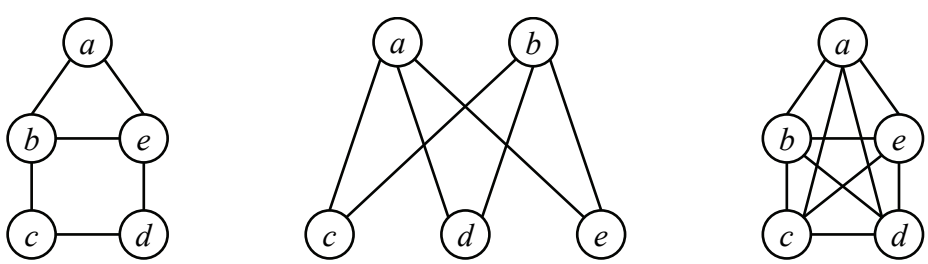
$$|N(u) \cap b_j| = |N(v) \cap b_j|. \tag{4}$$

Given two vertices, *u* and *v*, contained in an arbitrary block, b_i, *u* and *v* have the same number of neighbors in any block, b_j, including the case, $i = j$. However, vertices *u* and *v* do not have to share the same neighbors in each block, only the same *number* of neighbors. The number of neighbors may differ across blocks, i.e., it is *not* required that $|N(u) \cap b_j| = |N(u) \cap b_k|, j \ne k$ or $|N(v) \cap b_j| = |N(v) \cap b_k|, j \ne k$. Thus, it is acceptable if $|N(u) \cap b_j| \ne |N(u) \cap b_k|$ and $|N(v) \cap b_j| \ne |N(v) \cap b_k|$.

For example, the house graph shown in Figure 13(a) yields the ordered partition, $\left[\{a,c,d\},\{b,e\}\right]$. This partition is not equitable, since vertices *c* and *d* are each adjacent to one vertex in block $\{b,e\}$, whereas vertex *a* is a neighbor of both vertices contained in block $\{b,e\}$. A second reason this partition is not equitable is that block $\{a,c,d\}$ contains two adjacent vertices, *c* and *d*, whereas vertex *a* is not adjacent to any vertex contained in block $\{a,c,d\}$. Thus, the vertices contained in the block, $\{a,c,d\}$, do not share an equal number of neighbors with respect to any of the blocks contained in this partition.

Conversely, the graph, $K_{2,3}$, depicted in Figure 13(b) yields the degree partition, $\left[\{c,d,e\},\{a,b\}\right]$. This partition is equitable, since the vertices in each block are adjacent to all vertices in the other block, but none of the vertices in their own block. The graph, K_5, shown in Figure 13(c) similarly yields the equitable degree partition, $\left[\{a,b,c,d,e\}\right]$.

The graph, K_5, is 4-regular, where vertices in a *k*-regular graph have *k* neighbors. Thus, the degree partition of an arbitrary *k*-regular graph is also equitable. Moreover, the degree partition of an arbitrary *k*-regular graph equals its unit partition. However, the unit and degree partition of most graphs are not equitable, e.g., the house graph.



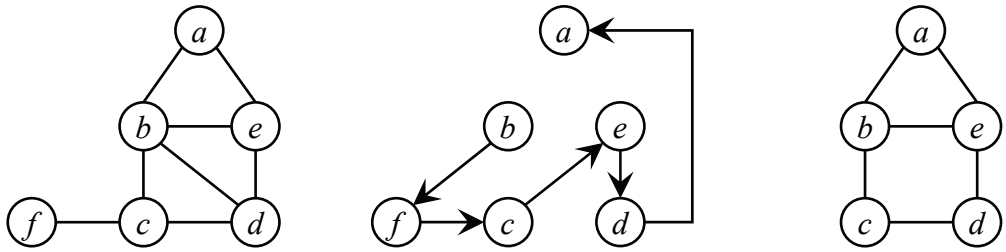
(a) G_1, House Graph (b) $G_2 = K_{2,3}$ (c) $G_3 = K_5$

Figure 13. Exploring Equitable Partitions

2.3.3. The Coarsest Equitable Partition

Fortunately, given an arbitrary graph and an arbitrary initial partition, a specific equitable partition is always guaranteed to exist. The *coarsest equitable partition* is the most refined partition that can be obtained using only the information that is derived from the neighbors of each vertex. The coarsest equitable partition is a key tool used to prune the partition tree while finding a canonical isomorph. For instance, on nearly all random graphs, the coarsest equitable partition is a discrete partition. Thus, the coarsest equitable partition of most random graphs prunes the partition tree to a single leaf, which induces a canonical vertex ordering. The results described in Chapters 3–5 establish the PageRank vector can be obtained more efficiently if the coarsest equitable partition is non-discrete, i.e., contains blocks composed of multiple vertices.

For example, the mansion graph shown in Figure 14(a) yields the discrete coarsest equitable partition, $\left[\{b\},\{f\},\{c\},\{e\},\{d\},\{a\}\right]$, and induces the canonical vertex order illustrated in Figure 14(b). Conversely, the house graph shown in Figure 14(c) yields the coarsest equitable partition, $\left[\{c,d\},\{a\},\{b,e\}\right]$, which cannot induce a canonical order. The results described in Chapters 4 and 5 improve the PageRank algorithm's performance on graphs whose coarsest equitable partition is non-discrete, such as the house graph.



(a) G_1, Mansion Graph (b) Canonical Ordering of G_1 (c) G_2, House Graph

Figure 14. Coarsest Equitable Partitions and Canonical Orderings

2.3.3.1. A Formal Method

The method described in this section for finding the coarsest equitable partition is derived from the definition of an equitable partition (4) and restated here for convenience. Given some partition, $B = [b_1, b_2, \ldots, b_k], 1 \le k \le n,$ a partition is *equitable* with respect to an arbitrary graph, $G = (V, E),$ if for all *i* and *j*, $1 \le i, j \le k,$ and for all $u, v \in b_i,$ [KrS98]

$$|N(u) \cap b_j| = |N(v) \cap b_j|, \tag{5}$$

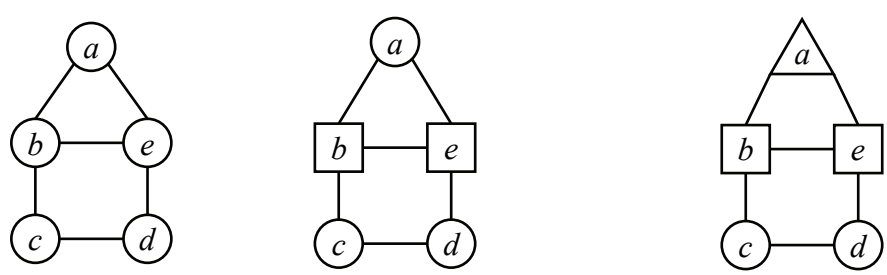
where $n = |V|$ and $N(u)$ denotes a set containing the neighbors of vertex *u*.

A simple method of satisfying this definition and computing the coarsest equitable partition is to proceed as follows. Given an arbitrary partition, such as the unit partition, select an arbitrary block in the partition containing two or more vertices. Then, select the first vertex in the block and identify the number of neighbors of that vertex relative to all blocks in the partition. Repeat this process for all vertices contained in the block. Vertices yielding the same number of neighbors with respect to every block, or identical sorted block degree sequences, are split into unique blocks. If the selected block cannot be split, repeat the process for all blocks containing multiple vertices until some block splits. A split block is ordered with respect to the vertex degree and block size in either ascending or descending order. If none of the blocks split after some iteration of this process, the vertex partition has stabilized to the coarsest equitable partition (5).

This naive partition refinement algorithm's execution time has an upper bound of $O(n^3),$ where $n = |V|$ [KaS83, PaT87]. If a graph is stored as a set of adjacency lists, the algorithm's upper bound is $O(m \cdot n),$ where $m = |E|.$ This algorithm's key contribution is as an easily described method of finding the graph's coarsest equitable partition.

For example, the unit partition yielded by the house graph shown in Figure 15(a) is $\left[\{a,b,c,d,e\}\right]$. Thus, the first block containing more than one vertex is the unit block, $\{a,b,c,d,e\}$. Three vertices, $\{a,c,d\}$, are adjacent to two vertices in block $\{a,b,c,d,e\}$, whereas two vertices, $\{b,e\}$, are adjacent to two vertices in block $\{a,b,c,d,e\}$. Thus, the refined partition yielded by this step, after ordering by the vertex degree and the number of elements contained in the resulting blocks, is $\left[\{a,c,d\},\{b,e\}\right]$. The degree partition is illustrated using unique shapes in the graph shown in Figure 15(b).

Another iteration must be performed to assess if the current partition is stable. The first block, $\{a,c,d\}$, contains multiple vertices where vertex *a* contains zero neighbors in block $\{a,c,d\}$ and two neighbors in block $\{b,e\}$. Conversely, vertices *c* and *d* each have one neighbor in block $\{a,c,d\}$, and one neighbor in block $\{b,e\}$. Thus, a new partition is obtained, $\left[\{a\},\{c,d\},\{b,e\}\right]$, as shown using unique shapes in Figure 15(c). Performing a third iteration confirms the partition is stable, i.e., no more block refinement will occur. Hence, the house graph's coarsest equitable partition is $\left[\{a\},\{c,d\},\{b,e\}\right]$.



(a) House Graph $\left[\{a,b,c,d,e\}\right]$ (b) Degree Partition $\left[\{a,c,d\},\{b,e\}\right]$ (c) Coarsest Equitable Partition $\left[\{a\},\{c,d\},\{b,e\}\right]$

Figure 15. Method 1: Finding the House Graph's Coarsest Equitable Partition

2.3.3.2. A Fast Method

More efficient methods exist for computing the coarsest equitable partition, e.g., the algorithm described by Cardon and Crochemore [CaC82]. Paige and Tarjan suggested a simpler algorithm [PaT87] described more extensively by Kreher and Stinson [KrS98]. The method's upper bound is $\mathrm{O}\left(n^2 \cdot \log n\right)$ and reduces to $\mathrm{O}\left(\left(m+n\right) \cdot \log n\right)$ if a graph is stored in sparse form, where $n = |V|$ and $m = |E|$ [BLS+06]. The key data structures are:

- the current vertex partition, *B*,
- a test vertex set, *U*,
- a test block set, *S*, and
- a block set, *Z*,

where *Z* is ordered on the number of vertices in a test block, $s_i \in S$, that are neighbors of vertices contained in blocks of the partition, *B*, where $s \subseteq U$. The partition, *B*, is said to be equitable after the potential test block set, *S*, has been emptied.

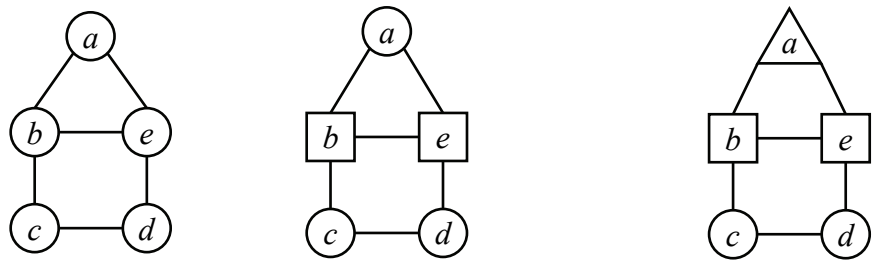
The algorithm listed in Figure 16 is a commented variant of Kreher and Stinson's method [KrS98]. The unit partitions for *B* and *S* are created on lines 2–4. The main loop is entered on line 5 and iterates until the test block set, *S*, is empty. A test block, $s \in S$, is selected on line 7 and its vertices are checked for validity on line 9 by assessing if $s \subseteq U$. If all vertices in *s* are valid, they are compared to each non-discrete block, $b_i \in B, |b_i| > 1$.

The number of vertices in *s* adjacent to the vertices in block b_i are stored in the block set, *Z*, as shown on lines 14–21. If *Z* contains two or more disjoint sub-blocks of block b_i, i.e., if $\left|\left\{z_{j=1,2,\ldots n-1} \in Z : z_j \neq \varnothing\right\}\right| \geq 2$, block b_i is subsequently replaced by those sub-blocks, $z_{j=1,2,\ldots n-1} \in Z : z_j \neq \varnothing$, on lines 24 and 25. The sets of potential test blocks, *S*, and valid test vertices, *U*, are updated on lines 7–10 and lines 26–29.

```
1.  findCoarsestPartition(G, n)
2.    # create initial unit partitions
3.    B ← {1,2,…,n}
4.    S ← [{1,2,…,n}]
5.    while S ≠ ∅
6.      # remove arbitrary unprocessed test block, s
7.      s ← S_1
8.      S ← S − s
9.      if s ⊆ U
10.       U ← U − s
11.       # initialize tracking sets for number of neighbors in block
12.       Z = [z_1, z_2, …, z_{n−1}], z_j = ∅
13.       # iterate over blocks in current partition
14.       foreach block, b_i ∈ B, |b_i| > 1
15.         # iterate over each vertex in current block, b_i ∈ B
16.         foreach vertex, v ∈ b_i, |b_i| > 1
17.           # compute number of neighbors of v in test block, s
18.           d = |getNeighborhood(v) ∩ s|
19.           # assign v to vertex set, z_d, with same number of neighbors
20.           z_d = z_d ∪ v
21.         end foreach
22.         # if block list, Z, contains multiple non-empty blocks, update
23.         if |{z_{j=1,2,…,n−1} ∈ Z : z_j ≠ ∅}| > 1
24.           # replace old block, b_i, with neighbor-degree ordered blocks in Z
25.           B = [b_1, b_2, …, b_{i−1},  z_1, z_2, …, z_{n−1},  b_{i+1}, b_{i+2}, …, b_{k=|B|}], z_j ≠ ∅
26.           # append blocks in neighbor lists, Z, to potential block list, S
27.           S = S ∪ Z
28.           # append vertices in neighbor lists, Z, to valid vertex list, U
29.           U = U ∪ z_{i=1,2,…,n−1} ∈ Z, z_j ≠ ∅
30.         end if
31.       end for
32.     end if
33.   end while
34.   return B
35. end findCoarsestPartition
```

Figure 16. Method 2: Fast Algorithm for Finding Equitable Partitions [PaT87, KrS98]

Applying the algorithm to the house graph illustrated in Figure 17 yields the data structures listed in Table 10, where the vertex set, U, is not listed, since each block placed in S is valid. The final refinement occurs when $b_i = \{b,e\}$ and $s = \{a,c,d\}$, which again shows the house graph's coarsest equitable partition is $[\{a\},\{c,d\},\{b,e\}]$.



(a) House Graph $[\{a,b,c,d,e\}]$ (b) Iteration 1 $[\{a,c,d\},\{b,e\}]$ (c) Coarsest Equitable Partition $[\{a\},\{c,d\},\{b,e\}]$

Figure 17. Method 2: Finding the House Graph's Coarsest Equitable Partition

Table 10. Method 2: Finding the House Graph's Coarsest Equitable Partition

i	b_i	B	s	S	Z
1	$\{a,b,c,d,e\}$	$[\{a,b,c,d,e\}]$	$\{a,b,c,d,e\}$	$[\{a,b,c,d,e\}]$	$\begin{bmatrix} 2=\{a,c,d\} \\ 3=\{b,e\} \end{bmatrix}$
2	$\{a,c,d\}$	$[\{a,c,d\},\{b,e\}]$	$\{a,c,d\}$	$[\{a,c,d\},\{b,e\}]$	$\begin{bmatrix} 0=\{a\} \\ 2=\{c,d\} \end{bmatrix}$
3	$\{b,e\}$	$[\{a\},\{c,d\},\{b,e\}]$		$[\{b,e\},\{a\},\{c,d\}]$	$[2=\{b,e\}]$
4	$\{c,d\}$	$[\{a\},\{c,d\},\{b,e\}]$	$\{b,e\}$	$[\{a\},\{c,d\}]$	$[1=\{c,d\}]$
5	$\{b,e\}$				$[0=\{b,e\}]$
6	$\{c,d\}$		$\{a\}$	$[\{c,d\}]$	$[0=\{c,d\}]$
7	$\{b,e\}$				$[2=\{b,e\}]$
8	$\{c,d\}$		$\{c,d\}$	[ ]	$[1=\{c,d\}]$
9	$\{b,e\}$				$[1=\{b,e\}]$

2.3.3.3. A Facile Method

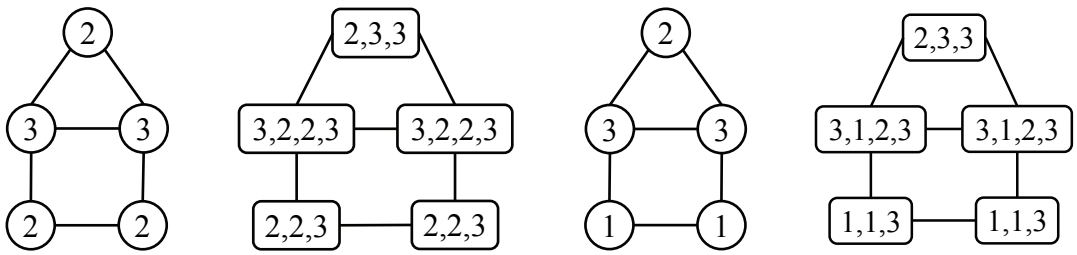
The third and final algorithm for computing a graph's coarsest equitable partition, 1-dimensional (1-D) Weisfeiler-Lehman stabilization, has been repeatedly discovered and is described most extensively by Weisfeiler and Lehman [Wei76, ReC77, CFI92, Bas02]. This stabilization method can also be used in many contexts, e.g., 1-D Weisfeiler-Lehman stabilization is used in Chapter 3 to establish vertices contained in the same block of the graph's coarsest equitable partition must have equal PageRank values.

All vertices are first labeled by its degree, as shown on line 3 in Figure 18. These labels are augmented with their sorted neighbor labels and replaced by shorter labels to conserve memory, as shown on lines 9–11. The labeling process iterates until the labels are unchanged with respect to consecutive iterations, i.e., if the old partition, *S*, equals the new partition, *B*, on line 7. The stabilized partition, *B*, is the coarsest equitable partition.

1. findCoarsestPartition(G, n)
2. # initialize labels to vertex degrees and partition
3. $Z_r \leftarrow \deg(v_r), v_r \in V$
4. $B \leftarrow [\{1, 2, \ldots, n\}]$
5. $S \leftarrow \varnothing$
6. # iterate until sorted vertex labels stabilize
7. while $S \neq B$
8. $S \leftarrow B$
9. $Z_r \leftarrow [Z_r, \text{sort}(\{Z_s : (v_r, v_s) \in E\})]$
10. $Z_r \leftarrow \text{getUniqueLabels}(Z_r)$
11. $B_i \leftarrow \{v_r : Z_r = i\}$
12. end while
13. return B
14. end findCoarsestPartition

Figure 18. Method 3: 1-D Weisfeiler-Lehman Stabilization

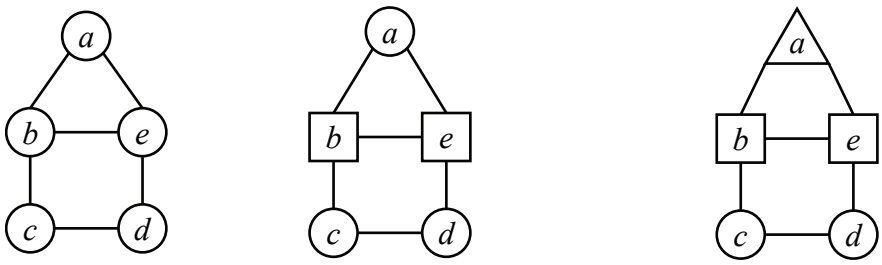
For example, the house graph yields the degree labels and neighbor labels shown in Figures 19(a) and (b), respectively. Shorter unique identifier labels are assigned to each vertex, as shown in Figure 19(c). In this example, no more partition refinement occurs if the vertex labels are augmented with the sorted neighbor labels, as shown in Figure 19(d). Repeating the process only yields replicates of the graphs shown in Figures 19(c) and (d). Thus, the house graph's coarsest equitable partition is $\left[\{c,d\},\{a\},\{b,e\}\right]$.



(a) Degree Labels (b) Adjacent Labels (c) Shorter Labels (d) Stable Labels

Figure 19. Method 3: 1-D Weisfeiler-Lehman Stabilization Example

Hence, 1-D Weisfeiler-Lehman stabilization refines the unit partition to the degree partition, as depicted in Figures 20(a) and (b), respectively. Sorting the adjacent labels of every vertex and appropriately assigning distinct labels until stabilization occurs produces the coarsest equitable partition, e.g., $\left[\{c,d\},\{a\},\{b,e\}\right]$, as illustrated in Figure 20(c).



(a) House Graph $\left[\{a,b,c,d,e\}\right]$ (b) Degree Partition $\left[\{a,c,d\},\{b,e\}\right]$ (c) Coarsest Equitable Partition $\left[\{c,d\},\{a\},\{b,e\}\right]$

Figure 20. Method 3: Finding the House Graph's Coarsest Equitable Partition

1-D Weisfeiler-Lehman stabilization can also be readily illustrated using matrices. For example, the adjacency matrix of the house graph is depicted in Table 11(a). Sorting the entries by row, from left to right, yields the matrix shown in Table 11(b). Assigning a shorter, but similarly unique, label to each of the sorted rows yields the labels shown in Table 12(a), where these labels correspond to the degree partition. Repeating the process using these shorter labels yields the matrices shown in Tables 12(b) and (c), respectively. Repeating the process yields the matrices listed in Tables 12(d)–(f) and future iterations also yield the same partition, i.e., the coarsest equitable partition, $\left[\{c,d\},\{a\},\{b,e\}\right]$.

Table 11. The House Graph's Adjacency and Sorted Degree Matrices

(a) **A**

	a	b	c	d	e
a	0	1	0	0	1
b	1	0	1	0	1
c	0	1	0	1	0
d	0	0	1	0	1
e	1	1	0	1	0

(b) $\text{sort}(\mathbf{A})$

	sorted left to right				
a	0	0	0	1	1
b	0	0	1	1	1
c	0	0	0	1	1
d	0	0	0	1	1
e	0	0	1	1	1

Table 12. Method 3: 1-D Weisfeiler-Lehman Stabilization of the House Graph

(a) Degree Labels

a	1
b	2
c	1
d	1
e	2

(b) Sorted Labels I

c	1	1	2	
d	1	1	2	
a	1	2	2	
b	2	1	1	2
e	2	1	1	2

(c) Shorter Labels I

c	1
d	1
a	2
b	3
e	3

(d) Unique Labels

a	2
b	3
c	1
d	1
e	3

(e) Sorted Labels II

c	1	1	3	
d	1	1	3	
a	2	3	3	
b	3	1	2	3
e	3	1	2	3

(f) Shorter Labels II

c	1
d	1
a	2
b	3
e	3

The coarsest equitable partition is unique up to a block permutation. For example, applying the methods described in Sections 2.3.3.1 or 2.3.3.2 to the house graph yields $\left[\{a\},\{c,d\},\{b,e\}\right]$. Blocks $\{a\}$ and $\{c,d\}$ exchange positions if 1-D Weisfeiler-Lehman stabilization is applied, yielding $\left[\{c,d\},\{a\},\{b,e\}\right]$. To avoid this issue, the same method should be used to compute the coarsest equitable partition throughout an application.

Each iteration of 1-D Weisfeiler-Lehman stabilization requires sorting *n* vectors of length *n* and finding shorter labels for each vector, where each step is $\mathrm{O}\left(n^2 \cdot \log n\right)$. Since up to $\log_2 n^2$ iterations are required to achieve stabilization, e.g., for path graphs [Bas02], the upper bound is $\mathrm{O}\left(n^2 \cdot \log n \cdot \log n^2\right)$. This bound reduces to $\mathrm{O}\left(d \cdot n \cdot \log n \cdot \log n^2\right)$ if a graph is stored using adjacency lists, where $d = \max\left(\deg\left(v_i\right)\right)$ [Bas02].

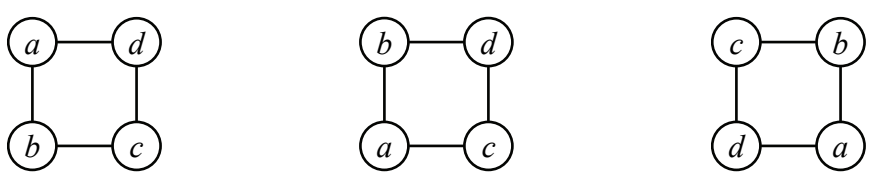
The method described in Section 2.3.3.2 for finding a coarsest equitable partition is more efficient, since its upper bound is merely $\mathrm{O}\left(n^2 \cdot \log n\right)$. Thus, Weisfeiler-Lehman stabilization is slower by a factor of $\left(2 \cdot n^2 \cdot \log n \cdot \log n^2\right) / \left(n^2 \cdot \log n\right) = 2 \cdot \log n^2 = 4 \cdot \log n$ with respect to the more efficient method. However, 1-D Weisfeiler-Lehman stabilization is simpler to describe and easier to implement correctly [PaT87].

Moreover, the similarity between performing 1-D Weisfeiler-Lehman stabilization and performing matrix multiplication yields the proof contained in Chapter 3 establishing a relationship between the coarsest equitable partition and dot product. Finally, the proof yields three progressively sophisticated methods of improving the PageRank algorithm's performance, as described in Chapters 4 and 5. These methods improve certain numerical properties of the PageRank vector and the latter two methods reduce execution time.

2.3.4. The Orbit Partition

The orbit partition is obtained by separating the graph's set of permutations into two disjoint classes: isomorphisms and automorphisms. An *isomorphism* is a permutation yielding a distinct isomorph, where one or more edges receive new labels. Conversely, an *automorphism* is a permutation yielding the original graph, i.e., edges retain their original label. If an isomorphism is applied, the associated adjacency matrices are different, but if an automorphism is applied, the adjacency matrices are identical.

For example, applying the permutation, $\phi_1 = [2,1,3,4]$, to the square illustrated in Figure 21(a) yields the isomorph shown in Figure 21(b), which is also unique, since some edges receive different labels, e.g., edges $\{a,d\}$ and $\{b,d\}$, respectively. Conversely, the automorphism illustrated in Figure 21(b) yields an isomorph with the same edge labels. The same effect is observed in Table 13, where the square's and the isomorph's adjacency matrices differ, but the square's and the automorph's adjacency matrices are identical.



(a) The Square, G (b) An Isomorph of G $\phi_1 = [2,1,3,4]$ (c) An Automorph of G $\phi_2 = [3,4,1,2]$

Figure 21. Isomorph and Automorph of the Square

Table 13. Isomorph and Automorph of the Square

(a) A

	a	b	c	d
a	0	1	0	1
b	1	0	1	0
c	0	1	0	1
d	1	0	1	0

(b) $\mathbf{A}_1 = \phi_{[2,1,3,4]}(\mathbf{A})$

	b	a	c	d
b	0	1	1	0
a	1	0	0	1
c	1	0	0	1
d	0	1	1	0

(c) $\mathbf{A}_2 = \phi_{[3,4,1,2]}(\mathbf{A})$

	c	d	a	b
c	0	1	0	1
d	1	0	1	0
a	0	1	0	1
b	1	0	1	0

All graphs yield the trivial automorphism, or identity permutation, $\phi = [1, 2, \ldots, n]$. Most automorphisms send some vertices to another, albeit logically equivalent, position. If an arbitrary vertex, *u*, is sent to another vertex, *v*, by some number of automorphisms, an equal number of automorphisms send *v* to *u*, where such vertices are in the same *orbit*. The *orbit partition* is the disjoint equitable vertex partition listing the graph's orbits.

The orbit partition equals or is more refined than the coarsest equitable partition. For example, the house graph's coarsest equitable and orbit partition are equal, as shown in Figure 22. Conversely, the coarsest equitable partition of the cuneane graph shown in Figure 23(a) [TiK99, StT99] contains a single block, $[\{a, b, \ldots, h\}]$, but its orbit partition contains three blocks, $[\{a, h\}, \{d, e\}, \{b, c, f, g\}]$, as shown in Figure 23(b).

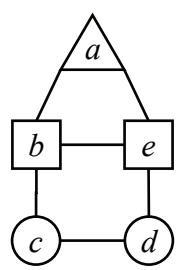
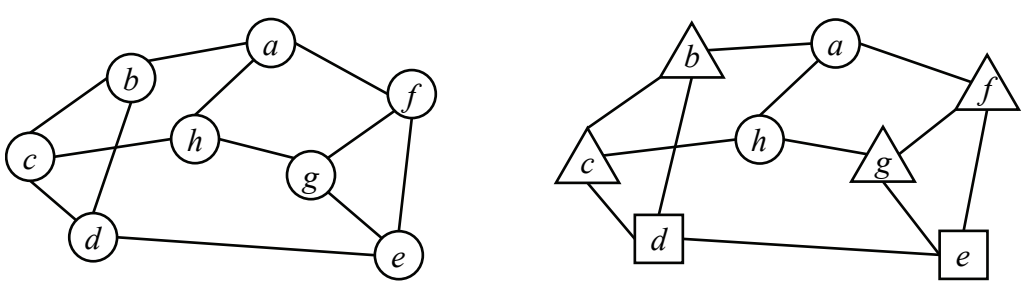


Figure 22. House Graph's Coarsest Equitable and Orbit Partition, $[\{c, d\}, \{a\}, \{b, e\}]$



(a) Coarsest Equitable Partition $[\{a, b, \ldots, h\}]$

(b) Orbit Partition $[\{a, h\}, \{d, e\}, \{b, c, f, g\}]$

Figure 23. Cuneane Graph's Distinct Coarsest Equitable and Orbit Partitions

2.3.4.1. Vertex Individualization and Arbitrary Partition Stabilization

Finding the orbit partition involves applying many techniques, especially if the coarsest equitable partition is not discrete, i.e., contains one or more blocks composed of multiple vertices. The *individualization* process involves splitting a non-discrete block and assigning a single vertex to its own block, where one or more blocks are successively split each possible way. Since the partition induced by applying individualization may not be equitable, the method used to compute the initial coarsest equitable partition is applied to the individualized partition, thus yielding a more refined and equitable partition.

For example, applying Weisfeiler-Lehman stabilization to the house graph yields the coarsest equitable partition, $\left[\{c,d\},\{a\},\{b,e\}\right]$, shown in Figure 24. Two blocks are not discrete, thus, one or more block's vertices are individualized. For example, splitting block $\{b,e\}$ by individualizing vertex *b* induces the partition shown in Figure 25(a). This partition is not equitable, since block $\{c,d\}$ contains a vertex, *c*, adjacent to one vertex in block $\{b\}$, namely, vertex *b*, whereas vertex *d* is not also adjacent to vertex *b*. Applying 1-D Weisfeiler-Lehman stabilization to the partition obtained by individualizing vertex *b* yields the discrete and equitable partition shown in Figure 25(b). The other three potential vertex individualizations are shown in Figures 25(c)–(h).

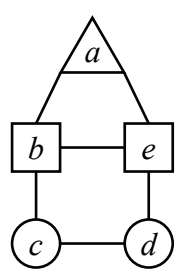
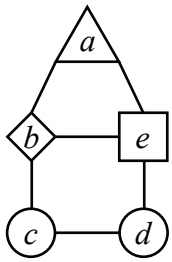
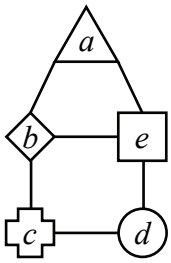


Figure 24. House Graph's Coarsest Equitable Partition, $\left[\{c,d\},\{a\},\{b,e\}\right]$



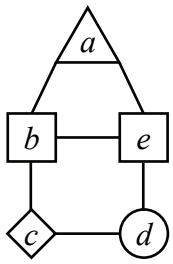
(a) Individualizing b

$\left[\{c,d\},\{a\},\{b\},\{e\}\right]$



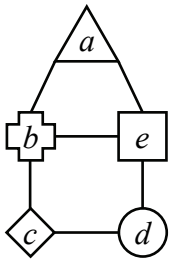
(b) Equitable Partition

$\left[\{c\},\{d\},\{a\},\{b\},\{e\}\right]$



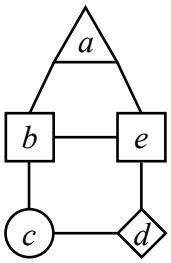
(c) Individualizing c

$\left[\{c\},\{d\},\{a\},\{b,e\}\right]$



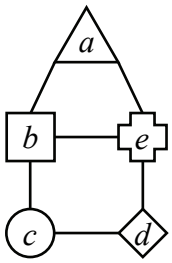
(d) Equitable Partition

$\left[\{c\},\{d\},\{a\},\{b\},\{e\}\right]$



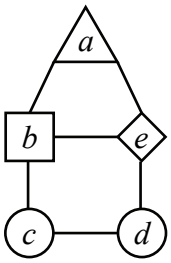
(e) Individualizing d

$\left[\{d\},\{c\},\{a\},\{b,e\}\right]$



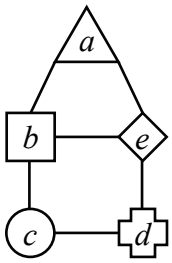
(f) Equitable Partition

$\left[\{c\},\{d\},\{a\},\{e\},\{b\}\right]$



(g) Individualizing e

$\left[\{c,d\},\{a\},\{e\},\{b\}\right]$



(h) Equitable Partition

$\left[\{d\},\{c\},\{a\},\{e\},\{b\}\right]$

Figure 25. Refining to Equitable Partitions after Vertex Individualization

2.3.4.2. Computing the Orbit Partition

Many methods have been described for determining graph isomorphism on some graphs [CoG70, Rea72, BES80, BaL83]. Many algorithms have also been developed that decide isomorphism for arbitrary graphs [CFS+04, BeE96] and many more are frequently proposed [KuS07, JuK07]. The traditional application used to decide graph isomorphism, *nauty* [McK04], is often used to provide a performance baseline to assess new algorithms that decide graph isomorphism. The key tools applied in *nauty* [McK81] are extensively described by Babai [Bab95], Kocay [Koc96], and Kreher and Stinson [KrS98]. In *nauty*, the key goal is to find a canonical isomorph, and in so doing, find the orbit partition.

Applications that are similar to *nauty*, such as *nice* [Mil07], perform a depth-first search of a graph's vertex partition tree to identify a canonical isomorph. However, many methods are used to dramatically prune this partition tree. For example, the house graph contains five vertices, thus, its original partition tree contains $5! = 120$ leaves. Applying the coarsest equitable partition and vertex individualization yields the pruned 4-leaf tree shown in Figure 26, where the equitable partitions are boxed. Since individualizing block $\{c,d\}$ yields discrete partitions, it is not necessary to individualize block $\{b,e\}$.

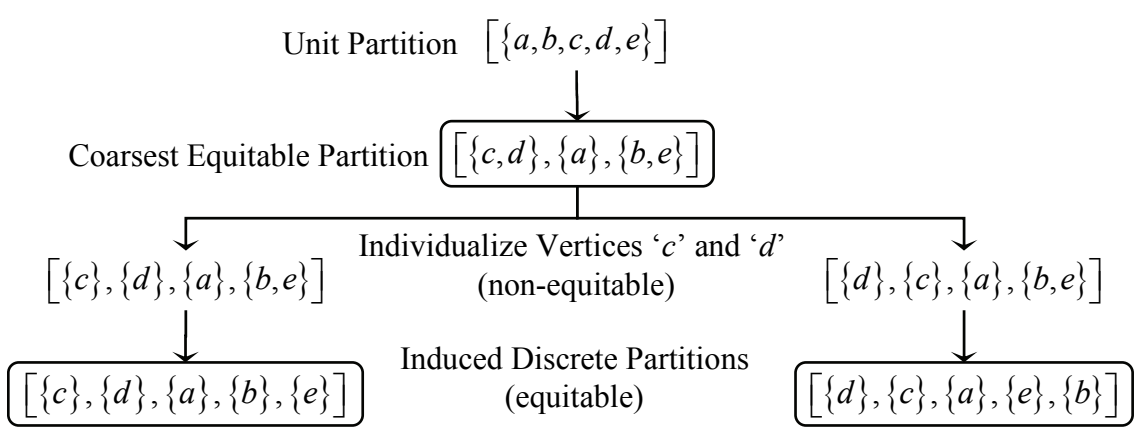


Figure 26. House Graph's Vertex Partition Tree (Equitable Partitions Boxed)

The next technique prunes the partition tree by comparing partial permutations, which compares the binary value induced by the partition's leading discrete blocks. That method extends the concept of retaining the minimum discovered isomorph described in Section 2.2.4. For example, the house graph and its corresponding adjacency matrix are shown in Figure 27 and Table 14(a), respectively.

Individualizing the two vertices, *c* and *d*, as shown in Figure 26 yields the two non-equitable partitions, $\left[\{c\},\{d\},\{a\},\{b,e\}\right]$ and $\left[\{d\},\{c\},\{a\},\{b,e\}\right]$, respectively. The leading discrete blocks in each partition is $\left[\{c\},\{d\},\{a\}\right]$ and $\left[\{d\},\{c\},\{a\}\right]$, thus inducing the partial permutations, $[3,4,1]$ and $[4,3,1]$, respectively. The resulting partial isomorphs are shown in Tables 14(b) and (c), respectively. Since the upper-right triangles in both matrices are identical, neither branch can be eliminated in this example, thus, both branches of the pruned vertex partition tree must be traversed.

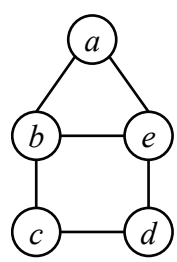


Figure 27. House Graph

Table 14. Partial Permutations of the House Graph's Adjacency Matrix

(a) $\mathbf{A}$

	a	*b*	*c*	*d*	*e*
a	0	1	0	0	1
b	1	0	1	0	1
c	0	1	0	1	0
d	0	0	1	0	1
e	1	1	0	1	0

(b) $\mathbf{A}_1 = \phi_{[3,4,1]}(\mathbf{A})$

	c	*d*	*a*
c	0	1	0
d	1	0	0
a	0	0	0

(c) $\mathbf{A}_2 = \phi_{[4,3,1]}(\mathbf{A})$

	d	*c*	*a*
d	0	1	0
c	1	0	0
a	0	0	0

Since individualizing vertex *c* or *d* does not yield a discrete partition in the vertex partition tree shown in Figure 26, both leaves are further refined. The first leaf is obtained by individualizing vertex *c* and refining to the coarsest equitable partition, which yields $\left[\{c\},\{d\},\{a\},\{b\},\{e\}\right]$ and induces the permutation, $\phi=[3,4,1,2,5]$. Similarly, the leaf obtained by individualizing vertex *d* and refining to a coarsest equitable partition yields $\left[\{d\},\{c\},\{a\},\{e\},\{b\}\right]$ and induces the permutation, $\phi=[4,3,1,5,2]$. Concatenating the entries contained in the first isomorph's upper-right triangle's columns yields the binary value, $1001010111_2=599_{10}$, where the other isomorph also yields $1001010111_2=599_{10}$, Since both isomorphs yield the same binary value, these isomorphs are automorphs.

In this example, the leaves that remain in the pruned vertex partition tree induce permutations that are also automorphisms. Alternatively stated, each leaf, or, permutation, yields the same isomorph, which is designated as the house graph's canonical isomorph. In sum, applying the coarsest equitable partition, individualization, and selection of the minimum remaining isomorph reduced the search space for canonical isomorph of the house graph from $5!=120$ potential permutations (leaves) to two permutations (leaves). However, the techniques applied thus far have not addressed computing the orbit partition while conducting this depth-first search.

Table 15. House Graph Isomorphs

(a) $\mathbf{A}_1=\phi_{[3,4,1,2,5]}(\mathbf{A})$

	c	*d*	*a*	*b*	*e*
c	0	1	0	1	0
d	1	0	0	0	1
a	0	0	0	1	1
b	1	0	1	0	1
e	0	1	1	1	0

(b) $\mathbf{A}_2=\phi_{[4,3,1,5,2]}(\mathbf{A})$

	d	*c*	*a*	*e*	*b*
d	0	1	0	1	0
c	1	0	0	0	1
a	0	0	0	1	1
e	1	0	1	0	1
b	0	1	1	1	0

The last technique that is used to prune the partition tree yields the orbit partition. However, this technique is the most complex method applied, and similarly, it is the most difficult to implement correctly. Recalling the previous example, it was observed that the permutations induced by the leaves remaining in the partition tree illustrated in Figure 26 yield the same isomorph of the house graph, i.e., they are relative automorphisms.

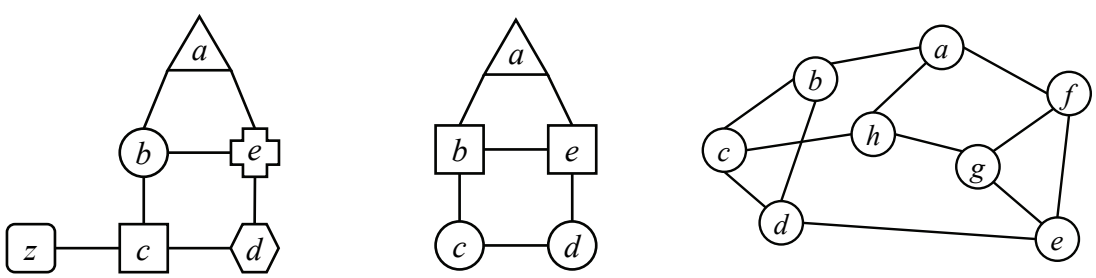
The first leaf contains the discrete vertex partition, $\left[\{c\},\{d\},\{a\},\{b\},\{e\}\right]$, thus inducing the permutation, $\phi=\left[3,4,1,2,5\right]$. Similarly, the second leaf contains the discrete partition, $\left[\{d\},\{c\},\{a\},\{e\},\{b\}\right]$, inducing the permutation, $\phi=\left[4,3,1,5,2\right]$. Inspection reveals the fundamental difference between the two permutations is the position exchange with respect to vertices *c* and *d*, or similarly, vertices *b* and *e*. Thus, vertices *c* and *d* are located in the same orbit and vertices *b* and *e* are located in the same orbit. Conversely, vertex *a* is not located in the orbit of any other vertex, thus, it is the only vertex contained in its block of the house graph's coarsest equitable partition. Therefore, the house graph's orbit partition is $\left[\{c,d\},\{a\},\{b,e\}\right]$.

The management of such automorphisms may enable pruning several branches in the partition tree, although that it is not true in this example. The use of such group theory machinery, e.g., orbits, stabilizers, the automorphism group, the Orbit-Stabilizer theorem, and the Schreier-Sims algorithm, are applied in methods that use automorphisms to prune the vertex partition tree [KrS98, Hof82, Mar02, Ser02]. However, even after applying all three tools of pruning the partition tree: coarsest equitable partitions, partial permutations, and automorphisms, applications similar in design to *nauty* still require exponential time to find the orbit partition or a canonical isomorph of certain graphs [Miy96].

2.3.4.3. Easy, Hard, and In-Between Graphs

A rigorous method of classifying an arbitrary graph with respect to the difficulty it causes algorithms that decide graph isomorphism, such as *nauty*, is not known to exist. However, various subjective notions of measuring relative graph difficulty are often used. For instance, almost every random graph is considered an easy graph, since the coarsest equitable partition of almost all random graphs is discrete. Alternatively stated, applying the coarsest equitable partition to an arbitrary random graph prunes its partition tree from having $n!$ leaves to a single leaf, yielding a canonical isomorph in $\mathrm{O}\left(n^2 \cdot \log n\right)$ time.

In contrast, the coarsest equitable partition of a non-trivial graph contains at least one block composed of multiple vertices, i.e., a non-discrete partition. In the extreme, the coarsest equitable partition may only contain one block, i.e., it is the unit partition. Thus, an arbitrary graph yields a coarsest equitable partition that is a discrete, non-discrete, or unit partition, e.g., the graphs illustrated in Figures 28(a)–(c), respectively. The methods described herein for improving the PageRank algorithm's performance can be applied to graphs whose coarsest equitable partition is non-discrete, e.g., the house or cuneane graph shown in Figures 28(b) and (c), respectively.



(a) Mansion Graph (discrete partition) $\left[\{b\},\{f\},\{c\},\{e\},\{d\},\{a\}\right]$

(b) House Graph (non-discrete partition) $\left[\{c,d\},\{a\},\{b,e\}\right]$

(c) Cuneane Graph (unit partition) $\left[\{a,b,\ldots,h\}\right]$

Figure 28. Easy, Medium, and Hard: Discrete, Non-Discrete and Unit Partitions

2.3.5. Induced Quotient Graphs and Matrices

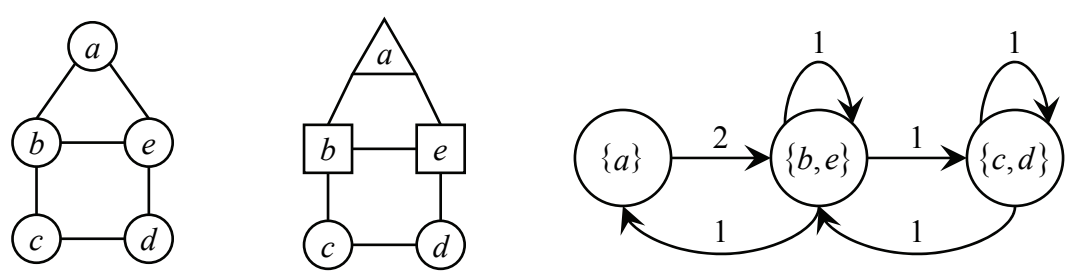
An equitable partition, such as the coarsest equitable partition and orbit partition, are closely related to the eigenvalues and eigenvectors of the graph's adjacency matrix. The concept of an equitable partition was first defined by Schwenk in the context of this relationship [Sch74], and not its usage as a tool for pruning search trees in algorithms that decide graph isomorphism. Sachs defined the equivalent graph divisors concept, as noted by Cvetković, Rowlinson and Simić [CRS97]. As its origin suggests, equitable partitions can be used to explore a graph's eigensystem. In particular, if some equitable partition is non-discrete, portions of a graph's eigensystem can be explored more efficiently using the quotient graph induced by the equitable partition [Hae95, God93, GoR01].

In simple terms, an equitable partition divides the eigenvalues, or equivalently, the characteristic polynomial, of the graph's adjacency matrix. The quotient graph induced by the equitable partition also defines an adjacency matrix whose eigenvalues are a subset of the eigenvalues yielded by the graph's adjacency matrix. This relationship also illustrates a graph can possess regular structure and that vertices can be mapped to some eigenvalue or eigenvector entry. This idea has also been used as the basis of algorithms that decide graph isomorphism [CRS97, HZL05].

The quotient graph induced by an arbitrary equitable partition, e.g., the coarsest equitable partition or orbit partition of an arbitrary graph, replaces the vertices contained in a given block of the partition with one vertex in the quotient graph. The edges between the vertices in the quotient graph correspond to the number of neighbors contained in the destination block with respect to each source block vertex. Quotient graphs often contain weighted edges, directed edges, and loops, i.e., they typically are not simple graphs.

For example, the house graph shown in Figure 29(a) yields the coarsest equitable partition, $\left[\{a\},\{c,d\},\{b,e\}\right]$, which is depicted using unique shapes in Figure 29(b). The partition contains three blocks, thus, the quotient graph contains three vertices, where one vertex corresponds to each block contained in the partition, as shown in Figure 29(c) and reflected in the corresponding adjacency and quotient matrices listed in Table 16.

Vertices *c* and *d* are each connected to one vertex in block $\{b,e\}$, thus, an edge of weight '1', or a 1-edge, links block $\{c,d\}$ to block $\{b,e\}$ and vice versa. Vertices *c* and *d* are connected to each other, as are vertices *b* and *e*, thus, a 1-loop is attached to blocks $\{c,d\}$ and $\{b,e\}$. Since vertex *a* is connected to both vertices contained in block $\{b,e\}$, a 2-edge links block $\{a\}$ to block $\{b,e\}$. Finally, a 1-edge connects block $\{b,e\}$ to block $\{a\}$, since vertices *b* and *e* each have one neighbor contained in block $\{a\}$, namely, *a*.



(a) House Graph (b) Coarsest Partition (c) Induced Quotient Graph

Figure 29. House Graph's Induced Quotient Graph

Table 16. House Graph's Induced Quotient Matrix

(a) **A**

	a	*b*	*c*	*d*	*e*
a	0	1	0	0	1
b	1	0	1	0	1
c	0	1	0	1	0
d	0	0	1	0	1
e	1	1	0	1	0

(b) **Q**

		destination		
		$\{a\}$	$\{b, e\}$	$\{c, d\}$
source	$\{a\}$	0	1	0
	$\{b, e\}$	2	1	1
	$\{c, d\}$	0	1	1

2.3.5.1. Interlacing Eigenvalues

Given an arbitrary matrix, **M**, and similarly arbitrary partition, *B*, the eigenvalues of the induced quotient matrix, **Q**, *interlace* **M**'s eigenvalues. That is, given an arbitrary matrix, $\mathbf{M}^{n,n}$, and an arbitrary partition, *B*, containing *m* blocks and the induced quotient matrix, $\mathbf{Q}^{m,m}$, **Q**'s i^{th} eigenvalue is less than or equal to **M**'s i^{th} eigenvalue and greater than or equal to **M**'s $(n-m-i)^{\text{th}}$ eigenvalue, i.e.,

$$\lambda_{n-m+i}(\mathbf{M}) \le \lambda_i(\mathbf{Q}) \le \lambda_i(\mathbf{M}),\ |\lambda_i| \le |\lambda_{i+1}|. \tag{6}$$

However, if the partition, *B*, used to construct **Q** is an equitable partition, e.g., the coarsest equitable partition, **Q**'s eigenvalues are a subset of **M**'s eigenvalues. Thus, given an arbitrary eigenvalue of **Q**, denoted $\lambda_i(\mathbf{Q})$, there exists some corresponding value, *j*, such that $\lambda_i(\mathbf{Q}) = \lambda_j(\mathbf{M})$. In certain sources, this restricted version is called interlacing; whereas generalized interlacing permits *B* to be non-equitable.

For example, the eigenvalues of the house graph and the quotient graph induced by its coarsest equitable partition, $[\{a\},\{c,d\},\{b,e\}]$, are listed in Tables 17(a) and (b), respectively. Since this partition is equitable, the quotient graph's eigenvalues are a subset of the house graph's eigenvalues, as shown in Table 17.

Table 17. House Graph's and Its Induced Quotient Graph's Eigenvalues

(a) λ, House Graph

i	λ_i
1	2.4812
2	0.6889
3	0.0000
4	-1.1701
5	-2.0000

(b) λ, Induced Quotient Graph

i	λ_i
1	2.4812
2	0.6889
3	-1.1701

2.3.5.2. Lifting Eigenvectors

Some eigenvectors of **M** can be *lifted* from eigenvectors of the quotient matrix, **Q**. This relationship is based on the *characteristic matrix*, **B**, the incidence matrix derived from the vertex partition, *B*, used to induce the quotient matrix, **Q**. Rows in **B** correspond to rows and columns in a source matrix, **M**, and columns in **B** correspond to blocks in *B*. For example, the house graph's coarsest equitable partition yields the 5×3 characteristic matrix, or block matrix, listed in Table 18.

The characteristic matrix, **B**, is the basis of several identities with respect to the original and quotient graphs. The first elementary result is

$$\mathbf{N} = \mathbf{B}^T \cdot \mathbf{B}, \tag{7}$$

where **N** is the diagonal matrix reflecting the number of vertices contained in the blocks of the associated coarsest equitable partition, as shown in Table 19(a). Since **N** is simply a diagonal matrix containing non-zero diagonal entries, its matrix inverse, $\mathbf{N}^{-1}$, is obtained by simply reciprocating **N**'s diagonal entries, as shown in Table 19(b).

Table 18. Block Matrix, **B**, of the House Graph's Coarsest Equitable Partition

	$\{a\}$	$\{b, e\}$	$\{c, d\}$
a	1	0	0
b	0	1	0
c	0	0	1
d	0	0	1
e	0	1	0

Table 19. Block Matrix, **N**, of the House Graph's Coarsest Equitable Partition

(a) $\mathbf{N} = \mathbf{B}^T \cdot \mathbf{B}, \mathbf{N}_{i,i} = \sum \mathbf{B}_{i,i}$

	$\{a\}$	$\{b, e\}$	$\{c, d\}$
$\{a\}$	1	0	0
$\{b, e\}$	0	2	0
$\{c, d\}$	0	0	2

(b) $\mathbf{N}^{-1}, \mathbf{N}_{i,i}^{-1} = 1/\mathbf{N}_{i,i}$

	$\{a\}$	$\{b, e\}$	$\{c, d\}$
$\{a\}$	1	0	0
$\{b, e\}$	0	1/2	0
$\{c, d\}$	0	0	1/2

More notably, it further can be shown that [God93, GoR01]

$$\mathbf{A} \cdot \mathbf{B} = \mathbf{B} \cdot \mathbf{Q}\,. \tag{8}$$

By left-multiplying each side with $\mathbf{B}^T$, it can also be shown that $\mathbf{Q} = \mathbf{B}^{\dagger} \cdot \mathbf{A} \cdot \mathbf{B}$, since

$$\begin{aligned} \mathbf{B}^T \cdot \mathbf{A} \cdot \mathbf{B} &= \mathbf{B}^T \cdot \mathbf{B} \cdot \mathbf{Q} \\ \mathbf{A} \cdot \mathbf{B} &= \left(\mathbf{B}^T \cdot \mathbf{B}\right)^{-1} \cdot \left(\mathbf{B}^T \cdot \mathbf{B}\right) \cdot \mathbf{Q} \\ \left(\mathbf{B}^T \cdot \mathbf{B}\right)^{-1} \cdot \mathbf{B}^T \cdot \mathbf{A} \cdot \mathbf{B} &= \cancel{\mathbf{N}^{-1} \cdot \mathbf{N}} \cdot \mathbf{Q} \\ \mathbf{B}^{\dagger} \cdot \mathbf{A} \cdot \mathbf{B} &= \mathbf{Q}, \end{aligned} \tag{9}$$

where $\mathbf{B}^{\dagger}$ denotes the pseudo-inverse of **B**, i.e., $\mathbf{B}^{\dagger} = \left(\mathbf{B}^T \cdot \mathbf{B}\right)^{-1} \cdot \mathbf{B}^T$. Given an eigenvalue, λ_i, and an eigenvector, $\mathbf{r}_i \neq 0$, of the quotient graph, **Q**, yielded by an equitable partition of **A**, it can be shown that [Hae95]

$$\mathbf{A} \cdot \mathbf{B} \cdot \mathbf{r}_i = \mathbf{B} \cdot \mathbf{Q} \cdot \mathbf{r}_i = \lambda_i \cdot \mathbf{B} \cdot \mathbf{r}_i\,, \tag{10}$$

where substituting $\mathbf{x}_i = \mathbf{B} \cdot \mathbf{r}_i$ in $\mathbf{A} \cdot \mathbf{B} \cdot \mathbf{r}_i = \lambda_i \cdot \mathbf{B} \cdot \mathbf{r}_i$ yields $\mathbf{A} \cdot \mathbf{x}_i = \lambda_i \cdot \mathbf{x}_i$.

Thus, this identity *lifts* the quotient matrix eigenvector, $\mathbf{r}_i$, to some eigenvector of the adjacency matrix, $\mathbf{x}_i = \mathbf{B} \cdot \mathbf{r}_i$. Quotient matrices may be smaller than the input matrix, thus, lifting can accelerate finding eigenvectors. For example, the quotient matrix induced by the house graph's coarsest equitable partition yields the dominant eigenvector listed in Table 20(a). Lifting yields the house graph's dominant eigenvector listed in Table 20(b).

Table 20. Lifting a Dominant Eigenvector of the House Graph

(a) $\mathbf{r}_i$

$\{a\}$	0.5555
$\{b, e\}$	0.6892
$\{c, d\}$	0.4653

(b) $\mathbf{x}_i = \mathbf{B} \cdot \mathbf{r}_i$

a	0.5555
b	0.6892
c	0.4653
d	0.4653
e	0.6892

2.3.5.3. Interlacing, Lifting, Arbitrary Matrices and Arbitrary Partitions

Thus, an arbitrary matrix, **M**, and partition, *B*, of **M**'s rows and columns yields the block matrix, **B**, and the quotient matrix, $\mathbf{Q} = \mathbf{B}^{\dagger} \cdot \mathbf{M} \cdot \mathbf{B} = \left(\mathbf{B}^{T} \cdot \mathbf{B}\right)^{-1} \cdot \mathbf{B}^{T} \cdot \mathbf{M} \cdot \mathbf{B}$. Moreover, **Q**'s eigenvalues interlace **M**'s eigenvalues, since **Q**'s eigenvalues are strictly contained in the intervals bounded by **M**'s eigenvalues, i.e.,

$$\lambda_{n-m+i}(\mathbf{M}) \le \lambda_i(\mathbf{Q}) \le \lambda_i(\mathbf{M}), |\lambda_i| \le |\lambda_{i+1}|, \tag{11}$$

where **Q** is an $k \times k$ matrix, **M** is an $n \times n$ matrix, $k \le n$, and $b = k = |B|$. If an arbitrary partition, *B*, is simply an equitable partition, such as **M**'s coarsest equitable partition, **Q**'s eigenvalues are a subset of **M**'s eigenvalues, i.e., given $1 \le i \le k$ and $1 \le j \le n$,

$$\lambda_i(\mathbf{Q}) = \lambda_j(\mathbf{M}). \tag{12}$$

If *B* is equitable, given some arbitrary eigenvalue of **Q**, $\lambda_i(\mathbf{Q})$, and an eigenvalue of **M**, $\lambda_j(\mathbf{M})$, such that $\lambda_i(\mathbf{Q}) = \lambda_j(\mathbf{M})$, an eigenvector, $\mathbf{r}_i$, associated with $\lambda_i(\mathbf{Q})$, is related to eigenvector, $\mathbf{x}_j$, associated with $\lambda_j(\mathbf{M})$ by the relationship,

$$\mathbf{x}_j = \mathbf{B} \cdot \mathbf{r}_i. \tag{13}$$

The quotient matrix, $\mathbf{Q} = \mathbf{B}^{\dagger} \cdot \mathbf{M} \cdot \mathbf{B}$, equals **M**'s corresponding average row sums, i.e., $\mathbf{Q}_{i,j} = \sum \mathbf{M}_{u,v}$ where $B = [b_1, b_2, \ldots, b_k]$, $u \in b_i$, $v \in b_j$, $i, j \le k$ and $k \le n$ [Hae95]. If *B* is equitable, the summation, $\mathbf{Q}_{i,j} = \sum \mathbf{M}_{u,v}$, simplifies to $\mathbf{Q}_{i,j} = z_{i,j} \cdot \mathbf{M}_{u,w}$, such that $u \in b_i$, $w \in b_j$, $\{u, w\} \in E$, and $z_{i,j} = |N(u) \cap b_j|$, the number of *r*'s neighbors contained in block b_j [Hae95]. Therefore, if *B* is an equitable partition, a summation of as many as *n* values can be reduced to a single multiplication operation.

2.4. A Brief Interlude: Eigen Decomposition Applications in Graph Theory

Eigen decomposition is the basis of several fundamental results in graph theory. A classic example is that the number of components in the graph, denoted $\kappa(G)$, equals the multiplicity of '0' in the eigenvalues of the graph's Laplacian matrix, denoted **L**, where $\kappa(G) = \left(\sum \lambda_i(\mathbf{L}) \equiv 0\right)$ [Chu94]. The Laplacian is constructed by subtracting the graph's adjacency matrix, **A**, from its degree matrix, **D**, where $\mathbf{D}_{i,i} = \deg(v_i)$, thus $\mathbf{L} = \mathbf{D} - \mathbf{A}$. Some recent results are based on applying the *signless Laplacian*, $\mathbf{L} = \mathbf{D} + \mathbf{A}$ [HaS04].

The eigen decomposition is used to partition graphs, e.g., mapping related parallel tasks to some pool of processors [HeL93, GuM95, Got03]. Another application provides solutions to the traveling salesman problem [Moh04]. Eigenvalues are also used to find the rate at which a graph's stationary distribution is reached, e.g., to assess data structure resiliency [AsW05] or determine how quickly the PageRank vector converges [HaK03]. A beautiful application uses eigenvectors as vertex coordinates [Kor05]. For example, the drawing of the Buckminsterfullerene molecule, C_{60}, or buckyball, illustrated in Figure 30 is based on three eigenvectors of the buckyball's signless Laplacian.

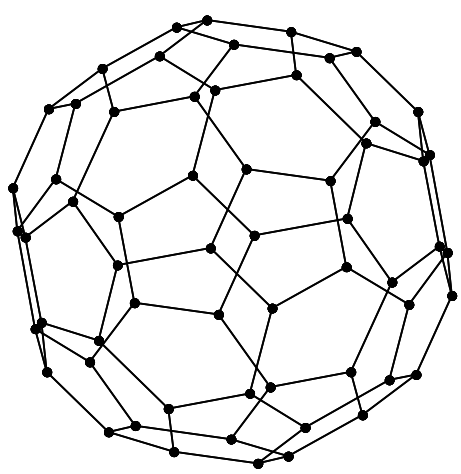
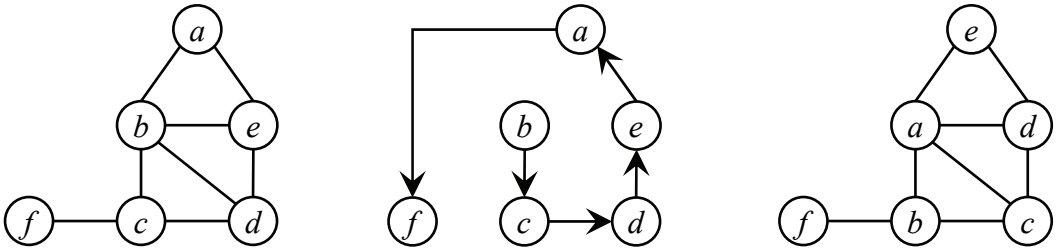


Figure 30. 3-D Buckyball Drawing Based on Its Signless Laplacian's Eigenvectors

2.5. The PageRank Algorithm

Another significant application of the eigen decomposition to graph theory is the PageRank algorithm used in certain search engines to order query responses, such as the web pages matching a user's search criteria [PBM+98]. For example, the mansion graph shown in Figure 31(a) yields the PageRank vector listed in Table 21(a), which is a unique eigenvector that always exists, as described in this section. Sorting the PageRank vector's entries yields the vector listed in Table 21(b), which is reflected in Figure 31(b).

The PageRank vector is unique up to isomorphism. Since every entry contained in the mansion graph's PageRank vector is distinct, every mansion graph isomorph yields a vertex ordering equivalent to the vertex ordering shown in Table 21(b). For example, the isomorph shown in Figure 31(c) yields the sorted PageRank vector shown in Table 21(c), which is equivalent to the ordering shown in Table 21(b) and reflected in Figure 31(b).



(a) Mansion Graph (b) PageRank Order (c) Another Isomorph

Figure 31. Mansion Graph

Table 21. Mansion Graph's PageRank Vector

(a) PageRank Vector

a	0.126
b	0.236
c	0.195
d	0.182
e	0.180
f	0.080

(b) PageRank Order

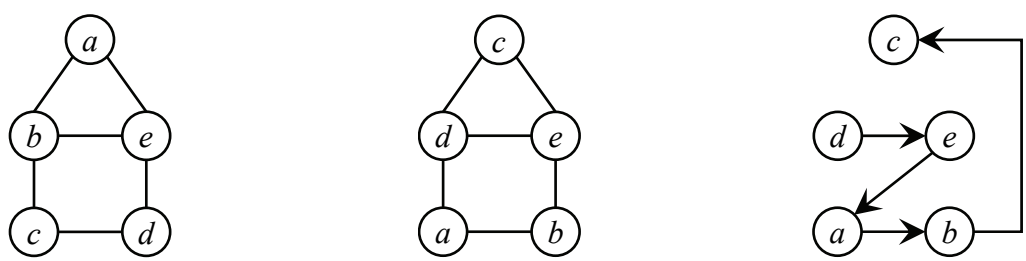
0.236	*b*
0.195	*c*
0.182	*d*
0.180	*e*
0.126	*a*
0.080	*f*

(c) Isomorph's Order

0.236	*a*
0.195	*b*
0.182	*c*
0.180	*d*
0.126	*e*
0.080	*f*

Conversely, if the PageRank vector contains duplicate entries, it cannot induce a canonical ordering. That issue can be resolved in some applications, e.g., search engines, by sorting on more keys, e.g., web page addresses. Alternatively, ties in PageRank value can be broken by the vertex order in a canonical isomorph produced by applications such as *nauty* (cf. Sections 1.2 and 2.3.4.2).

For example, the house graph depicted in Figure 32(a) yields the PageRank vector listed in Table 22(a). The house graph's canonical isomorph yielded by *nauty* is shown in Figure 32(b) and corresponds to applying the permutation, $[3,4,1,2,5]$, where the vertex mapping, $i \rightarrow j \Leftrightarrow r \rightarrow s$ denotes vertex v_i, labeled *r*, is mapped to vertex v_j, labeled *s*. The two graphs are isomorphs, thus, their PageRank vectors are also related by the same vertex permutation, as reflected in Tables 22(a) and (b), where the canonical isomorph induces the canonical vertex ordering listed in Table 22(c) and shown in Figure 32(c).



(a) House Graph (b) Canonical Isomorph (c) PageRank Order

Figure 32. House Graph

Table 22. House Graph's PageRank Vector

(a) House Graph

a	0.168
b	0.244
c	0.172
d	0.172
e	0.244

(b) Canonical Isomorph

$1 \rightarrow 3 \Leftrightarrow a \rightarrow c$	0.168
$2 \rightarrow 4 \Leftrightarrow b \rightarrow d$	0.244
$3 \rightarrow 1 \Leftrightarrow c \rightarrow a$	0.172
$4 \rightarrow 2 \Leftrightarrow d \rightarrow b$	0.172
$5 \rightarrow 5 \Leftrightarrow e \rightarrow e$	0.244

(c) PageRank Order

0.244	*d*
0.244	*e*
0.172	*a*
0.172	*b*
0.168	*c*

However, finding a canonical isomorph of certain graphs may require exponential time [Miy96]. Access to a canonical isomorph also does not reduce any numerical errors in the PageRank vector if the vector is obtained using standard finite-precision arithmetic, e.g., as is specified in the IEEE 754 standard [ISB85]. The three algorithms described in Chapters 4 and 5 eliminate a certain class of numerical errors that may be present in the PageRank vector. More importantly, two of these algorithms dramatically reduce the time needed to compute the PageRank vector of certain graphs.

To obtain this vertex order, the PageRank algorithm perturbs the adjacency matrix and builds a strictly positive stochastic matrix in which each entry reflects the probability of visiting a vertex from another vertex. Assuming the probabilities are independent, each entry reflects the probability of transitioning between states in a Markov chain. Applying the Perron-Frobenius and Perron theorems establishes the matrix must yield the dominant eigenvalue, one. Normalizing the unique and associated dominant eigenvector yields the Markov chain's stationary distribution, where an entry reflects the probability of visiting some vertex, e.g., nodes in a UAV swarm [PBM+98, LaM03, LaM06, PSC05].

Thus, the eigenvector orders a graph's vertices based on PageRank values yielded by its perturbed adjacency matrix. PageRank values are unique up to isomorphism, i.e., vertices have the same PageRank value across all input graph isomorphs. Hence, vertices contained in the same block of the orbit partition also must have equal PageRank values. In other words, given vertices u and v such that a set of automorphisms maps $u \to v$ and an equal number maps $u \to v$, the vertices, u and v, must have equal PageRank values. The proof developed in Section 3.4 shows the same result holds for the coarsest equitable partition, i.e., vertices contained in the same block must have equal PageRank values.

2.5.1. Computing the PageRank Perturbation

The first step in obtaining the PageRank vector is to perturb the adjacency matrix, **A**, to obtain a positive column-stochastic matrix, **S**, i.e., $\mathbf{S}_{i,j} > 0, \forall i, j$ and $\sum \mathbf{S}_{:,j} = 1, \forall j$. The PageRank perturbation applies the *degree matrix*, **D**, used to obtain several stochastic perturbations [Sin64, SiK67, HZL05], where $\mathbf{D}_{i,i}$, equals the degree of vertex v_i, i.e.,

$$\mathbf{D}_{i,j} = \begin{cases} 0 & i \neq j \\ \sum \mathbf{A}_{i,:} = \deg(v_i) = \deg(v_j) = \sum \mathbf{A}_{:,j} & i = j \end{cases}. \tag{14}$$

If **A** is an adjacency matrix of a connected graph, **D**'s diagonal entries are strictly non-zero. Hence, its inverse, denoted $\mathbf{D}^{-1}$, is found by reciprocating **D**'s diagonal entries, i.e., $\mathbf{D}_{i,i}^{-1} = 1/\mathbf{D}_{i,i}, \forall i$. For example, the house graph's adjacency matrix shown in Table 23 yields the degree and inverse matrices listed in Table 24. **D**'s diagonal entries correspond to the graph's vertex degree partition, which for the house graph is $\left[\{a,c,d\},\{b,e\}\right]$.

Table 23. House Graph's Adjacency Matrix, **A**

	a	*b*	*c*	*d*	*e*
a	0	1	0	0	1
b	1	0	1	0	1
c	0	1	0	1	0
d	0	0	1	0	1
e	1	1	0	1	0

Table 24. House Graph's Degree Matrix and Degree Matrix Inverse

(a) **D**

	a	*b*	*c*	*d*	*e*
a	2	0	0	0	0
b	0	3	0	0	0
c	0	0	2	0	0
d	0	0	0	2	0
e	0	0	0	0	3

(b) $\mathbf{D}^{-1}$

	a	*b*	*c*	*d*	*e*
a	1/2	0	0	0	0
b	0	1/3	0	0	0
c	0	0	1/2	0	0
d	0	0	0	1/2	0
e	0	0	0	0	1/3

A *row-stochastic* matrix, **S**, is a non-negative matrix whose rows each sum to one, i.e., $\sum \mathbf{S}_{i,:} = 1$. Given an arbitrary adjacency matrix, **A**, and its degree matrix, **D**, obtained by summing **A**'s rows, a row-stochastic matrix, **S**, can be obtained by computing

$$\mathbf{S} = \mathbf{D}^{-1} \cdot \mathbf{A}. \tag{15}$$

A *column-stochastic* matrix, **S**, is a non-negative matrix whose columns sum to one, i.e., $\sum \mathbf{S}_{:,j} = 1$. Thus, summing **A**'s columns to obtain a degree matrix, **D** enables constructing a column-stochastic matrix, **S**, by inverting the multiplication order, where

$$\mathbf{S} = \mathbf{A} \cdot \mathbf{D}^{-1}. \tag{16}$$

For example, applying the row-stochastic perturbation, (15), to the adjacency and degree matrices enumerated in Tables 23 and 24, respectively, yields the row-stochastic matrix listed in Table 25. Conversely, applying the column-stochastic perturbation, (16), to these same matrices yields the column-stochastic matrix listed in Table 26.

Table 25. A Row-Stochastic Matrix, $\sum \mathbf{S}(i,:) = 1$

0.0	0.5	0.0	0.0	0.5	
$0.\overline{3}$	0.0	$0.\overline{3}$	0.0	$0.\overline{3}$	
0.0	0.5	0.0	0.5	0.0	1
0.0	0.0	0.5	0.0	0.5	
$0.\overline{3}$	$0.\overline{3}$	0.0	$0.\overline{3}$	0.0	
$0.\overline{6}$	$1.\overline{3}$	$0.\overline{8}$	$0.\overline{8}$	$1.\overline{3}$	$\sum$

Table 26. A Column-Stochastic Matrix, $\sum \mathbf{S}(:,j) = 1$

0.0	$0.\overline{3}$	0.0	0.0	$0.\overline{3}$	$0.\overline{6}$
0.5	0.0	0.5	0.0	$0.\overline{3}$	$1.\overline{3}$
0.0	$0.\overline{3}$	0.0	0.5	0.0	$0.\overline{8}$
0.0	0.0	0.5	0.0	$0.\overline{3}$	$0.\overline{8}$
0.5	$0.\overline{3}$	0.0	0.5	0.0	$1.\overline{3}$
1					$\sum$

The perturbation applied in the PageRank algorithm modifies the original column-stochastic perturbation, (16), by applying a scaling factor, $\alpha \in [0.0, 1.0]$, and a shifting factor, $(1-\alpha)/n$, where $n = |V|$, the number of vertices. The stochastic PageRank matrix perturbation, or scaled and shifted modification of (16), is

$$\mathbf{S} = \alpha \cdot \mathbf{A} \cdot \mathbf{D}^{-1} + (1-\alpha)/n, \tag{17}$$

where **A** and **D** denote a graph's adjacency and degree matrices, respectively [PBM+98] and $1-\alpha$ is denoted δ, yielding $\mathbf{S} = \alpha \cdot \mathbf{A} \cdot \mathbf{D}^{-1} + \delta/n$. If $\alpha = 1$, the PageRank perturbation reduces to the column-stochastic perturbation (16), $\mathbf{S} = \mathbf{A} \cdot \mathbf{D}^{-1}$. In essence, decreasing α, and thus increasing δ, causes **S**'s entries to be less dependent on the value of **A**'s entries. In the opposite extreme, if $\alpha = 0$, $\delta = 1$ and $\mathbf{S}_{i,j} = 1/n, \forall i, j$. The PageRank algorithm's developers suggest a scaling, or damping factor of $\alpha = 0.85$, hence $\delta = 0.15$ [PBM+98].

For example, the adjacency matrix of the house graph listed in Table 23 yields the degree and inverse matrices listed in Table 24. Applying the PageRank perturbation, (17), to these matrices, where $\alpha = 0.85$ and $\delta = 0.15$, yields the column-stochastic matrix, i.e., the PageRank matrix, listed in Table 27. As this example demonstrates, all of the columns in the PageRank matrix sum to one, whereas its rows typically do not sum to one.

Table 27. House Graph's PageRank Matrix, **S**, $\alpha = 0.85$

	a	*b*	*c*	*d*	*e*	
a	0.0300	0.3133	0.0300	0.0300	0.3133	0.7167
b	0.4550	0.0300	0.4550	0.0300	0.4550	1.2833
c	0.0300	0.3133	0.0300	0.4550	0.0300	0.8583
d	0.0300	0.0300	0.4550	0.0300	0.4550	0.8583
e	0.4550	0.3133	0.0300	0.4550	0.0300	1.2833
	1					$\sum$

Applying the scaling and shifting modifications, α and δ, to the column-stochastic perturbation, (16), satisfies two objectives. First, the perturbation satisfies the assumption that any vertex, or web page, can be randomly visited. This assumption is often called the random surfer model [LaM03]. Moreover, the column-stochastic PageRank matrix, **S**, is a primitive matrix, where a matrix, **M**, is primitive:

1. if **M** is irreducible and **M** contains one or more positive diagonal entries,
2. or equivalently, if and only if $\mathbf{M}^k > 0$ for some $k > 0$.

The PageRank perturbation, and more specifically, the shifting factor, δ/n, forces every entry in **S** to be strictly positive, i.e., each entry is strictly positive, where $\mathbf{S}^k > 0, \forall\ k > 0$. The PageRank matrix, **S**, is also irreducible, where an arbitrary matrix, **M**, is irreducible if given any permutation matrix, **P**, the matrix, **M**, does not yield an isomorph such that

$$\mathbf{P} \cdot \mathbf{M} \cdot \mathbf{P} = \begin{bmatrix} \mathbf{X} & \mathbf{Y} \\ \mathbf{0} & \mathbf{Z} \end{bmatrix}, \tag{18}$$

where **X**, **Y**, and **Z** are square matrices. The PageRank matrix, **S**, is irreducible since each entry is strictly positive, i.e., **S** is constructed such that none of its entries equal zero.

An arbitrary PageRank matrix, **S**, is irreducible and satisfies the Perron-Frobenius theorem's conditions [LaM06]. Moreover, **S** is also a primitive matrix and satisfies the conditions of the Perron theorem, which is a more powerful theorem [LaM06]. Applying either theorem shows that the PageRank matrix, **S**, yields a positive eigenvalue associated with an eigenvector unique up to isomorphism and scaling, where eigenvectors are often not unique. However, applying the Perron theorem establishes it is the unique dominant eigenvalue, i.e., the eigenvalue having the largest magnitude. Similarly, the eigenvector is the unique dominant eigenvector defining the graph's stationary distribution.

A primitive, irreducible, and stochastic PageRank matrix, **S**, reflects the transition probabilities between pairs of states in an aperiodic Markov chain, a memoryless random process where the probability of entering the next state only depends on the current state. In an aperiodic Markov chain, any state may be entered during any transition, a property satisfied by **S** being irreducible and primitive [LaM06]. **S** is constant after the PageRank perturbation (16) is applied, i.e., all of **S**'s transition probabilities remain constant, thus, **S** represents a stationary Markov chain. Therefore, **S** yields a unique stationary probability distribution, $\mathbf{x}_j$, where for all *i*, $\mathbf{x}_{i+1} = \mathbf{S} \cdot \mathbf{x}_i$, and there exists some value, *j*, such that

$$\mathbf{x}_j = \mathbf{S} \cdot \mathbf{x}_j \,. \tag{19}$$

Given that the PageRank matrix, **S**, is primitive, irreducible, and stochastic, where **S** represents the transition probabilities of a stationary aperiodic Markov chain, it can be shown that its dominant eigenvalue equals one [LaM06]. More importantly, the dominant eigenvector, **x**, reflects the stationary distribution of a memoryless random process, or the unique probability a random surfer visits each vertex. This stationary result is guaranteed by the delta shift value, δ, which ensures each vertex has a nominal probability of being visited. Alternatively stated, applying the additive delta shift value, δ, ensures each vertex can be reached from any other vertex, or in the context of a Markov chain, that any state can be randomly reached from any other state.

Finally, and most significantly, the unique dominant eigenvector, **x**, produced by the PageRank matrix, **S**, is unique up to graph isomorphism. Thus, an arbitrary vertex, v_i, has the associated PageRank value, $\mathbf{x}_i$, with respect to any permutation matrix, **P**, where

$$\mathbf{P} \cdot \mathbf{A} \cdot \mathbf{P}^T \quad \leftrightarrow \quad \mathbf{P} \cdot \mathbf{S} \cdot \mathbf{P}^T \quad \leftrightarrow \quad \mathbf{P} \cdot \mathbf{x} \,. \tag{20}$$

2.5.2. Computing the PageRank Vector

2.5.2.1. Power Method Iteration

There are many ways to find the PageRank vector, e.g., by applying MATLAB's '`eig`' function for finding eigen decompositions. Another method is the *power method*, where PageRank matrix, **S**, is multiplied by the current PageRank vector, $\mathbf{x}_i$, yielding the revised estimate of the PageRank vector, $\mathbf{x}_{i+1}$ [LaM06]. The vector, $\mathbf{x}_{i+1}$, is normalized by applying an arbitrary norm, e.g., $\mathbf{x}_{i+1} \leftarrow \mathbf{x}_{i+1} / \|\mathbf{x}_{i+1}\|_1 = \mathbf{x}_{i+1} / \sum \mathbf{x}_{i+1}$. Thus, each iteration simply requires computing a normalized dot product, where

$$\mathbf{x}_{i+1} \leftarrow \mathbf{S} \cdot \mathbf{x}_i , \tag{21}$$

followed by

$$\mathbf{x}_{i+1} \leftarrow \mathbf{x}_{i+1} / \sum \mathbf{x}_{i+1} . \tag{22}$$

The initial entries in **x** can equal an arbitrary value, such that $\mathbf{x}_1(i) \in [0.0, 1.0]$ and $\sum \mathbf{x}_1 = 1$, e.g., $\mathbf{x}_1 = \mathbf{1}^{n,1}/n$. The power method terminates after some number of iterations is performed or some numerical tolerance, τ, is obtained, e.g., $\sum |\mathbf{x}_{i+1} - \mathbf{x}_i| < \tau, \tau \geq 0$. The power method derives its name from the observation the ith iteration can be expressed by the product of the ith power of **S**, denoted $\mathbf{S}^i$, and the initialization vector, $\mathbf{x}_0$, where

$$\begin{aligned}
\mathbf{x}_1 &\leftarrow [\mathbf{S} \cdot \mathbf{x}_0] / [\sum \mathbf{S} \cdot \mathbf{x}_0] \\
\mathbf{x}_2 &\leftarrow [\mathbf{S} \cdot \mathbf{x}_1] / [\sum \mathbf{S} \cdot \mathbf{x}_1] = [\mathbf{S} \cdot (\mathbf{S} \cdot \mathbf{x}_0)] / [\sum \mathbf{S} \cdot (\mathbf{S} \cdot \mathbf{x}_0)] \\
&\vdots \\
\mathbf{x}_i &\leftarrow [\mathbf{S}^i \cdot \mathbf{x}_0] / [\sum \mathbf{S}^i \cdot \mathbf{x}_0].
\end{aligned} \tag{23}$$

Finally, **S**'s left eigenvector, $\mathbf{y}$, is obtained by computing $\mathbf{y}_{i+1} \leftarrow \mathbf{y}_i^T \cdot \mathbf{S}$, followed by the requisite normalization, $\mathbf{y}_{i+1} \leftarrow \mathbf{y}_{i+1} / \sum \mathbf{y}_{i+1}$.

2.5.2.2. Expected Number of Power Method Iterations

The upper bound on the number of power method iterations required to obtain the PageRank vector, **x**, to some precision τ, e.g., $\tau < 0.001$, is based on the convergence rate, the rate that $c^t \to 0$ for some value c. The value, c, is bounded by the magnitude ratio of **S**'s two most dominant eigenvalues, i.e., $c \le |\lambda_2| / |\lambda_1|$. **S** is a stochastic matrix, therefore, $\lambda_1 = 1$, which yields $c \le |\lambda_2|$. It has also been shown that the second dominant eigenvalue is bounded by the scaling value, α, thus, $c \le |\lambda_2| \le \alpha$ [HaK03]. Since $c^t \le \tau$, the upper bound on the number of power method iterations, t, needed to obtain the PageRank vector using floating-point arithmetic computed in some base, b, is [GoV88, HaK03, LaM06]

$$t \le \frac{\log_b \tau}{\log_b c} \le \frac{\log_b \tau}{\log_b |\lambda_2(\mathbf{S})|} \le \frac{\log_b \tau}{\log_b \alpha} \quad \to \quad t \le \log_{|\lambda_2(\mathbf{S})|} \tau \le \log_\alpha \tau . \tag{24}$$

For example, if a PageRank vector is found using binary floating-point arithmetic, $b = 2$. Assuming $\alpha = 0.85$ and $\tau \le 0.001$,

$$t \le \log_{|\lambda_2(\mathbf{S})|} \tau \le \log_\alpha \tau = \frac{\log_b \tau}{\log_b \alpha} = \frac{\log_{10} 0.001}{\log_{10} 0.85} = \frac{\log_2 0.001}{\log_2 0.85} = 42.5043 . \tag{25}$$

Therefore, performing 43 power method iterations ensures the PageRank vector is correct to three decimal digits, or equivalently, approximately ten bits.

For many graphs, $\log_2 n$ power method iterations often appears to yield sufficient precision in the PageRank vector [PBM+98]. However, a formal lower bound has not yet been established on the number of iterations required to obtain the required precision, τ. As shown in Section 3.2, assuming $\tau \le 1/n$ and $\alpha \ge 0.5$, a practical lower bound on the number of necessary power method iterations, t, is $\log_2 n$, i.e., $\log_2 n \le t$.

2.5.3. PageRank: An Algorithm for Ranking Vertices

The PageRank algorithm consists of two key steps, applying a perturbation to the graph's adjacency matrix that yields a strictly positive stochastic matrix and determining the dominant eigenvector of the stochastic matrix by computing an iterated dot product. The eigenvector's entries correspond to the probability the corresponding vertices will be visited by an object that randomly selects its destination. The entire PageRank algorithm is listed in Figure 33, where the stochastic PageRank perturbation is applied on lines 2–6. Since the power method terminates based on the numerical differences with respect to the last eigenvector estimate, the two vectors, **s** and **x**, are initialized on lines 8 and 9, along with an iteration counter, *z*, on line 10.

The power method loop is entered on line 12 and the eigenvector estimate, **x**, is copied to the vector, **s**, on line 14. The eigenvector is revised on line 16, where $\mathbf{x} = \mathbf{S} \cdot \mathbf{x}$. Normalizing by the column sum norm on line 16, or 1-norm, $\|\mathbf{x}\|_1$, ensures **x** sums to one, i.e., that **x** is a probability distribution. The norm choice is arbitrary, e.g., the Euclidean, or spectral norm, $\|\mathbf{x}\|_2$, could be used. The power method terminates after the tolerance, τ, is obtained and the upper bound on the number of power method iterations is $t = \log_\alpha \tau$.

The PageRank perturbation requires $\Theta\left(n^2\right)$ time, where $n = |V|$, since each entry in **A** is scaled and shifted. Given a tolerance, τ, and scaling factor, α, the lower and upper bounds on the power method's complexity are $\Omega\left(n^2 \cdot \log n\right)$, and $\mathrm{O}\left(n^2 \cdot t\right)$, respectively, where $t = \log_\alpha \tau$. If the graph is stored using sparse matrices, the lower and upper bounds can be reduced to $\Omega\left(m \cdot \log n\right)$ and $\mathrm{O}\left(m \cdot t\right)$, respectively, where $m = |E|$, the number of edges contained in the graph.

A common modification of the PageRank algorithm uses a personalization vector, **v**, containing user-specified probabilities. A potential application is "to decrease the effect of spamming done by the so-called link farms" [LaM06]. The personalization vector, **v**, is used to modify the perturbation on line 6 in Figure 33, which becomes [LaM03, LaM06]

$$\mathbf{S} = \alpha \cdot \mathbf{A} \cdot \mathbf{D}^{-1} + (1-\alpha) \cdot \mathbf{v} \cdot \mathbf{1}^{1,n}. \tag{26}$$

```
1.  getPageRank(A, n, α, τ)
2.    # construct degree matrix
3.    d_i ← Σ_{i=1}^{n} A_{:,i}
4.    D ← diag(d)
5.    # apply PageRank perturbation
6.    S ← α · A · D^{-1} + (1 − α)/n

7.    # initialize vectors and counter
8.    x ← 1^{n,1}/n
9.    s ← 0^{n,1}
10.   z ← 0

11.   # iterate power method
12.   while(‖s − x‖_2 > τ)
13.     # save PageRank vector
14.     s ← x

15.     # update PageRank vector
16.     x ← S · x

17.     # normalize PageRank vector
18.     x ← x/Σx

19.     # increment loop counter
20.     z ← z + 1
21.   end while

22.   return x
23. end PageRank
```

Figure 33. PageRank: An Algorithm for Ordering Vertices [PBM+98]

The following example applies the PageRank algorithm to the paw graph shown in Figure 34 [Wes01]. The corresponding adjacency, degree, and inverse degree matrices are listed in Tables 28(a)–(c), respectively. The perturbation on lines 2–5 yields the graph shown in Figure 35, where edges created by the perturbation are depicted as dotted lines and the corresponding perturbed matrix is listed in Table 28(d). The initial PageRank and history vectors, $\mathbf{x}_0 = \mathbf{1}^{4,1}/4$ and $\mathbf{s}_0 = \mathbf{0}^{4,1}$, are shown in Tables 29(a) and (b), respectively. The revised vectors produced by executing the first power method iteration are shown in Tables 29(c) and (d), respectively. The normalization step performed on line 17 does not change **x** in this particular example.

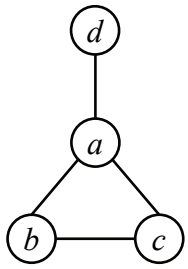


Figure 34. Paw Graph [Wes01]

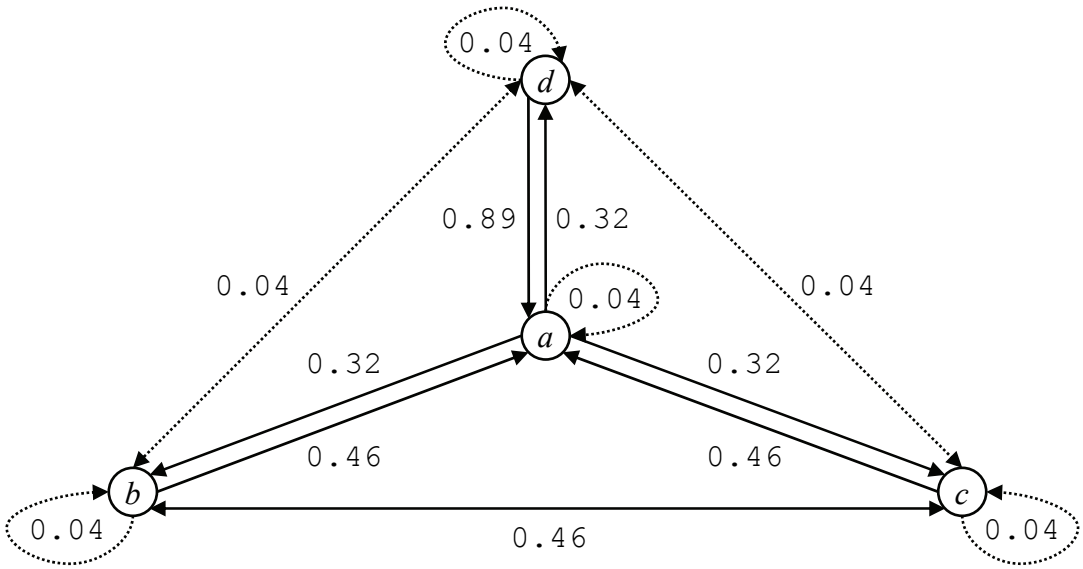


Figure 35. Applying the PageRank Perturbation to the Paw Graph, $\alpha = 0.85$

Table 28. Paw Graph's Adjacency and PageRank Matrices, $\alpha = 0.85$

(a) $\mathbf{A}$

	a	b	c	d
a	0	1	1	1
b	1	0	1	0
c	1	1	0	0
d	1	0	0	0

(b) $\mathbf{D}$

	a	b	c	d
a	3	0	0	0
b	0	2	0	0
c	0	0	2	0
d	0	0	0	1

(c) $\mathbf{D}^{-1}$

	a	b	c	d
a	1/3	0	0	0
b	0	1/2	0	0
c	0	0	1/2	0
d	0	0	0	1

(d) $\mathbf{S} = \alpha \cdot \mathbf{A} \cdot \mathbf{D}^{-1} + (1-\alpha)/n, \alpha = 0.85$

		source				
		a	b	c	d	
destination	a	0.04	0.46	0.46	0.89	1.85
	b	0.32	0.04	0.46	0.04	0.86
	c	0.32	0.46	0.04	0.04	0.86
	d	0.32	0.04	0.04	0.04	0.43
		1				$\sum$

The next iteration yields the PageRank vectors listed in Tables 29(e) and (f). The power method does not converge to four decimal places until the 20th iteration, as shown in the PageRank vector listed in Table 29(h), where $\left\lceil \log_{|\lambda_2(\mathbf{S})|=0.6194} 0.0001 \right\rceil = 20$.

Table 29. Paw Graph's PageRank Vector, $\alpha = 0.85$

(a) $\mathbf{x}_0$

a	0.25
b	0.25
c	0.25
d	0.25

(b) $\mathbf{s}_0$

a	0
b	0
c	0
d	0

(c) $\mathbf{s}_1 = \mathbf{x}_0$

a	0.25
b	0.25
c	0.25
d	0.25

(d) $\mathbf{x}_1 = \mathbf{S} \cdot \mathbf{x}_0$

a	0.4625
b	0.2146
c	0.2146
d	0.1083

(e) $\mathbf{s}_2 = \mathbf{x}_1$

a	0.4625
b	0.2146
c	0.2146
d	0.1083

(f) $\mathbf{x}_2 = \mathbf{S} \cdot \mathbf{x}_1$

a	0.3120
b	0.2597
c	0.2597
d	0.1685

(g) $\mathbf{s}_\infty = \mathbf{x}_\infty$

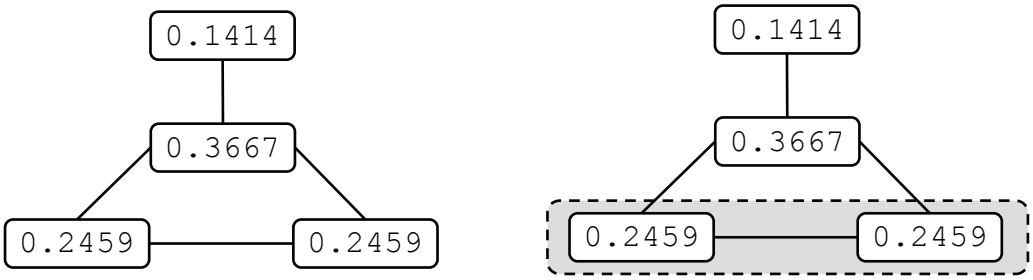
a	0.3667
b	0.2459
c	0.2459
d	0.1414

(h) $\mathbf{x}_\infty = \mathbf{S} \cdot \mathbf{x}_\infty$

a	0.3667
b	0.2459
c	0.2459
d	0.1414

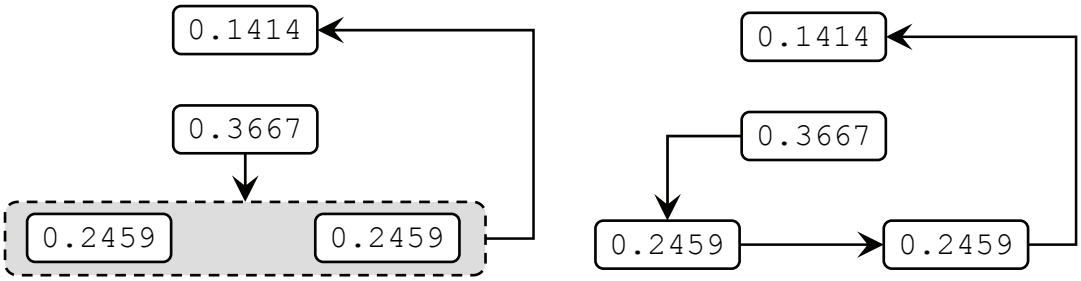
The PageRank vector yielded by the paw graph for $\alpha = 0.85$ corresponds to the weighted graph shown in Figure 36(a). Vertices *b* and *c* have the same PageRank value, 0.2459, which is illustrated in Figure 36(b) using a shaded overlay. The PageRank values of these two vertices preclude obtaining a canonical vertex order, where the most refined vertex order induced by this PageRank vector is illustrated in Figure 37(a).

The paw graph's canonical isomorph produced by *nauty* lists vertex *b* before *c*. Therefore, the tie between their PageRank values, 0.2459, is broken by sorting on their PageRank values, followed by their order in the canonical isomorph. Thus, the canonical isomorph induces the canonical vertex order shown in Figure 37(b). However, as noted in Section 2.3.4.2, determining a graph's canonical isomorph may require exponential time. For such graphs, a non-canonical vertex order, e.g., the order illustrated in Figure 37(a), may be the best that can be obtained.



(a) PageRank Values (b) PageRank Partition

Figure 36. Paw Graph's PageRank Vector, $\alpha = 0.85$



(a) PageRank Order (b) Canonical Order

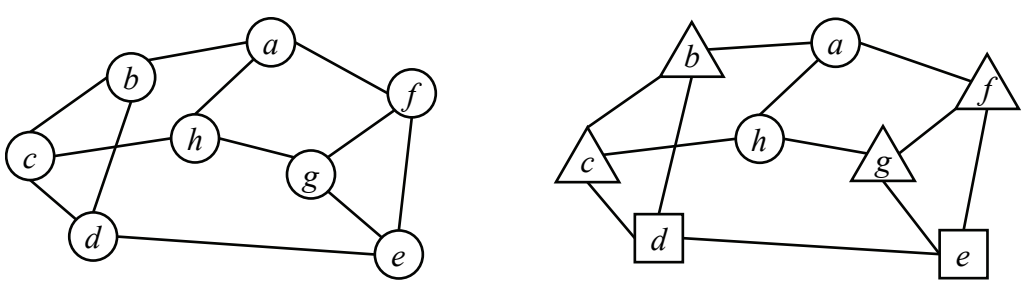
Figure 37. Paw Graph's PageRank Ordering, $\alpha = 0.85$

As described at the end of Section 2.5.1, the PageRank vector is also unique up to graph isomorphism. Thus, a vertex, v_i, that receives the PageRank value, $\mathbf{x}_i$, receives the same PageRank value after applying an arbitrary permutation matrix, **P**, i.e.,

$$\mathbf{P} \cdot \mathbf{A} \cdot \mathbf{P}^T \quad \leftrightarrow \quad \mathbf{P} \cdot \mathbf{S} \cdot \mathbf{P}^T \quad \leftrightarrow \quad \mathbf{P} \cdot \mathbf{x} \,. \tag{27}$$

Thus, two vertices contained in the same block of the orbit partition must yield equal PageRank values. For example, the paw graph's orbit partition is $\left[\{d\},\{b,c\},\{a\}\right]$, where each block's vertices yield the unique PageRank value, $\left[0.1414, 0.2459, 0.3667\right]$, respectively. The paw graph's orbit partition is the same as its coarsest equitable partition, up to a block permutation. For example, applying 1–D Weisfeiler-Lehman stabilization to the paw graph yields the coarsest equitable partition, $\left[\{d\},\{b,c\},\{a\}\right]$.

The orbit partition does not always coincide with the coarsest equitable partition. For example, the cuneane graph yields the coarsest equitable partition and orbit partition illustrated in Figures 38(a) and (b), respectively. Additionally, each vertex yields the same PageRank value, 0.1250. Thus, in this example, applying the coarsest equitable partition suffices to identify which vertices must have equal PageRank values.



(a) Coarsest Equitable Partition $\left[\{a,b,\ldots,h\}\right]$

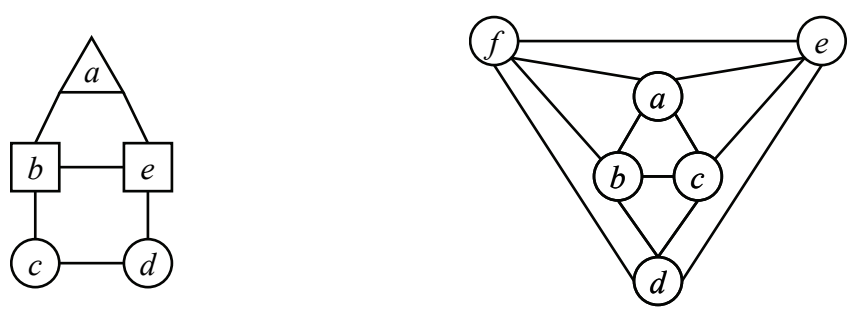
(b) Orbit Partition $\left[\{a,h\},\{d,e\},\{b,c,f,g\}\right]$

Figure 38. Cuneane Graph's Coarsest Equitable and Orbit Partitions

2.6. Observations about Equitable Vertices and PageRank Values

Vertices contained in the same block of the orbit and coarsest equitable partitions yield equal PageRank values. However, finite-precision arithmetic limitations may cause vertices contained in the same block of these partitions to have unequal PageRank values. The algorithms described in Chapters 4 and 5 ensure vertices contained in the same block of these partitions have equal PageRank values. Moreover, two of the methods reduce the time needed to find the PageRank vector if the coarsest equitable partition is non-discrete. Conversely, none of the algorithms improve the PageRank algorithm's performance if the graph's coarsest equitable partition is discrete.

For example, the house graph's coarsest equitable partition, $\left[\{c,d\},\{a\},\{b,e\}\right]$, which is non-discrete, is illustrated in Figure 39(a). The PageRank values yielded by the block's vertices are $\left[0.172, 0.168, 0.244\right]$, respectively. The most non-discrete coarsest equitable partition occurs if all vertices are contained in one block, i.e., the unit partition. Such partitions are yielded by all *k*-regular graphs, in which every vertex has *k* neighbors. For example, the coarsest equitable partition of the 4-regular octahedron graph shown in Figure 39(b) is $\left[\{a,b,c,d,e,f\}\right]$ and each vertex has the PageRank value, $1/6 = 0.1\overline{6}$.

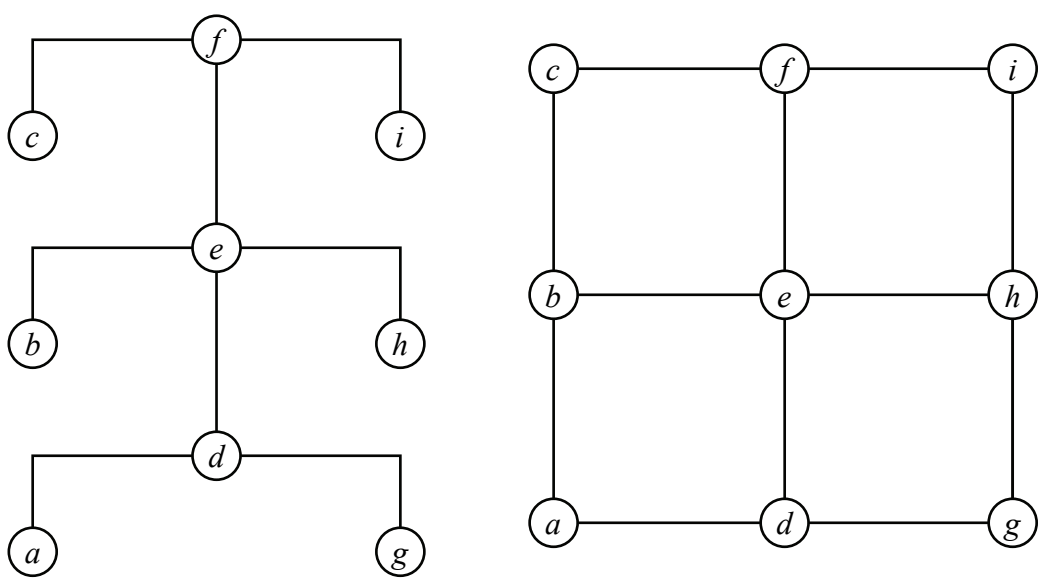


(a) House Graph, $\left[\{c,d\},\{a\},\{b,e\}\right]$ (b) Octahedron Graph, $\left[\{a,b,c,d,e,f\}\right]$

Figure 39. Coarsest Equitable Partitions of the House and Octahedron Graphs

Since the coarsest equitable partition of a *k*-regular graph contains one block, such graphs yields the maximum performance gain with respect to the algorithms described in Chapters 4 and 5. Although *k*-regular graphs are trivial for the PageRank algorithm, since each vertex has the same PageRank value, they have many important uses. For instance, *k*-regular graphs are used to assess applications such as *nauty*, since finding the canonical isomorph of some *k*-regular graphs may cause such applications to need exponential time.

Some more interesting graph families with respect to the PageRank algorithm and the results described in Chapters 3–5 are trees and grid graphs. For instance, many trees and grid graphs often yield a coarsest equitable partition containing blocks composed of multiple vertices. For example, the 9-vertex random tree shown in Figure 40(a) yields the coarsest equitable partition, $\left[\{a,c,g,i\},\{b,h\},\{d,f\},\{e\}\right]$. The 3×3 grid graph shown in Figure 40(b) yields a non-discrete coarsest equitable partition containing three blocks, $\left[\{a,c,g,i\},\{b,d,f,h\},\{e\}\right]$.



(a) 9-Vertex Tree (b) 3×3 Grid Graph

Figure 40. Two Graphs Yielding a Non-Discrete Coarsest Equitable Partition

2.7. Known Results

The PageRank algorithm is one of many methods that use eigenvector centrality to determine relative vertex importance. The HITS algorithm orders responses using the dominant eigenvector of two matrices [LaM06]. Recent social network research relaxes an equitable partition's definition to determine if results similar to those described herein can be obtained on larger graphs [BrL04, Ler05]. The mansion graph has also been used to define an "almost equitable partition" and its relationship to the Laplacian eigenvectors of a graph (cf. Figure 1 and Section 2.4) [CDR07].

The work of Boldi *et al*. is most directly related to the results described herein, since they first showed that vertices contained in the same block of the coarsest equitable partition have equal PageRank values [BLS+06]. The earlier proof uses tools drawn from category theory, the minimum base and its associated fibrations, which correspond to the coarsest equitable partition and its associated blocks, respectively. That proof establishes a Markov chain's limit distribution is constant within fibrations, i.e., the PageRank vector is constant within blocks.

Boldi *et al*. also show, in the proof accompanying their Theorem 9 [BLS+06], that the PageRank vector can be lifted from the quotient matrix using techniques described in Section 2.3.5. They suggest applying this theorem would reduce the time required to find PageRank vectors, but do not define such a method or analyze its performance. However, they do construct an algorithm for finding the coarsest equitable partition that minimizes memory usage. Their last result describes graphs based on European web pages that yield a non-discrete coarsest equitable partition. That result suggests the PageRank algorithm's performance can be improved on at least some web graphs.

2.8. Summary

Section 2.1 introduced an application of the PageRank algorithm to UAV swarms, determining which nodes are the most beneficial for injecting misinformation assumed to be randomly propagated in the network. That section also explored similar applications in social networks, e.g., finding members that facilitate spreading rumor or diseases.

The remainder of Chapter 2 defines tools applied in Chapters 3–5 to improve the PageRank algorithm's performance if two or more nodes yield equal PageRank values. Section 2.2 explores deciding graph isomorphism, where graphs are said to be isomorphs if they define equivalent edges up to a vertex permutation. Section 2.3 defines key vertex partitions commonly used to help decide graph isomorphism, such as equitable partitions.

The coarsest equitable partition is the most refined partition found if every vertex is placed in one block and the only operations used are the sorting and comparison of any adjacent vertex labels, e.g., as done in 1-D Weisfeiler-Lehman stabilization. The quotient graph (matrix) induced by a partition is defined in Section 2.3.5. The eigenvector of a key quotient matrix is used to obtain the most notable results herein, as shown in Chapter 5.

Section 2.4 describes some notable results that apply a graph's eigenvalues and its eigenvectors. Section 2.5 defines the PageRank algorithm that orders vertices by applying a certain eigenvector. The algorithm ensures the eigenvector exists by first perturbing the adjacency matrix to obtain a positive stochastic matrix that defines a Markov chain. The dominant eigenvector of that matrix represents the Markov chain's stationary distribution. The eigenvector can be obtained using the power method, an iterated dot product process. The results described in Chapters 3–5 improve the PageRank algorithm's performance by decreasing the number and size of the dot products computed by the power method.

III. Establishing Equitable Equivalency

3.1. Overview

A practical lower bound on the PageRank algorithm's execution time is developed in Section 3.2. This result is derived by applying two assumptions related to the scaling value, α, and required precision, τ. Combining these assumptions with recent results that determined the upper bound on the PageRank algorithm's complexity [HaK03] yields the lower bound. The existing upper bound and the new practical lower bound are essentially derived by applying a known bound on the number of power method iterations [GoV88], which is based on the two dominant eigenvalues of the PageRank matrix, **S**.

The material in Section 3.3 highlights the similarity between the dot product and 1-D Weisfeiler-Lehman stabilization. In particular, the coarsest equitable partition yielded by applying 1-D Weisfeiler-Lehman stabilization is identical to the partition yielded by a process based on the dot products, as illustrated by the example provided in Section 3.3.1. This method of finding the coarsest equitable partition is listed in Section 3.3.2 and has the same complexity bounds as 1-D Weisfeiler-Lehman stabilization. Its key contribution is to highlight the relationship between the coarsest equitable partition and dot product.

More precisely, the proof constructed in Section 3.4.1 shows vertices contained in the same block of the coarsest equitable partition must yield equal iterated dot products. Such vertices also must have equal PageRank values, as established in Section 3.4.2 and considered further in Section 3.4.3. This relationship between a graph's coarsest equitable partition and its PageRank values was previously and independently shown after applying different techniques [BLS+06]. The relationship's potential impact on the execution time needed to compute the dot product and PageRank vectors is explored in Section 3.4.4.

3.2. Lower Bound on the Expected Number of Power Method Iterations

Before exploring the relationship between the coarsest equitable partition and the PageRank vector, it is useful to determine a lower bound on the number of power method iterations needed to obtain sufficient numerical precision in the PageRank vector, **x**. The practical upper bound on the number of iterations, denoted *t*, is (cf. Section 2.5.2.2)

$$t \le \log_{|\lambda_2(\mathbf{s})|} \tau \le \log_{\alpha} \tau = \frac{\log_b \tau}{\log_b \alpha}. \tag{28}$$

The PageRank algorithm's developers report $\log_2 n$ iterations often suffices, and this behavior has been reported by other researchers [PBM+98, ANT+02]. However, no theoretical derivations about *t*'s lower bound are known. Applying three key assumptions does yield a practical lower bound on the number of power method iterations.

The key assumption is $\alpha \ge 0.5$, where the range, $\alpha \in [0.5, 1.0]$, also includes the default scaling factor, $\alpha = 0.85$. The second assumption is $b = 2$, where the upper bound, $\log_b \tau / \log_b \alpha$, is independent of any base, *b*. The third assumption is $\tau \le 1/n$, the largest tolerance that can potentially yield *n* distinct PageRank values.

Theorem 1 Assuming $\alpha \ge 0.5$ and $b = 2$, the practical lower bound on the number of power method iterations, *t*, to ensure $\tau \le 1/n$ is $\log_2 n$.

Proof Assuming $\alpha \ge 0.5$, $b = 2$, and $\tau \le 1/n$, substitution yields

$$\log_{\alpha} \tau = \frac{\log_b \tau}{\log_b \alpha} = \frac{\log_2 (1/n)}{\log_2 (0.5)} = \frac{\log_2 (1/n)}{-1} = -\log_2 (1/n) = \log_2 n \le t. \quad \blacksquare$$

Thus, combining the existing upper bound and the new practical lower bound, the number of power method iterations, *t*, needed to compute the PageRank vector, **x**, are

$$\log_2 n \le t \le \log_{|\lambda_2(\mathbf{s})|} \tau \le \log_{\alpha} \tau. \tag{29}$$

3.3. Motivating Equitable Dot Products and PageRank Values

The fundamental objective of this section is to motivate the techniques applied in the proofs constructed in Section 3.4. The proof of Theorem 2 shows vertices contained in the same block of a graph's coarsest equitable partition must yield equal dot products. The proof of Theorem 3 shows vertices contained in the same block of a graph's coarsest equitable partition must yield equal PageRank values. Hence, such vertices are equitable with respect to iterated dot products and PageRank values.

There are several equivalent methods of finding the coarsest equitable partition, where each method yields the same coarsest equitable partition up to a block permutation. For instance, three methods of computing the coarsest equitable partition are described in Section 2.3.3. However, neither the method based on the partition's formal definition that is described in Section 2.3.3.1, nor the most efficient method known of determining the partition described in Section 2.3.3.2 implicitly suggest a matrix dot product equivalency. Fortunately, 1-D Weisfeiler-Lehman stabilization, as described in Section 2.3.3.3, yields such an equivalency. Moreover, that method yields a parallel processing implementation of computing the coarsest equitable partition, since rows can be sorted independently.

The link between the coarsest equitable partition and dot product is predicated on observing 1-D Weisfeiler-Lehman stabilization sorts each matrix row and the dot product multiplies each matrix row by a vector. Appropriately substituting prime numbers yields a method that uses a modified dot product to perform 1-D Weisfeiler-Lehman stabilization. The method is described by example in Section 3.3.1 and formally listed in Section 3.3.2. The algorithm's key contribution is to motivate the proofs given in Section 3.4 that show vertices in the same block must have equal dot products and PageRank values.

3.3.1. From Weisfeiler-Lehman Stabilization to Iterated Dot Products

This section contains an example illustrating how to obtain the coarsest equitable partition using dot products. For example, the house graph's coarsest equitable partition is $\left[\{c,d\},\{a\},\{b,e\}\right]$, if computed using 1-D Weisfeiler-Lehman stabilization. Iterated dot products can be modified to yield the same partition.

First, multiplying the house graph's adjacency matrix listed in Table 30(a) with the ones vector listed in Table 30(b) yields the dot product vector listed in Table 30(c). Multiplying the house graph's adjacency matrix with that vector yields the vector listed in Table 30(d), where the entries associated with vertices *b* and *e* equal 7. Closer inspection reveals the corresponding summed intermediate products, $1\cdot 2+0\cdot 3+1\cdot 2+0\cdot 2+1\cdot 3$ and $1\cdot 2+1\cdot 3+0\cdot 2+1\cdot 2+0\cdot 3$, respectively. Sorting the summed intermediate products yields the same sorted intermediate products, $0\cdot 2+0\cdot 3+1\cdot 2+1\cdot 2+1\cdot 3$.

Similarly, multiplying the house graph's adjacency matrix with the vector listed in Table 30(d) yields the vector listed in Table 30(e), where vertices *b* and *e* yield the same sorted products, $0\cdot 5+0\cdot 7+1\cdot 5+1\cdot 6+1\cdot 7=18$, i.e., no further vertex refinement occurs. Thus, the house graph's coarsest equitable partition appears to be $\left[\{c,d\},\{a\},\{b,e\}\right]$.

Table 30. Iterated Dot Products of the House Graph's Adjacency Matrix

(a) $\mathbf{A}$

	a	*b*	*c*	*d*	*e*
a	0	1	0	0	1
b	1	0	1	0	1
c	0	1	0	1	0
d	0	0	1	0	1
e	1	1	0	1	0

(b) $\mathbf{x}_0=\mathbf{1}^{5,1}$

a	1
b	1
c	1
d	1
e	1

(c) $\mathbf{x}_1=\mathbf{A}\cdot\mathbf{x}_0$

a	2
b	3
c	2
d	2
e	3

(d) $\mathbf{x}_2=\mathbf{A}\cdot\mathbf{x}_1$

a	6
b	7
c	5
d	5
e	7

(e) $\mathbf{x}_3=\mathbf{A}\cdot\mathbf{x}_2$

a	14
b	18
c	12
d	12
e	18

The vector shown in Tables 30(d) and (e) coincide with the house graph's coarsest equitable partition, $\left[\{c,d\},\{a\},\{b,e\}\right]$. Vertices *b* and *e* yield 18, where vertices yielding the same dot product are contained in the same block of the coarsest equitable partition. However, it is erroneous to conclude iterated dot products of a graph's adjacency matrix must coincide with the coarsest equitable partition. In fact, a counter-example is obtained in the next step in the example, which is based on prime number substitution.

Before proceeding, it is worth noting the key goal is to show vertices contained in the same block of the coarsest equitable partition yield equal dot products. The example motivates this result by suitably modifying the iterated dot product process to show that the dot product *could* be used, albeit inefficiently, to find the coarsest equitable partition.

The first change substitutes all dot product entries with prime numbers. This step eliminates a key problem in finding the coarsest equitable partition using the dot product, namely, that two sets may yield equal products although each set contains distinct values. For example, the distinct sets, $\{6,6\}$ and $\{4,9\}$, yield the equal product, $6\cdot 6 = 4\cdot 9 = 36$. However, appropriately substituting distinct prime numbers often suffices to distinguish such sets, e.g., applying the substitution, $[4,6,9]\rightarrow[2,3,5]$, yields $3\cdot 3 \neq 2\cdot 5$.

For instance, substituting the '1's vector listed in Table 31(a) with a seed vector of '2's yields the seed vector listed in Table 31(b). Multiplying the house graph's adjacency matrix with the '2's seed vector yields the vector listed in Table 31(c). Replacing the '4's with '2's and '6's with '3's yields the vector shown in Table 31(d). Iterating that process yields the vectors shown in Tables 31(e)–(l), where the stabilization cycle is detected by comparing Tables 31(f)–(h) with Tables 31(j)–(l).

The vector listed in Table 31(f), $[3, 5, 2, 2, 5]$, corresponds with the house graph's coarsest equitable partition, $[\{c,d\},\{a\},\{b,e\}]$. However, the product, $[10, 10, 7, 7, 10]$, listed in Table 31(g) does not also correspond with the house graph's coarsest equitable partition. Thus, applying prime number substitution to each vector fails to yield a process equivalent to performing 1-D Weisfeiler-Lehman stabilization. More precisely, applying prime number to each vector resolves equal product issues, e.g., $6 \cdot 6 = 4 \cdot 9 \rightarrow 3 \cdot 3 \neq 2 \cdot 5$, but does not resolve equal summed prime products, e.g., $2 \cdot 3 + 2 \cdot 7 = 2 \cdot 5 + 2 \cdot 5 = 20$.

Table 31. Iterated Prime Dot Products of the House Graph's Adjacency Matrix

(a) $\mathbf{x}_0 = \mathbf{1}^{5,1}$

a	1
b	1
c	1
d	1
e	1

(b) $\mathbf{x}_1 = \text{getPrimes}(\mathbf{x}_0)$

a	2
b	2
c	2
d	2
e	2

(c) $\mathbf{x}_2 = \mathbf{A} \cdot \mathbf{x}_1$

a	4
b	6
c	4
d	4
e	6

(d) $\mathbf{x}_3 = \text{getPrimes}(\mathbf{x}_2)$

a	2
b	3
c	2
d	2
e	3

(e) $\mathbf{x}_4 = \mathbf{A} \cdot \mathbf{x}_3$

a	6
b	7
c	5
d	5
e	7

(f) $\mathbf{x}_5 = \text{getPrimes}(\mathbf{x}_4)$

a	3
b	5
c	2
d	2
e	5

(g) $\mathbf{x}_6 = \mathbf{A} \cdot \mathbf{x}_5$

a	10
b	10
c	7
d	7
e	10

(h) $\mathbf{x}_7 = \text{getPrimes}(\mathbf{x}_6)$

a	3
b	3
c	2
d	2
e	3

(i) $\mathbf{x}_8 = \mathbf{A} \cdot \mathbf{x}_7$

a	6
b	8
c	5
d	5
e	8

(j) $\mathbf{x}_9 = \text{getPrimes}(\mathbf{x}_8)$

a	3
b	5
c	2
d	2
e	5

(k) $\mathbf{x}_{10} = \mathbf{A} \cdot \mathbf{x}_9$

a	10
b	10
c	7
d	7
e	10

(l) $\mathbf{x}_{11} = \text{getPrimes}(\mathbf{x}_{10})$

a	3
b	3
c	2
d	2
e	3

Entry uniqueness is improved by multiplying rows and columns in the adjacency matrix with the prime number vector yielded after the most recent iteration. This change first constructs the temporary matrix, $\mathbf{T} = \mathbf{D} \cdot \mathbf{A} \cdot \mathbf{D}$, where $\mathbf{D} = \text{diag}(\mathbf{x}_i)$, and substitutes $\mathbf{x}_{i+1} = \mathbf{T} \cdot \mathbf{x}_i$ for $\mathbf{x}_{i+1} = \mathbf{A} \cdot \mathbf{x}_i$. For example, the house graph's initial iteration is shown in Table 32. Subsequently multiplying the house graph's adjacency matrix with the diagonal matrix, **D**, listed in Table 32(d), yields the temporary matrix, **T**, listed in Table 33(a).

The next iteration yields the vector listed in Table 33(c), where the resulting prime numbers precisely correspond to the house graph's coarsest equitable partition yielded by 1-D Weisfeiler-Lehman stabilization, $[\{c,d\},\{a\},\{b,e\}]$. Additional iterations yield the same prime numbers, hence, the dot product iteration process has stabilized.

Table 32. Constructing the First Prime Diagonal Matrix

(a) $\mathbf{x}_0$

a	2
b	2
c	2
d	2
e	2

(b) $\mathbf{x}_1 = \mathbf{A} \cdot \mathbf{x}_0$

a	4
b	6
c	4
d	4
e	6

(c) $\mathbf{x}_2 = \text{getPrimes}(\mathbf{x}_1)$

a	2
b	3
c	2
d	2
e	3

(d) $\mathbf{D} = \text{diag}(\mathbf{x}_2)$

	a	b	c	d	e
a	2	0	0	0	0
b	0	3	0	0	0
c	0	0	2	0	0
d	0	0	0	2	0
e	0	0	0	0	3

Table 33. First Prime Dot Product Iteration

(a) $\mathbf{T} = \mathbf{D} \cdot \mathbf{A} \cdot \mathbf{D}$

	a	b	c	d	e
a	0	6	0	0	6
b	6	0	6	0	9
c	0	6	0	4	0
d	0	0	4	0	6
e	6	9	0	6	0

(b) $\mathbf{x}_2$

a	2
b	3
c	2
d	2
e	3

(c) $\mathbf{x}_3 = \mathbf{Z} \cdot \mathbf{x}_2$

a	84
b	129
c	62
d	62
e	129

(d) $\mathbf{x}_4 = \text{getPrimes}(\mathbf{x}_3)$

a	3
b	5
c	2
d	2
e	5

The last step needed to illustrate vertices contained in the same block yield equal dot products is motivated by the impact of prime numbers on uniqueness. For example, if the vectors, **x**, **y**, and **z**, are composed of prime numbers, the intermediate multiplications contained in the dot products, $\mathbf{x} \cdot \mathbf{y}$ and $\mathbf{x} \cdot \mathbf{z}$, are distinct, but the summed multiplications, i.e., the dot products, may be equal. Various methods can be used to resolve this issue by appropriately replacing the dot product summation, e.g., by sorting, and not summing, the intermediate multiplications contained in each dot product.

For example, the vectors listed in Table 34 yield distinct intermediate products, but happen to have an equal summed value, i.e., $\mathbf{x} \cdot \mathbf{z}^T = 106 = 5 \cdot 2 + 5 \cdot 3 + 5 \cdot 5 + 3 \cdot 7 + 5 \cdot 7$ and $\mathbf{y} \cdot \mathbf{z}^T = 106 = 7 \cdot 2 + 7 \cdot 3 + 3 \cdot 5 + 3 \cdot 7 + 5 \cdot 7$. However, comparing sorted product pairs suffices to distinguish these dot products. For example, the products, $\mathbf{x} \cdot \mathbf{z}^T$ and $\mathbf{y} \cdot \mathbf{z}^T$, yield $[5 \cdot 2, 5 \cdot 3, 5 \cdot 5, 3 \cdot 7, 5 \cdot 7]$ and $[7 \cdot 2, 7 \cdot 3, 3 \cdot 5, 3 \cdot 7, 5 \cdot 7]$, respectively, corresponding to $[10, 15, 25, 21, 35]$ and $[14, 21, 15, 21, 35]$. Finally, sorting the intermediate products in ascending order yields $[10, 15, 21, 25, 35]$ and $[14, 15, 21, 21, 35]$, respectively.

Since the sorted intermediate products are distinct, if **x** and **y** are assumed to be in rows of an arbitrary matrix and **z** is the vector obtained after the last dot product iteration, **x** and **y** cannot be located in the same block of the coarsest equitable partition. Therefore, the dot product can be modified to obtain the same coarsest equitable partition yielded by 1-D Weisfeiler-Lehman stabilization.

Table 34. Two Equal Dot Products, $\mathbf{x} \cdot \mathbf{z}^T = \mathbf{y} \cdot \mathbf{z}^T$

(a) **x**

5	5	5	3	5

(b) **y**

7	7	3	3	5

(c) **z**

2	3	5	7	7

3.3.2. Finding the Coarsest Equitable Partition By Iterated Dot Products

The algorithm listed in Figure 41 determines a graph's coarsest equitable partition by similarly modifying the dot product. Initial values are assigned on lines 1–4, where the entries in the vector, **x**, equal the prime number, '2', and the partition archive vector, **s**, is set equal to '0'. The partition archive vector, **s**, stores the values contained in **x** obtained at the end of the most recent partition stabilization iteration.

The stabilization iteration occurs on lines 6–26, where the loop terminates if the partition is identical after two consecutive iterations. The current vertex partition is saved for future comparison on line 9, i.e., $\mathbf{s} = \mathbf{x}$. The diagonal prime number matrix based on the current partition is constructed on line 11 and applied to the rows and columns of the adjacency matrix, **A**, on line 13. The nested loops on lines 14–21 implement the last step, where the dot product, $\mathbf{x} = \mathbf{A} \cdot \mathbf{x}$, is replaced by the sorting of the summed multiplications defined by each of the *n* dot products. The *n* multiplications defined by the *n* dot products are computed on lines 16–18 and lexicographically sorted on line 20.

Each set of identical rows of sorted multiplications is assigned a unique identifier on line 20. These identifiers are matched to a similarly unique prime number on line 21. The previous and updated partitions are compared on line 6, where if the block identifiers entries are equal, i.e., if $\mathbf{s} = \mathbf{x}$, the vertex partition has stabilized and the main loop can be terminated. This alternative method is equivalent to 1-D Weisfeiler-Lehman stabilization and yields the same upper bound on execution time, $\mathrm{O}\left(n^2 \cdot \log n \cdot \log n^2\right)$, or equivalently, $\mathrm{O}\left(n^2 \cdot \log^2 n\right)$. That upper bound can be reduced to $\mathrm{O}\left(d \cdot n \cdot \log^2 n\right)$ if the graph is stored using adjacency lists, where $d = \max\left(\deg\left(v_i\right)\right), v_i \in V$, as described in Section 2.3.3.3.

```
findCoarsestPartition(A, n)
  # create prime partition vector
  x = 2 · 1^(n,1)

  # create archive partition vector
  s = 0^(n,1)

  # compute the coarsest equitable partition
  while (s ≠ x)
    # archive most recent partition
    s = x

    # construct diagonal matrix
    D = diag(x)
    # construct intermediate prime substitution matrix
    T = D · A · D

    for i from 1 to n
      # compute pair-wise product
      for j from 1 to n
        Z_(i,j) = T_(i,j) · x_j
      end for

      # sort row of pair-wise products
      Z_(i,:) = sort(Z_(i,:))
    end for

    # find unique lexicographically sorted rows
    x = getIdenticalRowIdentifiers(Z)

    # substitute primes for lexicographically unique rows
    x = getUniquePrimes(x)
  end while

  # return equitable partition
  return B = [b_(i=1), b_2, ... b_(max(x))], x_r = j → v_r ∈ b_j
end findCoarsestPartition
```

Figure 41. 1-D Weisfeiler-Lehman Stabilization Using Primes and Dot Products

3.4. Relating Equitable Dot Products and PageRank Values

3.4.1. Equitable Dot Products

Given an arbitrary graph, G, and its associated adjacency matrix, $\mathbf{A}$, the coarsest equitable partition, B, can be computed by applying Weisfeiler-Lehman stabilization. The method listed in Figure 42 is a matrix-based algorithm for performing Weisfeiler-Lehman stabilization (cf. Figure 18 in Section 2.3.3.3).

Theorem 2 Vertices contained in the same block of the coarsest equitable partition have equal iterated dot products, $\mathbf{x}_{t+1} = \mathbf{A} \cdot \mathbf{x}_t$, assuming the initial vector's entries are equal, i.e., $\mathbf{x}_0(i) = \mathbf{x}_0(j), \forall i, j$.

Proof Weisfeiler-Lehman stabilization sorts adjacent labels, whereas the dot product multiplies each row's values by some vector. Replacing the adjacent label sorting step on line 6 with $\mathbf{Z} = \mathbf{A} \cdot \mathbf{x}$ facilitates establishing $v_r, v_s \in b_i \rightarrow \mathbf{Z}_r = \mathbf{Z}_s$, since a contradiction is obtained if the implication is false, i.e., if $v_r, v_s \in b_i \not\rightarrow \mathbf{Z}_r = \mathbf{Z}_s$.

Assume that the dot products, $\mathbf{Z}_r = \mathbf{A}_{r,1} \cdot \mathbf{x}_1 + \mathbf{A}_{r,2} \cdot \mathbf{x}_2 + \cdots + \mathbf{A}_{r,n} \cdot \mathbf{x}_n$ and $\mathbf{Z}_s = \mathbf{A}_{s,1} \cdot \mathbf{x}_1 + \mathbf{A}_{s,2} \cdot \mathbf{x}_2 + \cdots + \mathbf{A}_{s,n} \cdot \mathbf{x}_n$ are not equal, i.e., that $\mathbf{Z}_r \neq \mathbf{Z}_s$. Then, sorting the adjacent labels, $\left[\mathbf{x}_i : \{v_r, v_i\} \in E\right]$ and $\left[\mathbf{x}_j : \{v_s, v_j\} \in E\right]$, should have showed the vertices are contained in different blocks, i.e., that $v_r \in b_i$ and $v_s \in b_j$, $i \neq j$. Otherwise, it must be the case that $\mathbf{Z}_r = \mathbf{Z}_s$. ■

1. findCoarsestPartition$(\mathbf{A}, n)$
2. $\quad \mathbf{x} = \mathbf{1}^{n,1}$
3. $\quad \mathbf{s} = \mathbf{0}^{n,1}$
4. $\quad$ while $(\mathbf{s} \neq \mathbf{x})$
5. $\quad\quad \mathbf{s} = \mathbf{x}$
6. $\quad\quad \mathbf{Z}_{r,1:n} = \left[\ \mathbf{x}_r \ \middle|\ \text{sort}\left(\mathbf{x}_s : \mathbf{A}_{r,s} = 1\right) \ \middle|\ \mathbf{0}^{1,n-\deg(v_r)-1}\ \right]$
7. $\quad\quad \mathbf{x} = \text{getUniqueRowIdentifiers}(\mathbf{Z})$
8. $\quad$ end while
9. $\quad$ return $B = \left[b_{i=1}, b_2, \ldots b_{\max(\mathbf{x})}\right], \mathbf{x}_r = i \rightarrow v_r \in b_i$
10. end findCoarsestPartition

Figure 42. 1-D Weisfeiler-Lehman Stabilization Using Matrices

More explicitly, during each iteration, each row contained in the label matrix, $\mathbf{Z}$, corresponds to the sorted labels of every adjacent vertex, i.e.,

$$\mathbf{Z}_{r,1:n} = \left[\ \mathbf{x}_r \ \middle| \ \text{sort}\left(\mathbf{x}_s : \mathbf{A}_{r,s} = 1\right) \ \middle| \ \mathbf{0}^{1,n-\deg(v_r)-1} \ \right]. \tag{30}$$

Thus, the rth row contains the current label, $\mathbf{x}_r$, given to vertex r, followed by the sorted labels of its adjacent neighbors, $\min\left(\mathbf{x}_{s=1}\right),\ldots,\max\left(\mathbf{x}_{s=n}\right)$, such that $\{v_r, v_s\} \in E$. Finally, the row, $\mathbf{Z}_{r,:}$, is padded with $n - \deg\left(v_r\right) - 1$ zero entries to ensure $\mathbf{Z}$ is a square matrix. At the end of this iteration, $\mathbf{x}_r = \mathbf{x}_s$ if and only if $\mathbf{Z}_{r,i} = \mathbf{Z}_{s,i}, \forall i$.

Replacing (30) with the dot product, i.e., $\mathbf{Z} = \mathbf{A} \cdot \mathbf{x}$, induces the computation of n dot products, where $\mathbf{Z}_r = \mathbf{A}_{r,1} \cdot \mathbf{x}_1 + \mathbf{A}_{r,2} \cdot \mathbf{x}_2 + \cdots + \mathbf{A}_{r,n} \cdot \mathbf{x}_n$. However, $\mathbf{A}$ is a graph's $\{0,1\}$ adjacency matrix, thus, $\mathbf{Z}_r = \sum \mathbf{x}_s : \mathbf{A}_{r,s} = 1$, where $\mathbf{A}_{r,s} = 1 \leftrightarrow \{v_r, v_s\} \in E$. Therefore, the dot product is simply the sum of the vertex labels adjacent to v_r, which for some graphs, suffices to construct the coarsest equitable partition. Thus, vertices contained in the same block of a graph's coarsest equitable partition yield equal iterated dot products, assuming the seed vector's entries are equal. Thus, if $\mathbf{x} = c^{n,1}$, where c is some constant, iterating $\mathbf{Z} = \mathbf{A} \cdot \mathbf{x}$ and $\mathbf{x} = \text{getUniqueRowIdentifiers}\left(\mathbf{Z}\right)$ yields $\mathbf{Z}_r = \mathbf{Z}_s$ if $v_r, v_s \in b_i$.

Vertices contained in different blocks may yield equal dot products, i.e., given two vertices, v_r and v_s, such that $v_r \in b_i \ \wedge \ v_s \in b_j$, it is possible that $\mathbf{Z}_r = \mathbf{Z}_s$. Therefore, the converse is false, i.e., $\mathbf{Z}_r = \mathbf{Z}_s \not\rightarrow v_r, v_s \in b_i$. Applying similar logic shows the inverse is also false, i.e., if $i \neq j$, such that $v_r \in b_i \ \wedge \ v_s \in b_j \not\rightarrow \mathbf{Z}_r \neq \mathbf{Z}_s$. Finally, the contrapositive is true, i.e., $\mathbf{Z}_r \neq \mathbf{Z}_s \rightarrow v_r \in b_i \ \wedge \ v_s \in b_j$, such that $i \neq j$.

3.4.2. Equitable PageRank Values

The relationship between the coarsest equitable partition and the dot product can be exploited to improve the PageRank algorithm's performance. The PageRank algorithm perturbs an adjacency matrix to obtain a strictly positive stochastic matrix. The PageRank algorithm then uses the power method to find the normalized dominant eigenvector of the that perturbed matrix, where the PageRank vector's entries are unique up to isomorphism. This relationship is leveraged in Chapters 4 and 5 to improve the PageRank algorithm's performance on graphs yielding a non-discrete coarsest equitable partition, i.e., partitions containing one or more blocks composed of multiple vertices.

> **Theorem 3** Vertices contained in the same block of the coarsest equitable partition must have equal PageRank values, i.e., $v_r, v_s \in b_i \rightarrow \mathbf{x}_r = \mathbf{x}_s$.
>
> **Proof** Assume the initial PageRank vector is the normalized ones vector, $\mathbf{x} = \mathbf{1}^{n,1}/n$. and the PageRank vector is computed using the power method, which computes a normalized iterated dot product. Applying Theorem 2 suffices to establish vertices contained in the same block of the coarsest equitable partition must have equal PageRank values.
>
> Identical results also hold with respect to the converse, inverse, and contrapositive. For instance, vertices contained in different blocks of the coarsest equitable partition may have the same PageRank value. However, vertices having different PageRank values must be in different blocks. ■

Boldi *et al*. first showed that vertices contained in the same block of the coarsest equitable partition must also have equal PageRank values [BLS+06]. Their proof is based on category theory, namely, the minimum base and its fibrations, which correspond to the coarsest equitable partition and its blocks, respectively. They show that a Markov chain's stationary distribution is constant within each fibration. Similarly, the PageRank value is constant within each block. Their work and Theorem 3 are only sufficient, i.e., no claims are made in either proof about the PageRank values of vertices in different blocks.

3.4.3. Additional Equitable Relationships

Notably, dot product iteration may yield a *less* refined partition than the coarsest equitable partition. Conversely, vertices contained in the same block of a graph's coarsest equitable partition yield equal dot products, assuming the first dot product iteration uses a constant vector, e.g., the all-ones seed vector. Additionally, vertices contained in the same block of the orbit partition yield equal dot product values, since vertices contained in the same orbit are necessarily contained in the same block of the coarsest equitable partition.

An orbit partition may be *more* refined than the coarsest equitable partition, i.e., contain more blocks. Since the coarsest equitable partition may be less refined than the graph's orbit partition, it may reveal more vertices that have equal PageRank values and is similarly more useful for improving the PageRank algorithm's performance. Moreover, the coarsest equitable partition can be obtained in deterministic polynomial time, whereas computing the orbit partition may require exponential time (cf. Sections 2.3.3 and 2.3.4).

For example, the 12-vertex graph illustrated in Figure 43 yields an orbit partition containing three 4-vertex blocks and a coarsest equitable partition containing an 8-vertex block and a 4-vertex block. Hence, more performance gains can be obtained by applying the coarsest equitable partition, since it only contains two blocks. In particular, using the coarsest equitable partition establishes eight vertices have equal PageRank values and the other four vertices also have equal PageRank values, for any scaling value, α.

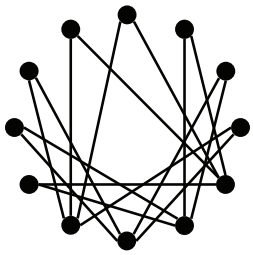


Figure 43. Graph Yielding Different Coarsest Equitable and Orbit Partitions

3.4.4. Complexity Analysis

A matrix is assumed to be a dense, or non-sparse, matrix, unless otherwise stated. This approach simplifies the analysis and suffices to demonstrate the utility of applying a graph's coarsest equitable partition to improve the PageRank algorithm's performance. An arbitrary graph, *G*, contains $n = |V|$ vertices and defines an adjacency matrix, **A**, that contains n^2 entries. *G*'s coarsest equitable partition, *B*, contains $b = k = |B|$ blocks, where $b \le n$, $B = \left[b_1, b_2, \ldots, b_{b=k=|B|}\right]$, and $n = \sum_{i=1}^{k} |b_i| = |V|$.

Given the number of vertices, *n*, finding the coarsest equitable partition requires $\mathrm{O}\left(n^2 \cdot \log n\right)$ time using the method described in Section 2.3.3.2. Furthermore, given the PageRank matrix, **S**, obtained using some arbitrary scaling value, α, the lower bound on using the power method to obtain the PageRank vector, **x**, is $\Omega\left(n^2 \cdot \log n\right)$, a new result derived in Section 3.2. The upper bound on using the power method to obtain an arbitrary precision, τ, in the PageRank vector, **x**, is $\mathrm{O}\left(n^2 \cdot t\right)$, where $t \le \log_{|\lambda_2(\mathbf{S})|} \tau \le \log_{\alpha} \tau$, as was described in Section 2.5.2.2. Thus, the PageRank algorithm, which is essentially a power method variant has a lower and upper bound of $\Omega\left(n^2 \cdot \log n\right)$ and $\mathrm{O}\left(n^2 \cdot t\right)$, respectively.

The coarsest equitable partition can be determined in $\mathrm{O}\left(n^2 \cdot \log n\right)$ time using the algorithm described in Section 2.3.3.2. Thus, a PageRank algorithm variant that applies the coarsest equitable partition increases its execution time bounds by a $n^2 \cdot \log n$ term. Thus, such a PageRank algorithm variant yields the lower and upper bounds, $2 \cdot n^2 \cdot \log n$ and $n^2 \cdot \log n + n^2 \cdot t$, respectively, which are $\Omega\left(n^2 \cdot \log n\right)$ and $\mathrm{O}\left(n^2 \cdot \log n + n^2 \cdot t\right)$.

Therefore, any algorithm that applies the coarsest equitable partition to reduce the execution time of the PageRank algorithm must recoup the cost of initially computing the partition, i.e., the $n^2 \cdot \log n$ term. Such efficiencies can be obtained by observing vertices contained in some arbitrary block, b_i, yield equal PageRank values, where $|b_i| - 1$ of the dot products are being (unnecessarily) computed. For instance, the PageRank algorithm's lower bound, $\Omega\left(n^2 \cdot \log n\right)$, can be written as $\Omega\left(\sum_{i=1}^{b=|B|} |b_i| \cdot n \cdot \log n\right)$, i.e., $n = \sum_{i=1}^{b=|B|} |b_i|$ dot products of length n for $\log n$ iterations. Similarly, the PageRank algorithm's upper bound, $\mathrm{O}\left(n^2 \cdot t\right)$, can be equivalently written as $\mathrm{O}\left(\sum_{i=1}^{b=|B|} |b_i| \cdot n \cdot t\right)$.

Hence, a PageRank algorithm variant that applies the coarsest equitable partition yields the lower bound, $\Omega\left(n^2 \cdot \log n + \sum_{i=1}^{b=|B|} |b_i| \cdot n \cdot \log n\right)$, and similarly, the upper bound, $\mathrm{O}\left(n^2 \cdot \log n + \sum_{i=1}^{b=|B|} |b_i| \cdot n \cdot t\right)$. Thus, eliminating $n^2 \cdot \log n$ or more operations will reduce the time required to obtain the PageRank vector. The ProductRank algorithm described in Section 4.3 eliminates $|b_i| - 1$ dot products from block b_i of a coarsest equitable partition. The QuotientRank algorithm described and analyzed in Chapter 5 uses a quotient matrix to reduce the time needed to obtain the PageRank vector more dramatically.

Finally, decreasing τ increases precision by increasing the executed number of power method iterations. However, increasing precision cannot ensure vertices contained in the same block have equal computed PageRank values. The three algorithms described in Chapters 4 and 5 guarantee each block's vertices have the same PageRank values. The latter two algorithms often compute the PageRank values more efficiently.

IV. Reducing Equitable Differences and Dot Products

4.1. Overview

The relationship of the PageRank vector to the coarsest equitable partition shown in Chapter 3 yields several methods of improving the PageRank algorithm's performance if the graph's coarsest equitable partition is non-discrete, i.e., contains one or more blocks composed of multiple vertices. For instance, the two algorithms described in this chapter leverage that vertices contained in the same block of the coarsest equitable partition have identical iterated dot products up to a permutation of the summed intermediate products.

The first algorithm, AverageRank, replaces the computed PageRank value of each vertex with the average PageRank value of vertices contained in the corresponding block. Thus, vertices contained in the same block will receive the same PageRank value, which eliminates any numerical differences in the computed PageRank values of such vertices. The second algorithm, ProductRank, computes one dot product for each block contained in the coarsest equitable partition during each power method iteration. The ProductRank algorithm guarantees vertices contained in the same block have the same PageRank value and reduces the execution time needed to compute the PageRank vector.

Both algorithms ensure vertices contained in the same block have equal PageRank values if the power method used to determine the PageRank vector is terminated after an arbitrary iteration. The ProductRank algorithm also reduces the time needed to obtain the PageRank vector by only computing certain dot products and thus is more useful than the AverageRank algorithm. Both algorithms are superseded by the QuotientRank algorithm described in Chapter 5, which uses significantly more robust techniques to further reduce the time needed to compute the PageRank vector.

4.2. Eliminating Equitable PageRank Differences

4.2.1. Numerical Differences and Equitable Vertices

Vertices that are contained in the same block of the coarsest equitable partition of an arbitrary graph must have equal PageRank values. However, the PageRank values may differ if the values are computed using finite-precision arithmetic, where such differences induce an invalid vertex ordering. Such differences may occur even after many iterations of the power method have been performed to obtain the PageRank vector. The following example is based on the tree shown in Figure 44 that yields the adjacency matrix listed in Table 35 and the coarsest equitable partition, $\left[\{a,c,g,i\},\{b,h\},\{d,f\},\{e\}\right]$.

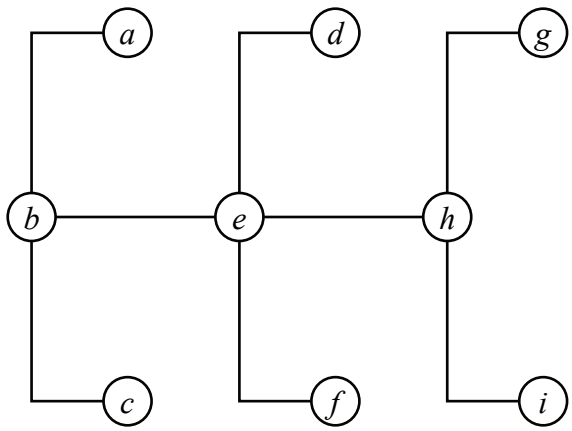


Figure 44. 9-Vertex Tree: A Graph Yielding a 4-Block Equitable Partition

Table 35. A 9-Vertex Tree's Adjacency Matrix

	a	b	c	d	e	f	g	h	i
a	0	1	0	0	0	0	0	0	0
b	1	0	1	0	1	0	0	0	0
c	0	1	0	0	0	0	0	0	0
d	0	0	0	0	1	0	0	0	0
e	0	1	0	1	0	1	0	1	0
f	0	0	0	0	1	0	0	0	0
g	0	0	0	0	0	0	0	1	0
h	0	0	0	0	1	0	1	0	1
i	0	0	0	0	0	0	0	1	0

This tree's PageRank vector for $\alpha = 0.85$ is listed in Table 36(a), where sorting its entries in descending order in Table 36(b). Since the tree's coarsest equitable partition is $\left[\{a,c,g,i\},\{b,h\},\{d,f\},\{e\}\right]$, vertices $\{a,c,g,i\}$ should have the same PageRank value. The PageRank vector listed in Table 36(a) initially confirms this expected behavior, since vertices $\{a,c,g,i\}$ have an equal PageRank value up to the fourth decimal place, 0.0682. However, close inspection shows the PageRank values of vertices *g* and *i* are $1.388 \cdot 10^{-17}$ greater than the PageRank values of vertices *a* and *c*. The difference occurs in the last bit of the double-precision format specified by the IEEE 754 standard [ISB85]. The value, $1.388 \cdot 10^{-17}$, is small in magnitude and induces a false order on vertices $\{a,c,g,i\}$, where vertices *g* and *i* erroneously receive a higher PageRank value than vertices *a* and *c*.

In this example, rounding to four decimal places ensures vertices contained in the same block have equal PageRank values. However, simply rounding the PageRank vector cannot guarantee such vertices have equal PageRank values. In particular, the computed PageRank values of vertices contained in each block may be above or below the rounding point and thus may receive different rounded PageRank values.

Table 36. A 9-Vertex Tree's PageRank Vector, $\alpha = 0.85$

(a) PageRank Vector

a	0.0682
b	0.1818
c	0.0682
d	0.0659
e	0.2318
f	0.0659
g	0.0682
h	0.1818
i	0.0682

(b) Sorted PageRank Vector

0.2318	*e*
0.1818	*b*
0.1818	*h*
0.0682	*g*
0.0682	*i*
0.0682	*a*
0.0682	*c*
0.0659	*d*
0.0659	*f*

4.2.2. AverageRank: An Algorithm for Eliminating Equitable Differences

Fortunately, differences in the computed PageRank values of vertices contained in the same block of the coarsest equitable partition can be easily eliminated. The algorithm listed in Figure 45 sets the PageRank value of each block's vertices to the average of their computed values without internally modifying the PageRank algorithm (cf. Section 2.5). First, the PageRank vector is determined on line 3 using the PageRank algorithm.

The coarsest equitable partition on line 5 is obtained using any method described in Section 2.3.3, e.g., 1-D Weisfeiler-Lehman stabilization. PageRank values differences among vertices in the same block are eliminated on lines 6–11 by setting their PageRank values to their corresponding average PageRank value. The median could be used in lieu of the average, but determining the median PageRank value requires $\Theta(n \cdot \log n)$ time, whereas determining the average PageRank value only requires $\Theta(n)$ time.

```
1.  findEquitablePageRank(A, n, α, τ)
2.    # compute the PageRank vector
3.    x ← getPageRank(A, n, α, τ)

4.    # compute the coarsest equitable partition
5.    B ← findCoarsestPartition(A, n)

6.    # assign average PageRank value of equitable vertices
7.    foreach block, b_i ∈ B
8.      foreach vertex, v_j ∈ b_i
9.        x(j) ← mean(x(b_i))
10.     end foreach
11.   end foreach

12.   return x
13. end EquitablePageRank
```

Figure 45. AverageRank: An Algorithm for Ensuring Equitable PageRank Values

4.2.3. Complexity Analysis

Applying the power method to an arbitrary PageRank matrix, **S**, requires at least $\Omega\left(n^2 \cdot \log n\right)$ time, a new result described in Section 3.2. The coarsest equitable partition can be found in $\mathrm{O}\left(n^2 \cdot \log n\right)$ time by applying the algorithm described in Section 2.3.3.2. Thus, the power method's lower bound equals the upper bound on computing the coarsest equitable partition. Obtaining the average PageRank value of the vertices contained in the same block requires $\Theta\left(n = \sum_{i=1}^{|B|} |b_i|\right)$ time and is dominated by $n^2 \cdot \log n.$ Thus, the lower bound on the AverageRank algorithm is $2 \cdot n^2 \cdot \log n + n,$ which is $\Omega\left(n^2 \cdot \log n\right).$

The upper bound on applying the power method to **S** is $\mathrm{O}\left(n^2 \cdot t\right),$ where t denotes the number of power method iterations and $t \le \log_\alpha \tau$ (cf. Section 2.5.2.1). Therefore, the power method's upper bound exceeds the complexity of computing the coarsest equitable partition, since $\log_\alpha \tau \ge \log n$ for most practical values of n, τ, and α. Thus, by combining the upper bounds of applying the power method, $\mathrm{O}\left(n^2 \cdot t\right),$ finding the coarsest equitable partition, $\mathrm{O}\left(n^2 \cdot \log n\right),$ and obtaining mean PageRank values, $\Theta\left(n\right),$ the upper bound on the AverageRank algorithm is $n^2 \cdot \log n + n^2 \cdot t + n,$ which is $\mathrm{O}\left(n^2 \cdot \log n + n^2 \cdot t\right).$

The AverageRank algorithm ensures each block's vertices receive equal PageRank values, but costs $n^2 \cdot \log n$ more time than the PageRank algorithm. The ProductRank and QuotientRank algorithms described in Section 4.3 and Chapter 5, respectively, provide this same assurance and decrease the PageRank algorithm's execution time if the graph's coarsest equitable partition is non-discrete.

4.3. Eliminating Equitable PageRank Dot Products

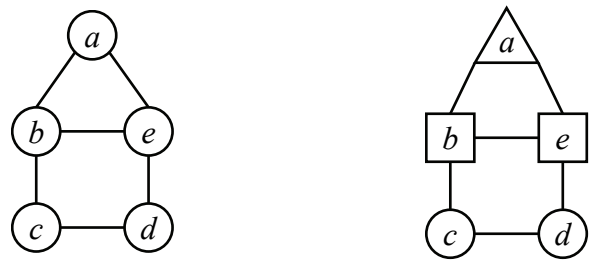
4.3.1. Excess Dot Products and Equitable Vertices

The AverageRank algorithm described in Section 4.2.2 eliminates differences in PageRank value between vertices contained in the same block of the coarsest equitable partition and *increases* the PageRank algorithm's execution time. Conversely, the method described in Section 4.3.2, the ProductRank algorithm ensures vertices in the same block have equal PageRank values and *decreases* the PageRank algorithm's execution time.

For example, the house graph shown in Figure 46(a) yields the coarsest equitable partition, $\left[\{c,d\},\{a\},\{b,e\}\right]$, shown in Figure 46(b). Since vertices *b* and *e* are contained in the same block, it suffices to obtain one of their associated dot products and assign its value to their associated PageRank vector entries during each power method iteration.

Multiplying the house graph's PageRank matrix listed in Table 38 with the initial PageRank vector, the normalized all-ones vector shown in Table 39(a), yields the updated PageRank vector shown in Table 39(b). More significantly, the intermediate product sums yielded by vertices *b* and *e* are $0.455 \cdot 0.2 + 0.030 \cdot 0.2 + 0.455 \cdot 0.2 + 0.030 \cdot 0.2 + 0.313 \cdot 0.2$ and $0.455 \cdot 0.2 + 0.313 \cdot 0.2 + 0.030 \cdot 0.2 + 0.455 \cdot 0.2 + 0.030 \cdot 0.2,$ respectively.

Inspection reveals ordering the intermediate products yields the same sorted list, $0.030 \cdot 0.2 + 0.030 \cdot 0.2 + 0.313 \cdot 0.2 + 0.455 \cdot 0.2 + 0.455 \cdot 0.2.$ Vertices *c* and *d* also yield the same intermediate products up to isomorphism, since vertices *c* and *d* are also contained in the same block of the coarsest equitable partition. Hence, a second dot product can be eliminated during each power method iteration. The ProductRank algorithm described in Section 4.3.2 applies this technique to decrease the PageRank algorithm's execution time. The potential performance improvement is formally determined in Section 4.3.3.



(a) House Graph (b) Coarsest Equitable Partition

Figure 46. House Graph and Its 3-Block Coarsest Equitable Partition

Table 37. House Graph's Adjacency and Degree Matrix

(a) **A**

	a	*b*	*c*	*d*	*e*
a	0	1	0	0	1
b	1	0	1	0	1
c	0	1	0	1	0
d	0	0	1	0	1
e	1	1	0	1	0

(b) **D**

	a	*b*	*c*	*d*	*e*
a	2	0	0	0	0
b	0	3	0	0	0
c	0	0	2	0	0
d	0	0	0	2	0
e	0	0	0	0	3

Table 38. House Graph's Stochastic PageRank Matrix, **S**, $\alpha = 0.85$

	a	*b*	*c*	*d*	*e*
a	0.030	0.313	0.030	0.030	0.313
b	0.455	0.030	0.455	0.030	0.313
c	0.030	0.313	0.030	0.455	0.030
d	0.030	0.030	0.455	0.030	0.313
e	0.455	0.313	0.030	0.455	0.030

Table 39. Initial PageRank Power Method Iterations of the House Graph

(a) $\mathbf{x}_0 = \frac{\mathbf{1}^{5,1}}{5}$

a	0.2
b	0.2
c	0.2
d	0.2
e	0.2

(b) $\mathbf{x}_1 = \frac{\mathbf{S}\cdot\mathbf{x}_0}{\sum(\mathbf{S}\cdot\mathbf{x}_0)}$

a	0.143
b	0.257
c	0.172
d	0.172
e	0.257

(c) $\mathbf{x}_2 = \frac{\mathbf{S}\cdot\mathbf{x}_1}{\sum(\mathbf{S}\cdot\mathbf{x}_1)}$

a	0.175
b	0.237
c	0.176
d	0.176
e	0.237

(d) $\mathbf{x}_3 = \frac{\mathbf{S}\cdot\mathbf{x}_2}{\sum(\mathbf{S}\cdot\mathbf{x}_2)}$

a	0.164
b	0.246
c	0.172
d	0.172
e	0.246

4.3.2. ProductRank: An Algorithm for Eliminating Equitable Dot Products

In the AverageRank algorithm described in Section 4.2.2, the PageRank value of each block's vertices was set to their average PageRank value. That method exploits the relationship established in Section 3.4.2, that such vertices yield identical dot products up to a permutation and thus have equal PageRank values. In that algorithm, the average is obtained after the power method is terminated, i.e., no internal modifications are made to the PageRank algorithm. The iterated dot product relationship is further exploited in this section to develop an algorithm that reduces the time needed to compute the PageRank vector by computing one dot product for each block during each power method iteration. However, to achieve that result, the PageRank algorithm must be internally modified.

The unaltered PageRank algorithm listed in Figure 33 is repeated in Figure 47(a). Since lines 1–10 are unchanged in the ProductRank algorithm listed in Figure 47(b), they are not listed. The first change is to obtain the coarsest equitable partition on line 11 using the *findCoarsestPartition* function. That function can implement any of the three methods described in Section 2.3.3 for computing the graph's coarsest equitable partition.

The next change is to replace the matrix multiplication step in line 16, $\mathbf{x} \leftarrow \mathbf{S} \cdot \mathbf{x}$, of the original PageRank algorithm with lines 17–24 in the ProductRank algorithm. The lines are similar to lines 6–11 in Figure 45, which assign the average PageRank value of each block's vertices to those same vertices. In this algorithm, the dot product of the first vertex in every block is computed on lines 19 and 20. The resulting updated PageRank value is subsequently assigned to each vertex in that block using the loop on lines 21–23. The ProductRank algorithm's remaining lines, 25–31, are identical to lines 17–23 in the original PageRank algorithm listed in Figure 47(a).

```
getPageRank(A, n, α, τ)
  # perturb adjacency matrix
  d_i ← Σ_{i=1}^{n} A_{:,i}
  D ← diag(d)
  # apply PageRank perturbation
  S ← α · A · D^{-1} + (1 − α)/n

  # initialize vectors and counter
  x ← 1^{n,1}/n
  s ← 0^{n,1}
  z ← 0

  # iterate power method
  while(‖s − x‖_2 > τ)
    # save PageRank vector
    s ← x

    # update PageRank vector
    x ← S · x

    # normalize PageRank vector
    x ← x/Σ x

    # increment loop counter
    z ← z + 1
  end while

  return x
end PageRank
```

(a) PageRank Algorithm

```
getPageRank(A, n, α, τ)
  ⋮
  # get coarsest equitable partition
  B = findCoarsestPartition(A, n)

  # iterate power method
  while(‖s − x‖_2 > τ)
    # save PageRank vector
    s ← x

    # update PageRank vector
    foreach block, b_i ∈ B
      v ← b_i(1)
      x ← S_{:,v} · x

      foreach vertex, v ∈ b_i
        x_v ← x
      end foreach
    end foreach

    # normalize PageRank vector
    x ← x/Σ x

    # increment loop counter
    z ← z + 1
  end while

  return x
end PageRank
```

(b) ProductRank Algorithm

Figure 47. ProductRank: An Algorithm for Eliminating Equitable Dot Products

4.3.3. Complexity Analysis

The AverageRank algorithm described in Section 4.2.2 ensures vertices contained in the same block have equal PageRank values. The ProductRank algorithm described in Section 4.3.2 similarly ensures vertices contained in the same block have equal PageRank values. However, the ProductRank algorithm reduces the number of operations needed to compute the PageRank vector if a graph's coarsest equitable partition is non-discrete. The performance gain is obtained by only computing one dot product for every block during each power method iteration. This method succeeds since vertices contained in the same block yield equal dot products up to a permutation of their intermediate multiplications. Therefore, the ProductRank algorithm reduces the time needed to compute the PageRank vector if the coarsest equitable partition is non-discrete. Thus, the ProductRank algorithm listed in Section 4.3.2 supersedes the AverageRank algorithm listed in Section 4.2.2.

For example, it was shown in Section 4.3.1 that applying the PageRank algorithm to the house graph requires five dot products to be computed during each power method iteration, which requires a total of 25 multiplications and 20 additions. The house graph's coarsest equitable partition contains three blocks, where two blocks contain two vertices, thus, two dot products can be eliminated from each block in each power method iteration.

Each iteration performed by the AverageRank algorithm computes 5 dot products, which requires 25 multiplications and 20 additions. Conversely, each iteration performed in the AverageRank algorithm only computes 3 dot products, requiring 15 multiplications and 12 additions. Thus, if three iterations are executed, ensuring three bits of precision, the AverageRank algorithm performs $(25+20)\cdot 3 = 135$ operations, but the ProductRank algorithm only performs $(15+12)\cdot 3 = 81$ operations.

If $\alpha \geq 0.5$ and $\tau \leq 1/n$, the PageRank algorithm needs at least $\Omega\left(n^2 \cdot \log n\right)$ time, as shown in Section 3.2. Obtaining the coarsest equitable partition requires $\Theta\left(n^2 \cdot \log n\right)$ time, as noted in Section 2.3.3.2. Hence, the ProductRank algorithm obtains the coarsest equitable partition in $\Omega\left(n^2 \cdot \log n\right)$ time and applies the power method in $\Theta\left(n^2 \cdot \log n\right)$ time, yielding an overall lower bound of $\Omega\left(n^2 \cdot \log n\right)$.

The PageRank algorithm's upper bound is $\mathrm{O}\left(n^2 \cdot t\right)$, where $t \leq \log_\alpha \tau$ denotes the maximum number of required power method iterations, as described in Section 2.5.2. The ProductRank algorithm computes the graph's coarsest equitable partition in $\Theta\left(n^2 \cdot \log n\right)$ time and applies the power method in $\mathrm{O}\left(n^2 \cdot t\right)$ time, yielding an overall upper bound of $\mathrm{O}\left(n^2 \cdot \log n + n^2 \cdot t\right)$.

Sharper bounds can be derived by accounting for the dot products eliminated by applying the coarsest equitable partition. If a block contains *s* vertices, $s-1$ dot products, or $(s-1)\cdot n$ multiplications and $(s-1)\cdot(n-1)$ additions are saved in each power method iteration. If *t* iterations are performed, $(s-1)\cdot(2\cdot n-1)\cdot t$ operations are eliminated.

A coarsest equitable partition, *B*, contains $b = |B|$ blocks, where $n = \sum_{i=1}^{|B|} |b_i|$. Only one dot product is computed for each block, therefore, the algorithm yields a lower bound of $\Omega\left(n^2 \cdot \log n + b \cdot n \cdot \log n\right)$, where $n^2 \cdot \log n$ operations are used to compute the coarsest equitable partition and $b \cdot n \cdot \log n$ operations are used to obtain *b* dot products of length *n* for at least $\log n$ iterations. Similarly, the upper bound is $\mathrm{O}\left(n^2 \cdot \log n + b \cdot n \cdot t\right)$.

For example, the 4×4 grid graph shown in Figure 48 yields the coarsest equitable partition listed in Table 40. The partition contains 3 blocks, hence, only 3 dot products are computed during every power method iteration. Eliminating 13 of 16 dot products saves $13 \cdot (16+15) = 403$ floating-point operations. Computing the remaining three dot products only requires $3 \cdot (16+15) = 93$ floating-point operations.

Obtaining the coarsest equitable partition requires as many as $16^2 \cdot \log_2 16 = 1024$ operations. Since $\lceil 1024/403 \rceil = 3,$ at least four power method iterations must be executed before the PageRank algorithm's execution time is reduced. However, since $\log_2 n = 4,$ at least that many iterations are performed to obtain the PageRank vector. Thus, if $\alpha \geq 0.5$ and $\tau \leq 1/n \leq 1/16 \leq 0.0625,$ the Product Rank algorithm shown in Figure 47(b) needs less time than the PageRank algorithm to compute this grid graph's PageRank vector.

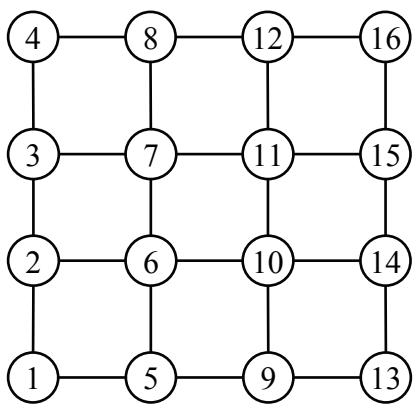


Figure 48. 4×4 Grid: A Graph Yielding a 3-Block Equitable Partition

Table 40. 4×4 Grid Graph's 3-Block Coarsest Equitable Partition

i	Block $b_i \in B$
1	$\{1, 4, 13, 16\}$
2	$\{2, 3, 5, 8, 9, 12, 14, 15\}$
3	$\{6, 7, 10, 11\}$

4.3.4. Algorithm Applicability

The decision to use the ProductRank algorithm listed in Figure 47(b) or the more efficient QuotientRank algorithm developed in Chapter 5 to obtain the PageRank vector hinges on several factors, where this section prefaces the more in-depth analysis provided in Section 5.7. The key factors are the number of blocks, *b*, in a graph's coarsest equitable partition, *B*, PageRank scaling value, α, and required precision in the PageRank vector, τ. To simplify the analysis, it is assumed $\alpha \geq 0.5$ and $\tau \leq 1/n$, where $n = |V|$, the number of vertices in the input graph, *G*.

Loosely stated, if *b* is sufficiently less than *n*, applying either method dramatically reduces the time needed to obtain the PageRank vector. However, if *b* is sufficiently large with respect to *n*, where $b \leq n$, applying the ProductRank or QuotientRank algorithm can increase the time needed to obtain a PageRank vector by a factor of two, i.e., $2 \cdot n^2 \cdot \log n$. The worst case occurs if the coarsest equitable partition is discrete, i.e., if $b = n$, since the time needed to obtain the coarsest equitable partition equals the lower bound of obtaining the PageRank vector, $n^2 \cdot \log n$. However, the PageRank algorithm's upper bound can be significantly reduced if the graph's coarsest equitable partition is discrete, i.e., if $b < n$.

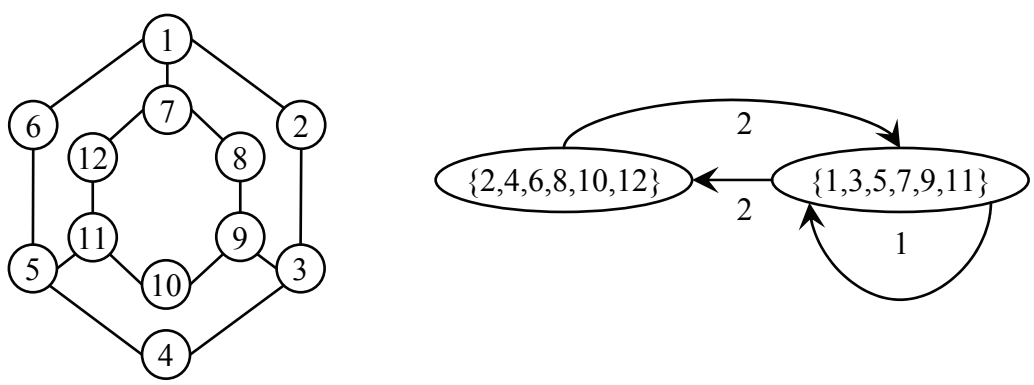
Unfortunately, it is impossible to assess *a priori* if the graph yields a non-discrete coarsest equitable partition containing a sufficiently small number of blocks, *b*. Assuming *b* is sufficiently small with respect to *n*, the PageRank vector is obtained more efficiently using either the ProductRank algorithm listed in Section 4.3.2 or QuotientRank algorithm constructed in Chapter 5. Although both algorithms are more efficient than the PageRank algorithm, the ProductRank algorithm is easier to implement, whereas the QuotientRank algorithm more dramatically reduces the time needed to compute the PageRank vector.

V. Lifting PageRank Values

5.1. Overview

A more dramatic decrease in the PageRank algorithm's execution time is obtained using the quotient matrix induced by the graph's coarsest equitable partition, as shown in Section 5.2. An example that applies the quotient matrix to obtain the PageRank vector is constructed in Section 5.3. The corresponding algorithm is described in Section 5.4 and its analysis is contained in Section 5.5. The utility of the quotient graph was also obtained using different tools by Boldi *et al*. [BLS+06]. However, they did not develop a method similar to the QuotientRank algorithm described and analyzed in Sections 5.4 and 5.5.

For example, the coarsest equitable partition of the graph shown in Figure 49(a) is $\left[\{2,4,6,8,10,12\},\{1,3,5,7,9,11\}\right]$ and induces the quotient graph shown in Figure 49(b). Every vertex in each block is linked to two vertices in the other block, hence, the pair of 2-edges, and odd-labeled vertices are linked to one odd-labeled vertex, hence, the 1-loop. As will be shown in this chapter, the PageRank vector can be obtained more efficiently by lifting the dominant eigenvector of a 2×2 quotient matrix (cf. Section 2.3.5), instead of computing the dominant eigenvector of the original graph's 12×12 PageRank matrix.



(a) Pseudo-Benzene (b) Quotient Graph

Figure 49. Pseudo-Benzene: A Graph Yielding a 2-Block Equitable Partition [StT99]

5.2. Quotient Computations

As described in Section 2.3.5, given some arbitrary matrix, **M**, and some similarly arbitrary partition, *B*, the entries in the quotient matrix, **Q**, induced by *B* correspond to the average row sums in **M** with respect to *B*. Formally, **Q** is obtained by computing [Hae95]

$$\begin{aligned}\mathbf{Q} &= \left(\mathbf{B}^T \cdot \mathbf{B}\right)^{-1} \cdot \mathbf{B}^T \cdot \mathbf{M} \cdot \mathbf{B} \\ &= \mathbf{B}^{\dagger} \cdot \mathbf{M} \cdot \mathbf{B},\end{aligned} \tag{31}$$

where **B** is the characteristic block matrix whose $n = |V|$ rows correspond to the vertices contained in V and $b = |B|$ columns correspond to the blocks contained in *B*, respectively.

Given an arbitrary graph, *G*, and its associated adjacency matrix, **A**, it is critical to construct the appropriate quotient matrix, **Q**. For instance, an incorrect approach is to first compute the quotient matrix based on **A**, where

$$\begin{aligned}\mathbf{Q} &= \left(\mathbf{B}^T \cdot \mathbf{B}\right)^{-1} \cdot \mathbf{B}^T \cdot \mathbf{A} \cdot \mathbf{B} \\ &= \mathbf{B}^{\dagger} \cdot \mathbf{A} \cdot \mathbf{B},\end{aligned} \tag{32}$$

and subsequently apply the PageRank perturbation, where

$$\mathbf{S}_{\mathbf{Q}} = \alpha \cdot \mathbf{Q} \cdot \mathbf{D}_{\mathbf{Q}}^{-1} + (1-\alpha)/n. \tag{33}$$

However, applying the PageRank algorithm to $\mathbf{S}_{\mathbf{Q}}$ yields the correct PageRank vector if $\alpha = 1$. Given an adjacency matrix, **A**, the correct approach for any value of α, as shown in this section, is to apply the traditional PageRank perturbation to **A**, where

$$\mathbf{S}_{\mathbf{A}} = \alpha \cdot \mathbf{A} \cdot \mathbf{D}_{\mathbf{A}}^{-1} + (1-\alpha)/n. \tag{34}$$

The correct quotient matrix, **Q**, subsequently can be obtained by computing

$$\begin{aligned}\mathbf{Q} &= \left(\mathbf{B}^T \cdot \mathbf{B}\right)^{-1} \cdot \mathbf{B}^T \cdot \mathbf{S}_{\mathbf{A}} \cdot \mathbf{B} \\ &= \mathbf{B}^{\dagger} \cdot \mathbf{S}_{\mathbf{A}} \cdot \mathbf{B}.\end{aligned} \tag{35}$$

To show the PageRank vector, **x**, associated with a PageRank matrix, $\mathbf{S} = \mathbf{S}_{\mathbf{A}}$, can be lifted from a quotient matrix, **Q**, using (35), three key results must be obtained. First, it must be established **Q** and **S** yield the same dominant eigenvalue, one. Second, it must be shown a unique eigenvector, **r**, is associated with **Q**'s dominant eigenvalue, one. Finally, it must be shown that **S**'s PageRank vector, **x**, can be lifted from **Q**'s eigenvector, **r**.

The quotient matrix, **Q**, yielded by applying (35) is often not stochastic. However, **Q**'s row sums equal one or more of **S**'s corresponding row sums. Moreover, **Q**'s diagonal entries, as well as many of **Q**'s non-diagonal entries, may equal zero, i.e., **Q** is often not primitive or irreducible. Therefore, **Q** does not satisfy the conditions of the Perron or the Perron-Frobenius theorems, hence, they cannot be directly applied to ensure that **Q** yields a unique eigenvector associated with the dominant eigenvalue, one. Instead, more robust machinery, namely, the interlacing and lifting properties described in Section 2.3.5 must be used to establish the eigen decomposition relationships between a PageRank matrix, **S**, and the quotient matrix, **Q**, induced by **S**'s coarsest equitable partition, *B*.

Theorem 4 The dominant eigenvalue of an equitable quotient matrix, **Q**, of a positive stochastic matrix, **S**, equals **S**'s dominant eigenvalue, one.

Proof Given the quotient matrix, **Q**, induced by an arbitrary partition, *B*, of an arbitrary matrix, **M**, the eigenvalues of **M** and **Q** are interlaced, such that $\lambda_{n-m+i}(\mathbf{M}) \le \lambda_i(\mathbf{Q}) \le \lambda_i(\mathbf{M}), |\lambda_i| \le |\lambda_{i+1}|$. If *B* is equitable, e.g., if *B* is the coarsest equitable partition of $\mathbf{M} = \mathbf{S}$, **Q**'s eigenvalues are some subset of **S**'s eigenvalues. If **S** is the weighted adjacency matrix of some strongly connected graph, *G*, **Q** and **S** must have the same dominant eigenvalues, i.e., $\lambda_1(\mathbf{Q}) = \lambda_1(\mathbf{S})$ [God93, CRS97].

A stochastic positive matrix, e.g., a PageRank matrix, **S**, yields the dominant eigenvalue, one. **Q** may not be stochastic. However, applying the Perron and Perron-Frobenius theorems to the PageRank matrix, **S**, and the interlacing property to an equitable quotient matrix, **Q**, shows that **Q** also yields the dominant eigenvalue, one, i.e., $\lambda_1(\mathbf{Q}) = \lambda_1(\mathbf{S}) = 1$. ■

The next step establishes **Q** has a unique eigenvector, **r**, whose entries are a subset of the entries in **S**'s dominant eigenvector, **x**, where **x** can be obtained more efficiently by simply lifting it from **Q**'s dominant eigenvector, **r**.

> **Theorem 5** The entries contained in the dominant eigenvector, r, yielded by the quotient matrix, **Q**, defined by the coarsest equitable partition, *B*, of a positive stochastic matrix, **S**, are a subset of the entries contained in **S**'s dominant eigenvector, **x**. Each entry in **x** can be obtained by simply lifting it from **Q**'s dominant eigenvector, *r*.
>
> **Proof** A coarsest equitable partition, *B*, is equitable, therefore, the lifting identity, $\mathbf{S}\cdot\mathbf{x}=\mathbf{S}\cdot\mathbf{B}\cdot\mathbf{r}=\mathbf{B}\cdot\mathbf{Q}\cdot\mathbf{r}=\mathbf{B}\cdot\mathbf{r}=\mathbf{x}$, ensures that each entry in **Q**'s dominant eigenvector, **r**, also equals some entry in **S**'s PageRank vector, **x**.
>
> Applying the Perron-Frobenius theorem ensures elements in **x** are positive and unique, thus, **r**'s elements are positive and unique. The lifting identity, $\mathbf{x}=\mathbf{B}\cdot\mathbf{r}$, maps entries in **Q**'s eigenvector, **r**, to **S**'s eigenvector, **x**. Each of **B**'s *n* non-zero entries equal one, thus, lifting is simply an exercise in memory copying, where $\mathbf{x}=\mathbf{B}\cdot\mathbf{r}$ and $\mathbf{r}=\mathbf{B}^{\dagger}\cdot\mathbf{x}=\left(\mathbf{B}^{T}\cdot\mathbf{B}\right)^{-1}\cdot\mathbf{B}^{T}\cdot\mathbf{x}$. ■

Thus, given some arbitrary graph, *G*, and its adjacency matrix, **A**, the PageRank vector can be obtained by normalizing the dominant eigenvector of the PageRank matrix, $\mathbf{S}=\alpha\cdot\mathbf{A}\cdot\mathbf{D}^{-1}+\left(1-\alpha\right)/n$. *G*'s PageRank vector, **x**, can be constructed more efficiently by lifting it from **Q**'s PageRank vector, **r**. The quotient matrix, **Q**, is yielded by applying the block matrix, **B**, defined by the partition, *B*, to the PageRank matrix, **S**, where

$$\mathbf{Q}=\mathbf{B}^{\dagger}\cdot\mathbf{S}\cdot\mathbf{B}=\left(\mathbf{B}^{T}\cdot\mathbf{B}\right)^{-1}\cdot\mathbf{B}^{T}\cdot\mathbf{S}\cdot\mathbf{B}. \tag{36}$$

Assuming **Q**'s dominant eigenvector, **r**, is available, e.g., by applying the power method, **S**'s dominant eigenvector, **x**, is obtained by simply lifting, or copying, it from **r**, where

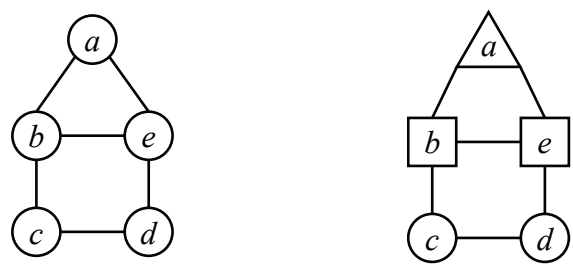
$$\mathbf{x}=\mathbf{B}\cdot\mathbf{r}. \tag{37}$$

Finally, that vector is normalized by its sum, yielding the PageRank vector, **x**, where

$$\mathbf{x}=\mathbf{x}/\sum\mathbf{x} \quad\rightarrow\quad \sum\mathbf{x}=1. \tag{38}$$

5.3. Lifting the House Graph's PageRank Vector

The example in this section illustrates how the PageRank vector can be efficiently lifted from the dominant eigenvector of the quotient matrix. The house graph depicted in Figure 50(a) yields the coarsest equitable partition, $B = \left[\{c,d\},\{a\},\{b,e\}\right]$, illustrated in Figure 50(b). Its associated block matrix, **B**, is listed in Table 41, where '1's reflect B's vertex block membership. The intermediate product, $\mathbf{N} = \mathbf{B}^T \cdot \mathbf{B}$, is listed in Table 42(a), where a diagonal entry equals its corresponding column sum from **B**. Reciprocating all of the diagonal entries yields its matrix inverse, $\mathbf{N}^{-1}$, as shown in Table 42(b).



(a) House Graph (b) Coarsest Equitable Partition

Figure 50. House Graph's 3-Block Coarsest Equitable Partition

Table 41. Characteristic Block Matrix, **B**

	$\{c, d\}$	$\{a\}$	$\{b, e\}$
a	0	1	0
b	0	0	1
c	1	0	0
d	1	0	0
e	0	0	1

Table 42. Characteristic Block Matrix Products

(a) $\mathbf{N} = \mathbf{B}^T \cdot \mathbf{B},\ \mathbf{N}_{i,:} = \sum \mathbf{B}_{i,:}$

	$\{c, d\}$	$\{a\}$	$\{b, e\}$
$\{c, d\}$	2	0	0
$\{a\}$	0	1	0
$\{b, e\}$	0	0	2

(b) $\mathbf{N}^{-1} = \left(\mathbf{B}^T \cdot \mathbf{B}\right)^{-1},\ \mathbf{N}^{-1}_{i,i} = 1/\mathbf{N}_{i,i}$

	$\{c, d\}$	$\{a\}$	$\{b, e\}$
$\{c, d\}$	1/2	0	0
$\{a\}$	0	1	0
$\{b, e\}$	0	0	1/2

The house graph's adjacency matrix is listed in Table 43. Given the scaling factor, $\alpha = 0.85$, the house graph yields the PageRank matrix shown in Table 44. Subsequently applying the quotient matrix identity,

$$\mathbf{Q} = \mathbf{B}^{\dagger} \cdot \mathbf{S} \cdot \mathbf{B} = \left(\mathbf{B}^{T} \cdot \mathbf{B}\right)^{-1} \cdot \mathbf{B}^{T} \cdot \mathbf{S} \cdot \mathbf{B}, \tag{39}$$

yields the quotient matrix, **Q**, listed in Table 45. As this example illustrates, a stochastic PageRank matrix, **S**, may yield a non-stochastic quotient matrix, **Q**, i.e., none of the rows or columns in **Q** sum to one, although the row sums remain constant.

Table 43. House Graph's Adjacency Matrix, **A**

	a	b	c	d	e
a	0	1	0	0	1
b	1	0	1	0	1
c	0	1	0	1	0
d	0	0	1	0	1
e	1	1	0	1	0

Table 44. House Graph's PageRank Matrix, **S**, $\alpha = 0.85$

	a	b	c	d	e	
a	0.0300	0.3133	0.0300	0.0300	0.3133	0.7167
b	0.4550	0.0300	0.4550	0.0300	0.3133	1.2833
c	0.0300	0.3133	0.0300	0.4550	0.0300	0.8583
d	0.0300	0.0300	0.4550	0.0300	0.3133	0.8583
e	0.4550	0.3133	0.0300	0.4550	0.0300	1.2833
	1					$\sum$

Table 45. Quotient Matrix, **Q**, of the House Graph's PageRank Matrix, **S**, $\alpha = 0.85$

	$\{c, d\}$	$\{a\}$	$\{b, e\}$	
$\{c, d\}$	0.4850	0.0300	0.3433	0.8583
$\{a\}$	0.0600	0.0300	0.6267	0.7167
$\{b, e\}$	0.4850	0.4550	0.3433	1.2833
	1.0300	0.5150	1.3133	$\sum$

As described in Section 2.3.5, the eigenvalues of the quotient matrix, **Q**, interlace the eigenvalues of the PageRank matrix, **S**. Moreover, since a coarsest equitable partition, *B*, is equitable, **Q**'s eigenvalues are a subset of **S**'s eigenvalues, as reflected in Table 46. The dominant eigenvectors of the PageRank matrix, **S**, and quotient matrix, **Q**, are listed in Tables 47(a) and (b), respectively. The PageRank vector of **S** is assumed to be obtained by multiplying a 5×5 matrix with a 5×1 vector during each power method iteration. The PageRank vector of the quotient matrix, **Q**, is similarly assumed to be obtained by simply multiplying a 3×3 matrix with a 3×1 vector during each power method iteration.

However, **S**'s PageRank vector, **x**, can be obtained by simply lifting it from **Q**'s dominant eigenvector, **x**. This process multiplies **Q**'s eigenvector, **r**, with **B**, as reflected in Table 47(c). Normalizing that lifted eigenvector yields the PageRank vector, **x**, listed in Table 47(d), which, as required, equals the PageRank vector listed in Table 47(a).

Table 46. Eigenvalues of the PageRank and Quotient Matrices

(a) $\text{sort}\left(\left|\lambda_i(\mathbf{S})\right|\right)$

1.0000
-0.7083
-0.4250
0.2833
0.0000

(b) $\text{sort}\left(\left|\lambda_i(\mathbf{Q})\right|\right)$

1.0000
-0.4250
0.2833

Table 47. Dominant Eigenvectors of the PageRank and Quotient Matrices

(a) **x**

a	0.1681
b	0.2437
c	0.1723
d	0.1723
e	0.2437
$\sum$	1.0000

(b) **r**

$\{c, d\}$	0.2949
$\{a\}$	0.2878
$\{b, e\}$	0.4173
$\sum$	1.0000

(c) $\mathbf{x} = \mathbf{B} \cdot \mathbf{r}$

a	0.2878
b	0.4173
c	0.2949
d	0.2949
e	0.4173
$\sum$	1.7122

(d) $\mathbf{x} = \mathbf{x} / \sum \mathbf{x}$

a	0.1681
b	0.2437
c	0.1723
d	0.1723
e	0.2437
$\sum$	1.0000

5.4. QuotientRank: An Algorithm for Lifting PageRank Vectors

The algorithm listed in Figure 51 obtains the dominant eigenvector of the quotient matrix induced by the coarsest equitable partition and lifts this eigenvector to obtain the PageRank vector of the input matrix. The standard PageRank matrix, **S**, is constructed with respect to the adjacency matrix, **A**, on lines 1–5. The coarsest equitable partition is determined on lines 6–7 by applying one of the methods described in Section 2.3.3. The characteristic block matrix, **B**, is computed on lines 9–11, and subsequently applied to the PageRank matrix, **S**, to obtain the quotient matrix, **Q**, on line 12. The power method is applied to **Q** on lines 13–27 to obtain its dominant eigenvector. Any vector norm could be applied on line 24, where the sum norm is the most efficient to obtain. The final step is to lift **Q**'s dominant eigenvector, **r**, on line 29, where normalizing the lifted vector, **x**, with respect to its sum yields the PageRank vector of the input matrix, **A**, on line 30.

It is assumed that an arbitrary adjacency matrix, **A**, is dense. However, the block matrix, **B**, should be a sparse matrix. As indicated on lines 10 and 11, **B** has a lone '1' on each row that reflects vertex membership in a coarsest equitable partition's blocks. Thus, one of the intermediate products, $\mathbf{N} = \mathbf{B}^T \cdot \mathbf{B}$, computed on line 12 is a diagonal matrix, such that a diagonal entry of **N** corresponds to the number of vertices in some block, i.e., $\mathbf{N}_{i,i} = |B_i| = \sum \mathbf{B}_{:,i}$, Hence, its inverse, $\mathbf{N}^{-1} = \left(\mathbf{B}^T \cdot \mathbf{B}\right)^{-1}$, is obtained by reciprocating each diagonal entry, i.e., $\mathbf{N}_{i,i}^{-1} = 1/\mathbf{N}_{i,i}$. Therefore, line 12 can be written as $\mathbf{Q} \leftarrow \mathbf{N}^{-1} \cdot \mathbf{B}^T \cdot \mathbf{S} \cdot \mathbf{B}$, where $\mathbf{N}^{-1}$ is obtained by reciprocating **B**'s column sums. The lifting step performed on line 29, $\mathbf{x} = \mathbf{B} \cdot \mathbf{r}$, is equivalent to simply copying eigenvalues in **r** to their corresponding positions in the PageRank vector, **x**.

```
getPageRank(A, n, α, τ)
  # perturb adjacency matrix
  d_i ← Σ_{i=1}^{n} A_{:,i}
  D ← diag(d)
  S ← α · A · D^{-1} + (1 − α)/n

  # get coarsest equitable partition
  B ← findCoarsestPartition(A, n)
  # construct block and quotient matrices
  b ← |B|
  B ← 0^{n,b}
  B_{i,j} ← 1, ∀i, j : v_i ∈ B_j
  Q ← (B^T · B)^{-1} · B^T · S · B ← B^† · S · B

  # initialize vectors & iteration counter
  r ← 1^{b,1}/b
  s ← 0^{b,1}
  z ← 0
  # iterate power method
  while(‖s − r‖_2 > τ)
    # save eigenvector
    s ← r
    # update eigenvector
    r ← Q · r
    # normalize eigenvector
    r ← r/Σ r
    # increment loop counter
    z ← z + 1
  end while

  # lift and normalize PageRank vector
  x ← B · r
  x ← x/Σ x

  return x
end PageRank
```

Figure 51. QuotientRank: An Algorithm for Lifting PageRank Vectors

5.5. Complexity Analysis

Applying the PageRank perturbation on lines 2–5 is $\Theta\left(n^2\right)$, since all entries must be scaled and shifted. Computing the coarsest equitable partition, *B*, on lines 6–7 requires $\Theta\left(n^2 \cdot \log n\right)$ time, as described in Section 2.3.3.2. Constructing the block matrix, **B**, on lines 9–11 from the coarsest equitable partition, *B*, requires $\Theta\left(n\right)$ time, where *n* ones are placed at **B**'s rows and column corresponding to the vertex block membership in *B*. Since the $n \times b$ matrix, **B**, only contains *n* non-zero entries, it is stored as a sparse matrix, which only consumes $\Theta\left(n\right)$ space.

The quotient matrix construction, $\mathbf{Q} = \left(\mathbf{B}^T \cdot \mathbf{B}\right)^{-1} \cdot \mathbf{B}^T \cdot \mathbf{S} \cdot \mathbf{B}$, performed on line 12 can be decomposed into two steps, $\mathbf{Q} = \mathbf{N}^{-1} \cdot \mathbf{Z}$, where $\mathbf{N}^{-1} = \left(\mathbf{B}^T \cdot \mathbf{B}\right)^{-1}$ and $\mathbf{Z} = \mathbf{B}^T \cdot \mathbf{S} \cdot \mathbf{B}$. The first step, $\mathbf{N}^{-1} = \left(\mathbf{B}^T \cdot \mathbf{B}\right)^{-1}$, yields a diagonal matrix whose diagonal entries equal the reciprocal of the number of vertices contained in the corresponding block in *B*. Thus, the diagonal entries in $\mathbf{N}^{-1}$ equal **B**'s reciprocated column sums. Moreover, since $\mathbf{N} = \mathbf{B}^T \cdot \mathbf{B}$ is a diagonal matrix, its inverse, $\mathbf{N}^{-1}$, can be computed in $\Theta\left(n\right)$ time.

The second step, $\mathbf{Z} = \mathbf{B}^T \cdot \mathbf{S} \cdot \mathbf{B}$, multiplies a $b \times n$ matrix, $\mathbf{B}^T$, by an $n \times n$ matrix, **S**, yielding the $b \times n$ product, $\mathbf{B}^T \cdot \mathbf{S}$. This $b \times n$ matrix is multiplied by a $n \times b$ matrix, **B**, yielding a $b \times b$ matrix, $\mathbf{Z} = \mathbf{B}^T \cdot \mathbf{S} \cdot \mathbf{B}$. These multiplications are $\mathrm{O}\left(b \cdot n^2\right)$ and $\mathrm{O}\left(b^2 \cdot n\right)$, respectively, where $b \le n$. Thus, if *B* is a discrete partition, which is the worst case, these bounds become $\mathrm{O}\left(n^3\right)$. Since **B** contains *n* non-zero entries, storing **B** as a sparse matrix reduces the bounds to $\mathrm{O}\left(n^2\right)$ and $\mathrm{O}\left(b \cdot n\right)$, respectively, yielding $\mathrm{O}\left(n^2\right)$ if $b = n$.

The same bounds can also be derived using the tools described in Section 2.3.5.2. Given an arbitrary vertex partition, B, $\mathbf{Q}_{i,j} = \sum \mathbf{S}_{r,k}$, where $1 \le i, j \le b$, $1 \le k \le n$, $v_r \in b_i$, and $v_k \in b_j$. The complexity of this summation is $\mathrm{O}\left(b^2 \cdot n\right)$, since each entry in the $b \times b$ matrix, $\mathbf{Q}$, may consume as many as n summations. If $b = n$, which is the worst possible case, the best upper bound that can be obtained initially appears to be $\mathrm{O}\left(n^3\right)$.

However, since B is assumed to be the coarsest equitable partition, which is an equitable partition, the summation can be computed more efficiently. In particular, each vertex contained in block b_i has an equal number of neighbors with respect to block b_j, where $1 \le i, j \le b$. Therefore, the number of neighbors with respect to block b_j must only be determined for a single vertex in block b_i, i.e., $z_{i,j} = \left|N(u) \cap b_j\right| = \left|N(v) \cap b_j\right|$, where $1 \le i, j \le b$, $u\ v \in b_i$, and $N(u)$ denotes u's neighborhood (cf. Sections 2.3.2 and 2.3.3). Thus, $\mathbf{Q}_{i,j} = z_{i,j} \cdot \mathbf{S}_{u,w}$, for some arbitrary vertex, $u \in b_i$, and some arbitrary neighbor of u, $w \in b_j$. This approach yields the same bound on building $\mathbf{Q}$, $\mathrm{O}\left(n^2\right)$ (cf. Section 2.3.5.3).

The most interesting analysis step occurs when assessing the complexity of using the power method on lines 13–27 to determine the dominant eigenvector of the quotient matrix, $\mathbf{Q}$. As described in Sections 2.5.2.2 and 3.2, given some precision, τ, and scaling factor, $\alpha > 0.5$, using the power method to obtain the PageRank vector of the PageRank matrix, $\mathbf{S}$, yields the lower and upper bounds of $\Omega\left(n^2 \cdot \log n\right)$ and $\mathrm{O}\left(n^2 \cdot t\right)$, respectively. The value of t is bounded by $t \le \log_{|\lambda_2(\mathbf{S})|} \tau \le \log_\alpha \tau$, where $\lambda_2(\mathbf{S})$ denotes the eigenvalue with the second largest magnitude and $\left|\lambda_2(\mathbf{S})\right| \le \alpha$.

The power method is used to obtain the dominant eigenvector, **r**, of the quotient matrix, **Q**, where **r** is lifted to obtain the PageRank vector, **x**, of the PageRank matrix, **S**. Since **S** and **Q** are $n \times n$ and $b \times b$ matrices, respectively, where $b \le n$, substitution into the bounds $\Omega\left(n^2 \cdot \log n\right)$ and $\mathrm{O}\left(n^2 \cdot t\right)$ yields $\Omega\left(b^2 \cdot \log b\right)$ and $\mathrm{O}\left(b^2 \cdot t\right)$, respectively.

Theorem 6 The practical lower bound on the execution time needed to obtain the dominant eigenvector of $\mathbf{Q}^{b,b}$ is $\log b$.

Proof The practical lower bound on the number of iterations was based on assuming $\tau \le 1/n$. Since vertices contained in a given block have equal PageRank value, the maximum precision increases to $\tau \le 1/b$. Substitution yields the new practical minimum number of iterations, $b = \log_2 1/b$. ■

The upper bound on the number of power method iterations, *r*, needed to compute **Q**'s PageRank vector, **r**, may be less than the bound on the number of iterations needed to compute **S**'s PageRank vector, **x**, since $r \le \log_{|\lambda_2(\mathbf{Q})|} \tau \le \log_{|\lambda_2(\mathbf{S})|} \tau \le \log_\alpha \tau$.

Theorem 7 The upper bound on the number of power method iterations, *r*, required to obtain the dominant eigenvector, **x**, of an equitable quotient matrix, **Q**, is bounded by the number of iterations, *t*, needed to obtain the PageRank vector of the PageRank matrix, **S**.

Proof The number of iterations, *r*, required to determine **x** is bounded by $\log_{\lambda_2(\mathbf{Q})} \tau$ [GoV88]. Interlacing ensures $\lambda_2\left(\mathbf{Q}\right) = \lambda_i\left(\mathbf{S}\right) \le \lambda_2\left(\mathbf{S}\right)$ for some *i*, therefore, $r \le \log_{|\lambda_2(\mathbf{Q})|} \tau \le t \le \log_{|\lambda_2(\mathbf{S})|} \tau \le \log_\alpha \tau$. ■

If $\left|\lambda_2\left(\mathbf{Q}\right)\right| < \left|\lambda_2\left(\mathbf{S}\right)\right|$, the upper bound on the number of power method iterations may differ with respect to **S** and **Q**, where $r = \log_{|\lambda_2(\mathbf{Q})|} \tau$, $t = \log_{|\lambda_2(\mathbf{S})|} \tau \le \log_\alpha \tau$, and $r \le t$. In the worst case, if the coarsest equitable partition, *B*, is a discrete partition, i.e., if none of *B*'s blocks are composed of multiple vertices, $b = n \rightarrow \lambda_2\left(\mathbf{Q}\right) = \lambda_2\left(\mathbf{S}\right) \rightarrow r = t$. The converse, is not true, since for certain graphs, $b < n$, yet $\lambda_2\left(\mathbf{Q}\right) = \lambda_2\left(\mathbf{S}\right)$, thus, $r = t$.

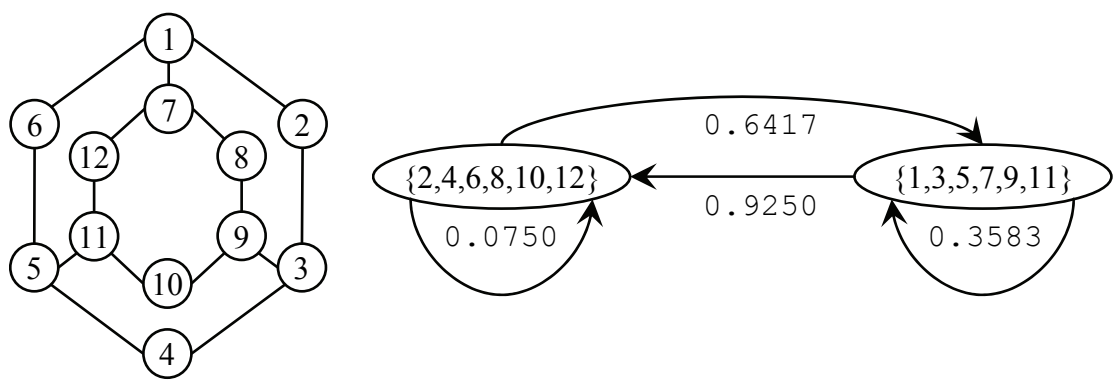
Thus, applying power method iteration to the quotient matrix, **Q**, of the PageRank matrix, **S**, reduces the lower and upper bounds on the complexity of the power method in two ways. The matrix size reduction, from $\mathbf{S}^{n,n}$ to $\mathbf{Q}^{b,b}$, reduced the bounds on the power method from $\Omega\left(n^2 \cdot \log n\right)$ and $\mathrm{O}\left(n^2 \cdot t\right)$ to $\Omega\left(b^2 \cdot \log b\right)$ and $\mathrm{O}\left(b^2 \cdot t\right)$, respectively, where the performance scales in proportion to n^2/b^2. The upper bound on the number of power method iterations, *t*, is replaced by *r*, yielding $\mathrm{O}\left(b^2 \cdot r\right)$, where *r* depends on **Q**'s, not **S**'s, second dominant eigenvalue, i.e., $r \le \log_{|\lambda_2(\mathbf{Q})|} \tau \le \log_{|\lambda_2(\mathbf{S})|} \tau \le \log_{\alpha} \tau$. If **B** is stored as a sparse matrix, the lifting step is $\Theta\left(n\right)$, since **B** contains *n* non-zero entries.

Thus, the lower bound of the QuotientRank algorithm described in Section 5.4 is based on computing the coarsest equitable partition and applying the power method to **Q**, which are $\Omega\left(n^2 \cdot \log n\right)$ and $\Omega\left(b^2 \cdot \log b\right)$, respectively. Hence, its overall lower bound is $\Omega\left(n^2 \cdot \log n + b^2 \cdot \log b\right)$. The corresponding upper bound on the QuotientRank algorithm is $\mathrm{O}\left(n^2 \cdot \log n + b^2 \cdot r\right)$, where $b \le n$, $r \le t$, $r \le \log_{|\lambda_2(\mathbf{Q})|} \tau$, and $t \le \log_{|\lambda_2(\mathbf{S})|} \tau \le \log_{\alpha} \tau$.

If $b \ll n$, a QuotientRank algorithm implementation that uses an efficient method of computing the coarsest equitable partition outperforms the PageRank algorithm, which is $\Omega\left(n^2 \cdot \log n\right)$ and $\mathrm{O}\left(n^2 \cdot t\right)$. The gains are proportional to $\left(n^2 \cdot t\right)/\left(b^2 \cdot r\right)$, where $b \le n$, $r \le \log_{|\lambda_2(\mathbf{Q})|} \tau$, $t \le \log_{|\lambda_2(\mathbf{S})|} \tau \le \log_{\alpha} \tau$, and $r \le t$. The worst case again occurs if the graph's equitable partition, *B*, is discrete, where if $b = n$, the PageRank algorithm's lower bound is doubled by the cost to find *B*, for a total of $2 \cdot n^2 \cdot \log n$ operations. The QuotientRank algorithm is more efficient, however, if the coarsest equitable partition is non-discrete.

5.6. A QuotientRank Example

This example illustrates the potential performance gain obtained by applying the QuotientRank algorithm. The pseudo-benzene graph shown in Figure 52(a) has a coarsest equitable partition of two blocks, $\left[\{2,4,6,8,10,12\},\{1,3,5,7,9,11\}\right]$. The graph's 12×12 adjacency and PageRank matrices are not listed. The 2×2 quotient matrix induced by its coarsest equitable partition and PageRank matrix is shown in Table 48 and illustrated in Figure 52(b). Finding its coarsest equitable partition requires as many as $n^2\cdot\log_2 n$ (517) operations. Constructing the PageRank matrix, **S**, which is used in both the QuotientRank and PageRank algorithms, requires n^2 (144) operations. Storing a 12×2 block matrix, **B**, as a sparse matrix only requires n (12) operations. Constructing a 2×2 quotient matrix, $\mathbf{Q}=\left(\mathbf{B}^T\cdot\mathbf{B}\right)^{-1}\cdot\mathbf{B}^T\cdot\mathbf{S}\cdot\mathbf{B}$, requires $b^2\cdot n$ (48) operations.



(a) Pseudo-Benzene Graph (b) PageRank-Induced Quotient Graph

Figure 52. Pseudo-Benzene Graph and Its PageRank-Induced Quotient Graph

Table 48. A 2×2 PageRank-Induced Quotient Matrix, **Q**

		destination		
		{2,4,6,8,10,12}	{1,3,5,7,9,11}	
source	{2,4,6,8,10,12}	0.0750	0.6417	0.7167
	{1,3,5,7,9,11}	0.9250	0.3583	1.2833
		1		$\sum$

A PageRank vector computed to 52 bits of precision, $\tau = 2^{-52}$, or approximately 15 decimal digits, correspond to IEEE 754 double-precision values [ISB85]. Applying the power method to the PageRank matrix, **S**, and the induced quotient matrix, **Q**, requires $n^2 \cdot t, t \le \log_{|\lambda_2(\mathbf{S})|} \tau$ and $b^2 \cdot r, r \le \log_{|\lambda_2(\mathbf{Q})|} \tau$, time, respectively. The eigenvalues of **S** and **Q** reveal that $|\lambda_2(\mathbf{S})| = 0.85$ and $|\lambda_2(\mathbf{Q})| = 0.5667$, i.e., the largest number of power method iterations that must be performed with respect to **S** or **Q** are $t = \lceil \log_{0.85} 2^{-52} \rceil = 222$ and $r = \lceil \log_{0.5667} 2^{-52} \rceil = 64$, respectively. Thus, in the worst case, applying the power method to **S** and **Q** costs $12^2 \cdot 222 = 31{,}968$ and $2^2 \cdot 64 = 256$ operations, respectively.

The dominant eigenvector, **r**, of **Q** is shown in Table 49(a). Lifting **S**'s dominant eigenvector, **x**, from **r**, results in the vector shown in Table 49(b). Normalizing the vector by its sum yields **S**'s PageRank vector listed in Table 49(c). These lifting and normalizing steps require $3 \cdot n$ (36) operations, 12 to compute $\mathbf{B} \cdot \mathbf{r}$ and 24 to compute $\mathbf{x} / \sum \mathbf{x}$.

Table 49. Lifting the PageRank Vector from the Dominant Eigenvector

(a) **r**

1	0.4096
2	0.5904
$\sum$	1.0000

(b) $\mathbf{x} = \mathbf{B} \cdot \mathbf{r}$

1	0.5904
2	0.4096
3	0.5904
4	0.4096
5	0.5904
6	0.4096
7	0.5904
8	0.4096
9	0.5904
10	0.4096
11	0.5904
12	0.4096
$\sum$	6.0000

(c) $\mathbf{x} = \mathbf{x} / \sum \mathbf{x}$

1	0.0984
2	0.0683
3	0.0984
4	0.0683
5	0.0984
6	0.0683
7	0.0984
8	0.0683
9	0.0984
10	0.0683
11	0.0984
12	0.0683
$\sum$	1.0000

Summing each algorithm's number of operations yields Table 50, which confirms using the QuotientRank algorithm is more efficient than using the PageRank algorithm to obtain the PageRank vector of the graph shown in Figure 52. Although some overhead is required to build the quotient matrix and lift the PageRank vector, the execution time is dramatically reduced in proportion to $32112/1109 \approx 29$. Executing each algorithm shows the number of observed power method iterations for **S** and **Q** are 60 and 62, respectively, as shown in Table 51. Although the PageRank algorithm executes two fewer iterations, it still needs more execution time than the QuotientRank algorithm, where $8784/1101 \approx 8$.

Table 50. Theoretical Iterations: $n = 12,\ b = 2,\ \tau = 2^{-52} \quad \rightarrow \quad t = 222,\ r = 64$

(a) PageRank Algorithm

Task	Time	Total
Construct **S**	n^2	144
Power Method	$n^2 \cdot t$	31968
	$\sum$	32112

(b) QuotientRank Algorithm

Task	Time	Total
Compute B	$n^2 \cdot \log_2 n$	517
Construct **S**	n^2	144
Construct **B**	n	12
Construct **Q**	n^2	144
Power Method	$b^2 \cdot r$	256
Lift & Normalize **x**	$3 \cdot n$	36
	$\sum$	1109

Table 51. Observed Iterations: $n = 12,\ b = 2,\ \tau = 2^{-52} \quad \rightarrow \quad t = 60,\ r = 62$

(a) PageRank Algorithm

Task	Time	Total
Construct **S**	n^2	144
Power Method	$n^2 \cdot t$	8640
	$\sum$	8784

(b) QuotientRank Algorithm

Task	Time	Total
Compute B	$n^2 \cdot \log_2 n$	517
Construct **S**	n^2	144
Construct **B**	n	12
Construct **Q**	n^2	144
Power Method	$b^2 \cdot r$	248
Lift & Normalize **x**	$3 \cdot n$	36
	$\sum$	1101

5.7. QuotientRank Applicability

Section 4.3.4 informally considered when the algorithm described in Section 4.3.2 or QuotientRank algorithm described in Section 5.4 should be used to compute a graph's PageRank vector instead of applying the PageRank algorithm. The material contained in this section provides a more robust analysis with respect to the QuotientRank algorithm, which is the most significant new algorithm described herein. If dense matrices are used, with the notable exception of the block matrix, **B**, which is stored as a sparse matrix, the key factors in assessing if the PageRank algorithm or QuotientRank algorithm is the more efficient method of computing the PageRank vector, **x**, of an arbitrary graph, *G*, are:

- $n = |V|$, number of vertices contained in *G*
- $b = |B|$, number of blocks contained in *G*'s coarsest equitable partition, **B**
- $\alpha, \alpha \geq 0.5$, scaling factor used to construct the PageRank matrix, **S**
- $\tau, \tau \leq 1/n$, numerical precision of the PageRank vector, **x**
- $t, t \leq \log_{|\lambda_2(\mathbf{S})|} \tau \leq \log_{\alpha} \tau$, maximum iterations with respect to **S**
- $r, r \leq \log_{|\lambda_2(\mathbf{Q})|} \tau$ maximum iterations with respect to quotient matrix, **Q**

The number of blocks, *b*, contained in the graph's coarsest equitable partition, *B*, is bounded by the number of vertices, *n*, in *G*, i.e., $b \leq n$. Furthermore, as established in Section 5.5, the maximum number of required power method iterations with respect to **S** equals or exceeds the number of maximum iterations needed with respect to **Q**, i.e., $r \leq t$, since $r \leq \log_{|\lambda_2(\mathbf{Q})|} \tau \leq t \leq \log_{|\lambda_2(\mathbf{S})|} \tau \leq \log_{\alpha} \tau$. The following bounds, in particular, the lower bounds are derived by assuming the scaling factor, α, used in the PageRank perturbation is contained in the range, $0.5 \leq \alpha \leq 1$ and the PageRank vector is computed to a precision that nominally provides for at least *n* unique PageRank values, i.e., $\tau \leq 1/n$.

The PageRank algorithm's practical lower bound on its required execution time is $\Omega\left(n^2 \cdot \log n\right)$, a new result derived in Section 3.2. The analysis in Section 5.5 showed the QuotientRank algorithm's lower bound is $\Omega\left(n^2 \cdot \log n + b^2 \cdot \log b\right)$. The number of blocks, b, contained in the coarsest equitable partition, B, is bounded by the number of vertices, $n = |V,|$ where $b \leq n$. The coarsest equitable partition can be found in $\Theta\left(n^2 \cdot \log n\right)$ time.

In the worst case, if G yields a discrete coarsest equitable partition, i.e., if $b = n$, the QuotientRank algorithm increases the PageRank algorithm's lower bound by a factor of two, where $n^2 \cdot \log n + b^2 \cdot \log b = 2 \cdot n^2 \cdot \log n$. The impact of this worst-case scenario is offset by noting two more significant results. First, the QuotientRank algorithm decreases the execution time required to determine the PageRank vector of those graphs that yield a non-discrete coarsest equitable partition, i.e., if $b < n$. For such graphs, the QuotientRank algorithm also ensures vertices contained in the same block have equal PageRank values.

The PageRank algorithm's upper bound is $\mathrm{O}\left(n^2 \cdot t\right)$, where $t \leq \log_{|\lambda_2(\mathbf{S})|} \tau \leq \log_{\alpha} \tau$, as shown in Section 2.5.2.2. The analysis in Section 5.5 establishes that the QuotientRank algorithm's upper bound is $\mathrm{O}\left(n^2 \cdot \log n + b^2 \cdot r\right)$, where $r \leq \log_{|\lambda_2(\mathbf{Q})|} \tau \leq t$. This bound also degenerates in the worst-case, i.e., if the coarsest equitable partition is discrete, where $b = n$ yields $\mathrm{O}\left(n^2 \cdot \log n + n^2 \cdot t\right)$. Thus, in the worst case, the QuotientRank algorithm may need $n^2 \cdot \log n$ more time than the PageRank algorithm. However, the QuotientRank algorithm outperforms the PageRank algorithm if the graph's coarsest equitable partition is non-discrete, i.e., contains sufficiently fewer blocks than the graph contains vertices.

The upper bounds on the PageRank and QuotientRank algorithm's execution time are $\mathrm{O}\left(n^2 \cdot t\right)$ and $\mathrm{O}\left(n^2 \cdot \log n + b^2 \cdot r\right)$, respectively. The $n^2 \cdot t$ and $b^2 \cdot r$ terms denote the time needed to apply the power method to the PageRank and quotient matrices, **S** and **Q**, respectively. Thus, the QuotientRank algorithm outperforms the PageRank algorithm if $\left(n^2 \cdot t - b^2 \cdot r\right) \geq n^2 \cdot \log n$, where $n^2 \cdot \log n$ denotes the time needed to obtain the coarsest equitable partition. The anticipated reduction in execution time is $\left(n^2 \cdot t\right) / \left(b^2 \cdot r\right)$, where $r \leq t$ and $b \leq n$. Although $r \leq \log_{|\lambda_2(\mathbf{Q})|} \tau \leq t \leq \log_{|\lambda_2(\mathbf{S})|} \tau \leq \log_{\alpha} \tau$, preliminary data suggests $\left(r \approx t\right) \leq \log_{|\lambda_2(\mathbf{Q})|} \tau \leq \log_{|\lambda_2(\mathbf{S})|} \tau \leq \log_{\alpha} \tau$, reducing the relative performance gain to n^2 / b^2.

If b is sufficiently small relative to n, i.e., if a coarsest equitable partition contains fewer blocks than vertices, the QuotientRank algorithm obtains a PageRank vector in less time than the PageRank algorithm. However, b's value can only be known by computing the coarsest equitable partition, which requires $n^2 \cdot \log n$ time. Graphs containing regular structure yield such gains, e.g., the grid graph depicted in Figure 53.

These relative performance gains increase as the number of blocks decreases with respect to the number of vertices, i.e., as b decreases with respect to n. For instance, many trees have a coarsest equitable partition containing fewer blocks than vertices. Finally, at least certain web graphs yield a coarsest equitable partition containing fewer blocks than vertices [BLS+06].

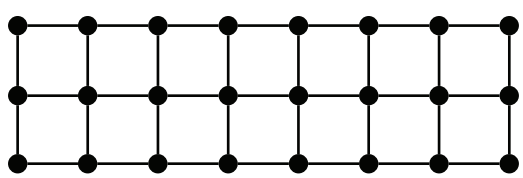


Figure 53. 8×3 Grid: A Graph Yielding a Non-Discrete Coarsest Equitable Partition

VI. Conclusions and Future Research

6.1. Conclusions

6.1.1. Complexity Bounds

The first new result, which was derived in Section 3.2, is a practical lower bound on the number of power method iterations that is obtained by applying recent work that defined an upper bound [HaK03, LaM06]. The lower bound is derived by applying two assumptions related to the required precision, τ, and PageRank scaling value, α. The first assumption is $\tau \le 1/n$, where $\tau = 1/n$ is the largest precision that can provide n unique PageRank values, i.e., one value for each of the n vertices. Assuming $\alpha \ge 0.5$ includes the suggested default PageRank scaling value, $\alpha = 0.85$. Applying these two assumptions and applying base-2 logarithms yields a practical lower bound on the number of required power method iterations, t, needed to compute the PageRank vector, where

$$\frac{\log_2 1/n}{\log_2 0.5} = \log_2 n \le t \le \log_\alpha \tau = \frac{\log_2 \tau}{\log_2 \alpha}. \tag{40}$$

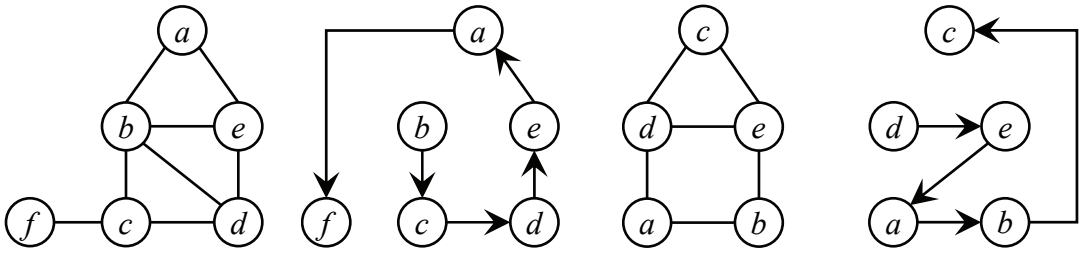
The practical lower bound, $\log_2 n$, matches experimental results reported by the PageRank algorithm's creators [PBM+98] and other researchers [ANT+02]. Each power method iteration performed in the PageRank algorithm multiplies an $n \times n$ matrix with an $n \times 1$ vector. If dense matrices are used, the lower and upper bounds on the time needed to determine the PageRank vector are $\Omega\left(n^2 \cdot \log n\right)$ and $\mathrm{O}\left(n^2 \cdot \log_\alpha \tau\right)$, respectively, where $\log n$ and $\log_\alpha \tau$ are the minimum and maximum required number of iterations. If sparse matrices are being used, the lower and upper bounds are $\Omega\left(m \cdot \log n\right)$ and $\mathrm{O}\left(m \cdot \log_\alpha \tau\right)$, respectively, where $m = |E|$ denotes the number of edges contained in the graph.

6.1.2. Obtaining Canonical Vertex Orderings from the PageRank Vector

Another result, which is not known to be formally stated elsewhere with respect to the PageRank algorithm, is related to the concept of finding a canonical vertex ordering. Assuming sufficient computing resources exist, the two methods described in Section 1.2 can obtain a canonical order, where the first method only uses the PageRank algorithm. For example, the mansion graph illustrated in Figure 54(a) yields a PageRank vector that induces a canonical vertex order, as shown in Figure 54(b).

However, graphs relevant to the work described herein yield non-discrete coarsest equitable partitions. Thus, their PageRank vectors contain at least two equal entries and cannot induce a canonical ordering. For such graphs, a canonical ordering can be found using an application that computes a canonical isomorph, e.g., *nauty*, and applying the PageRank algorithm to the canonical isomorph. For example, the canonical isomorph that is yielded by applying *nauty* to house graph is shown in Figure 54(c), where its PageRank vector induces the canonical vertex ordering illustrated in Figure 54(d).

However, finding a canonical isomorph may require exponential time, therefore, the latter approach only succeeds if it is applied to sufficiently small and irregular graphs. For instance, nearly every random graphs and random trees is sufficiently irregular, even if their coarsest equitable partition is not discrete.



(a) Mansion Graph (b) PageRank Order (c) House Graph (d) PageRank Order

Figure 54. Canonical PageRank Orderings of the Mansion and House Graphs

6.1.3. Relating the PageRank Vector and the Coarsest Equitable Partition

The most theoretical contribution constructed herein is contained in Section 3.3, where it is shown that vertices contained in the same block of a graph's coarsest equitable partition must have equal iterated dot products. Since a PageRank vector can be obtained by applying the power method, which simply computes an iterated dot product, vertices contained in the same block yield equal PageRank values, as described in Section 3.4. This result mirrors another proof of the relationship between the PageRank vector and the coarsest equitable partition [BLS+06]. Boldi *et al.*'s earlier proof uses tools from category theory, e.g., the graph's minimum base and its fibrations, which correspond to the graph's coarsest equitable partition and its blocks.

Both of the proofs suggest many methods of improving the PageRank algorithm's performance. The first improvement is obtained by the AverageRank algorithm listed in Section 4.2, which uses the average block PageRank values. That method's lower bound is the same as the PageRank algorithm's, $\Omega\left(n^2 \cdot \log n\right)$, where n denotes the number of vertices. However, its upper bound is increased from $\mathrm{O}\left(n^2 \cdot t\right)$ to $\mathrm{O}\left(n^2 \cdot \log n + n^2 \cdot t\right)$, where $t \le \log_{|\lambda_2(\mathbf{S})|} \le \log_\alpha \tau$ and **S** denotes the PageRank matrix.

That method is superseded by the ProductRank algorithm listed in Section 4.3, in which one PageRank value is computed for each block in the coarsest equitable partition. The ProductRank algorithm's lower bound is also the same as the PageRank algorithm's lower bound, $\Omega\left(n^2 \cdot \log n\right)$. More importantly, the upper bound reduces from $\mathrm{O}\left(n^2 \cdot t\right)$ to $\mathrm{O}\left(n^2 \cdot \log n + b \cdot n \cdot t\right)$, where b denotes the number of blocks. The ProductRank algorithm ensures vertices contained in the same block have the same PageRank value.

Both algorithms are superseded by the proof provided in Section 5.2 that shows a certain quotient matrix induced by the coarsest equitable partition can be used to compute the PageRank vector. That proof is the basis of the QuotientRank algorithm described in Section 5.4. As shown in Section 5.5, the lower and upper bounds of the QuotientRank algorithm are $\Omega\left(n^2 \cdot \log n\right)$ and $\mathrm{O}\left(n^2 \cdot \log n + b^2 \cdot r\right)$, respectively, where b denotes the number of blocks contained in the coarsest equitable partition. The value of r is based on the second eigenvalue of the quotient matrix, $\mathbf{Q}$, the precision, τ, and scaling factor, α, such that $r \le \log_{|\lambda_2(\mathbf{Q})|} \tau \le t \le \log_{|\lambda_2(\mathbf{S})|} \tau \le \log_\alpha \tau$.

The QuotientRank algorithm further extends [BLS+06], which also showed that the quotient matrix can be used to reduce the time needed to obtain the PageRank vector. However, that proof's authors did not develop or analyze a method based on the quotient matrix. As established in Sections 5.5– 5.7, the QuotientRank algorithm's upper bound is $\mathrm{O}\left(n^2 \cdot \log n + b^2 \cdot r\right)$, whereas the PageRank algorithm's upper bound is $\mathrm{O}\left(n^2 \cdot t\right)$. Thus, if $\left(n^2 \cdot t\right) - \left(b^2 \cdot r\right) \ge n^2 \cdot \log n$, the potential performance gain is $\left(n^2 \cdot t\right) / \left(b^2 \cdot r\right)$, such that $r \le \log_{|\lambda_2(\mathbf{Q})|} \tau \le t \le \log_{|\lambda_2(\mathbf{S})|} \tau \le \log_\alpha \tau$.

In sum, PageRank vectors of graphs containing at least some regular structure can be obtained in less time using the ProductRank and QuotientRank algorithms described in Sections 4.3 and 5.2, respectively. The relative gain is based on the number of blocks in the coarsest equitable partition and number of vertices, such that the performance gain is $\left(n^2 \cdot t\right) / \left(b^2 \cdot r\right)$. Preliminary data suggests $\left(r \approx t\right) \le \log_{|\lambda_2(\mathbf{S})|} \tau$, reducing the gain to n^2 / b^2. Such gains can be obtained on some web graphs [BLS+06], many trees, and grid graphs.

6.2. Future Work

6.2.1. Implementation Improvements

Many of the algorithms described herein were implemented in MATLAB to verify their correctness. A key area of future work is to design robust implementations, e.g., the methods described in Sections 2.3.3.2 and 5.2. For instance, the current implementation find the coarsest equitable partition using 1-D Weisfeiler-Lehman stabilization, due to its relevance to the results described herein. However, the method listed in Section 2.3.3.2 is more efficient.

The quotient matrix can be constructed more efficiently, as noted in Section 5.5. The current versions also use dense matrices, but sparse matrices must be used to process large graphs, e.g., web graphs. Adding sparse matrix support also motivates supporting personalization vectors, where provided a probability vector, **v**, the PageRank matrix, **S**, yielded by applying the PageRank perturbation is (cf. Section 2.5.3)

$$\mathbf{S} = \alpha \cdot \mathbf{A} \cdot \mathbf{D}^{-1} + \left(1 - \alpha\right) \cdot \mathbf{v} \cdot \mathbf{1}^{1,n}. \tag{41}$$

This modification adjusts the PageRank of certain vertices, e.g., to decrease the effect of "spamming done by the so-called link farms" [LaM06]. Personalization may decrease the effectiveness of applying the graph's coarsest equitable partition, since a personalization vector may increase the number of blocks contained in the coarsest equitable partition.

A robust implementation should also leverage parallel hardware, e.g., multi-core processors. The power method can be executed in parallel, since PageRank matrix rows can be independently multiplied by a PageRank vector. Weisfeiler-Lehman stabilization can be implemented in parallel, but similarly implementing the algorithm for finding the coarsest equitable partition described in Section 2.3.3.2 in parallel requires more effort.

6.2.2. Other Linear Algebra Applications

Throughout this research, it was assumed the PageRank vector is being computed by applying the power method to the stochastic PageRank matrix, **S**, or its corresponding quotient matrix, **Q**. For instance, given **Q**, and the normalized all-ones vector, $\mathbf{r} = \mathbf{1}^{b,1}/b$, the PageRank vector is obtained by iteratively computing

$$\begin{aligned} \mathbf{r} &\leftarrow \mathbf{S} \cdot \mathbf{r} \\ \mathbf{r} &\leftarrow \mathbf{r} / \sum \mathbf{r} \end{aligned} \tag{42}$$

until **r** converges to the required numerical precision, τ. Given **Q**'s dominant eigenvector, **r**, the PageRank vector, **x**, is lifted and normalized using the block matrix, **B**, where

$$\begin{aligned} \mathbf{x} &\leftarrow \mathbf{B} \cdot \mathbf{r} \\ \mathbf{x} &\leftarrow \mathbf{x} / \sum \mathbf{x}. \end{aligned} \tag{43}$$

One conjecture is that a similar process reduces the time needed to perform other linear algebra tasks. For example, given an arbitrary linear system,

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b}, \tag{44}$$

the solution vector, **x**, preliminary work suggests **x** can be obtained by solving the system,

$$\mathbf{Q} \cdot \mathbf{r} = \mathbf{c}, \tag{45}$$

where **Q** denotes **A**'s quotient matrix and **c** denotes the associated elements in **b**. Solving for **r** and applying the lifting identity and block matrix, **B** to **r**, yields the solution vector, **x**, where again, $\mathbf{x} \leftarrow \mathbf{B} \cdot \mathbf{r}$.

A key nuance is that the coarsest equitable partition must be found with respect to **A** and each unique vector, **b**. Loosely stated, **b** similarly partitions **A** in the same way the personalization vector, **v**, partitions the PageRank matrix (cf. Sections 2.5.3 and 6.2.1). This approach may decrease the time needed to find **x** in applications that require solving structured linear systems, e.g., certain linear regression or finite element systems.

6.2.3. *k*-D Weisfeiler-Lehman Stabilization

The coarsest equitable partition is the most refined partition that can be obtained by only considering the neighbors of each vertex. Thus, the coarsest equitable partition is obtained by applying information in the first dimension, which contains the current labels of all adjacent vertices. This concept gives rise to the method listed in Section 2.3.3.3 for finding the coarsest equitable partition, 1-D Weisfeiler-Lehman stabilization. The concept also can be generalized to *k* dimensions, or *k*-D Weisfeiler-Lehman stabilization [Bas02]. Although *k*-D stabilization was thought to decide graph isomorphism in polynomial time, both parts of that conjecture have been shown to be false [Für87, CFI92, Für01].

The *k*-D equitable partition is often more refined than the 1-D equitable partition. Since the *k*-D partition may contain more blocks than the 1-D partition, it yields fewer performance gains with respect to the PageRank algorithm or the future work described in Section 6.2.2. However, graphs that yield finer partitions based on a value of *k* relative to smaller values of *k* are useful for assessing algorithms that decide graph isomorphism. Although a library of graphs strictly based on *k* does not exist, the libraries constructed by Weisfeiler [Wei76], Mathon [Mat78], and Junttila and Kaski [JuK07] are commonly used. An avenue of future research is to construct a graph library strictly based on values of *k*.

In addition, the proof developed in Chapter 3 can be extended to show that some linear algebra methods of determining isomorphism are no more powerful than applying *k*-D stabilization. For instance, one avenue of work described elsewhere considered using the matrix inverse to determine graph isomorphism [AMB+07]. The proof constructed in Chapter 3 can be extended to show the matrix inverse does not provide more information than the equitable partition yielded by applying 2-D Weisfeiler-Lehman stabilization.

6.2.4. Open Call for Parallel Software that Decides Graph Isomorphism

Many algorithms decide graph isomorphism for either a restricted set of graphs or two arbitrary graphs (cf. Section 2.3.4.2). Algorithms that decide graph isomorphism can be classified into two groups, where an algorithm either computes an explicit permutation between two graphs or computes a canonical certificate of each graph, i.e., two graphs are isomorphs if some permutation between their vertices exists or their canonical certificates are identical. Both types of algorithms apply graph invariants, e.g., the coarsest equitable partition, to prune the search tree. For example, the classic algorithm developed by Babai, Erdös, and Selkow uses an invariant that can be computed in linear time [BES80]. Their algorithm yields a canonical isomorph of nearly all random graphs.

More robust algorithms decide graph isomorphism between two arbitrary graphs. Some readily accessible software tools include Boost [BST], Groups & Graphs [KoK06], LEDA [LED], Mathematica [MTH], and NetworkX [HSS]. Other algorithms that decide graph isomorphism include VF2 [CFS+04], Bliss [JuK07], and ScrewBox [KuS07]. The VF2 algorithm is the most modern algorithm that decides graph isomorphism by finding a permutation between two input graphs instead of finding their canonical isomorphs.

The standard algorithm used to decide graph isomorphism, *nauty*, includes many other isomorphism-related functions [McK81, McK04]. Software tools that require such functions often interface with *nauty*, such as GAP's GRAPE package [GAP, Soi06] and MAGMA [MAG]. MATLAB's documentation hints that its bioinformatics package links with *nauty* [MAT]. *Nauty*'s widespread usage can be attributed to many factors, e.g., its performance and function set, which includes functions for finding canonical isomorphs, orbit partitions, quotient graphs, and automorphism groups.

Unfortunately, *nauty* does have some shortcomings, prompting this open call for improvements to *nauty* or an alternative tool that implements its functions and addresses its shortcomings. For example, *nauty* is freely accessible, however, its source code is not open-source. The only known open-source package that is similar to *nauty* is the N.I.C.E. library and that was recently written for the SAGE mathematics package [SAG, Mil07]. The N.I.C.E. library applies techniques used in *nauty* and summarized in Section 2.3.4.2. The N.I.C.E. library can be integrated in military applications, whereas *nauty*'s copyright statement currently precludes its use in military applications [McK04].

Currently, N.I.C.E. is the only known alternative to *nauty* that provides a similar set of functions, such as obtaining canonical isomorphs and orbit partitions. N.I.C.E. also appears to support non-simple graphs, such as weighted and directed graphs. In contrast, *nauty*'s documentation describes how to transform theses graphs for processing [McK04]. However, N.I.C.E.'s stated objectives are to provide an open-source and understandable alternative to *nauty*, i.e., N.I.C.E. is not designed to provide the raw speed with respect to deciding graph isomorphism that is presently yielded by *nauty*.

Hence, the most ambitious future goal is to construct an open-source software tool that finds canonical isomorphs and orbit partitions and whose performance is competitive with *nauty*'s performance. This application would leverage the multi-core processors and other parallel tools available in modern computing devices. For instance, this application could apply the method used to find the coarsest equitable partition to be implemented in the parallel algorithm for determining the PageRank vector, as described in Section 6.2.1. A software tool that decides graph isomorphism in parallel can benefit many applications, to include work related to UAV swarms, as described in Sections 2.1, 2.2.2, and 6.1.2.

6.3. Summary

The PageRank algorithm is often used to order query responses, e.g., web pages matching some search criteria. However, the PageRank algorithm has other applications. For instance, the results described herein were motivated by exploring how the PageRank algorithm or tools used to decide graph isomorphism could be applied to sensor networks, e.g., UAV swarms. One natural problem to consider orders the nodes based on some measure of importance. If an attacker knows the $1^{\text{st}}, 2^{\text{nd}}, \ldots, n^{\text{th}}$ most critical nodes, the attacker can determine which nodes to attack first. A similar application is finding nodes that can facilitate spreading (mis)information, or in social networks, diseases and rumors. In general, the PageRank algorithm is useful for analyzing scenarios that require knowing the probable behavior of an object that traverses the underlying graph.

The results described in Chapters 3–5 improve the performance of the PageRank algorithm in several ways, e.g., if it is being applied to networks such as UAV swarms. For instance, the practical lower bound derived in Section 3.2 provides the minimum time needed to obtain the PageRank vector of an arbitrary UAV swarm. The proof described in Sections 3.3 and 3.4 establishes it is possible to easily identify nodes that must have equal PageRank values by applying the graph's coarsest equitable partition.

The AverageRank algorithm listed in Section 4.2 ensures such nodes have equal PageRank values, even if a PageRank vector is numerically computed. The ProductRank algorithm listed in Section 4.3 finds the PageRank vector of graphs containing such nodes more efficiently. Both methods are superseded by the QuotientRank algorithm listed in Section 5.4, the most efficient algorithm described herein. Finally, these results generated many promising avenues of future research, as described in Section 6.2.

[AMB+07] C. Augeri, B. Mullins, L. Baird III, D. Bulutoglu, and R. Baldwin. "An algorithm for determining isomorphism using lexicographic sorting and the matrix inverse", *Congressus Numerantium*, Utilitas Mathematica Publishing, 184:97–120, 2007.

[ANT+02] 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 World Wide Web Conference* (WWW), 2002.

[AsW05] J. Aspnes and U. Weider. "The expansion and mixing time of skip graphs with applications", in *Proceedings of the 17th Symposium on Parallelism in Algorithms and Architectures* (SPAA), ACM, 2005.

[Bab95] L. Babai. "Automorphism groups, isomorphism, reconstruction", Chapter 27 in *Handbook of Combinatorics*, R. Graham, M. Grötschel, and L. Lovász, eds., Elsevier and MIT Press, vol. 2, 1995.

[BaL83] L. Babai and E. Luks. "Canonical labeling of graphs", in *Proceedings of the Symposium on Theory of Computing* (STOC), 1983.

[Bas02] O. Bastert. *Stabilization Procedures and Applications*, Ph.D. Thesis, Zentrum Mathematik, Technische Universtät München (TUM, Technical University of Munich), 2002. Online: http://tumb1.biblio.tu-muenchen.de/publ/diss/ma/2002/bastert.pdf

[BeC06] T. Berners-Lee and D. Connolly. "Delta: An ontology for the distribution of differences between RDF graphs", World Wide Web Consortium (W3C), 2006. Online: http://www.w3.org/DesignIssues/Diff

[BeE96] J. Bennett and J. Edwards. "A graph isomorphism algorithm using pseudo-inverses", *BIT Numerical Mathematics*, 36:41–53, 1996.

[BES80] L. Babai, P. Erdös, and S. Selkow. "Random graph isomorphism", *Journal on Computing*, SIAM, 9(3):628–635, 1980.

[BGM82] L. Babai, D. Grigoryev, and D. Mount. "Isomorphism of graphs with bounded eigenvalue multiplicity", in *Proc. of the Symposium on the Theory of Computing* (STOC), ACM, 1982.

[BLS+06] P. Boldi, V. Lonati, M. Santini, and S. Vigna. "Graph fibrations, graph isomorphism, and PageRank", *RAIRO: Informatique Théorique et Applications*, EDP Sciences, 40:227–253, 2006.

[BMT+04] S. Brennan, A. Mielke, D. Torney, and A. Maccabe. "Radiation detection with distributed sensor networks", *Computer*, IEEE, 37(8):57–59, 2004.

[BrL04] U. Brandes and J. Lerner. "Structural similarity in graphs: a relaxation approach for role assignment", in *Proceeding of the 15th Int'l Sym. on Algorithms and Computation* (ISAAC), Springer, Lecture Notes in Computer Science (LNCS), vol. 3341, 2004.

[BST] *Boost C++ Libraries*. Online: http://www.boost.org/

[CaC82] A. Cardon and M. Crochemore. "Partitioning a graph in $O(|A|\log_2|V|)$", *Theoretical Computer Science*, North-Holland, 19:85–98, 1982.

[Car03] J. Carroll. *Signing RDF graphs*, Tech. Report HPL-2003-142, HP Labs, 2003. Online: http://www.hpl.hp.com/techreports/2003/HPL-2003-142.html

[CDR07] D. Cardoso, C. Delorme, and P. Rama. "Laplacian eigenvectors and eigenvalues and almost equitable partitions", *European Journal of Combinatorics*, Elsevier, 28(3):665–673, 2007.

[CFI92] J. Cai, M. Fürer, and N. Immerman. "An optimal lower bound on the number of variables for graph identification", *Combinatorica*, Springer, 12(4):389–410, 1992.

[CFS+04] L. Cordella, P. Foggia, C. Sansone, and M. Vento. "A (sub)graph isomorphism algorithm for matching large graphs", *Transactions on Pattern Analysis and Machine Intelligence*, IEEE, 26(10):1367–1372, 2004.

[CHP+04] P. Corke, S. Hrabar, R. Peterson, D. Rus, S. Saripalli, and G. Sukhatme. "Autonomous deployment and repair of a sensor network using an unmanned aerial vehicle", in *Proceedings of the International Conference on Robotics and Automation*, IEEE, 2004.

[Chu94] F. Chung. *Spectral Graph Theory*, Regional Conference Series in Mathematics, vol. 92, American Mathematical Society, 1994.

[Cla00] R. Clark. *Uninhabited Combat Aerial Vehicles*, Air University, CADRE Paper, vol. 8, 2000. Online: http://handle.dtic.mil/100.2/ADA382577

[CMI00] *The Millennium Prize Problems*, Clay Mathematics Institute, 2000. Online: http://www.claymath.org/millennium/

[CoG70] D. Corneil and C. Gotlieb. "An efficient algorithm for graph isomorphism", *Journal of the ACM*, ACM, 17(1):51–64, 1970.

[CRS97] D. Cvetković, P. Rowlinson, and S. Simić. *Eigenspaces of Graphs*, Cambridge University Press, 1997.

[DBC+98] G. Duckworth, J. Barger, S. Carlson, D. Gilbert, M. Knack, J. Korn, and R. Mullen. "Fixed and wearable acoustic counter-sniper systems for law enforcement", in *Proceedings of the International Symposium on Enabling Technologies for Law Enforcement and Security*, SPIE, vol. 3577, 1998.

[ElE04] J. Elson and D. Estrin. “Sensor networks: a bridge to the physical world”, Chapter 1 in *Wireless Sensor Networks*, C. Raghavendra, K. Sivalingam, and T. Znati, eds., Kluwer, pgs. 3–20, 2004.

[Fau98] J. Faulon. “Isomorphism, automorphism partitioning, and canonical labeling can be solved in polynomial-time for molecular graphs”, *Journal of Chemical Information and Computer Sciences*, 38:432–444, 1998.

[Für01] M. Fürer. “Weisfeiler-Lehman refinement requires at least a linear number of iterations”, in *Proceedings of the International Colloquium in Automata, Languages, and Programming* (ICALP), Springer-Verlag, Lecture Notes in Computer Science (LNCS), 2076:322–333, 2001.

[Für87] M. Fürer. *A Counterexample in Graph Isomorphism Testing*, Technical Report CS-87-36, Department of Computer Science, Pennsylvania State University, 1987.

[GAP] *Groups, Algorithms, Programming (GAP): A System for Computational Discrete Algebra*, ver. 4.4, Computational Algebra Group, School of Mathematics and Statistics, University of Sydney. Online: http://www.gap-system.org/

[Gat79] G. Gati. “Further annotated bibliography on the isomorphism disease”, *Journal of Graph Theory*, John Wiley & Sons, 3(2):95–109, 1979.

[GCM06] S. Greenblatt, T. Coffman, and S. Marcus. “Behavioral network analysis for terrorist detection”, Chapter 17 in *Emergent Information Technologies and Enabling Policies for Counter-Terrorism*, R. Popp and J. Yen, eds., pgs. 331–348, 2006.

[GHK+03] R. Grossman, D. Hamelberg, P. Kasturi, and B. Liu. “Experimental studies of the universal chemical key (UCK) algorithm on the NCI database of chemical compounds”, in *Proceedings of the IEEE Bioinformatics Conference* (CSB), 2003.

[God93] C. Godsil. *Algebraic Combinatorics*, Chapman & Hall, 1993.

[GoV88] G. Golub and C. Van Loan. *Matrix Computations*, Hopkins Univ. Press, 1988.

[Gol04] M. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*, 2nd ed., Elsevier, 2004.

[Gol80] M. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*, 1st ed., Elsevier, 1980.

[GoR01] C. Godsil and G. Royle. *Algebraic Graph Theory*, Springer-Verlag, 2001.

[Got03] C. Gotsman. “On graph partitioning, spectral analysis, and digital mesh processing”, in *Proceedings of Shape Modeling International* (SMI), IEEE, 2003.

[Gui05] E. Guizzo. “Into deep ice”, *Spectrum*, 42(12):28–35, 2005.

[GuM95] S. Guattery and G. Miller. “On the performance of spectral graph partitioning methods”, in *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms* (SODA), 1995.

[Hae95] W. Haemers. “Interlacing eigenvalues and graphs”, *Linear Algebra and its Applications*, North-Holland, 227–228:593–616, 1995.

[HaK03] T. Haveliwala and S. Kamvar. *The Second Eigenvalue of the Google Matrix*, Technical Report, Department of Computer Science, Stanford University, 2003.

[HaS04] W. Haemers and E. Spence. “Enumeration of cospectral graphs”, *European Journal of Combinatorics*, 25:199–211, 2004.

[HeL93] B. Hendrickson and R. Leland. “An improved spectral graph partitioning algorithm for mapping parallel computations”, *Journal on Scientific Computing*, SIAM, 16(2):452–469, 1995.

[Hof82] C. Hoffmann. *Group-Theoretic Algorithms and Graph Isomorphism*, Springer-Verlag, Lecture Notes in Computer Science (LNCS), vol. 136, 1982.

[HSS] A. Hagberg, D. Schult, and P. Swart. *NetworkX*, Dept. of Mathematical Modeling and Analysis, Los Alamos National Laboratory. Online: https://networkx.lanl.gov/

[HZL05] P. He, W. Zhang, and Q. Li. “Some further development on the eigensystem approach for graph isomorphism detection”, *Journal of the Franklin Institute*, Elsevier, 342(6):657–673, 2005.

[ISB85] *IEEE Standard for Binary Floating-Point Arithmetic*, IEEE Standard 754-1985. Online: http://grouper.ieee.org/groups/754/

[IyB05] S. Iyengar and R. Brooks, eds. *Distributed Sensor Networks*, Chapman and Hall, 2005.

[JAU] *Joint Architecture for Unmanned Systems* (JAUS), JAUS Working Group. Online: http://www.jauswg.org/

[JuK07] T. Junttila and P. Kaski. “Engineering an efficient canonical labeling tool for large and sparse graphs”, in *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments* (ALENEX), SIAM, 2007.

[KaS83] P. Kanellakis and S. Smolka. “CCS expressions, finite state processes, and three problems of equivalence”, in *Proceedings of the Second Annual Symposium on Principles of Distributed Computing* (PODC), ACM, 1983.

[Knu97] D. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*, Addison-Wesley, 1997.

[Koc96] W. Kocay. “On writing isomorphism programs”, in *Computational and Constructive Design Theory*, W. Wallis, ed., Kluwer Academic Publishers, 1996.

[KoK06] W. Kocay and D. Kreher. *Groups & Graphs*, University of Manitoba. Online: http://www.combinatorialmath.ca/G&G/index.html

[Kor05] Y. Koren. “Drawing graphs by eigenvectors: theory and practice”, *Computers and Mathematics with Applications*, Elsevier, 49(11–12):1867–1888, 2005.

[KrS98] D. Kreher and D. Stinson. *Combinatorial Algorithms*, CRC Press, 1998.

[KuS07] M. Kutz and P. Schweitzer. “ScrewBox: a randomized certifying graph-non-isomorphism algorithm”, in *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments* (ALENEX), SIAM, 2007.

[LaM03] A. Langville and C. Meyer. “Deeper inside PageRank”, *Internet Mathematics*, A K Peters, 1(3):335–380, 2003.

[LaM06] A. Langville and C. Meyer. *Google’s PageRank and Beyond: The Science of Search Engine Rankings*, Princeton University Press, 2006.

[LED] *Library of Efficient Data types and Algorithms* (LEDA), Algorithmic Solutions Software. Online: http://www.algorithmic-solutions.com/leda/index.htm

[Ler05] J. Lerner. “Role assignments”, Chapter 9 in *Network Analysis: Methodological Foundations*, U. Brandes and T. Erlebach, eds., Springer, Lecture Notes in Computer Science (LNCS), 3418:216–251, 2005.

[LNV+05] Á. Lédeczi, A. Nádas, P. Völgyesi, G. Balogh, B. Kusy, J. Sallai, G. Pap, S. Dóra, K. Molnár, M. Maróti, and G. Simon. “Countersniper system for urban warfare”, *Transactions on Sensor Networks*, ACM, 1(2):153–177, 2005.

[MAG] MAGMA Computational Algebra System Home Page, ver. 2.14-11, Computational Algebra Group, School of Mathematics and Statistics, University of Sydney. Online: http://magma.maths.usyd.edu.au/magma/

[Mar02] F. Margot. “Pruning by isomorphism in branch-and-cut”, *Mathematical Programming*, Springer, 94(1):71–90, 2002.

[MAT] *MATLAB*, The MathWorks. Online: http://www.mathworks.com/

[Mat78] R. Mathon. “Sample graphs for isomorphism testing”, *Congressus Numerantium*, Utilitas Mathematica Publishing, 9:499–517, 1978.

[McK04] B. McKay. *NAUTY (no AUTormorphisms, yes?)*, Dept. of Computer Science, Australian National University. Online: http://cs.anu.edu.au/~bdm/nauty/

[McK81] B. McKay. “Practical graph isomorphism”, *Congressus Numerantium*, Utilitas Mathematica Publishing, 30:45–87, 1981.

[Mil07] R. Miller. *Nice (as in open source) Isomorphism Check Engine* (N.I.C.E.), 2007. Online: http://www.sagemath.org/doc/html/ref/module-sage.graphs.graph-isom.html

[Miy96] T. Miyazaki. “The complexity of McKay’s canonical labeling algorithm”, in *Groups and Computation II*, *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, AMS, 28:239–256, 1996.

[MMP+06] K. Morris, B. Mullins, D. Pack, G. York, and R. Baldwin. “Impact of limited communications on a cooperative search algorithm for multiple UAVs”, in *Proc. of the IEEE International Conference On Networking, Sensing and Control* (ICNSC), 2006.

[Moh04] B. Mohar. “Graph Laplacians”, Chapter 4 in *Topics in Algebraic Graph Theory*, L. Beineke and R. Wilson, eds., in consultation with P. Cameron, Cambridge University Press, pgs. 113–136, 2004.

[Mor06] K. Morris. *Performance Analysis of a Cooperative Search Algorithm for Multiple Unmanned Aerial Vehicles under Limited Communication Conditions*, M.S. Thesis, Air Force Institute of Technology, 2006. Online: http://handle.dtic.mil/100.2/ADA447093

[MTH] *Mathematica*, Wolfram Research. Online: http://www.wolfram.com/

[NIS] *NIST Chemistry WebBook (Standard Reference Database 69)*, National Institute of Science and Technology (NIST). Online: http://webbook.nist.gov/chemistry/

[OEG+93] M. Ohlrich, C. Ebeling, E. Ginting, and L. Sather. “SubGemini: identifying subcircuits using a fast subgraph isomorphism algorithm”, in *Proceedings of the 30th Int'l Conference on Design Automation* (DAC), 1993.

[OSD05] *Unmanned Aircraft Systems Roadmap, 2005–2030*, Office of the Secretary of Defense (OSD), 2005. Online: http://www.acq.osd.mil/usd/Roadmap%20Final2.pdf

[PaT87] R. Paige and R. Tarjan. “Three partition refinement algorithms”, *Journal on Computing*, SIAM, 16(6):973–989, 1987.

[PBM+98] L. Page, S. Brin, R. Motwani, and T. Winograd. *The PageRank Citation Ranking: Bringing Order to the Web*, Technical Report 1999-66, The Stanford Digital Library Technologies Project, 1998. Online: http://dbpubs.stanford.edu/pub/1999-66

[PBO03] H. Parunak, S. Brueckner, and J. Odell. “Swarming coordination of multiple UAV’s for collaborative sensing”, in *Proceedings of the “Unmanned Unlimited” Systems Technologies and Operations—Aerospace, Land, and Sea Conference*, AIAA, 2003.

[PSC05] S. Pillai, T. Suel, and S. Cha. “The Perron-Frobenius theorem: some of its applications”, *Signal Processing Magazine,* 22:62–75, 2005.

[Rea72] R. Read. “The coding of various kinds of unlabeled trees”, in *Graph Theory and Computing*, R. Read and C. Berge, eds., Academic Press, pgs. 153–182, 1972.

[ReC77] R. Read and D. Corneil. “The graph isomorphism disease”, *Journal of Graph Theory*, John Wiley & Sons, 1:339–363, 1977.

[Rig03] B. Rigling. *Signal Processing Strategies for Bistatic Synthetic Aperture Radar*, Ph.D. Thesis, Department of Electrical Engineering, Ohio State University, 2003. Online: http://www.ohiolink.edu/etd/view.cgi?acc_num=osu1052835606

[Ros00] K. Rosen. *Handbook of Discrete and Combinatorial Mathematics*, CRC, 2000.

[SAG] *SAGE: Open Source Mathematics Software*, The SAGE Foundation. Online: http://www.sagemath.org/

[Sch74] A. Schwenk. “Computing the characteristic polynomial of a graph”, in *Proceedings of the Capital Conference on Graph Theory and Combinatorics*, Springer, Lecture Notes in Mathematics, 406:153–172, 1974.

[Ser02] Á. Seress. *Permutation Group Algorithms*, Cambridge University Press, Cambridge Tracts in Mathematics, vol. 152, 2002.

[SiK67] R. Sinkhorn and P. Knopp. “Concerning nonnegative matrices and doubly stochastic matrices”, *Pacific Journal of Mathematics*, 21:343–348, 1967.

[Sin64] R. Sinkhorn. “A relationship between arbitrary positive matrices and doubly stochastic matrices”, *Annals of Mathematical Statistics*, Institute of Mathematical Statistics, 35:876–879, 1964.

[Soi06] L. Soicher. *GRAPE Package for GAP*, ver. 4.3, School of Math. Sciences, Queen Mary, University of London, 2006. Online: http://www.maths.qmul.ac.uk/~leonard/grape/

[SOP+04] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring, and D. Estrin. “Habitat monitoring with sensor networks”, *Communications of the ACM*, 47(6):34–40, 2004.

[Spi06] C. Spinneli. *Development and Testing of a High-Speed Real-Time Kinematic Precise DGPS Positioning System between Two Aircraft*, M.S. Thesis, Air Force Institute of Technology, 2006. Online: http://handle.dtic.mil/100.2/ADA454831

[StT99] P. Stadler and G. Tinhofer. “Equitable partitions, coherent algebras and random walks: applications to the correlation structure of landscapes”, *MATCH Communications in Mathematical and in Computer Chemistry*, 40:215–261, 1999.

[TBH+04] N. Tatbul, M. Buller, R. Hoyt, S. Mullen, and S. Zdonik. “Confidence-based data management for personal area sensor networks”, in *Proceedings of the Workshop on Data Management for Sensor Networks*, ACM, 2004.

[TiK99] G. Tinhofer and M. Klin. *Algebraic Combinatorics in Mathematical Chemistry. Methods and Algorithms*, *III. Graph Invariants and Stabilization Methods*, Tech. Report TUM-M9902, Technische Universtät München (TUM, Technical University of Munich), 1999. Online: http://www-lit.ma.tum.de/veroeff/html/990.05005.html

[Tra07] *Traffic.com*, NavTeq. Online: http://www.traffic.com/

[USA05] *The U.S. Air Force Remotely Piloted Aircraft and Unmanned Aerial Vehicle Strategic Vision*, Technical Report, United States Air Force, Headquarters, Directorate of Strategic Planning, Future Concepts and Transformation Div., HQ USAF/A8XC, 2005. Online: http://www.af.mil/shared/media/document/AFD-060322-009.pdf

[WaG04] M. Walch and D. Gantz. "Pictographic matching: A graph-based approach towards a language independent document exploitation platform", in *Proceedings of the ACM Workshop on Hardcopy Document Processing*, 2004.

[Wei76] B. Weisfeiler. *On Construction and Identification of Graphs*, Springer-Verlag, Lecture Notes in Mathematics, vol. 558, 1976.

[Wes01] D. West. *Introduction to Graph Theory*, Prentice-Hall, 2001.

[ZhG04] F. Zhao and L. Guibas. *Wireless Sensor Networks*, Morgan-Kaufmann, 2004.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to an penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
11 Sep 2008	Doctoral Dissertation	Aug 2004 – Sep 2008

4. TITLE AND SUBTITLE

On Graph Isomorphism and the PageRank Algorithm

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Augeri, Christopher J.

5d. PROJECT NUMBER
ENG 08-314

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/DCS/ENG/08-08

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Communications Agency
Attn: ENSA (Mr. John Resen)
203 West Losey St
Scott AFB, IL 62225
(618) 229-5341
John.Resen@scott.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)
AFCA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Graphs express relationships among objects, such as the radio connectivity among nodes in unmanned vehicle swarms. Some applications may rank a swarm's nodes by their relative importance, for example, using the PageRank algorithm applied in certain search engines to order query responses. The PageRank values of the nodes correspond to a unique eigenvector that can be computed using the power method, an iterative technique based on matrix multiplication. The first result is a practical lower bound on the PageRank algorithm's execution time that is derived by applying assumptions to the PageRank perturbation's scaling value and the PageRank vector's required numerical precision. The second result establishes nodes contained in the same block of the graph's coarsest equitable partition must have equal PageRank values. The third result, the AverageRank algorithm, ensures such nodes are assigned equal PageRank values. The fourth result, the ProductRank algorithm, reduces the time needed to find the PageRank vector by eliminating certain dot products in the power method if the graph's coarsest equitable partition contains blocks composed of multiple vertices. The fifth result, the QuotientRank algorithm, uses a quotient matrix induced by the coarsest equitable partition to further reduce the time needed to compute a swarm's PageRank vector.

15. SUBJECT TERMS

PageRank algorithm, Markov chain, stochastic matrix, stationary probability distribution, dominant eigenvalue, normalized dominant eigenvector graph isomorphism, canonical isomorph, coarsest equitable partition, orbit partition, automorphism group, Weisfeiler-Lehman stabilization quotient graph, quotient matrix, characteristic block matrix, eigenvalue interlacing, eigenvector lifting, matrix multiplication, dot product

16. SECURITY CLASSIFICATION:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
REPORT	ABSTRACT	c. THIS PAGE			Dr. Barry E. Mullins, AFIT/ENG
U	U	U	UU	153	19b. TELEPHONE NUMBER *(Include area code)* (937) 255-3636, x7979; e-mail: Barry.Mullins@afit.edu

Standard Form 298 (Rev: 8-98)
Prescribed by ANSI Std. Z39-18