Distribution Agreement

In presenting this thesis or dissertation as a partial fulfillment of the requirements for an advanced degree from Emory University, I hereby grant to Emory University and its agents the non-exclusive license to archive, make accessible, and display my thesis or dissertation in whole or in part in all forms of media, now or hereafter known, including display on the world wide web. I understand that I may select some access restrictions as part of the online submission of this thesis or dissertation. I retain all ownership rights to the copyright of the thesis or dissertation. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation.

Signature:

_____ _____

Verena Kuhlemann Date

Iterative Methods and Partitioning Techniques for Large Sparse Problems in Network Analysis

By

Verena Kuhlemann
Doctor of Philosophy

Mathematics and Computer Science

Michele Benzi, Ph.D.
Advisor

James Nagy, Ph.D.
Committee Member

Alessandro Veneziani, Ph.D.
Committee Member

Daniel B. Szyld, Ph.D.
Committee Member

Accepted:

Lisa A. Tedesco, Ph.D.
Dean of the James T. Laney School of Graduate Studies

Date

Iterative Methods and Partitioning Techniques for Large Sparse Problems in Network Analysis

by

Verena Kuhlemann
Master of Science in Mathematics, Emory University, 2010

Advisor: Michele Benzi, Ph.D.

An abstract of
A dissertation submitted to the Faculty of the
James T. Laney School of Graduate Studies of Emory University
in partial fulfillment of the requirements of the degree of
Doctor of Philosophy
in Mathematics and Computer Science
2012.

Abstract

Iterative Methods and Partitioning Techniques for Large Sparse Problems in Network Analysis

By Verena Kuhlemann

The analysis of networks is an important aspect in many fields. Here we consider three different topics: the numerical solution of Markov chains, ranking of genes, and parallel computations with large scale-free graphs.

First, additive Schwarz methods are a class of domain decomposition methods that are suitable for the solution of large linear systems in serial as well as in parallel mode. We adapt the Restricted Additive Schwarz (RAS) method to the computation of the stationary probability distribution vector of large, sparse, irreducible stochastic matrices. The convergence properties are analyzed and extensive numerical experiments aimed at assessing the effect of varying the number of subdomains and the choice of overlap are discussed.

Next, the ranking of genes plays an important role in the identification of key genes for a specific disease. A modification of the PageRank algorithm that is used to rank web pages is the GeneRank algorithm. We assessed the performance of additive Schwarz methods as well as that of Chebyshev iteration for the solution of the GeneRank problem.

Finally, many large networks are scale-free. That is, the degree distribution follows a power-law. Currently available graph partitioners are not efficient for such an irregular degree distribution. The lack of a good partitioning leads to excessive inter-processor communication requirements during each matrix-vector product on parallel distributed-memory computers. We present an approach to alleviate this problem based on embedding the original irregular graph into a more regular one by disaggregating (splitting up) vertices in the original graph. Even though the latter graph is larger, we are able to decrease the communication requirements considerably and improve the performance of the matrix-vector product.

Iterative Methods and Partitioning Techniques for Large Sparse Problems in Network Analysis

by

Verena Kuhlemann
Master of Science in Mathematics, Emory University, 2010

Advisor: Michele Benzi, Ph.D.

A dissertation submitted to the Faculty of the
James T. Laney School of Graduate Studies of Emory University
in partial fulfillment of the requirements of the degree of
Doctor of Philosophy
in Mathematics and Computer Science
2012.

Acknowledgments

First and foremost, I would like to thank my advisor, Dr. Michele Benzi, for his patience and guidance throughout the process. He always seems to have the answer for everything.

I also want to thank my committee members, Dr. James Nagy, Dr. Alessandro Veneziani, and Dr. Daniel Szyld. The time and effort that you spent to enhance my work is truly appreciated.

Furthermore, I would like to express my gratitude to the faculty and staff of the Department of Mathematics and Computer Science at Emory University and the Institut für Mathematik at the Technische Universität Berlin.

In addition, I thank Dr. Panayot Vassilevski and Dr. Van Emden Henson of Lawrence Livermore National Laboratory for mentoring me during a summer internship, and introducing me to an interesting project. I would also like to thank the other members of the Eigensolvers group at Lawrence Livermore National Laboratory for all the help during my internship.

Last but not least, I would like to thank my family and friends for their support and encouragement throughout the years. Special thanks go to Alexis: Mein Fels in der Brandung der auch immer dann ruhig geblieben ist, wenn ich (wie üblich) dafür gesorgt habe, dass alles andere um mich herum im Chaos versinkt.

Contents

1 Introduction and preliminaries **2**

1.1 Background and motivation 2

1.1.1 Networks 2

1.1.2 Overview of the thesis 3

1.2 Basic notions from graph theory and linear algebra 4

1.2.1 Background from graph theory 4

1.2.2 Linear algebra tools 10

1.2.3 Markov chains 15

1.2.4 Complex networks 20

1.3 Overview of iterative methods and graph partitioning 23

1.3.1 Matrix splittings and stationary iterative methods 24

1.3.2 Krylov subspace methods 27

1.3.3 Preconditioning techniques 32

1.3.4 Graph partitioning 34

2 Restricted Additive Schwarz Methods for Markov Chains **38**

2.1 Introduction 38

2.2 Definitions and auxiliary results 39

2.3 Introduction to algebraic domain decomposition methods 40

2.4 Algebraic formulation of Schwarz methods 42

2.5 Nonsingularity of the RAS preconditioner 46
2.6 Properties of the RAS splitting 50
2.7 Extension to inexact solves 53
2.8 Two-level method 55
2.9 Description of the test problems 57
2.10 Numerical Experiments 61
2.10.1 Serial results 61
2.10.1.1 Results with incomplete LU factorization and small amount of overlap 71
2.10.1.2 Results with incomplete LU factorization and large amount of overlap 81
2.10.1.3 Results with incomplete LU factorization, small amount of overlap, and edge weights during partitioning . . . 89
2.10.1.4 Results with the two-level method 97
2.10.2 Parallel results 99
2.11 Summary and conclusions 112

3 Iterative solvers for the GeneRank problem 114

3.1 Introduction 114
3.2 Definitions and auxiliary results 115
3.3 Symmetric formulation of GeneRank 116
3.4 Properties of the symmetric GeneRank matrix 117
3.5 Methods tested 119
3.6 Description of test problems 120
3.7 Numerical experiments 121
3.8 Summary 123

4 Disaggregation techniques for large scale-free graphs **124**

4.1 Indroduction 124

4.2 Disaggregation 125

4.3 Parallel Disaggregation 132

4.4 Numerical Results 140

4.5 Conclusions 148

5 Conclusions and suggestions for future work **150**

Appendix A Algorithms **152**

Appendix B Additional numerical experiments **154**

List of Figures

1.1 Main connected component of the social network of injecting drug users in Colorado Springs. The size of the nodes is proportional to the degree. Two drug addicts are connected if they have exchanged a needle. (Image courtesy of Ernesto Estrada [42].) 20

2.1 Nonzero patterns of matrices used to construct the coarse grid correction. Left: sparsity pattern of the matrix permuted by Luby's maximal independent set reordering. Right: sparsity pattern of the corresponding Schur complement, also permuted by Luby's maximal independent set reordering. The original matrix comes from the *ncd* family. 57

2.2 Transition rate matrix for the reliability model. 59

2.3 Left: nonzero pattern of an *ncd* matrix. Right: nonzero pattern of the same matrix when four subdomains are used and the matrix is reordered accordingly by Metis. 62

2.4 Left: nonzero pattern of a *mutex* matrix. Right: nonzero pattern of the same matrix when four subdomains are used and the matrix is reordered accordingly by Metis. 62

2.5 Left: nonzero pattern of a block of an *ncd* matrix. Right: nonzero pattern of the same matrix after reordering with reverse Cuthill–McKee. 64

2.6 Nonzero pattern of the ILU factorization of a block of an *ncd* matrix without RCM reordering. Left: nonzero pattern L. Right: nonzero pattern of U. 64
2.7 Nonzero pattern of the ILU factorization of a block of an *ncd* matrix with RCM reordering. Left: nonzero pattern L. Right: nonzero pattern of U. 65

3.1 Nonzero pattern of the SNPa matrix. 121

4.1 Disaggregating a node and using a circle as connections between the new nodes. 126
4.2 Disaggregating a node and using a complete graph as connections between the new nodes. 126
4.3 Presenting the disaggregated graph as a combination of the graph with internal and external edges. 127
4.4 Non-zero pattern of the original matrix, and the original matrix after redistributing with ParMETIS. 137
4.5 Non-zero pattern after using disaggregation to limit communication to 75%, 50%, or 25% of the other processors. 138
4.6 Numerical results for matrices with average degree two. The matrices have 10,000 nodes per processor. Before disaggregating the matrices, we first repartitioned them with ParMETIS. The time needed for 10 matrix-vector products is given in seconds. 142
4.7 Numerical results for matrices with average degree five. The matrices have 10,000 nodes per processor. Before disaggregating the matrices, we first repartitioned them with ParMETIS. The time needed for 10 matrix-vector products is given in seconds. 143

4.8 Numerical results for matrices with average degree two. The matrices have 10,000 nodes per processor. No repartitioning is used. The time needed for 10 matrix-vector products is given in seconds. 144
4.9 Numerical results for matrices with average degree five. The matrices have 10,000 nodes per processor. No repartitioning is used. The time needed for 10 matrix-vector products is given in seconds. 145
4.10 Numerical results for the hollywood-2011 matrix. Before disaggregating the matrix, we first repartitioned it with ParMETIS. The time needed for 10 matrix-vector products is given in seconds. 146
4.11 Time in seconds needed to find the 4 smallest eigenvalues with Lanczos algorithm. The matrix has 100 nodes per processor and average degree 2. 147

List of Tables

2.1 Properties of the generator matrices 58

2.2 Subdominant eigenvalue, reliability models 60

2.3 ILUTH results, reliability models 61

2.4 Time in seconds required for factoring the diagonal blocks. The resulting fill-in is given as the ratio of the number of non zeros in $L \cdot U$ and the matrix. K is the number of domains. The matrix is of the *ncd* family and has 12341 rows and columns. 66

2.5 Complete LU factorization vs. incomplete LU factorization. 70

2.6 Results for the *ncd* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A small amount of overlap was chosen. 71

2.7 Results for the *ncd* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A small amount of overlap was chosen. 72

2.8 Results for the *twod* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A small amount of overlap was chosen. 73

2.9 Results for the *twod* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A small amount of overlap was chosen. 74

2.10 Results for the *tcomm* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A small amount of overlap was chosen. 75

2.11 Results for the *mutex* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-3} was used. A small amount of overlap was chosen. 76

2.12 Results for the *reliab1* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-3} was used. A small amount of overlap was chosen. 77

2.13 Results for the *reliab1* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-3} was used. A small amount of overlap was chosen. 78

2.14 Results for the *reliab2* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-3} was used. A small amount of overlap was chosen. 79

2.15 Results for the *reliab2* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-3} was used. A small amount of overlap was chosen. 80

2.16 Results for the *ncd* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A large amount of overlap was chosen. 81

2.17 Results for the *ncd* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A large amount of overlap was chosen. 82

2.18 Results for the *twod* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A large amount of overlap was chosen. 83

2.19 Results for the *twod* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A large amount of overlap was chosen. 84

2.20 Results for the *tcomm* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A large amount of overlap was chosen. 85

2.21 Results for the *reliab1* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-3} was used. A large amount of overlap was chosen. 86

2.22 Results for the *reliab1* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-3} was used. A large amount of overlap was chosen. 87

2.23 Results for the *reliab2* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-3} was used. A large amount of overlap was chosen. 88

2.24 Results for the *ncd* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A small amount of overlap was chosen, and we used weighted edges in Metis. 89

2.25 Results for the *ncd* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A small amount of overlap was chosen, and we used weighted edges in Metis. 90

2.26 Results for the *twod* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A small amount of overlap was chosen, and we used weighted edges in Metis. For each matrix, the best overall timings are in boldface. 91

2.27 Results for the *twod* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A small amount of overlap was chosen, and we used weighted edges in Metis. For each matrix, the best overall timings are in boldface. 92

2.28 Results for the *twod* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A small amount of overlap was chosen, and we used weighted edges in Metis. For each matrix, the best overall timings are in boldface. 93

2.29 Results for the *mutex* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-3} was used. A small amount of overlap was chosen, and we used weighted edges in Metis. For each matrix, the best overall timings are in boldface. 94

2.30 Results for the *reliab1* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-3} was used. A small amount of overlap was chosen, and we used weighted edges in Metis. For each matrix, the best overall timings are in boldface. 95

2.31 Results for the *reliab1* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-3} was used. A small amount of overlap was chosen, and we used weighted edges in Metis. 96

2.32 Results with the 2-level method for the *ncd* and *twod* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. 97

2.33 Results with the 2-level method for the *reliability* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the RAS preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-3} was used. 98

2.34 Results for the *ncd(20)* matrix. K is the number of subdomains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For the incomplete LU 3 levels of fill-in were used. 102

2.35 Results for the *ncd(25)* matrix. K is the number of subdomains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For the incomplete LU 3 levels of fill-in were used. 103

2.36 Results for the *twod(14)* matrix. K is the number of subdomains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For the incomplete LU 20 levels of fill-in were used. 104

2.37 Results for the *twod(17)* matrix. K is the number of subdomains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For the incomplete LU 20 levels of fill-in were used. 105

2.38 Results for the *reliab1(1200)* matrix. K is the number of subdomains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For the incomplete LU 20 levels of fill-in were used. 106

2.39 Results for the *reliab1(2000)* matrix. K is the number of subdomains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For the incomplete LU 20 levels of fill-in were used. 107

2.40 Results for the *reliab1(2800)* matrix. K is the number of subdomains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For the incomplete LU 20 levels of fill-in were used. 108

2.41 Results for the *reliab2(1200)* matrix. K is the number of subdomains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For the incomplete LU 20 levels of fill-in were used. 109

2.42 Results for the *reliab2(2000)* matrix. K is the number of subdomains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For the incomplete LU 20 levels of fill-in were used. 110

2.43 Results for the *reliab2(2800)* matrix. K is the number of subdomains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For the incomplete LU 20 levels of fill-in were used. 111

3.1 Results for the SNPa matrix. The matrix has $n = 152,520$ rows and columns. The tolerance used is 10^{-10}. Here $\boldsymbol{ex} = (1/n)\boldsymbol{e}$, where $\boldsymbol{e}$ is the vector of all ones. The number of iterations and the CPU time in seconds (in brackets) are given. 122

3.2 Results for the SNPa matrix. The matrix has $n = 152,520$ rows and columns. The tolerance used is 10^{-10}. Here $\boldsymbol{ex} = p$, where p is a random probability vector. The number of iterations and the CPU time in seconds (in brackets) are given. 122

3.3 Results for the RENGA matrices. The tolerance used is 10^{-10}. Here $\boldsymbol{ex} = (1/n)\boldsymbol{e}$, where $\boldsymbol{e}$ is the vector of all ones. The number of iterations and the CPU time in seconds (in brackets) are given. 122

3.4 Results for the RENGA matrices. The tolerance used is 10^{-10}. Here $\boldsymbol{ex} = p$, where p is a random probability vector. The number of iterations and the CPU time in seconds (in brackets) are given. 123

4.1 Ratio of the number of nodes of the disaggregated matrix and the original matrix. 141

4.2 Ratio of the number of nodes of the disaggregated matrix and the original matrix. The matrix was re-partitioned with ParMETIS before applying disaggregation. 141
4.3 Ratio of the number of edges of the disaggregated matrix and the original matrix. 141

List of Algorithms

3.1 Chebyshev iteration 120
4.1 Parallel Disaggregation 135
A.1 Arnoldi iteration (modified Gram-Schmidt variant) 152
A.2 Lanczos iteration 152
A.3 Gaussian Elimination 152
A.4 GMRES (basic form) 153
A.5 GMRES(m) 153
A.6 CG 153

Chapter 1

Introduction and preliminaries

1.1 Background and motivation

1.1.1 Networks

Recently there has been a great interest in the field of networks. Networks are ubiquitous. Broadly speaking, a network models interactions between components. As such, they can be used to model many different real life problems. Family ties, collaborations between scientists, and other relations can be modeled by social networks. Complex biological systems such as protein-protein interactions, gene regulations, connections between neurons in the brain, just to name a few, are examples of networks as well. The size of these networks is increasing and they are becoming more and more complex. This is mainly due to technical advances that make it possible to gather larger and larger amounts of data. Social networking sites are widely used; Facebook recently reached a billion users.

The need to analyze these networks as well as the discovery of properties that are shared by many real life networks has led to an increased interest from scientists from diverse fields such as biology, sociology, neuroscience, computer science, and mathematics, to name a few [8]. There is a lot of information that can be deduced

from the structure of a network. Scientists are interested in finding the nodes in the network that play a central role (centrality). For example, search engines rely on ranking of web pages based on the network given by hyperlinks between web pages [90]. Epidemiologists are interested in how a disease spreads in a social network [84]. Identifying central figures can be used to better focus treatment or immunization for contagious diseases. The power grid, where generators and substations are connected by transmission lines, is studied to prevent cascading failures from the failure of a few substations. By identifying high-risk substations the network can be protected more cost-efficiently [96, 120]. Measures of centrality range from simple local measures such as the number of connections of a component (degree centrality), to more complex measures that rely on eigenvectors of a matrix that can be associated with the network (eigenvector centrality) or on the number of shortest paths in the network that pass through a node (betweenness centrality). Identifying groups or communities of closely related components in a network is another important aspect of network analysis (clustering). Applications of community detections range from tailoring of marketing schemes to the identification of terrorist cells [70]. Efficient methods for network analysis are in great demand. In this thesis we address a few fundamental issues of a computational nature arising in the analysis of large-scale complex networks.

1.1.2 Overview of the thesis

This thesis is structured as follows. In the remainder of Chapter 1 we provide some background information from graph theory, linear algebra, and iterative methods. In the second chapter we investigate a restricted additive Schwarz (RAS) preconditioner for computing the stationary distribution vector of Markov chains. We give the first theoretical analysis of RAS in the context of Markov chains. In addition, we provide an extensive collection of numerical experiments, including some experiments with a two-level method, and parallel results. In Chapter 3 we consider the GeneRank

algorithm, a modification of Google's PageRank algorithm, which was originally proposed in [81] for the ranking of genes. Finally, in the last chapter we introduce a new method to improve the performance of the parallel matrix-vector multiplication for scale-free graphs. The results from Chapter 4 are joint work with Panayot S. Vassilevski from the Lawrence Livermore National Laboratory in Livermore, CA.

1.2 Basic notions from graph theory and linear algebra

1.2.1 Background from graph theory

Here we will review basic terminology and background related to graphs that will be used throughout the thesis. A good introduction to graph theoretic concepts can be found for example in [19, 20, 37].

A *graph* is a representation of a set of objects together with connections between these objects. For example, each object could represent a person and the connection between the objects represents a relationship between the two persons. This is referred to as a *social graph*.

Mathematically, we will describe a graph as an ordered pair $\mathcal{G} = (V, E)$, where V denotes a set of objects called *nodes* or *vertices*. The set E describes the connections between the nodes. These connections, called *edges*, are given as a pair of nodes (u, v) with $u, v \in V$. These pairs can be ordered or unordered. Edges that are given by ordered pairs of nodes are called *directed edges*, and a graph whose edges are directed is called *directed graph* or *digraph*. For a directed edge $e = (u, v)$, also denoted with $u \mapsto v$, u is called the *tail* of the edge and v is called the *head*. Similarly, if the edges are given by unordered pairs of nodes, the edges have no orientation and the graph is called an *undirected graph*. We will usually omit the term undirected, and in the

following the term *graph* will be used interchangeably with the term *undirected graph*.

Both vertices and edges of a graph can have weights associated with them. For example, in a graph that describes a road network, weights on the edges can describe distances between points in the network, while weights on the nodes could describe for example the price of gas at this location. In our case there will be weights on the edges but not on the nodes of the graph. For an unweighted graph we will set each edge weight to one.

An edge that starts and ends at the same vertex is called a *loop*. If a pair of nodes is connected by more than one edge, we say the graph has *multiple edges*. If a graph has no loops or multiple edges it is referred to as a *simple* graph. If two nodes u and v are connected by an edge in the graph, that is $e = (u, v) \in E$, we say that the vertices u and v are *adjacent*. The edge $e = (u, v)$ is said to be *incident* to both nodes u and v. For every node the *degree* of the node is the number of edges that are incident to this node. The degree of $v \in V$ will be denoted by $deg(v)$. Since every edge is incident to two nodes, the sum of the degrees of all nodes of a simple graph is twice as large as the number of edges. That is, $\sum_{v \in V} deg(v) = 2 \cdot |E|$. This result is known as *degree sum formula* or *handshaking lemma* and is due to Leonhard Euler. In the case of digraphs we can distinguish between the in-degree and out-degree of a node. The in-degree $deg^-(v)$ of node v is the number of edges in the graph that have node v as a head. Similarly, the out-degree $deg^+(v)$ of node v is the number of edges in the graph that have node v as a tail. Since every edge has only one head and one tail, the sum of all in-degrees or out-degrees in a simple graph is the number of edges in this graph. That is, $\sum_{v \in V} deg^-(v) = |E|$ and $\sum_{v \in V} deg^+(v) = |E|$.

A *walk* from node u to node v is a sequence of adjacent vertices and edges incident to these vertices of the form

$$u = v_0, e_1, v_1, e_2, \ldots, v_{n-1}, e_n, v_n = v,$$

where each v_i represents a node and $e_i = (v_{i-1}, v_i)$ represents an edge. A walk with distinct vertices $v_1, \ldots, v_{n-1}$ is called *path*. If a walk starts and ends at the same vertex, that is, $u = v_0 = v_n = v$, it is called *closed walk*, while a path that starts and ends at the same vertex is usually referred to as a *cycle*. The *length of a walk (path)* equals the number of edges on the walk (path). In a directed graph walks are usually directed as well. That is, a walk from u to v is given by

$$u = v_0, e_1, v_1, e_2, \ldots, v_{n-1}, e_n, v_n = v,$$

where e_i is a directed edge of the form $v_{i-1} \mapsto v_i$. Paths can also be used to determine the distance between two vertices. The distance of two vertices u and v, denoted with $d(u, v)$, is the minimum length of a path between these two vertices. One is often interested in the longest possible distance between two nodes in a graph. This is referred to as *diameter* of the graph and is given by

$$\operatorname{diam}(\mathcal{G}) = \max\{d(u, v) \mid u, v \in V\}.$$

An important concept in graph theory is the connectivity of a graph. An undirected graph is *connected* if for every pair of vertices $u, v \in V$ there exists a path in the graph that starts with u and ends with v. For a directed graph we distinguish between two forms of connectedness. We say a directed graph $\mathcal{G}$ is *weakly connected* if the undirected graph that results from removing the directions on the edges in $\mathcal{G}$ is connected. A directed graph is said to be *strongly connected* if every pair of vertices in the graph lies on a common directed cycle.

If a graph is not connected, or *disconnected*, one is interested in the number of connected maximal subgraphs. A graph $\mathcal{H} = (V_{\mathcal{H}}, E_{\mathcal{H}})$ is a *subgraph* of $\mathcal{G} = (V, E)$, if $V_{\mathcal{H}}$ is a subset of V and $E_{\mathcal{H}}$ is a subset of E. A subgraph $\mathcal{H}$ is called a *connected component* of $\mathcal{G}$ if $\mathcal{H}$ is connected and there is no other connected subgraph of $\mathcal{G}$

that has $\mathcal{H}$ as a subgraph. For a digraph we distinguish between weak and strong components. $\mathcal{H}$ is called a *strong component* if it is strongly connected and there is no strongly connected subgraph of $\mathcal{G}$ that has $\mathcal{H}$ as a subgraph. If $\mathcal{H}$ is weakly connected and not contained in any weakly connected subgraph, it is called a *weak component*. If a graph is partitioned into weak components, every node and every edge is contained in exactly one weak component. On the other hand, if the graph is partitioned into strong components, every node is contained in exactly one strong component, but there might be edges that are not contained in any of the strong components.

Properties of graphs can also be represented by matrices. For a graph $\mathcal{G} = (V, E)$ the adjacency properties can be represented by the *adjacency matrix*. To derive the adjacency matrix we number the vertices with $1, \ldots, n$ where $n = |V|$. Then the adjacency matrix $A_{\mathcal{G}}$ is an $n \times n$ matrix given by

$$[A_{\mathcal{G}}]_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise.} \end{cases}$$

For an undirected graph the adjacency matrix is symmetric, while for a digraph the adjacency matrix is usually not symmetric. For a graph with weights on the edges the *weighted adjacency matrix* is given by

$$[A_{\mathcal{G}}]_{ij} = \begin{cases} w_{i,j} & \text{if } (i,j) \in E \\ 0 & \text{otherwise,} \end{cases}$$

where $w_{i,j}$ is the weight of edge (i, j). There is an interesting connection between the powers of the unweighted adjacency matrix and the number of walks of a specific length between two vertices. The (i, j) entry in ${A_{\mathcal{G}}}^r$ equals the number of walks of length r from node i to node j. That also means that a graph with n nodes is

connected if and only if all entries of the matrix $\sum_{r=1}^{n} {A_{\mathcal{G}}}^r$ are positive. For a digraph the (i,j) entry in ${A_{\mathcal{G}}}^r$ will give the number of directed walks of length r from i to j, and it is strongly connected if and only if all entries of $\sum_{r=1}^{n} {A_{\mathcal{G}}}^r$ are positive.

Another matrix that can be used to represent the adjacency structure of a graph is the *incidence matrix*. If the nodes of the graph are numbered from $1,\ldots,n$ and the edges are numbered from $1,\ldots,m$, where $n=|V|$ and $m=|E|$, then the incidence matrix is an $m \times n$ matrix with exactly two entries per row. For an unweighted digraph $\mathcal{G}$ the incidence matrix is given by

$$[EV_{\mathcal{G}}]_{e,i} = \begin{cases} 1 & \text{if } e = i \mapsto j, \\ -1 & \text{if } e = j \mapsto i, \\ 0 & \text{otherwise.} \end{cases}$$

If the graph has weights on the edges we can define a *weighted incidence matrix* as follows

$$[EV_{\mathcal{G}}]_{e,i} = \begin{cases} w_e & \text{if } e = i \mapsto j, \\ -w_e & \text{if } e = j \mapsto i, \\ 0 & \text{otherwise,} \end{cases}$$

where w_e denotes the weight of edge e. For an undirected graph a random direction for the edges is chosen and the incidence matrix is then given as for the digraph. Note that the incidence matrix is also called *edge-vertex matrix*, while the transpose of the incidence matrix is referred to as *vertex-edge matrix*. The incidence matrix of a graph can also be used to derive information about the connectivity of this graph. A graph is connected if and only if the rank of the incidence matrix equals $|V|-1$. Furthermore, if the graph has c components, then the rank of the incidence matrix equals $|V|-c$; see [26].

For an undirected graph $\mathcal{G}$, the $|V| \times |V|$ diagonal matrix $D_{\mathcal{G}}$ with the degrees of

the vertices on the diagonal is called *degree matrix*. That is,

$$[D_{\mathcal{G}}]_{i,j} = \begin{cases} deg(i) & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

Note that the ith diagonal entry of $D_{\mathcal{G}}$ is just the ith row sum of the adjacency matrix $A_{\mathcal{G}}$. If the graph has weights on the edges the *weighted degree matrix* has the sum of the weights of the edges incident to the corresponding node on the diagonal rather than the degree of this node. Thus, the weighted degree matrix is given by

$$[D_{\mathcal{G}}]_{i,j} = \begin{cases} \sum_{(i,k)\in E} w_{i,k} & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

The *Laplacian matrix* of a graph, also called *Kirchhoff matrix*, is given by

$$L_{\mathcal{G}} = D_{\mathcal{G}} - A_{\mathcal{G}}.$$

We can either use the unweighted degree and unweighted adjacency matrix to derive an unweighted graph Laplacian, or the weighted matrices can be used to derive a weighted graph Laplacian. The graph Laplacian can also be written in terms of the edge-vertex and vertex-edge incidence matrix [17, 19, 55]. That is, the unweighted graph Laplacian can be written as a product of the unweighted vertex-edge and edge-vertex incidence matrix

$$L_{\mathcal{G}} = (EV_{\mathcal{G}})^T EV_{\mathcal{G}},$$

while the weighted graph Laplacian can be written as a triple product of the unweighted vertex-edge incidence matrix, a weight matrix, and the unweighted edge-vertex incidence matrix

$$L_{\mathcal{G}} = (EV_{\mathcal{G}})^T \cdot W \cdot EV_{\mathcal{G}},$$

where W is a $|E| \times |E|$ diagonal matrix with the weight of the edges on the diagonal. Similar to the other matrices associated with graphs, there is a relationship between the graph Laplacian and the connectivity of the graph. The graph Laplacian has n nonnegative eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$. The multiplicity of the zero eigenvalue equals the number of connected components of the graph [79, 80].

1.2.2 Linear algebra tools

In this section we present background information and tools from the area of linear algebra. We start with a few definitions related to matrices.

We say a real matrix A is *positive* or *element-wise positive* if all of its entries are positive. Similarly, a matrix is called *nonnegative* if all of its entries are nonnegative. We will use $A > 0$ to denote a positive matrix, and $A \geq 0$ to denote a nonnegative matrix, and we write $B \geq A$ if $B - A \geq 0$. The set of eigenvalues of a matrix A is called *spectrum* and is denoted with $\sigma(A)$. The *spectral radius* $\rho(A)$ of A is given by

$$\rho(A) = \max\{|\lambda|, \lambda \in \sigma(A)\}.$$

The *spectral circle* of A is a circle of radius $\rho(A)$ with the origin as its center. The range of A is denoted by $\mathcal{R}(A)$.

Definition 1.2.1. A square matrix A is said to be a *reducible matrix* if there exists a permutation matrix P such that

$$P^T A P = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix},$$

where A_{11} and A_{22} are square matrices. If a matrix is not reducible it is called *irreducible* or *nondecomposable*.

Another characterization of irreducible matrices is given by the following. Let A

be an $n \times n$ matrix, and let $\mathcal{G}(A)$ denote the underlying digraph. That is, $\mathcal{G}(A)$ is a graph with n nodes, $v_1, \ldots, v_n$, and there is a directed edge from v_i to v_j if and only if $a_{ij} \neq 0$. A matrix A is irreducible if and only if the digraph $\mathcal{G}(A)$ is strongly connected. The connection between the irreducibility of a matrix and the strong connectivity of the underlying digraph is not hard to see. The underlying digraph of a matrix and that of a symmetric permutation of this matrix are the same up to a relabeling of the nodes. If we consider the reducible matrix

$$A = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix}$$

with $A_{11} \in \mathbb{R}^{r \times r}$ and $A_{22} \in \mathbb{R}^{n-r \times n-r}$, we can decompose the set of vertices V of the underlying digraph $\mathcal{G}(A)$ into two subsets $V(A_{11})$ and $V(A_{22})$, where the set $V(A_{11})$ corresponds to the first r rows of the matrix and $V(A_{22})$ to the last $n-r$ rows. Since $a_{ij} = 0$ for $i > r$ and $j \leq r$, there is no edge in $\mathcal{G}(A)$ that connects a node from $V(A_{22})$ with a node from $V(A_{11})$. Thus, there is no path from node u to node v if $u \in V(A_{22})$ and $v \in V(A_{11})$, and $\mathcal{G}$ is not strongly connected. On the other hand, if $\mathcal{G}$ is not strongly connected we can decompose the graph into strong components. There must be a strong component that has only outgoing edges. If we number the nodes from this component first, the adjacency matrix has the form

$$\begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix},$$

where A_{11} is the adjacency matrix of the strong component with only outgoing edges. For nonnegative matrices, irreducibility can be characterized by the following result [16].

Theorem 1.2.2. *A nonnegative matrix A is irreducible if and only if for every (i, j)*

there exists a natural number q such that

$$[A^q]_{ij} > 0.$$

The following well known theorem gives useful information about the spectra of nonnegative irreducible matrices. It can be found with proof for example in [16, 78, 110].

Theorem 1.2.3 (Perron-Frobenius Theorem)**.** *If $A \geq 0$ is an $n \times n$ irreducible matrix, then each of the following is true.*

1. $r = \rho(A) > 0$.
2. $r \in \sigma(A)$ *(r is called the* Perron root*).*
3. *The algebraic multiplicity of the eigenvalue r is one, that is, the Perron root is simple.*
4. *There exists an eigenvector $x > 0$ such that $Ax = rx$.*
5. *The* Perron vector *is the unique vector defined by*

$$Ap = rp, \quad p > 0, \ \|p\|_1 = 1,$$

   and, except for positive multiples of p, there are no other nonnegative eigenvectors of A, regardless of the eigenvalue.
6. *r need not be the only eigenvalue on the spectral circle of A.*

The Perron-Frobenius Theorem for nonnegative irreducible matrices is a generalization of the Perron Theorem for positive matrices. For positive matrices it can be guaranteed that there is only one eigenvalue on the spectral circle. For nonnegative irreducible matrices this cannot be guaranteed. Thus, the set of nonnegative

irreducible matrices is divided into two classes, those that have only one eigenvalue on the spectral circle and those that have multiple eigenvalues on it. Nonnegative irreducible matrices with only one eigenvalue on the spectral circle are called *primitive*. If a nonnegative irreducible matrix has more than one eigenvalue on its spectral circle it is referred to as *imprimitive*, and the number of eigenvalues on the spectral radius is called *index of imprimitivity*. For a nonnegative irreducible matrix A the primitivity can be determined by its diagonal entries. The matrix is primitive if it has at least one nonzero diagonal entry. Another sufficient condition for primitivity is that there is a constant m such that $A^m > 0$.

An important class of matrices is that of M-matrices, defined below.

Definition 1.2.4. A square (real) matrix A is called an *M-matrix* whenever there exists a matrix $B \geq 0$ and a real number $r \geq \rho(B)$ such that $A = rI - B$.

The Laplacian matrix $L_{\mathcal{G}}$ is an example of a symmetric M-matrix. Recall that $L_{\mathcal{G}}$ is given by $L_{\mathcal{G}} = D_{\mathcal{G}} - A_{\mathcal{G}}$ where $A_{\mathcal{G}} \geq 0$ is the adjacency matrix of the graph and $D_{\mathcal{G}} \geq 0$ is a diagonal matrix with the degrees of the nodes on the diagonal. We can write $L_{\mathcal{G}}$ as $L_{\mathcal{G}} = \Delta(\mathcal{G})I - B$ with $B = A_{\mathcal{G}} + (\Delta(\mathcal{G})I - D_{\mathcal{G}})$, where $\Delta(\mathcal{G})$ is the maximum degree of $\mathcal{G}$. Since $A_{\mathcal{G}}$ and $\Delta(\mathcal{G})I - D_{\mathcal{G}}$ are nonnegative matrices, B is nonnegative as well. From the Gershgorin Circle Theorem [48, Theorem 7.2.1] it follows that $\rho(B) \leq \Delta(G)$. Thus, $L_{\mathcal{G}}$ is an M-matrix.

M-matrices have several useful properties. First, we note that if $A = rI - B$ is an M-matrix with $r < \rho(B)$, then A is nonsingular. If $r = \rho(B)$, then A is singular. The graph Laplacian $L_{\mathcal{G}}$ is a singular matrix and it is easy to see that $\Delta(G)$ is an eigenvalue of $B = A_{\mathcal{G}} + (\Delta(\mathcal{G})I - D_{\mathcal{G}})$. For the vector of all ones e, $Be = (A_{\mathcal{G}} - D_{\mathcal{G}})e + \Delta(\mathcal{G})e = \Delta(\mathcal{G})e$. Thus, $\Delta(G)$ is an eigenvalue of B, and with $\rho(B) \leq \Delta(\mathcal{G})$ it follows that $\rho(B) = \Delta(\mathcal{G})$. That is, for the graph Laplacian, $\Delta(\mathcal{G}) = \|A\|_\infty = \|A\|_1 = \rho(A)$

The following gives some characterization of nonsingular M-matrices [93].

Theorem 1.2.5. *Suppose A is a square real matrix with positive diagonal entries and nonpositive off-diagonal entries. Then the following statements are equivalent.*

1. *A is a nonsingular M-matrix.*
2. *$A = rI - B$ for some nonnegative B and real $r > \rho(B)$.*
3. *The real part of of each eigenvalue of A is positive. In particular, if A is a symmetric nonsingular M-matrix then A is positive definite.*
4. *All principal minors of A are positive.*
5. *A^{-1} exists and $A^{-1} \geq 0$.*
6. *There exists a vector $x > 0$ such that $Ax > 0$.*
7. *If B is a matrix with positive diagonal entries and nonpositive off-diagonal entries and $B \geq A$, then B^{-1} exists.*

In our applications we will also deal with singular M-matrices. Several equivalent characterizations of singular M-matrices are given next [93].

Theorem 1.2.6. *Suppose A is a square real matrix with positive diagonal entries and nonpositive off-diagonal entries. Then the following statements are equivalent.*

1. *A is a singular M-matrix.*
2. *$A = rI - B$ for some nonnegative B and real $r = \rho(B)$.*
3. *All principal minors of A are nonnegative.*
4. *The real part of each nonzero eigenvalue of A is positive. In particular, if A is a singular symmetric M-matrix, it is positive semidefinite.*

In the next section we will see that irreducible singular M-matrices are of particular interest to us.

1.2.3 Markov chains

In this part we will review basic material on Markov chains. For additional information about Markov chains, see [106]. We start with a few definitions from the theory of stochastic processes.

A *row-stochastic matrix* is a nonnegative matrix in which each row sum is equal to one, and a *column-stochastic matrix* is a nonnegative matrix in which each column sum is equal to one. A *stochastic* or *random process* is a collection of random variables $\{X(t)\}_{t\in T}$, where T is a time parameter set, that take on values from the same set S, called the *state space*. The time parameter set can be discrete or continuous and we distinguish between *discrete-time stochastic processes* and *continuous-time stochastic processes*. A special class of stochastic processes are Markov chains defined below.

Definition 1.2.7. A *Markov chain* is a stochastic process with a finite or countable state space that satisfies the Markov property. A collection of random variables $\{X(t)\}_{t\in T}$ with state space $S = \{S_1, S_2, \dots\}$ satisfies the Markov property if, for all integers m and $t_0 < t_1 < \cdots < t_m < t$, the following holds true:

$$P(X(t) = S_j \mid X(t_m) = S_{i_m}, \dots, X(t_0) = S_{i_0}) = P(X(t) = S_j \mid X(t_m) = S_{i_m}).$$

That is, a Markov chain is memoryless in the sense that the state of the chain at the next time period only depends on the current time period and not on the previous states of the chain. If the time parameter set is discrete we refer to the Markov chain as a *discrete-time Markov chain (DTMC)*, otherwise it is referred to as a *continous-time Markov chain (CTMC)*. We are mainly interested in discrete-time Markov chains with finite state space. However, a new discrete-time Markov chain, called the *embedded Markov chain (EMC)*, can be defined for a CTMC; see [106, Chapter 1.4.3]. The embedded Markov chain is constructed by ignoring the time spent in any state and considering only the state transitions that are being made.

The EMC can be used to deduce many of the properties of the CTMC.

A discrete-time finite state space Markov chain can be represented by a matrix. Let $p_{ij}(t)$ denote the probability that the chain is in state S_j at time t provided it was in state S_i at time $t-1$. That is,

$$p_{ij}(t) = P(X(t) = S_j \mid X(t-1) = S_j),$$

and $p_{ij}(t)$ is called *transition probability*. The *transition probability matrix* $P(t) = [p_{ij}(t)]$ is an $n \times n$ matrix, where n is the number of states, and each row of $P(t)$ sums up to one. That is, $P(t)$ is a row-stochastic matrix for all t. If the transition probabilities do not vary over time the Markov chain is called *stationary*. In the following we will only consider stationary Markov chains, and the matrix P denotes the transition probability matrix of the chain.

A Markov chain is called *ergodic* or *irreducible* if it is possible to get from every state to every other state. The transition probability matrix of an ergodic Markov chain is irreducible. If the transition probability matrix P is also imprimitive, the Markov chain is called *periodic*. This terminology is motivated by the following. For each imprimitive matrix A with index of imprimitivity h there exists a permutation matrix π such that

$$\pi^t A \pi = \begin{bmatrix} 0 & A_{12} & 0 & \dots & 0 \\ 0 & 0 & A_{23} & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & A_{h-1,h} \\ A_{h1} & 0 & \dots & 0 & 0 \end{bmatrix}.$$

In [71] this is called the *Frobenius form*. A consequence of this form is that we can partition the chain into h clusters in such a way that it is only possible to move from

state S_i to S_j if i is in the kth cluster and j is in the $(k+1)$th cluster, or if i is in the last cluster and j is in the first cluster. Thus, each state can only be visited at periodic points in time, where the period is the index of imprimitivity of the matrix. Otherwise, if P is primitive, the Markov chain is called *aperiodic*.

A nonnegative vector $\boldsymbol{\pi} = (\boldsymbol{\pi}_1,\ \boldsymbol{\pi}_2, \ldots,\ \boldsymbol{\pi}_n)^t$ whose entries sum up to one, that is, $\sum_{i=1}^{n} \boldsymbol{\pi}_i = 1$, is called a *probability distribution vector*. A probability distribution vector can be used to describe the evolution of a Markov chain at a given time step. The ith entry in the vector $\boldsymbol{\pi}^{(k)}$ is the probability that the chain is in state S_i at time step k, that is,

$$\boldsymbol{\pi}_i^{(k)} := P(X(k) = S_i).$$

We adopt the convention that probability distribution vectors are row vectors. Given $\boldsymbol{\pi}^{(k)}$, the evolution of the chain in the next time step is given by

$$\boldsymbol{\pi}^{(k+1)} = \boldsymbol{\pi}^{(k)} P, \quad k = 0, 1, \ldots$$

Thus, $\boldsymbol{\pi}^{(k+1)}$ can be given in terms of the initial state distribution $\boldsymbol{\pi}^{(0)}$ and the transition probability matrix

$$\boldsymbol{\pi}^{(k+1)} = \boldsymbol{\pi}^{(0)} P^{k+1}, \quad k = 0, 1, \ldots$$

The vectors $\boldsymbol{\pi}^{(k)}$ are called the *transient distributions* of the Markov chain. If a transient distribution does not change from one time step to the next, it is called a *stationary probability distribution*. That is, a stationary probability distribution is a probability distribution vector $\boldsymbol{\pi}$ such that $\boldsymbol{\pi} = \boldsymbol{\pi} P$. The existence and uniqueness of a stationary probability distribution depends on the properties of the Markov chain.

Of particular interest is what happens to the Markov chain in the long run. If

$\lim_{k\to\infty} P^k$ exists, then the probability distribution

$$\boldsymbol{\pi} = \lim_{k\to\infty} \boldsymbol{\pi}^{(k)} = \boldsymbol{\pi}^{(0)} \lim_{k\to\infty} P^k$$

exists, and is called a *limiting distribution* of the Markov chain. If a limiting distribution exists, is unique and does not depend on the initial distribution, it is called the *steady-state distribution* of the Markov chain. In that case the steady-state distribution is also the only stationary distribution. A Markov chain that is ergodic and aperiodic always has a unique steady-state distribution. Note that if the chain is ergodic and periodic no limiting distribution exists, but a consequence of the Perron-Frobenius Theorem (see Theorem 1.2.3) is that the chain has a unique and positive stationary distribution. The ith entry of the unique stationary distribution $\boldsymbol{\pi}$ may be interpreted as the proportion of time that the chain spends in state S_i in the long run.

Finding the stationary distribution of a Markov chain amounts to solving the linear system $(I - P^t)\boldsymbol{\pi}^t = 0$. The matrix $A = I - P^t$ is a singular M-matrix. First note that $P \geq 0$, and $1 \in \sigma(P)$, since $P\boldsymbol{e} = 0$, where $\boldsymbol{e}$ is the vector of all ones. Furthermore, from the Gershgorin Circle Theorem it follows that $\rho(P) \leq 1$. Hence, $\rho(P) = 1$ and A is singular.

An important application of Markov chains is the PageRank algorithm [90]. The algorithm is based on the world wide web graph that models web pages as nodes and hyperlinks from one page to another as edges. Each edge $e = (u, v)$ is weighted depending on the out degree $\deg^+(u)$ of u. We set $w_e = 1/\deg^+(u)$ so that the weights of the outgoing edges of every node sum up to one. This leads to a weighted adjacency matrix H, called *hyperlink matrix*, where every row either sums up to one or is an all zero row. The idea is that a random walk on this graph models the behavior of a random surfer. At every page the surfer randomly follows one of the

hyperlinks to another page. A Markov chain based on this model is constructed, and the steady-state distribution of this chain is used to find a ranking of the web pages.

Recall that the chain needs to be ergodic and aperiodic to guarantee the existence of a steady-state distribution. With this model a surfer would get stuck whenever he reaches a web page that does not have any outlinks. These nodes in the graph are called *dangling nodes*. In addition, this model also does not incorporate the possibility that a surfer might not follow a hyperlink, but chooses a random web page as the next destination. To remedy these shortcomings, two modifications are made. The matrix H is modified to a row-stochastic matrix $\bar{H}$ as follows:

$$\bar{h}_{ij} = \begin{cases} h_{ij} & \text{if } \sum_{k=1}^{n} h_{ik} = 1, \\ 1/n & \text{otherwise,} \end{cases}$$

where n is the number of nodes in the graph. This means that for a page with no outlinks the next page is chosen at random. Note that $\bar{H}$ is a row-stochastic matrix, but it might not be irreducible. To ensure irreducibility a constant $0 < \alpha < 1$ is introduced. This constant represents the probability that a surfer does not follow a hyperlink, but instead goes to any web page with a predefined probability. Using this idea we can define a row-stochastic matrix

$$P = \alpha\bar{H} + (1 - \alpha)ev.$$

Here, v is a probability vector and e is the column vector of all ones. Often, v is chosen as $v = (1/n)e^t$, and a common value for α is 0.85. Note that if $v > 0$, then $P > 0$ which guarantees that the Markov chain is ergodic and aperiodic.

Markov chains also have applications in a variety of other fields. In biology and neuroscience, population processes or simulation of brain activities can be modeled with a Markov chain. In economics, Markov chains can be found in asset pricing

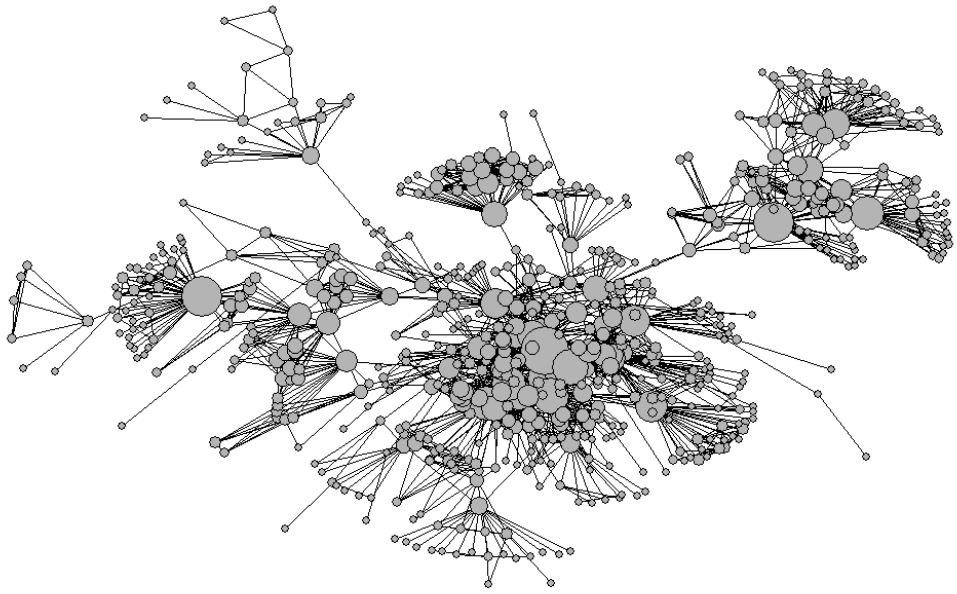


Figure 1.1: Main connected component of the social network of injecting drug users in Colorado Springs. The size of the nodes is proportional to the degree. Two drug addicts are connected if they have exchanged a needle. (Image courtesy of Ernesto Estrada [42].)

models. Queueing theory and game theory both depend heavily on Markov chain models.

1.2.4 Complex networks

There is no formal definition of a complex network. A network can broadly be described as a number of interacting entities. A complex network usually is very large, has nontrivial topological features, and is dynamically changing. A snapshot of a network at a given time can be modeled by a graph. An example of such a graph can be seen in Figure 1.1. For an excellent survey on complex networks, see [85].

Examples of complex networks are social, information, technological, and biological networks. Social networks may describe relationships between humans, such as

collaborations between scientists. An example of a very large information network is the world wide web, where interactions between webpages are given by hyperlinks. Examples of technological networks are power grids, road, airline, and railway networks. In biology, complex networks appear in the form of neural networks, or protein-protein interaction networks.

Early on, real life networks were assumed to be approximately random. More recent work has led to the realization that many real life complex network share some properties that distinguish them from random networks. In a random network nodes are randomly connected, and thus, the degree distribution of the network follows a bell-shaped curve. In 1999 Barabási and Albert analyzed the topology of a portion of the world wide web and found the degree distribution of this network to follow a power law [9]. That is, the number of nodes of a certain degree decreases exponentially with the degree. If $P(k)$ denotes the number of nodes with degree k, then $P(k) \sim k^{-\gamma}$, for some $\gamma > 0$. Thus, these networks have a fairly small average degree while the maximum degree is very large. The parameter γ typically lies in the range between 1.5 and 4. Networks with this property are called *scale-free* or *power-law* networks. The term scale-free is motivated in this case by the fact that the variance of the degree distribution has no scale and does not depend on the size of the graph. Scale-free networks appear in a variety of applications including social network analysis [6, 31, 40, 86, 87, 88], web mining [21, 32, 69, 100], and bioinformatics [82, 58]. Examples include the world wide web, and protein-protein interaction networks. For additional information refer to [8, 114].

The degree distribution of scale-free networks makes this type of network desirable in some applications. For example, scale-free networks are robust in the sense that the random removal of a node has very little likelihood of having a negative effect on the diameter of the network. On the other hand, the removal of highly connected nodes has a strong effect on the lengths of the shortest paths in the network. That

is, these networks are very sensitive to coordinated attacks [33].

Another property that is shared by many complex networks is the *small world property*, discussed for example by Watts and Strogatz [115]. Here each node is connected to any other node by a fairly short path, and the graph shows a fairly high clustering (see below). In addition, the average degree of the graph is bounded. In particular, complete graphs, where every node is connected to every other node, are not called small-world. Measures for the small world property are the diameter or mean shortest distance, and the clustering coefficient. The *mean shortest distance* is given by

$$l = \frac{1}{n(n-1)} \sum_{u,v \in V} d(u,v),$$

where n is the number of nodes in the graph. Note that the mean shortest distance is bounded above by the diameter of the graph. The *clustering coefficient* c is a measure of the density of the graph. For $v \in V$ the *local clustering coefficient* c_v is a measure of the local density in the sense that a relatively high local clustering coefficient suggests that relatively many of the direct neighbors of v in the graph are also connected. That means, two neighbors of a node v are more likely to be connected that two nodes that do not share a common neighbor. This is a feature that is not found in random graphs, where every possible pair of vertices is connected with the same likelihood. The local clustering coefficient is given by

$$c_v = \frac{2 \cdot |\{(u,w) \in E \mid u,w \in \mathcal{N}(v)\}|}{|\mathcal{N}(v)| \cdot (|\mathcal{N}(v)| - 1)},$$

where $\mathcal{N}(v)$ is the neighborhood of v given by

$$\mathcal{N}(v) = \{u \mid (u,v) \in E\}.$$

The clustering coefficient is given as the mean of the local clustering coefficient of the

nodes, that is,

$$c = \frac{1}{n} \sum_{v \in V} c_v.$$

Note that the above coefficient is sometimes referred to as *network average clustering coefficient*, and the *global clustering coefficient* is described as the ratio of three times the number of triangles in the graph to the number of connected triples in the graph (that is, paths of length 2).

Formally, a graph is said to have the small-world property, if the following three conditions hold:

1. There is a constant $C_1 \geq 0$ such that the average degree of the graph is bounded from above by $C_1 log(n)$ where n is the number of nodes.

2. There is a constant $C_2 > 0$ such that the the mean shortest distance is bounded from above by $C_2 log(n)$.

3. The clustering coefficient c is bounded away from zero as $n \to \infty$. In particular, it is higher than the clustering coefficient of a random graph.

While many scale-free networks are also small-world, the small-world property is not unique to scale-free networks. That is, there are small-world networks that are not scale-free; see [114].

1.3 Overview of iterative methods and graph partitioning

In this section we will give an overview of iterative methods and graph partitioning. Many matrices that arise from applications are large and sparse. Direct methods are often too expensive to solve linear systems with these types of matrices. In the following we will first introduce matrix splittings and stationary iterative methods, give an

overview of Krylov subspace methods, and comment on preconditioning techniques to improve the convergence of iterative methods. Lastly, we will give some background on graph partitioning, an important method for parallel computations with large and sparse matrices.

1.3.1 Matrix splittings and stationary iterative methods

Stationary iterative methods are methods to solve a linear system $Ax = b$. A stationary iterative method has the form

$$x^{(k+1)} = Tx^{(k)} + c,$$

where $k \geq 0$ is an integer, and $x^{(0)}$ is a given initial guess. T is a matrix, called the *iteration matrix*, c is a vector, and both only depend on the linear system and do not change with k.

Stationary iterative methods can be derived by a suitable *splitting* $A = M - N$, with M nonsingular. Given a splitting $A = M - N$, the stationary iterative method has the form

$$Mx^{(k+1)} = Nx^{(k)} + b,$$

or equivalently,

$$x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b.$$

For any nonsingular matrix A and stationary iteration with iteration matrix T such that $I - T$ is nonsingular, there exists a unique splitting $A = M - N$ such that $T = M^{-1}N$ [72]. The splitting is given by $M = A(I - T)^{-1}$ and $N = M - A$. If A is singular, then there exists a splitting $A = M - N$ with $T = M^{-1}N$, if $\mathcal{N}(A) = \mathcal{N}(I - T)$ [14]. Note that the splitting is not unique in the case of singular matrices.

The stationary iterative method converges for all choices of $x^{(0)}$ if either T is *zero-convergent*, that is $\lim_{k\to\infty} T^k = 0$, or if $\rho(T) = 1$ and T is *convergent*, that is, $\lim_{k\to\infty} T^k$ exists. If a splitting $A = M - N$ induces a zero-convergent iteration matrix $T = M^{-1}N$, then we say it is a *zero-convergent splitting*, if it induces a convergent iteration matrix we say it is a *convergent splitting*. The terminology regarding convergent and zero-convergent matrices is not uniform in the literature. In [83], zero-convergent matrices are called convergent, and convergent matrices are called *semiconvergent*. The following well known theorem gives an equivalent condition for the zero-convergence of a matrix.

Theorem 1.3.1. *A matrix T is zero-convergent, that is $\lim_{k\to\infty} T^k = 0$, if and only if $\rho(T) < 1$.*

We distinguish between different types of splittings. Varga [109, 110] introduced regular splittings, and the following definition gives an overview of other types of splittings.

Definition 1.3.2. Let $A = M - N$ be a splitting, and $T = M^{-1}N$ the corresponding iteration matrix. Then the splitting is called

- *regular* if $M^{-1} \geq 0$ and $N \geq 0$ [109, 110],
- *weak regular* if $M^{-1} \geq 0$ and $T \geq 0$ [16],
- *nonnegative* if $M^{-1} \geq 0$, $M^{-1}N \geq 0$, and $NM^{-1} \geq 0$ [116],
- *weak* if $T \geq 0$ [75],
- *M-splitting* if M is an M-matrix and $N \geq 0$ [101],
- *P-regular* if $M^t + N$ is positive definite [89].

Splittings can be used to determine if an iteration matrix is zero-convergent. The following gives an overview of convergence results.

Lemma 1.3.3. *Let A be nonsingular. A splitting $A = M - N$ is zero-convergent, that is, $\rho(M^{-1}N) < 1$, if*

1. *The splitting is regular and $A^{-1} \geq 0$. The spectral radius of $M^{-1}N$ is given by $\rho(M^{-1}N) = \frac{\rho(A^{-1}N)}{1+\rho(A^{-1}N)} < 1$. Conversely, if $A = M - N$ is a regular splitting with $\rho(M^{-1}N) < 1$, then $A^{-1} \geq 0$* [110].

2. *The splitting is weak regular and $A^{-1} \geq 0$. The spectral radius of $M^{-1}N$ is given by $\rho(M^{-1}N) = \frac{\rho(A^{-1}N)}{1+\rho(A^{-1}N)} < 1$. Conversely, if $A = M - N$ is a weak regular splitting with $\rho(M^{-1}N) < 1$, then $A^{-1} \geq 0$* [116].

3. *A is nonnegative, $M^{-1}N \geq 0$, and $A^{-1}N \geq 0$. The spectral radius of $M^{-1}N$ is given by $\rho(M^{-1}N) = \frac{\rho(A^{-1}N)}{1+\rho(A^{-1}N)} < 1$* [116].

4. *The splitting is weak, and $A^{-1}M \geq 0$. The spectral radius of $M^{-1}N$ is given by $\rho(M^{-1}N) = \frac{\rho(A^{-1}M)-1}{\rho(A^{-1}M)} < 1$. Conversely, if $A = M - N$ is a weak splitting with $\rho(M^{-1}N) < 1$, then $A^{-1}M \geq 0$* [104].

Splittings can also be used to determine if an iteration matrix is convergent. For A SPD, a splitting $A = M - N$ is convergent if the splitting is P-regular [67]. Note that this is a sufficient but not a necessary condition and a splitting might be convergent even if it is not P-regular [73].

Many stationary iterative methods start with a decomposition of the matrix into its diagonal part D, strictly upper triangular part U, and strictly lower triangular part L. That is, $A = D + U + L$. The splittings are then defined in terms of D, U, and L. The following are a few stationary iterative methods than can be derived in this form.

Jacobi iteration: $M = D$, $N = -(L + U)$,

Gauss-Seidel iteration: $M = L + D$, $N = -U$,

SOR iteration: $M = D + \omega L$, $N = (\omega - 1)L - U$, where $0 < \omega < 2$.

Of particular interest for us are block iterative methods. The $n \times n$ matrix A is partitioned into K blocks as follows

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \dots & A_{1K} \\ A_{21} & A_{22} & A_{23} & \dots & A_{2K} \\ A_{31} & A_{32} & A_{33} & \dots & A_{3K} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{K1} & A_{K2} & A_{K3} & \dots & A_{KK} \end{bmatrix}$$

where $A_{ii} \in \mathbb{R}^{n_i \times n_i}$, for $i = 1, \dots, K$, and $\sum_{i=1}^{K} n_i = n$. Then the block Jacobi, block Gauss-Seidel, and block SOR iterations can be derived with the same splittings as above by taking D as the block diagonal, U as the strictly upper block triangular, and L as the strictly lower block triangular part of A.

1.3.2 Krylov subspace methods

Stationary iterative methods often converge slowly. In contrast to stationary iterative methods, *non-stationary iterative methods* use steps that differ from iteration to iteration. These methods often show more potential in solving large sparse linear systems. Here we give an introduction to Krylov subspace methods, a special class of non-stationary iterative methods. For more details, refer to [98]. We start with some motivation. As before, we want to solve a large and sparse linear system $Ax = b$. Consider a simple stationary iterative method $x^{(k+1)} = (I - A)x^{(k)} + b$. Note that this stationary iterative method is induced by the splitting $A = I - (I - A)$. In particlar, the iterative method is not preconditioned. We can rewrite the iterates as $x^{(k+1)} = (I - A)x^{(k)} + b = x^{(k)} - Ax^{(k)} + b = x^{(k)} + r^{(k)}$, where $r^{(k)} = b - Ax^{(k)}$ is the

residual generated by the kth iterate. Continuing in the same fashion we get

$$x^{(k+1)} = x^{(0)} + r^{(0)} + r^{(1)} + \cdots + r^{(k)}. \tag{1.1}$$

We can also rewrite the residuals as $r^{(k)} = b - Ax^{(k)} = b - A\left[(I-A)x^{(k-1)} + b\right] = b - Ax^{(k-1)} - A(b - Ax^{(k-1)}) = (I-A)r^{(k-1)}$. Thus, if we continue the same procedure iteratively, every residual can be rewritten in terms of the residual generated by the initial guess and powers of the iteration matrix, that is $r^{(k)} = (I-A)^k r^{(0)}$. Substituting in (1.1) we get

$$x^{(k+1)} = x^{(0)} + r^{(0)} + (I-A)r^{(0)} + \cdots + (I-A)^k r^{(0)}, \tag{1.2}$$

that is, $x^{(k+1)}$ is of the form

$$x^{(k+1)} = x^{(0)} + \hat{x}^{(k+1)},$$

where $\hat{x}^{(k+1)} \in \text{span}\{r^{(0)}, Ar^{(0)}, A^2r^{(0)}, \ldots, A^k r^{(0)}\}$. One can ask, if there is a better choice for $\hat{x}^{(k+1)}$ in $\text{span}\{r^{(0)}, Ar^{(0)}, A^2r^{(0)}, \ldots, A^k r^{(0)}\}$ than the one given in (1.2). This motivates the idea behind Krylov subspace methods, where in the kth iteration a suitable $\hat{x}^{(k)}$ is chosen from a specific subspace of dimension k. The subspace used belongs to a specific kind called a Krylov subspace. Given a matrix A, and a vector v, the *rth Krylov subspace* generated by A and v is given by

$$\mathcal{K}_r(A, v) = \text{span}\{v, Av, A^2v, \ldots, A^{r-1}v\}.$$

Note that $\mathcal{K}_1(A,v) \subset \mathcal{K}_2(A,v) \subset \cdots \subset \mathcal{K}_d(A,v) = \cdots = \mathcal{K}_n(A,v)$ for some d. A general description of a Krylov subspace method is as follows. If $v, Av, A^2v, \ldots, A^{r-1}v$ are linearly independent, they form a basis of $\mathcal{K}_r(A,v)$. This basis is usually not very

good, that is, it can be ill-conditioned. For a point of view of design and implementation an orthonormal basis is often preferred. That is, a Krylov subspace method generates an orthonormal basis $\{v_1, v_2, \ldots, v_{r-1}\}$ of $\mathcal{K}_r(A, v)$. Popular methods to generate such a basis are the Arnoldi iteration [5] or the Lanczos iteration for symmetric matrices. Details of the methods can be found in Appendix A in Algorithm A.1 and A.2. Assuming exact arithmetic, after m iterations the Arnoldi iteration produces an $m \times m$ upper Hessenberg matrix

$$H_m = \begin{bmatrix} h_{11} & h_{12} & h_{13} & \ldots & h_{1m} \\ h_{21} & h_{22} & h_{23} & \ldots & h_{2m} \\ & h_{32} & h_{33} & \ldots & h_{3m} \\ & & \ddots & \ddots & \vdots \\ & & & h_{m,m-1} & h_{mm} \end{bmatrix} \tag{1.3}$$

and the vectors v_i $(i = 1, \ldots, m)$ form the columns of a $n \times m$ matrix V_m with $V_m^t V_m = I_m$, such that

$$H_m = V_m^t A V_m. \tag{1.4}$$

The Lanczos iteration uses the symmetry of the matrix A. Thus, it produces a tridiagonal matrix

$$T_m = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{m-1} & \alpha_{m-1} & \beta_m \\ & & & \beta_m & \alpha_m \end{bmatrix} \tag{1.5}$$

and the vectors v_i $(i = 1, \ldots, m)$ form the columns of a $n \times m$ matrix V_m with $V_m^t V_m = I_m$, such that

$$T_m = V_m^t A V_m. \tag{1.6}$$

In both cases the new iterate x_m will be given by $x_m = x_0 + V_m u_m$, where $u_m \in \mathcal{R}^m$ is a vector that has to be determined.

In general, a Krylov subspace method is a projection method, where two spaces $\mathcal{K}_m(A, r_0)$ and $\mathcal{L}_m$ are given, and u_m is chosen such that $r_m = b - Ax_m$ is orthogonal to $\mathcal{L}_m$, where $x_m = x_0 + V_m u_m$. We will briefly describe two Krylov subspace methods that will be used later on.

The *generalized minimum residual (GMRES) algorithm* [99] uses $\mathcal{L}_m = A \cdot \mathcal{K}_m(A, r_0)$, and can be used for any type of matrix. An $(m+1) \times m$ matrix $\bar{H}_m$ is constructed as follows

$$\bar{H}_m = \begin{bmatrix} & & H_m & \\ & & & \\ 0 & \dots & 0 & h_{m+1,m} \end{bmatrix}, \tag{1.7}$$

and satisfies

$$AV_m = V_{m+1}\bar{H}_m. \tag{1.8}$$

Thus, the residual generated by the *mth* iterate is given by

$$\begin{aligned} r_m &= b - A(x_0 + V_m u_m) \\ &= r_0 - AV_m u_m \\ &= \beta_1 v_1 - V_{m+1}\bar{H}_m u_m \\ &= V_{m+1}(\beta_1 e_1 - \bar{H}_m u_m), \end{aligned}$$

where $\beta_1 = \|r_0\|_2$, and $e_1 = (1, 0, \dots, 0)^t$. Since the columns of V_{m+1} are orthogonal, $\|r_m\|_2 = \|\beta_1 e_1 - \bar{H}_m u_m\|_2$. A unique solution for u_m that minimizes $\|\beta_1 e_1 - \bar{H}_m u_m\|_2$ can be found, since $\bar{H}_m$ has full rank. A summary of GMRES can be found in Appendix A in Algorithm A.4. One disadvantage of GMRES is that we have to save the vectors $v_1, \dots, v_m$. For large matrices the memory demand can become too large as m increases. For this purpose a restarted GMRES method, denoted GMRES(m), can be used. After m iterations the vectors $v_1, \dots, v_m$ are discarded and the methods restarts with x_m as initial guess. The details can be seen in Appendix A in Algorithm A.5. The GMRES algorithm usually shows a good convergence rate if the eigenvalues

of A are clustered away from zero. However, characterizing the rate of convergence of GMRES is a difficult matter.

The *conjugate gradient (CG) method* [57] uses $\mathcal{L}_m = \mathcal{K}_m(A, r_0)$ and can be used for symmetric (Hermitian) positive definite matrices. Similarly to the GMRES algorithm, $r_m = r_0 - AV_m u_m$. Since r_m is orthogonal to $\mathcal{K}_m(A, r_0)$, it follows that $V_m^t(r_0 - AV_m u_m) = 0$. Thus, $V_m^t A V_m u_m = T_m u_m = V_m^t r_0 = \beta_1 e_1$, and u_m can be found by

$$u_m = T_m^{-1}(\beta_1 e_1).$$

Using the LU factorization of T_m

$$\begin{aligned} T_m &= L_m U_m \\ &= \begin{bmatrix} 1 & & & & \\ \lambda_2 & 1 & & & \\ & \ddots & \ddots & & \\ & & \lambda_{m-1} & 1 & \\ & & & \lambda_m & 1 \end{bmatrix} \times \begin{bmatrix} \eta_1 & \beta_2 & & & \\ & \eta_2 & \beta_3 & & \\ & & \ddots & \ddots & \\ & & & \eta_{m-1} & \beta_m \\ & & & & \eta_m \end{bmatrix} \end{aligned}$$

a set of coupled recurrences of the form

$$\begin{aligned} x_m &= x_{m-1} + \alpha_{m-1} p_{m-1}, \\ r_m &= r_{m-1} - \alpha_{m-1} A p_{m-1}, \\ p_m &= r_m + \beta_{m-1} p_{m-1} \end{aligned}$$

can be found. Here, $\alpha_m = \eta_m + \lambda_m \beta_m$. Thus, for the conjugate gradient algorithm we do not need to store the vectors $v_1, \ldots, v_m$. A summary of CG can be found in Appendix A in Algorithm A.6. The rate of convergence of CG depends on the distribution of the eigenvalues of A. A clustering of the eigenvalues around 1, or a relatively small spectral condition number, imply fast convergence.

1.3.3 Preconditioning techniques

Here we will give an overview of preconditioning techniques. For additional background on preconditioning, see [11].

Preconditioning techniques are used to modify a linear system. The goal of this modification is to improve the performance and stability of Krylov subspace methods. A *preconditioner* is a matrix that is used in this modification. By (implicitly) applying a matrix to the linear system, the spectral properties of the coefficient matrix can be improved. For a linear system $Ax = b$ there are several ways to apply a preconditioner. The preconditioner can be applied simultaneously on both sides of the equation. This is called *left preconditioning*, and the linear system has the form

$$M^{-1}Ax = M^{-1}b,$$

where M is the preconditioner. One can also apply the preconditioner to the right of the matrix, that is, use *right preconditioning*. In this case the linear system is modified to

$$AM^{-1}y = b,$$

and the solution of the original system is given by $x = M^{-1}y$. Also possible is a *split preconditioning*

$$M_1^{-1}AM_2^{-1}y = M_1^{-1}b,$$

where the solution of the original system is given by $x = M_2^{-1}y$, and the preconditioner is $M = M_1M_2$.

The type of preconditioning that should be used depends on the underlying problem and the chosen iterative method. For example, CG requires a symmetric positive definite matrix. In general, a good preconditioner should make the system easier to solve, and the construction of the preconditioner should be relatively cheap.

A matrix splitting defines a preconditioner in a natural way. If we consider a splitting $A = M - N$, M is nonsingular and can be used as preconditioner. Thus, common preconditioners arise from the splittings that we have seen in section 1.3.1.

Jacobi preconditioner: $M = D$,

Gauss-Seidel preconditioner: $M = L + D$,

SOR preconditioner: $M = D + \omega L$, with $0 < \omega < 2$.

Other popular precoditioners correspond to the block versions of these splittings.

Another common class of preconditioners comes from incomplete LU factorization (ILU). The ILU factorization was first introduced by Varga [109], but it became popular only after the work of Meijerink and van der Vorst [77]. In general, an incomplete LU factorization of a matrix A computes a sparse lower triangular matrix $\hat{L}$ and a sparse upper triangular matrix $\hat{U}$ so that $\hat{L}\hat{U}$ is the complete LU factorization of a perturbed matrix $A+R$, where R is called the *residual matrix*. There are different forms of ILU factorizations that are based on different constraints on the residual matrix or the nonzero pattern of $\hat{L} \cdot \hat{U}$. One of the simplest versions of ILU is the zero fill-in ILU, denoted ILU(0). In this case the off-diagonal nonzero pattern of $\hat{L} \cdot \hat{U}$ is the same as the off-diagonal nonzero pattern of A. However, the zero fill-in ILU is usually fairly inaccurate. A more accurate version allows new non-zeroes based on the levels of fill-in. Initially, each pair (i, j) receives a level of fill of $\text{lev}_{ij} = 0$ if $a_{ij} \neq 0$ and $\text{lev}_{ij} = \infty$ otherwise. Each time a_{ij} is updated in step 5 in the Gaussian Elimination (see Algorithm A.3 in Appendix A) the level of fill lev_{ij} is updated to $\text{lev}_{ij} = \min\{\text{lev}_{ij}, \text{lev}_{ik} + \text{lev}_{kj} + 1\}$. An entry is only kept if lev_{ij} is smaller than a given level of fill-in. Note that with this construction the level of fill of the nonzero entries of A is kept at 0. Thus, entries in the nonzero locations of A are always kept. This ILU factorization is denoted ILU(k), were k is the level of fill-in. Incomplete LU factorizations that depend on levels of fill-in only consider the nonzero structure

of the matrix but not the numerical values. An ILU factorization that depends on the numerical values of the entries is a threshold-based scheme. The factorization is computed row by row. Entries in each row are dropped if their absolute value falls below a certain threshold. An exception are the diagonal entries, which are always kept. We refer to this factorization as ILUTH. To control the amount of memory needed, an additional dropping based on the number of entries per row can be introduced. In every row only the largest p entries from the ones that have not been dropped based on the threshold property are kept. This dual threshold-based factorization is referred to as ILUT preconditioning [97].

1.3.4 Graph partitioning

In graph partitioning problems we are interested in dividing the graph's vertex set into a fixed number of subsets so that every set has roughly the same number of vertices and the number of edges connecting vertices from different subsets is minimized. Applications for graph partitioning can be found in image processing [102], VLSI circuit layout [41], domain decomposition of unstructured graphs, and parallel computing. In parallel scientific computing, graph partitioning is used to balance the workload among the processors while at the same time limit the communication between the processors. Graph partitioning is an NP-hard problem [47], but there are good heuristics for many types of graphs. For a graph $\mathcal{G} = (V, E)$, a *k-way partitioning* is a partitioning of the vertex set V into k subsets $V_1, \ldots, V_k$, such that $V_i \cap V_j = \emptyset$, $1 \leq i < j \leq k$, and $\bigcup_{1 \leq i \leq k} V_i = V$. The number of edges whose incident vertices belong to different subsets is called *edge-cut*. Most available partitioners try to approximately minimize the edge-cut while balancing the size of every subset. There are several readily available software packages for graph partitioning. In serial mode one can choose among the following: Metis [61], Scotch [91], Party [94], and Jostle [113]. In parallel mode, well known graph partitioning libraries include ParMETIS

[65], PT-Scotch [30], Chaco [54], and Zoltan [36].

We are particularly interested in using graph partitioning methods to speed up matrix-vector multiplication for a sparse matrix A in parallel. A k-way partitioning gives rise to a symmetric permutation of A such that

$$P^tAP = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \dots & A_{1K} \\ A_{21} & A_{22} & A_{23} & \dots & A_{2K} \\ A_{31} & A_{32} & A_{33} & \dots & A_{3K} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{K1} & A_{K2} & A_{K3} & \dots & A_{KK} \end{bmatrix},$$

where A_{ii} is an $n_i \times n_i$ matrix and n_i is the size of the ith subset in the partitioning. Since in a k-way partitioning the edge-cut is minimized (subject to balanced domains), the submatrices that are not on the block diagonal are very sparse. In a parallel environment the matrix is distributed row-wise among the processors, that is, the first n_1 rows of P^tAP are assigned to the first processor, the next n_2 rows are assigned to processor two, and so on.

A k-way partitioning can also be found for weighted graphs. In that case, rather than trying to minimize the number of edges whose incident vertices belong to different subsets, the sum of the weights of the edges in the edge-cut is minimized. If there are weights on the nodes of the graph, a k-way partitioning balances the sum of the weights of the nodes in every subset instead of the number of nodes in each subset. If we weigh every vertex according to its degree (or out-degree for digraphs), we can balance the number of edges that have to be stored on every processor. Note that only integer weights may be used.

Instead of partitioning the vertex set of a graph one can also partition the edge set. The resulting partitioning of the matrix is a 2D partitioning where both the rows and columns of the matrix are distributed among the processors. This partitioning

is particularly useful for dense matrices. In our case the matrices are sparse, and we partition the nodes of the underlying graph.

Chapter 2

Restricted Additive Schwarz Methods for Markov Chains

2.1 Introduction

Domain decomposition methods are widely used for solving large-scale linear systems of equations arising from the discretization of partial differential equations (PDEs) on parallel computers [108]. Of particular importance among domain decomposition schemes are variants of the additive and multiplicative Schwarz methods, with or without overlap. These algorithms can be cast as stationary iterations associated with matrix splittings, making a purely algebraic analysis possible; see, e.g., [45, 12]. Although there are situations where Schwarz-type methods exhibit rapid convergence, they are usually more efficient when used as preconditioners for Krylov subspace methods. Our goal is to investigate the application of various additive Schwarz methods as solvers and as preconditioners for computing the stationary probability distribution vector of ergodic Markov chains with large and sparse transition matrices. In particular, we give the first analysis and implementation of the *restricted additive Schwarz* (RAS) method in the Markov chain context. The RAS method, first proposed in [28]

as a preconditioner for nonsingular linear systems $Ax = b$, is a variant of the additive Schwarz method that requires less communication and is therefore better suited for parallel implementation. Somewhat surprisingly, this method tends to exhibit convergence rates that are no worse and often better than those of the standard additive Schwarz method; see [46] for an algebraic analysis in the nonsingular case. We note that RAS is the default Schwarz preconditioner for linear systems in the popular PETSc package [7]. Additive Schwarz methods (but not RAS) for Markov chains have been analyzed in [23, 76]. The focus of these papers is primarily theoretical, and the use of Krylov subspace acceleration is not considered. Here we present some analysis and the results of computational experiments on realistic Markov models, including a selection of problems from the MARCA collection [105] and certain reliability models. Our results show that RAS preconditioning (with inexact subdomain solves) is a promising approach to the solution of large, sparse Markov chain models. This chapter is based in part on the publication [13].

2.2 Definitions and auxiliary results

First we introduce some additional terminology. For $\lambda \in \sigma(A)$ the *index* of A with respect to λ, denoted by $\mathrm{ind}_\lambda(A)$, is the smallest integer k for which $\mathcal{R}((\lambda I - A)^{k+1}) = \mathcal{R}((\lambda I - A)^k)$. Note that this is the size of the largest Jordan block associated with the eigenvalue λ in the Jordan normal form of A.

We are interested in computing the stationary probability distribution vector of finite, ergodic Markov chains. Following [16], we identify ergodic chains with irreducible ones; in particular, we allow periodic chains. The problem amounts to finding a non-trivial solution to a homogeneous system of linear equations $Ax = 0$, where A is a singular, irreducible M-matrix and $x = \boldsymbol{\pi}^t$. As we saw in Section 1.2.3, up to normalization, such solution vector x is unique. This formulation applies to Discrete-

Time (DT) as well as to Continuous-Time (CT) Markov Chains. For a DTMC, we have $A = I - P^t$ where P is the row-stochastic matrix of transition probabilities.

A matrix A is *weak semipositive* if there exists a vector $x > 0$ (i.e., an entry-wise positive vector) such that $Ax \geq 0$; see [83]. By the Perron-Frobenius Theorem (see Theorem 1.2.3) it is then clear that any singular, irreducible M-matrix is weak semipositive. The standard stationary iteration associated with the splitting $A = M - N$ is of the form $x^{(k+1)} = Tx^{(k)} + c,\ k = 0, 1, \ldots$, where $T = M^{-1}N$, $c = M^{-1}b$, and $x^{(0)}$ is arbitrary. In the cases of interest to us, T has unit spectral radius. Such a matrix is convergent if and only if $\text{ind}_1(T) = 1$ and $\lambda \in \rho(T)\backslash\{1\}$ implies $|\lambda| < 1$. For $T \geq 0$, the latter condition can be replaced with T having positive diagonal entries [3]. Also observe that in the Markov chain context we have $c = 0$, and that the sequence $\{x^{(k)}\}$ converges to a nontrivial solution of $Ax = 0$ only if $x^{(0)} \notin \mathcal{R}(M^{-1}A)$.

Finally, we will need the following result from [83, Thm. 6] in our analysis of convergence.

Theorem 2.2.1. *Let $A = M - N$ be a weak regular splitting of $A \in \mathbb{R}^{n\times n}$, and let $T = I - M^{-1}A = M^{-1}N$. If A is weak semipositive and singular, then $\rho(T) = 1$ and $ind_1(T) = 1$.*

2.3 Introduction to algebraic domain decomposition methods

Domain decomposition methods are widely used for solving large-scale linear systems of equations arising from the discretization of PDEs. The basic idea is to partition the physical domain Ω of the PDE into K subdomains, that is, $\Omega = \bigcup_{i=1}^{K} \Omega_i$. The PDE is then solved on every Ω_i with appropriate boundary conditions, and a global solution is approximated using the local solutions (see for example [29, 95, 103, 108]). General sparse linear systems do not have an associated physical domain. In this

case an algebraic approach to domain decomposition can be used [27]. That is, the domains are defined by a partitioning of the matrix. A partitioning can be achieved for example by applying a graph partitioner to the underlying graph of the matrix.

In section 1.3.1 we have seen block iterative methods such as the block Jacobi and block Gauss-Seidel methods. Those two methods are examples of algebraic domain decomposition methods. We first want to motivate why these methods can be viewed as domain decomposition methods. In the algebraic case the domain is the set of indices $S = \{1, \ldots, n\}$, where n is the size of the matrix. For the block Jacobi and block Gauss-Seidel method a domain decomposition is given by a partition of the index set into K subsets S_i, $i = 1, \ldots, K$, such that $S = \bigcup_{i=1}^{K} S_i$ and $S_i \cap S_j = \emptyset$ for $i \neq j$. Thus, if $n_i = |S_i|$, then $n = \sum_{i=1}^{K} n_i$. Assume the subsets are given by $S_i = \{s_{i1}, \ldots, s_{i,n_i}\}$, $i = 1, \ldots, K$, and let π be the matrix presentation of the permutation that maps $(1, \ldots, n)$ to $(s_{11}, \ldots, s_{1,n_1}, s_{21}, \ldots, s_{2,n_2}, \ldots, s_{K1}, \ldots, s_{K,n_K})$. Then

$$\pi A \pi^t = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \ldots & A_{1K} \\ A_{21} & A_{22} & A_{23} & \ldots & A_{2K} \\ A_{31} & A_{32} & A_{33} & \ldots & A_{3K} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{K1} & A_{K2} & A_{K3} & \ldots & A_{KK} \end{bmatrix}.$$

Thus, the block Jacobi method applied to $\pi A \pi^t$ can be written as a stationary iterative method with iteration matrix

$$T_{BJ} = I - \begin{bmatrix} A_{11}^{-1} & & \\ & \ddots & \\ & & A_{KK}^{-1} \end{bmatrix} \pi A \pi^t. \tag{2.1}$$

Each iteration of the block Jacobi method involves solving systems of the form $A_{ii} x_i = y_i$. These are the local subdomain solves.

Block Jacobi and block Gauss-Seidel iterations are often fairly slow to converge or do not converge at all, since the domains are non-overlapping and information derived from the local solves only slowly propagate to other domains. To offset this drawback, overlapping methods can be considered. If we introduce overlap the methods are referred to as *Schwarz methods*. Block Jacobi with overlap corresponds to additive Schwarz, while Gauss-Seidel with overlap describes multiplicative Schwarz. In the next section we will describe the algebraic formulation of Schwarz methods. We adopt the notation and the approach developed in [23, 76].

2.4 Algebraic formulation of Schwarz methods

As before, we consider the set of indices $S = \{1, \ldots, n\}$. To distinguish between overlapping and non-overlapping subdomains we introduce some additional notation. Let $S = \cup_{i=1}^{K} S_{i,0}$ be a partition of S into K disjoint, non-empty subsets. For each $S_{i,0}$ consider $S_{i,\delta}$ with $S_{i,0} \subseteq S_{i,\delta} \subset S$. Thus, $S = \cup_{i=1}^{K} S_{i,\delta}$ for all values of δ, with not necessarily pairwise disjoint $S_{i,\delta}$. For $\delta > 1$ this notation introduces overlap. Thus, if $n_{i,0} = |S_{i,0}|$ and $n_{i,\delta} = |S_{i,\delta}|$, then

$$n = \sum_{i=1}^{K} n_{i,0} < \sum_{i=1}^{K} n_{i,\delta}. \tag{2.2}$$

One way to find this overlap is by considering the underlying undirected graph of A. If A does not have a symmetric pattern, we use the graph of $A + A^t$ instead. To each set of nodes $S_{i,0}$ we add all nodes with distance at most δ in the underlying graph. Here, as usual, the distance is defined as the length of the shortest path in the graph connecting a given node to any node in $S_{i,0}$.

Next, we define a restriction operator $R_{i,\delta}$ from the whole space $\mathbb{R}^n$ to the subspace defined by $S_{i,\delta}$. Let $\pi_{i,\delta}$ be the matrix representation of a permutation that relabels the states in $S = \{1, \ldots, n\}$ in such a way that $S_{i,0}$ is ordered first, $S_{i,\delta} \backslash S_{i,0}$ is ordered

next, and $S \backslash S_{i,\delta}$ is ordered last. Then, the first $n_{i,0}$ rows of $\pi_{i,\delta}$ form the restriction operator $R_{i,0}$, and the restriction operator $R_{i,\delta}$ is given by the first $n_{i,\delta}$ rows of $\pi_{i,\delta}$. $R_{i,\delta}$ is an $n_{i,\delta} \times n$ matrix such that $R_{i,\delta}\pi_{i,\delta}^t = [I_{i,\delta}|0]$, where $I_{i,\delta}$ is the $n_{i,\delta} \times n_{i,\delta}$ identity matrix. For example if $n = 7$, $S_{i,0} = \{3, 5\}$, and $S_{i,\delta} = \{3, 5, 1\}$, then $R_{i,0}$ and $R_{i,\delta}$ are given by

$$R_{i,0} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad \text{and} \quad R_{i,\delta} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{2.3}$$

The matrix

$$A_{i,\delta} = R_{i,\delta} A R_{i,\delta}^t \tag{2.4}$$

is the restriction of A to the subspace corresponding to $S_{i,\delta}$. Thus, $A_{i,\delta}$ is an $n_{i,\delta} \times n_{i,\delta}$ principal submatrix of A. In the case of ergodic Markov chains, A is an irreducible singular M-matrix. Since principal minors of irreducible singular M-matrices are nonsingular M-matrices, $A_{i,\delta}$ is a nonsingular M-matrix [106]. The additive Schwarz method can be given in the form of a stationary iteration, $x^{(k+1)} = T_{AS,\delta} x^{(k)} + c$, where

$$T_{AS,\delta} = I - \sum_{i=1}^{K} R_{i,\delta}^t A_{i,\delta}^{-1} R_{i,\delta} A \tag{2.5}$$

and c is a certain vector. Note that for the solution of Markov chains c will be zero. The additive Schwarz method might not converge. To ensure convergence, a damping parameter θ with $0 < \theta \leq 1$ is introduced. The damped additive Schwarz method has an iteration matrix of the form

$$T_{\theta_{AS,\delta}} = I - \theta \sum_{i=1}^{K} R_{i,\delta}^t A_{i,\delta}^{-1} R_{i,\delta} A; \tag{2.6}$$

see [29, 38, 39, 50, 103]. The damping parameter depends on the chosen overlap. Note

that if we use the same notation for the block Jacobi method, the iteration matrix for block Jacobi is of the form

$$T_{BJ} = I - \theta \sum_{i=1}^{K} R_{i,0}^t A_{i,0}^{-1} R_{i,0} A. \tag{2.7}$$

The additive Schwarz method can be used as a preconditioner for a Krylov subspace method. In that case

$$M_{AS,\delta} = \sum_{i=1}^{K} R_{i,\delta}^t A_{i,\delta}^{-1} R_{i,\delta} \tag{2.8}$$

is used as preconditioner and no damping parameter is needed.

In the kth iteration, the additive Schwarz method follows the following steps for every subdomain:

I. Restrict $y^{(k)} = Ax^{(k-1)}$ to the space defined by $S_{i,\delta}$. That is, compute $y_{i,\delta} = R_{i,\delta} y^{(k)}$.

II. Solve a linear system of the form $A_{i,\delta} x_{i,\delta} = y_{i,\delta}$.

III. Prolongate $x_{i,\delta}$ to the whole space. That is, compute $\hat{y}^{(k)} = R_{i,\delta}^t x_{i,\delta}$.

In a parallel implementation, the matrix is distributed among the processors according to the non-overlapping domains. That is, processor p holds row i if $i \in S_{p,0}$. Thus, the overlap of a domain, $S_{p,\delta} \backslash S_{p,0}$, is stored on a different processor, and communication between the processors is needed in step I and III of the above method. In 1999 Cai and Sarkis described a modification of this method that converges as fast as additive Schwarz but avoids communication in step III [28]. Instead of prolongating $\hat{y}^{(k)}$, only the part of $\hat{y}^{(k)}$ that corresponds to the local non-overlapping domain $S_{i,0}$ is prolongated. This method was named restricted Additive Schwarz method. The advantage of the method in the parallel case is that no communication is needed in step III.

For the restricted additive Schwarz iteration the prolongation operator $R_{i,\delta}^t$ will be replaced by a prolongation operator that does not consider overlap. Define $E_{i,0}$ as

$$E_{i,0} = R_{i,0}^t R_{i,0} = \pi_{i,0}^t \begin{bmatrix} I_{i,0} & 0 \\ 0 & 0 \end{bmatrix} \pi_{i,0}. \tag{2.9}$$

$E_{i,0}$ is an $n \times n$ diagonal matrix with a one on the diagonal for every row where $R_{i,0}^t$ has a one, and zeros otherwise. Thus, $\sum_{i=1}^{K} E_{i,0} = I$. Note that if we define $E_{i,\delta} = R_{i,\delta}^t R_{i,\delta}$, then

$$\sum_{i=1}^{K} E_{i,\delta} = \begin{bmatrix} q_1 & & & \\ & q_2 & & \\ & & \ddots & \\ & & & q_n \end{bmatrix} \leq qI, \tag{2.10}$$

where q_i denotes the number of subdomains that contain index i, and $q = \max\{q_i\}$. The parameter q plays a role in the convergence theory for the damped additive Schwarz method. In fact, it has been shown that for M-matrices the method converges for $\theta \leq 1/q$ [46].

We can use the matrix $E_{i,0}$ to define the "restricted" operator $\tilde{R}_{i,\delta}$ as

$$\tilde{R}_{i,\delta} = R_{i,\delta} E_{i,0}. \tag{2.11}$$

Using the same example as in (2.3), the restricted operator is given by

$$\tilde{R}_{i,\delta} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{2.12}$$

That is, all rows corresponding to overlap are set to zero. Thus, $\tilde{R}_{i,\delta}$ is a $n_{i,\delta} \times n$ matrix of the form

$$\tilde{R}_{i,\delta} = \begin{bmatrix} I_{i,0} & 0 \\ 0 & 0 \end{bmatrix} \pi_{i,0}. \tag{2.13}$$

With this notation, the restricted additive Schwarz method has the form of a stationary iteration, $x^{(k+1)} = T_{RAS,\delta} x^{(k)} + c$, where

$$T_{RAS,\delta} = I - \sum_{i=1}^{K} \tilde{R}_{i,\delta}^t A_{i,\delta}^{-1} R_{i,\delta} A. \tag{2.14}$$

For the restricted additive Schwarz method no damping parameter is needed. The restricted additive Schwarz preconditioner is given by

$$M_{RAS,\delta}^{-1} := \sum_{i=1}^{K} \tilde{R}_{i,\delta}^t A_{i,\delta}^{-1} R_{i,\delta}. \tag{2.15}$$

We will use the restricted additive Schwarz preconditioner as a right preconditioner for GMRES. That is, we will use GMRES to solve $AM_{RAS,\delta}^{-1} y = 0$, with $y = M_{RAS,\delta} x$.

With the same notations we can write the block Gauss-Seidel, multiplicative Schwarz, and restricted multiplicative Schwarz methods as follows:

$$T_{GS} = \prod_{i=K}^{1} (I - R_{i,0}^t A_{i,0}^{-1} R_{i,0} A), \tag{2.16}$$

$$T_{MS,\delta} = \prod_{i=K}^{1} (I - R_{i,\delta}^t A_{i,\delta}^{-1} R_{i,\delta} A), \tag{2.17}$$

$$T_{RMS,\delta} = \prod_{i=K}^{1} (I - \tilde{R}_{i,\delta}^t A_{i,\delta}^{-1} R_{i,\delta} A). \tag{2.18}$$

2.5 Nonsingularity of the RAS preconditioner

The notation $M_{RAS,\delta}^{-1}$ only makes sense if $\sum_{i=1}^{K} \tilde{R}_{i,\delta}^t A_{i,\delta}^{-1} R_{i,\delta}$ is a nonsingular matrix. Guaranteeing the nonsingularity of this matrix is not entirely straightforward. A

simple solution would be to construct the RAS preconditioner not using the original singular matrix A, but a slightly perturbed matrix $\tilde{A} = A + \epsilon I$ for some $\epsilon > 0$. Since this matrix is guaranteed to be a nonsingular M-matrix, it follows from the general results in [46] that the corresponding RAS preconditioner is well-defined, i.e., nonsingular. The resulting ϵ-dependent RAS preconditioner would then be applied, of course, to the original (singular) system, $Ax = 0$. This approach, however, besides not being very elegant requires the introduction of an additional parameter ϵ.

An alternative approach is to work with the original matrix A, but to impose some conditions on the decomposition (graph partition) that will automatically produce a nonsingular RAS preconditioner. We begin by writing

$$\pi_{i,\delta} A {\pi_{i,\delta}}^t = \begin{bmatrix} A_{i,\delta} & K_{i,\delta} \\ L_{i,\delta} & A_{\neg i,\delta} \end{bmatrix}, \tag{2.19}$$

where $A_{\neg i,\delta}$ is the principal submatrix of A "complementary" to $A_{i,\delta}$. Thus, $A_{\neg i,\delta}$ is a nonsingular M-matrix. Let $D_{\neg i,\delta} = \mathrm{diag}(A_{\neg i,\delta})$ and note that since $A_{\neg i,\delta}$ is an irreducible nonsingular M-matrix, $D_{\neg i,\delta}$ is nonnegative with positive diagonal entries. We construct a matrix $M_{i,\delta}$ corresponding to $R_{i,\delta}$ as follows:

$$M_{i,\delta} = \pi_{i,\delta}^t \begin{bmatrix} A_{i,\delta} & 0 \\ 0 & D_{\neg i,\delta} \end{bmatrix} \pi_{i,\delta}\,. \tag{2.20}$$

Since $A_{i,\delta}$ is nonsingular and $D_{\neg i,\delta}$ has positive entries on the diagonal, $M_{i,\delta}$ is invertible. It has been proven by Frommer and Szyld [46] that

$$\tilde{R}_{i,\delta}^t A_{i,\delta}^{-1} R_{i,\delta} = E_{i,0} M_{i,\delta}^{-1}. \tag{2.21}$$

Thus, $M_{RAS,\delta}^{-1}$ can be written as

$$M_{RAS,\delta}^{-1} = \sum_{i=1}^{K} E_{i,0} M_{i,\delta}^{-1}, \tag{2.22}$$

and we will use this representation in our analysis of the restricted additive Schwarz method for Markov chains in the remainder of this section and the next.

We will make use of the following lemma from [66].

Lemma 2.5.1. *Assume that A is an irreducible singular M-matrix, the splittings $A = M_l - N_l$, $l = 1, \ldots K$, are regular, and that E_l, $l = 1, \ldots, K$ are nonnegative diagonal matrices such that $\sum_{l=1}^{K} E_l = I$. If there exist indices $1 \leq i,\ j \leq n$ such that the (i,j)-entry in N_l is nonzero for each $l = 1, \ldots, K$, then for sufficiently small $\epsilon > 0$, the splittings*

$$A + \epsilon E_{i,j} = M_l - (N_l - \epsilon E_{i,j}), \quad l = 1, \ldots, K$$

are regular splittings of a nonsingular M-matrix. Thus $M = \sum_{l=1}^{K} E_l M_l^{-1}$ is nonsingular regardless of the choice of the weighting matrices E_l.

We emphasize that in the above lemma, the matrices M_l are independent of ϵ. We can now state sufficient conditions that guarantee the nonsingularity of the RAS preconditioner.

Theorem 2.5.2. *Given any set of overlapping subdomains $S_{1,\delta}, \ldots, S_{K,\delta}$, there exists a set of overlapping subdomains $\hat{S}_{1,\delta}, \ldots, \hat{S}_{K,\delta}$ such that $S_{i,0} \subseteq \hat{S}_{i,\delta} \subseteq S_{i,\delta}$ and the preconditioning matrix $M_{RAS}^{-1} = \sum_{l=1}^{K} E_{l,\delta} M_{l,\delta}^{-1}$ corresponding to $\hat{S}_{1,\delta}, \ldots, \hat{S}_{K,\delta}$ is nonsingular.*

Proof. Since A is irreducible, there exist $i \in S_{1,0}$ and $j \notin S_{1,0}$ such that the (i,j)-entry of A is nonzero. Now we construct the sets $\hat{S}_{1,\delta}, \ldots, \hat{S}_{K,\delta}$ from the sets $S_{1,\delta}, \ldots, S_{K,\delta}$ as follows. Set $\hat{S}_{1,\delta} = S_{1,\delta} \backslash \{j\}$. Let s be the index of the set such that $j \in S_{s,0}$. Set

$\hat{S}_{s,\delta} = S_{s,\delta}\backslash\{i\}$. Also, we remove i and j from all other sets, that is $\hat{S}_{l,\delta} = S_{l,\delta}\backslash\{i,j\}$ for $l \in \{2,\ldots,K\}\backslash\{s\}$. Consider the matrix splittings $A = M_{l,\delta} - N_{l,\delta}$ corresponding to the new subdomains. With

$$A = \pi_{l,\delta}^t \begin{bmatrix} A_{l,0} & K_{l,\delta} \\ L_{l,\delta} & A_{\neg l,\delta} \end{bmatrix} \pi_{l,\delta} \qquad \text{and} \qquad M_{l,\delta} = \pi_{l,\delta}^t \begin{bmatrix} A_{l,0} & 0 \\ 0 & D_{\neg l,\delta} \end{bmatrix} \pi_{l,\delta}\,,$$

it follows that

$$N_{l,\delta} = \pi_{l,\delta}^t \begin{bmatrix} 0 & -K_{l,\delta} \\ -L_{l,\delta} & -A_{\neg l,\delta} + D_{\neg l,\delta} \end{bmatrix} \pi_{l,\delta}.$$

Since A has nonpositive off-diagonal entries it follows that $N_{l,\delta} \geq 0$. Since $M_{l,\delta}^{-1} \geq 0$, the splittings $A = M_{l,\delta} - N_{l,\delta}$ are regular. With the updated set of overlapping subdomains $\hat{S}_{1,\delta},\ldots,\hat{S}_{K,\delta}$ the conditions for the previous lemma are satisfied. If the (i,j)-entry of A is given by $a_{i,j}$ ($\neq 0$ by the choice of i and j), then the (i,j)-entry in $N_{l,\delta}$, $l = 1,\ldots,K$, is given by $-a_{i,j}$, since i and j are never in the same subdomain. It follows from the previous lemma that $M = \sum_{l=1}^{K} E_{l,\delta} M_{l,\delta}^{-1}$ is nonsingular. □

It is worth stressing that the conditions expressed in the foregoing result are sufficient, but not necessary. In practice we have not found any cases where a given overlapping set of subdomains produced a singular preconditioner, and therefore we have not needed to modify any subdomains so as to satisfy the theorem's conditions. On the other hand, we have been unable to prove nonsingularity of the RAS preconditioner without any additional assumptions. Since nonsingularity of the preconditioner is an essential requirement for the analysis in the next sections, we henceforth assume that the conditions in the previous theorem are satisfied.

In addition, we would like to note that the nonsingularity of the additive Schwarz preconditioner for Markov chains $M_{AS,\delta}$ has also been proven with some restrictions on the choice of overlap. Bru, Pedroche, and Szyld showed that a $M_{AS,\delta}$ is nonsingular if the overlap is chosen from consecutive domains [23]. That is, the overlapping domains

$S_{i,\delta}$ are chosen such that $S_{i,\delta} \cap S_{i+k,\delta} = \emptyset$, for $k \geq 2$.

2.6 Properties of the RAS splitting

In this section we analyze the convergence of the restricted additive Schwarz iteration for irreducible singular M-matrices. In particular, we will formulate sufficient conditions under which the stationary RAS iteration converges to a nontrivial solution of the linear system $Ax = 0$ for almost all initial guesses—i.e., conditions under which the iteration matrix of the RAS method is convergent.

Recall that the iterative method $x^{(k+1)} = Tx^{(k)}$ is convergent if the $\lim_{k\to\infty} T^k$ exists. As is well known (see, e.g., [83]), a matrix T is convergent if and only if (i) $\rho(T) \leq 1$, and if $\rho(T) = 1$ then (ii) $1 \in \sigma(T)$ and there are no other eigenvalues on the unit circle, and (iii) $\mathrm{ind}_1 T = 1$, i.e., T has only Jordan blocks of size 1 associated with the eigenvalue 1. Note that given an iteration matrix $T = M^{-1}N = I - M^{-1}A$ associated with a splitting $A = M - N$ with A singular, one necessarily has $1 \in \sigma(T)$ and thus $\rho(T) \geq 1$.

We begin our discussion of convergence by first showing that the iteration matrix of the RAS method satisfies $\rho(T) = 1$ and $\mathrm{ind}_1 T = 1$. Because of Theorem 2.2.1, it suffices to show that the RAS splitting is weak regular.

Theorem 2.6.1. *Let A be an irreducible singular M-matrix, and let $M^{-1}_{RAS,\delta}$ be given by (2.22). Then the splitting $A = M_{RAS,\delta} - N_{RAS,\delta}$ is weak regular.*

Proof. With $E_{i,0} \geq 0$ and $M^{-1}_{i,\delta} \geq 0$ it follows that $M^{-1}_{RAS,\delta} = \sum_{i=1}^{K} E_{i,0} M^{-1}_{i,\delta} \geq 0$. Let

$N_{i,\delta} = M_{i,\delta} - A$, then

$$\begin{aligned} M_{RAS,\delta}^{-1} N_{RAS,\delta} &= I - \sum_{i=1}^{K} \left(E_{i,0} M_{i,\delta}^{-1} A \right) \\ &= I - \sum_{i=1}^{K} E_{i,0} + \sum_{i=1}^{K} E_{i,0} M_{i,\delta}^{-1} N_{i,\delta} \\ &= \sum_{i=1}^{K} E_{i,0} M_{i,\delta}^{-1} N_{i,\delta}. \end{aligned}$$

The last equality holds since $\sum_{i=1}^{K} E_{i,0} = I$. Since $M_{i,\delta}^{-1} N_{i,\delta} \geq 0$ and $E_{i,0} \geq 0$, we obtain $M_{RAS,\delta}^{-1} N_{RAS,\delta} \geq 0$. □

The previous result, combined with Theorem 2.2.1, implies that the iteration matrix T of the RAS method is semiconvergent. For it to be convergent, we additionally need to show that $\lambda = 1$ is the only eigenvalue of T on the unit circle. As mentioned in section 2, this is equivalent to T having no zero diagonal entries, see [3]. In general, T may not have all positive entries along the diagonal. There are essentially two ways to enforce this condition. One approach (see, e.g., [23, 76]) is to slightly modify the splittings of A so as to ensure positive diagonals. Adding a small positive constant to the diagonal entries of $M_{i,\delta}$ makes the splittings $A = M_{i,\delta} - N_{i,\delta}$ regular, and the diagonal entries of $T_{i,\delta} = M_{i,\delta}^{-1} N_{i,\delta}$ positive [76]. The following two results show that with this modification, the RAS iteration matrix $T = M_{RAS,\delta}^{-1} N_{RAS,\delta}$ has positive diagonal entries.

Proposition 2.6.2. *Let A be an irreducible singular M-matrix, and let $M_{RAS,\delta}$ and $M_{i,\delta}$ be defined as in (2.22) and (2.20). Set $N_{i,\delta} = M_{i,\delta} - A$ and $N_{RAS,\delta} = M_{RAS,\delta} - A$. Assume that $M_{i,\delta}^{-1} N_{i,\delta}$ has positive entries on the main diagonal. Then, $T = M_{RAS,\delta}^{-1} N_{RAS,\delta}$ has positive entries on the main diagonal.*

Proof. As seen in the proof of Theorem 2.6.1,

$$M_{RAS,\delta}^{-1} N_{RAS,\delta} = \sum_{i=1}^{K} E_{i,0} M_{i,\delta}^{-1} N_{i,\delta}.$$

For a row in which $E_{i,0}$ is zero, the diagonal entry of $E_{i,0} M_{i,\delta}^{-1} N_{i,\delta}$ is zero. For a row in which $E_{i,0}$ has a one, the diagonal entry of $E_{i,0} M_{i,\delta}^{-1} N_{i,\delta}$ equals the diagonal entry of $M_{i,\delta}^{-1} N_{i,\delta}$ which is positive by assumption. Since $\sum_{i=1}^{K} E_{i,0} = I$, for each row exactly one of the $E_{i,0}$ has a one. Thus, the diagonal entries of $M_{RAS,\delta}^{-1} N_{RAS,\delta}$ are positive. □

The following proposition describes the modification that is needed to ensure that $M_{i,\delta}^{-1} N_{i,\delta}$ has positive entries along its diagonal. The result can be found in [76].

Proposition 2.6.3. *Let $B \geq 0$, $B^t e = e$. Let $K > 1$ be a positive integer and let $\alpha_1, \ldots, \alpha_K$ be any positive real numbers. Let $A = I - B = M_{i,\delta} - N_{i,\delta}$, $i = 1, \ldots, K$, be defined by*

$$M_{i,\delta} = \pi_{i,\delta}^{t} \begin{bmatrix} \alpha_i I + A_{i,\delta} & 0 \\ 0 & \alpha_i I + D_{\neg i,\delta} \end{bmatrix} \pi_{i,\delta},$$

and $N_{i,\delta} = M_{i,\delta} - A$ where $\pi_{i,\delta}$, $A_{i,\delta}$, and $D_{\neg i,\delta}$ are defined as before. Then, the splittings $A = M_{i,\delta} - N_{i,\delta}$ are regular, and the diagonals of $T_{i,\delta} = M_{i,\delta}^{-1} N_{i,\delta}$ are positive, for $i = 1, \ldots, K$.

Another way to ensure convergence was already proposed in [83]. The idea is to replace T by $T_\alpha = \alpha I + (1 - \alpha) T$ for some $\alpha \in (0, 1)$. Such a matrix is guaranteed to be convergent if $T \geq 0$, $\rho(T) = 1$ and $\mathrm{ind}_1(T) = 1$; indeed, such a T_α has positive diagonal entries.

In concluding this section, we emphasize that such modifications are needed only when the RAS iteration is used as a stationary iterative method. No modifications are necessary if Krylov subspace acceleration is used, since in this case the convergence of T is not required.

2.7 Extension to inexact solves

In this section we extend the convergence results to the case of inexact local solves. Instead of solving the linear systems $A_{i,\delta}y_i = z_i$ exactly, we want to approximate the matrices $A_{i,\delta}$ by $\hat{A}_{i,\delta}$ so that the systems $\hat{A}_{i,\delta}y_i = z_i$ are easier to solve. The diagonal modifications mentioned above can also be regarded as a type of inexact solve. The following propositions shows that under certain assumptions the restricted additive Schwarz method is also convergent in the case of inexact local solves.

Proposition 2.7.1. *If $\hat{A}_{i,\delta}$ is a nonsingular M-matrix with $\hat{A}_{i,\delta} \geq A_{i,\delta}$ and*

$$\hat{M}_{i,\delta} = \pi_{i,\delta}^t \begin{bmatrix} \hat{A}_{i,\delta} & 0 \\ 0 & D_{\neg i,\delta} \end{bmatrix} \pi_{i,\delta},$$

then the splittings $A = \hat{M}_{i,\delta} - \hat{N}_{i,\delta}$ are weak regular.

Proof. First, note that $\hat{M}_{i,\delta}^{-1}$ is nonnegative, since $\hat{A}_{i,\delta}$ has a nonnegative inverse. Second, we will consider $\hat{M}_{i,\delta}^{-1}\hat{N}_{i,\delta}$ and show that it is nonnegative. Recall that A can be written as

$$A = \pi_{i,\delta}^t \begin{bmatrix} A_{i,\delta} & K_{i,\delta} \\ L_{i,\delta} & A_{\neg i,\delta} \end{bmatrix} \pi_{i,\delta},$$

and it follows that

$$\hat{M}_{i,\delta}^{-1}\hat{N}_{i,\delta} = I - \pi_{i,\delta}^t \begin{bmatrix} \hat{A}_{i,\delta}^{-1}A_{i,\delta} & \hat{A}_{i,\delta}^{-1}K_{i,\delta} \\ D_{\neg i,\delta}^{-1}L_{i,\delta} & D_{\neg i,\delta}^{-1}A_{\neg i,\delta} \end{bmatrix} \pi_{i,\delta}.$$

With $\hat{A}_{i,\delta} \geq A_{i,\delta}$, and the fact that both matrices are M-matrices it follows that $\hat{A}_{i,\delta}^{-1} \leq A_{i,\delta}^{-1}$ and $I \geq \hat{A}_{i,\delta}^{-1}A_{i,\delta}$. Furthermore, $I - D_{\neg i,\delta}^{-1}A_{\neg i,\delta}$ is nonnegative, since $D_{\neg i,\delta} = \mathrm{diag}(A_{\neg i,\delta})$. With $\hat{A}_{i,\delta}^{-1}$, $D_{\neg i,\delta}^{-1} \geq 0$, and $K_{i,\delta}$, $L_{i,\delta} \leq 0$ we can conclude that $\hat{M}_{i,\delta}^{-1}\hat{N}_{i,\delta}$ is nonnegative and that the splittings $A = \hat{M}_{i,\delta} - \hat{N}_{i,\delta}$ are weak regular. □

Consider the splitting $A = \hat{M}_{RAS,\delta} - \hat{N}_{RAS,\delta}$ with $\hat{M}_{RAS,\delta} = \sum_{i=1}^{K} E_{i,0}\hat{M}_{i,\delta}^{-1}$. As seen in the proof of Theorem 2.6.1, $\hat{M}_{RAS,\delta}^{-1}\hat{N}_{RAS,\delta} = \sum_{i=1}^{K} E_{i,0}\hat{M}_{i,\delta}^{-1}\hat{N}_{i,\delta}$. From the proposition above it follows that $A = \hat{M}_{RAS,\delta} - \hat{N}_{RAS,\delta}$ is weak regular as well. Following the idea of adding a positive value to the diagonal of each $\hat{M}_{i,\delta}$ as seen in Section 2.6 will lead to positive diagonal entries in $\hat{M}_{RAS,\delta}^{-1}\hat{N}_{RAS,\delta}$. Alternatively, the corresponding iteration matrix can be shifted and scaled using a parameter $\alpha \in (0, 1)$, as shown at the end of Section 2.6. With either of these modifications, the restricted additive Schwarz method with inexact local solves associated with matrices $\hat{A}_{i,\delta} \geq A_{i,\delta}$ is convergent.

We are particularly interested in the case where an incomplete LU factorization is used for approximating the solution of $A_{i,\delta}y_i = z_i$. Meijerink and van der Vorst [77] showed that for an M-matrix A and every zero pattern $Q \subset P_n = \{(i, j) \,|\, 1 \leq i, j \leq n\}$ not containing the diagonals (i, i), there is a unique lower triangular matrix L with unit diagonal, a unique upper triangular matrix U with positive diagonals, and a matrix R with

$$\begin{aligned} l_{ij} &= 0 \text{ if } (i,\ j) \in Q, \\ u_{ij} &= 0 \text{ if } (i,\ j) \in Q, \\ r_{ij} &= 0 \text{ if } (i,\ j) \notin Q, \end{aligned}$$

such that the splitting $A = LU - R$ is regular. Note that Q can always be chosen such that $\text{diag}(R) = 0$. So, in our case $A_{i,\delta} = L_{i,\delta}U_{i,\delta} - R_{i,\delta}$ and we use $\hat{A}_{i,\delta} = L_{i,\delta}U_{i,\delta}$ instead. Since the splitting $A_{i,\delta} = L_{i,\delta}U_{i,\delta} - R_{i,\delta}$ is regular, $R_{i,\delta}$ is nonnegative and $\hat{A}_{i,\delta} \geq A_{i,\delta}$. Varga showed in [110] that if A is a nonsingular M-matrix and B is a matrix satisfying

$$\begin{aligned} a_{ij} \leq b_{ij} \leq 0, &\quad \text{for } i \neq j \text{ and} \\ 0 < a_{ii} \leq b_{ii}, & \end{aligned}$$

then B is also a nonsingular M-matrix. If the drop tolerance in the ILU factorization is sufficiently small, $\hat{A}_{i,\delta}$ will satisfy both of the above conditions. In this case $\hat{A}_{i,\delta}$ is a

nonsingular M-matrix and the previous discussion shows that replacing $A_{i,\delta}$ with $\hat{A}_{i,\delta}$ will preserve the convergence results. Again, this is only needed if the inexact RAS method is used as a stationary iteration. If Krylov acceleration is used, the above conditions are not needed for convergence.

2.8 Two-level method

The rate of convergence of the additive Schwarz method may be improved with the help of a 'coarse grid' correction, leading to a two-level approach. This correction can be formed algebraically, without any reference to an actual grid; for simplicity, however, we will use the same terminology in use in the field of numerical PDEs. The correction can be applied in two ways, additively or multiplicatively; see for example [12, 29, 45, 95, 103].

In this section we describe the use of a coarse grid correction for the restricted additive Schwarz method in the case of irreducible Markov chains. Let P_0, R_0 denote prolongation and restriction operators (respectively), and let $A_0 = R_0 A P_0$ be the resulting coarse representation of A. Let $\hat{A}_0$ denote an invertible approximation of A_0. The additive variant of the two-level method corresponds to the iteration matrix

$$T = I - \theta(P_0 \hat{A}_0^{-1} R_0 + M_{RAS}^{-1})A,$$

where $\theta > 0$ is a parameter. For the multiplicative variant the iteration matrix is given by

$$T = (I - P_0 \hat{A}_0^{-1} R_0 A)(I - M_{RAS}^{-1} A).$$

Here we consider the additive approach only. For use as a preconditioner, the parameter θ has no effect and can be set to 1.

One way to construct a coarse grid problem is to use independent sets of the

underlying undirected graph of the matrix and the Schur complement. A subset $\mathcal{F}$ of the nodes of a graph is an *independent set* if no pair of nodes from $\mathcal{F}$ is connected by an edge. We proceed as follows. First a maximal independent set $\mathcal{F}$ in the underlying graph of A is found. An independent set $\mathcal{F}$ is *maximal* if there is no independent set that has $\mathcal{F}$ as a subset. We can partition A such that

$$\pi A \pi^t = \begin{bmatrix} A_{\mathcal{FF}} & A_{\mathcal{FC}} \\ A_{\mathcal{CF}} & A_{\mathcal{CC}} \end{bmatrix}.$$

Since $\mathcal{F}$ is an independent set, $A_{\mathcal{FF}}$ is a diagonal matrix. Note that $A_{\mathcal{FF}}$ is non-singular, since A has positive entries along its diagonal. We define the restriction operator

$$R_0 = [-A_{\mathcal{CF}} A_{\mathcal{FF}}^{-1} \quad I]$$

and the prolongation operator as

$$P_0 = \begin{bmatrix} -A_{\mathcal{FF}}^{-1} A_{\mathcal{FC}} \\ I \end{bmatrix}.$$

With these operators $A_0 = R_0(\pi A \pi^t) P_0$ is given by the Schur complement

$$A_0 = A_{\mathcal{CC}} - A_{\mathcal{CF}} A_{\mathcal{FF}}^{-1} A_{\mathcal{FC}}.$$

Note that A_0 is a singular, irreducible M-matrix [15, Lemma 1], and that an ILU factorization can be used to inexactly solve systems of the form $A_0 z = y$. In other words, we set $\hat{A}_0 = \hat{L}_0 \hat{U}_0$, where $\hat{L}_0$ and $\hat{U}_0$ are incomplete factors of A_0, which can be assumed to be nonsingular; see [25] for some sufficient conditions that guarantee the existence of nonsingular ILU factors of singular M-matrices. In our experiments, the ILU factors were always nonsingular.

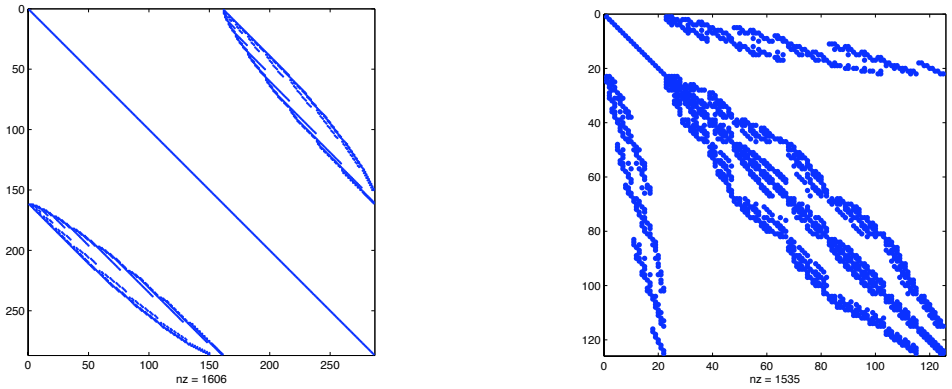


Figure 2.1: Nonzero patterns of matrices used to construct the coarse grid correction. Left: sparsity pattern of the matrix permuted by Luby's maximal independent set reordering. Right: sparsity pattern of the corresponding Schur complement, also permuted by Luby's maximal independent set reordering. The original matrix comes from the *ncd* family.

To reduce the size of the coarse grid correction we apply the above scheme twice. That is, a maximal independent set in the underlying graph of A_0 is found and the Schur complement of A_0 is used. We refer to Figure 2.1 for an example. A few numerical results using our two-level method with additive correction using the above independent set approach are given in Section 2.10. A greedy algorithm was used to find the maximal independent sets [74].

For a discussion of different coarsening strategies within algebraic multi-level methods for Markov chain problems we refer the reader to, e.g., [35, 112].

2.9 Description of the test problems

For our numerical experiments we used the generator matrices of some Markov chain models provided in the MARCA (MARkov Chain Analyzer) collection [105]. These matrices are infinitesimal generators of CTMCs. For our purposes we converted them to the form $A = I - P^t$, with P row-stochastic. A corresponds to the embedded Markov chain. In Table 2.1 the dimensions, the number of non-zeroes and the chosen

Table 2.1: Properties of the generator matrices

Matrix	Number of rows/cols	Number of nonzeroes	Parameter
ncd(07)	62,196	420,036	$N = 70$
ncd(10)	176,851	1,207,051	$N = 80$
ncd(15)	585,276	4,028,076	$N = 150$
ncd(20)	1,373,701	9,494,101	$N = 200$
ncd(25)	2,667,126	18,480,126	$N = 250$
mutex(09)	65,535	1,114,079	$N = 15$, $R = 15$
mutex(12)	263,950	4,031,310	$N = 20$, $R = 8$
tcomm(47)	603,201	3,009,401	$K_1 = 200$, $K_2 = 3000$
tcomm(49)	1,204,301	6,012,601	$K_1 = 300$, $K_2 = 4000$
twod(06)	66,049	263,169	$N_x = 256$, $N_y = 256$
twod(10)	263,169	1,050,625	$N_x = 512$, $N_y = 512$
twod(12)	591,361	2,362,369	$N_x = 768$, $N_y = 768$
twod(14)	1,050,625	4,198,401	$N_x = 1024$, $N_y = 1024$
twod(17)	2,563,201	10,246,401	$N_x = 1600$, $N_y = 1600$
$\text{reliab}_1(m)$	m^2	$5m^2 - 4m$	$\lambda_1 = 2$, $\lambda_2 = 3$, $\mu_1 = 5$, and $\mu_2 = 6$
$\text{reliab}_2(m)$			$\lambda_1 = 2$, $\lambda_2 = 0.9$, $\mu_1 = 0.5$, $\mu_2 = 6$

parameters of our selected test matrices are shown. Note that some of the larger matrices are only used in parallel mode. Each matrix is named by its family and an index. The matrices from the *ncd* family come from a queuing network of the central server type. The size of the matrices depends on the number of terminals N. The *ncd* matrices are structurally symmetric and nearly completely decomposable. A description of the model can be found in [92]. The matrices from the *twod* family come from a 2D epidemic model. Only transitions to the South, East and North-West are permitted, therefore the matrices are structurally non-symmetric. The size of the matrices depends on the size of the dimensions; N_x is the size of the first dimension and N_y is the size of the second dimension. The matrices from the *mutex* family come from a resource sharing model. N processes alternate between a sleeping state and a resource using state. The number of processes that may concurrently use a resource is limited to R where $1 \leq R \leq N$. The matrices from the *mutex* family are structurally symmetric. The model has been discussed in [44]. The matrices from the *tcomm* family come from a telecommunication model that has been used to determine the effect of impatient telephone customers on a computerized telephone exchange. The model includes two server stations and the parameters K_1 and K_2

$$
Q = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \\ 16 \end{array}
\begin{array}{c}
\begin{array}{cccccccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \end{array} \\
\begin{pmatrix}
* & 3\lambda_2 & & & 3\lambda_1 & & & & & & & & & & & \\
\mu_2 & * & 2\lambda_2 & & & 3\lambda_1 & & & & & & & & & & \\
 & 2\mu_2 & * & \lambda_2 & & & 3\lambda_1 & & & & & & & & & \\
 & & 3\mu_2 & * & & & & 3\lambda_1 & & & & & & & & \\
\mu_1 & & & & * & 3\lambda_2 & & & 2\lambda_1 & & & & & & & \\
 & \mu_1 & & & \mu_2 & * & 2\lambda_2 & & & 2\lambda_1 & & & & & & \\
 & & \mu_1 & & & 2\mu_2 & * & \lambda_2 & & & 2\lambda_1 & & & & & \\
 & & & \mu_1 & & & 3\mu_2 & * & & & & 2\lambda_1 & & & & \\
 & & & & 2\mu_1 & & & & * & 3\lambda_2 & & & \lambda_1 & & & \\
 & & & & & 2\mu_1 & & & \mu_2 & * & 2\lambda_2 & & & \lambda_1 & & \\
 & & & & & & 2\mu_1 & & & 2\mu_2 & * & \lambda_2 & & & \lambda_1 & \\
 & & & & & & & 2\mu_1 & & & 3\mu_2 & * & & & & \lambda_1 \\
 & & & & & & & & 3\mu_1 & & & & * & 3\lambda_2 & & \\
 & & & & & & & & & 3\mu_1 & & & \mu_2 & * & 2\lambda_2 & \\
 & & & & & & & & & & 3\mu_1 & & & 2\mu_2 & * & \lambda_2 \\
 & & & & & & & & & & & 3\mu_1 & & & 3\mu_2 & *
\end{pmatrix}
\end{array}.
$$

Figure 2.2: Transition rate matrix for the reliability model.

are the maximum number of customers permitted in each station. The matrices are close to being structurally symmetric. The model has been described for example in [92]. A description of all of the models can also be found in [106]. Refer to [105] for information on the transition probabilities for all of the models.

We also run some test with matrices that arise from a reliability problem. We consider a simple reliability model with two different classes of machines. We assume that each class has the same number of machines. Each machine is subject to breakdown and a subsequent repair. A state is completely specified by the ordered pair (n_1, n_2), where n_1 denotes the number of intact machines of the first class and n_2 denotes the number of intact machines of the second class. Thus, if there are m machines in each class, the total number of possible states is $(m+1)^2$. We order these states such that state (i, j) has index $(m+1)(m-i)+m-j+1$. The times between successive breakdowns and successive repairs are both exponentially distributed. The breakdown rates of class 1 machines and class 2 machines are respectively λ_1 and λ_2. Similarly, the repair rates of the two classes of machines are μ_1 and μ_2. The transition rate matrix for the described reliability model is then given by Q; see Figure 2.2. Here $m = 3$ machines per class where used. The diagonal elements indicated by asterisks are the negated sums of the off-diagonal elements in their corresponding rows. We

note that the stationary distribution for these models is known analytically (i.e., a closed form solution exist), which makes them well suited for testing codes and for checking the accuracy of computed solutions. Our goal is to solve the singular system $\boldsymbol{\pi}Q = 0$ subject to $\|\boldsymbol{\pi}\|_1 = 1$ (the normalization condition). However, one can solve the equivalent system $-Q^t\boldsymbol{\pi}^t = 0$ instead. Here, the coefficient matrix is a singular irreducible M-matrix and the theory developed in the previous sections applies. From this point on, we let $A = -Q^t$ and $x = \boldsymbol{\pi}^t$, so that we can use the notation introduced in the earlier sections. We tested two different reliability matrices. We choose different parameters to vary the difficulty. The first corresponds to a reliability problem with parameters $\lambda_1 = 2$, $\lambda_2 = 3$, $\mu_1 = 5$, and $\mu_2 = 6$, while the second corresponds to a reliability problem with parameters $\lambda_1 = 2$, $\lambda_2 = 0.9$, $\mu_1 = 0.5$, and $\mu_2 = 6$. In serial mode the largest reliability matrix tested has $1,440,000$ rows and columns, and $7,195,200$ non-zeros. In parallel mode the largest reliability matrix tested has $7,840,000$ rows and columns, and $39,188,800$ non-zeros.

Table 2.2: Subdominant eigenvalue, reliability models

Matrix	subdominant eigenvalue
$\text{reliab}_1(100)$	0.9807
$\text{reliab}_1(400)$	0.9952
$\text{reliab}_1(700)$	0.9972
$\text{reliab}_1(1000)$	0.9981
$\text{reliab}_1(1200)$	0.9984
$\text{reliab}_2(100)$	0.9894
$\text{reliab}_2(400)$	0.9974
$\text{reliab}_2(700)$	0.9985
$\text{reliab}_2(1000)$	0.9989
$\text{reliab}_2(1200)$	0.9991

In Table 2.2 we report the value of the subdominant eigenvalue (i.e., the second largest eigenvalue) of the stochastic matrices for the embedded Markov chains describing the reliability models. The fact that the gap between the dominant eigenvalue

Table 2.3: ILUTH results, reliability models

Matrix	ILUTH time	Its
$\text{reliab}_1(100)$	0.23	32
$\text{reliab}_1(400)$	10.0	43
$\text{reliab}_1(700)$	70.1	50
$\text{reliab}_1(1000)$	195.	> 500
$\text{reliab}_1(1200)$	325.	> 500

$\lambda = 1$ and the subdominant eigenvalue shrinks as the number of states is increased indicates that these problems become increasingly difficult as their size grows. In Table 2.3 we report results for a few reliability models from the first class using GMRES(50) with the drop tolerance-based ILUTH preconditioner [106]. In all cases, the drop tolerance was 10^{-3}. Note the high construction costs and very slow convergence behavior in the case of sufficiently large problems.

2.10 Numerical Experiments

2.10.1 Serial results

In this section we provide the results of our numerical experiments on a single compute node. The primary goal of these tests is to study the convergence behavior of RAS and of RAS-preconditioned GMRES for large Markov chain problems as a function of algorithmic parameters like the amount of overlap and the number of subdomains, and to carry out a comparison with standard additive Schwarz (AS). We also present some experiments with the two-level method.

The implementation was done in Matlab 7.8.0 on a 2.3 GHz Intel Core i7 Processor with 4GB main memory. We performed a large number of tests on numerous matrices. Here we present a selection of these results to show our overall findings. For additive Schwarz one has different options to handle local results in the overlapping regions.

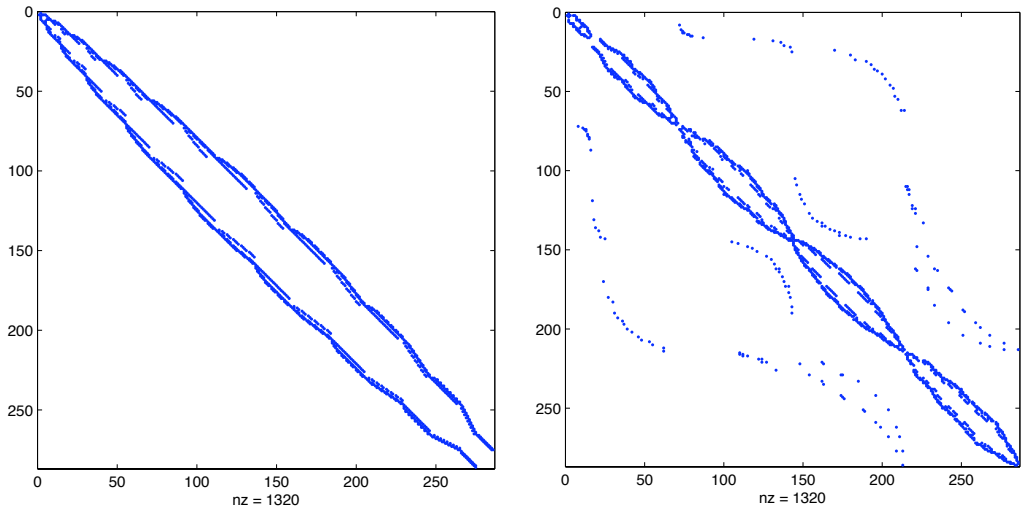


Figure 2.3: Left: nonzero pattern of an *ncd* matrix. Right: nonzero pattern of the same matrix when four subdomains are used and the matrix is reordered accordingly by Metis.

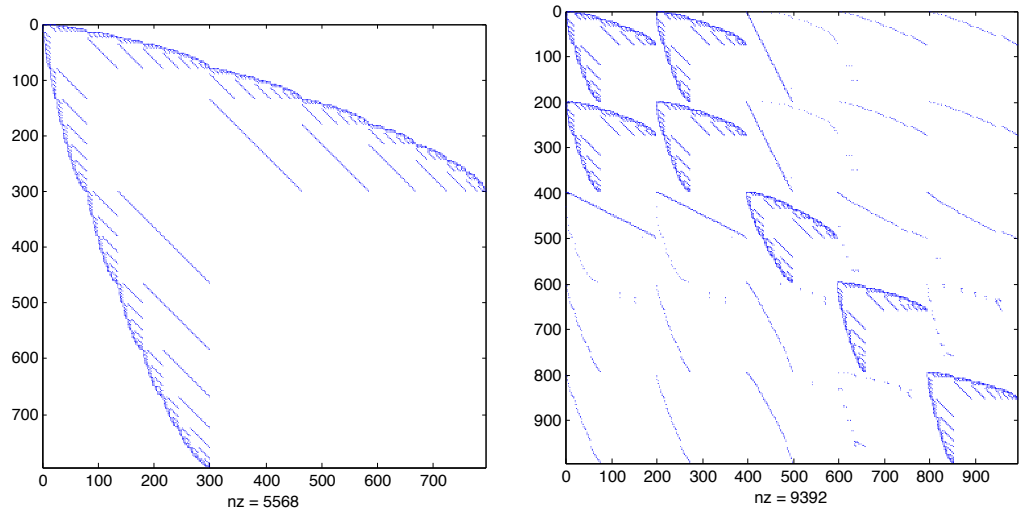


Figure 2.4: Left: nonzero pattern of a *mutex* matrix. Right: nonzero pattern of the same matrix when four subdomains are used and the matrix is reordered accordingly by Metis.

One can simply add the different results from the subdomains in the overlapping regions, or combine them in a weighted way. We used the average of the results on the overlapping regions as it reduces the number of iterations needed. The Krylov method used was GMRES [99] with restart $m = 50$. As initial vector we used a unit basis vector.

For the partitioning of our matrices we used the Metis library [61]. In our tests we used $K = 2$, 4, 8, 16, 32, 64, and 80 domains. Note that a partitioning that only considers the non-zero pattern of the matrix but not the actual coefficients might not be a good domain decomposition for Schwarz methods [111]. For this reason we also use Metis with weights on the edges of the underlying graph of $P + P^t$. The edge weights may only be integers. Each edge $e = (i, j)$ is given the weight $w_e = 1 + 1000 \cdot \lfloor p_{ij} + p_{ji} \rfloor$. The idea is that two nodes that are connected by an edge with a high weight are more likely to be in the same non-overlapping domain than nodes that are connected by an edge with a very low weight. Now, the number of edges between the non-overlapping domains might be higher, but these edges have relatively low weights and may be not as important. See Figure 2.3 for the sparsity pattern of an *ncd* matrix and Figure 2.4 for the sparsity pattern of a *mutex* matrix before and after the reordering induced by Metis.

Metis requires the matrices to be structurally symmetric. Therefore we applied Metis to the underlying graph of $P + P^t$ for the *twod* matrices. The amount of overlap was chosen according to the distance to the domain. For a small choice of overlap, we chose all vertices within a distance of 1 in the underlying graph, that is, all vertices that are connected to a vertex in the domain. The *mutex* matrices have a large separator set and choosing the overlap in the described way leads to domains that are close to the size of the matrix. For these matrices we restricted the total size of a domain, that is the partition and the overlap, by $\frac{4}{3} \cdot \frac{n}{K}$, where n is the size of the matrix and K is the number of domains. For a large choice of overlap we chose all vertices that lie within a distance of ten in the underlying graph. In a parallel implementation it is important to reduce the amount of communication between the processors. Therefore, we also run experiments where the overlap was only chosen from neighboring domains. That is, the first domain can only overlap with the second domain, the second with the first and the third, and so on. Thus, the domains $S_{i,\delta}$

satisfy $S_{i,\delta} \cap S_{i+k,\delta} = \emptyset$, for $k \geq 2$.

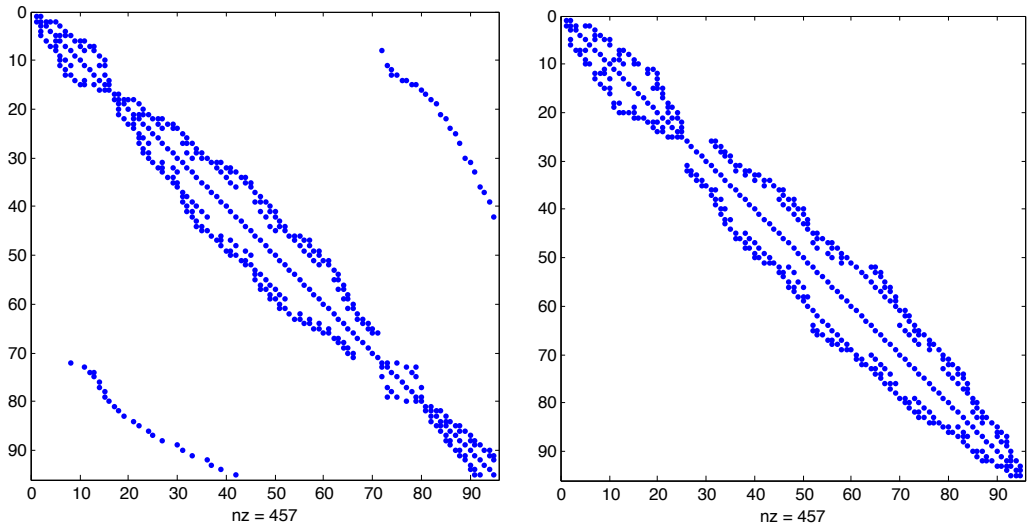


Figure 2.5: Left: nonzero pattern of a block of an *ncd* matrix. Right: nonzero pattern of the same matrix after reordering with reverse Cuthill–McKee.

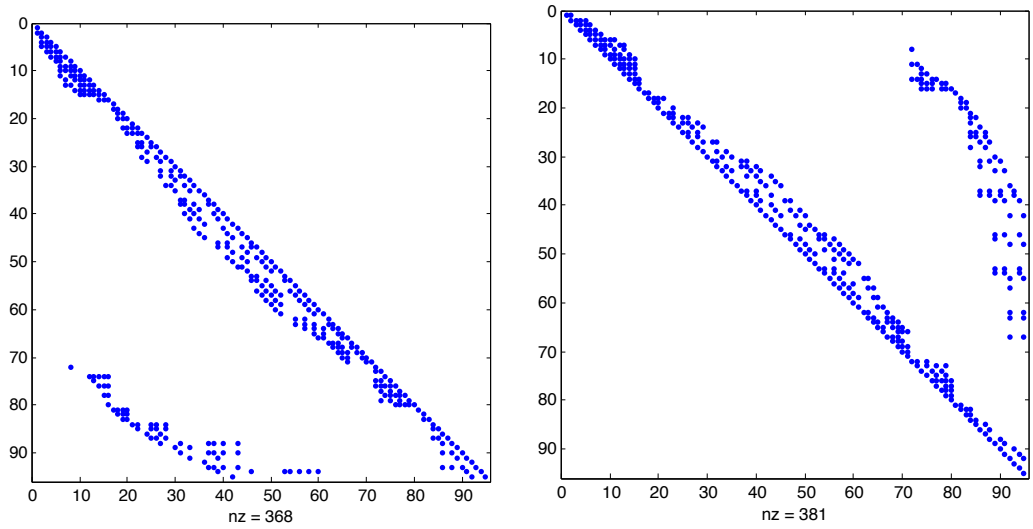


Figure 2.6: Nonzero pattern of the ILU factorization of a block of an *ncd* matrix without RCM reordering. Left: nonzero pattern L. Right: nonzero pattern of U.

To reduce fill-in during the ILU factorization we first reorder each block with a symmetric reverse Cuthill–McKee reordering [34]. See Figure 2.5 for the sparsity pattern of a block of an *ncd* matrix before and after the reordering. The effect on the

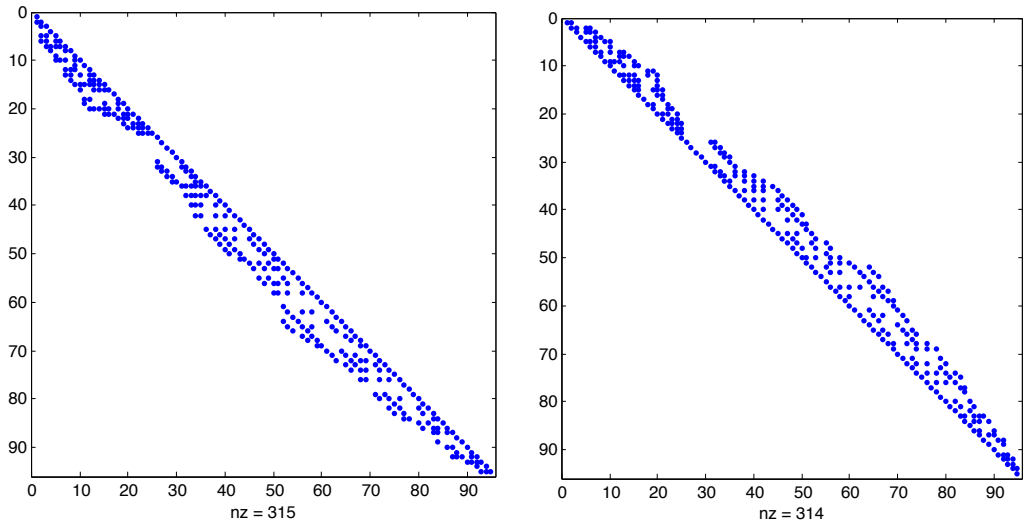


Figure 2.7: Nonzero pattern of the ILU factorization of a block of an *ncd* matrix with RCM reordering. Left: nonzero pattern L. Right: nonzero pattern of U.

sparsity pattern of the ILU factorization can be seen in Figure 2.6 and Figure 2.7. We factored the blocks using an incomplete LU factorization with threshold parameter $\tau = 10^{-3}$ for the *mutex* and *reliability* matrices and $\tau = 10^{-4}$ for the other matrices. Compared to exact LU factorization the timings and storage requirements were dramatically reduced, while the number of iterations remained essentially unchanged. In Table 2.4 we compare the time needed for factoring the domains of a matrix of the *ncd* family with ILU and LU. The amount of fill-in is also given.

We found that RAS usually performs as well or better than the standard additive Schwarz preconditioner in terms of number of iterations, whereas in terms of total runtime the standard additive Schwarz preconditioner tends to perform slightly better. This can be explained by observing that in RAS we have to keep track of the nodes that are in the non-overlapping domains and the ones that are in the overlap. This needs some extra "book-keeping" which might lead to RAS performing slightly slower than AS for the same number of iterations, at least with our implementation. Note, however, that AS requires more interprocessor communication and thus we expect it to be less efficient than RAS in a parallel implementation with many processors. Our

Table 2.4: Time in seconds required for factoring the diagonal blocks. The resulting fill-in is given as the ratio of the number of non zeros in $L \cdot U$ and the matrix. K is the number of domains. The matrix is of the *ncd* family and has 12341 rows and columns.

	K	without RCM		with RCM	
		time	fill-in	time	fill-in
ILU	2	0.15	2.41	0.15	1.51
	4	0.13	2.35	0.13	1.53
	8	0.14	2.44	0.14	1.62
	16	0.13	2.39	0.13	1.66
	32	0.11	2.29	0.11	1.59
	64	0.09	2.17	0.09	1.49
	80	0.08	2.08	0.08	1.44
	96	0.08	2.05	0.08	1.43
LU	2	26.06	32.78	4.49	12.34
	4	21.09	28.05	2.05	8.50
	8	13.49	21.20	1.39	6.58
	16	6.95	14.58	0.79	4.76
	32	3.47	9.89	0.46	3.47
	64	1.41	6.37	0.24	2.52
	80	1.02	5.51	0.20	2.27
	96	0.76	4.88	0.16	2.05

results are presented in the following tables. We compare both runtime and iteration counts for GMRES preconditioned with RAS and AS, and the stationary versions of RAS and AS. For every example we also compare the effect on the restriction of domains from which overlap may be chosen. In the "no restr."case overlap may be chosen from any of the domains, while in the "consec."case overlap may only be chosen from neighboring domains. Results in bold indicate the best total runtime, that is the time to construct the preconditioner plus the time taken by the Krylov method. We first give a comparison of the construction time for the complete LU factorization vs. the incomplete LU factorization, and compare the change in number of iterations. The results can be seen in Table 2.5. In Section 2.10.1.1 we use the unweighted Metis version and a small amount of overlap, in Section 2.10.1.2 a large amount of overlap

was used. The effect of using weighted graphs during the graph partitioning are given in Section 2.10.1.3 Results using the two-level method are given in Section 2.10.1.4.

Our results indicate many favorable properties of the RAS preconditioner for the numerical solution of Markov chains. In particular, we focused on the rate of convergence depending on the size of the problem, the number of domains used, and the amount of overlap chosen. First of all, note that the time needed to compute the incomplete LU factorizations is considerably lower than the time needed for the complete LU factorizations (see Table 2.5). The difference is most notable the ncd(07) example, where the construction time for two domains dropped from about 28 seconds to 0.20 seconds. Also note that there is only a small increase in the number of iterations. Due to a cheaper application of the incomplete LU factorization the time to solve the problems was in most cases lower when an incomplete LU was used, even though the number of iterations was slightly higher.

In general, the time required to construct the preconditioner, when incomplete LU factorizations are used, is very small. Also note that with an increased number of domains the size of the submatrices that require an incomplete LU factorization decreases, and thus the construction time for the preconditioner decreases. This can be best observed in the largest example of the first reliability problem ($\text{reliab}_1(1200)$, see Table 2.13). Here the time to construct the preconditioner can be cut from 40 seconds for two domains to about 8 seconds for 80 domains. Furthermore, the construction expenses mainly consist of the work needed to compute the incomplete LU factorizations as the time spent to find the non-overlapping domains with the Metis software is almost negligible. The task of computing several incomplete LU factorizations of different submatrices is trivial to parallelize, and we expect an even better construction time in a parallel setting. Another observation that can be made from our results is that there is only a slight increase in the number of iterations when the size of the problem is increased. This property is particularly desirable for precondi-

tioners. The choice of number of domains also has an influence on the convergence. As the number of domains increases so does the number of iterations. This behavior is not surprising, since the propagation of information across the computational domain takes longer for an increased number of subdomains. Note, however, that for several matrices the increase in the number of iterations is moderate. As mentioned earlier, the construction time is faster if more domains are used. In most cases the time saved during the construction is smaller than the increase in the time required by the Krylov method, and thus the overall time needed increases with the number of domains. This behavior seems to be reversed for some very large problems. Here the overall time needed is reduced for a large number of domains, since the decreased construction time outweighs the larger time needed to solve the linear system. See for example the results for larger problems from the reliability models, Table 2.13. The time to construct the preconditioner was cut from about 40 to 8 seconds, while the time required to solve the system increased from 13 to 30 seconds as the number of domains increased from 2 to 80 subdomains.

Another aspect that is worth mentioning is that for most of the problems the number of iterations increased quite a bit if overlap was only chosen from neighboring domains (an exception are the *tcomm* matrices, see Table 2.10). This effect is more noticeable for a large number of domains, as the restriction on the overlap is more rigid. For example for the first reliability model of size $n = 360,000$ the number of iterations increased from 40 to about 80 when overlap was restricted to neighboring domains and 80 domains were used, see Table 2.12 The increase in number of iterations is not surprising as less information among the subdomains is shared. An advantage of choosing overlap only among neighboring subdomains lies in the reduced communication in a parallel environment.

Also interesting is the effect of the amount of overlap on the convergence. While larger overlap leads to an increase in construction time, the number of iterations is

decreased. For a large problem from the first reliability model, 43 iterations were needed for 80 subdomains (see Table 2.13), while for a large amount of overlap only 22 iterations were needed (see Table 2.22). The construction time only increased slightly from 4.49 to 6.16 seconds, but the time required for the solving decreased sharply from about 18 to 8 seconds.

Next, we comment on our results if edge weights were used in Metis. While the usage of edge weights neither hurts or helps in general, for some examples there can be a big improvement, most notably for the *ncd* matrices, especially when overlap is chosen from consecutive subdomains; compare Table 2.6 and 2.7 with Table 2.24 and 2.25.

We also compared GMRES with RAS (AS) preconditioner with the stationary version of RAS (AS). While the number of iterations is generally higher without the acceleration of a Krylov subspace solver, the cost per iteration is much cheaper. Sometimes the stationary versions outperform the preconditioned Krylov subspace method in terms of solution times. This happens for the *ncd* matrices, see Table 2.6 and Table 2.7. Also, for the *tcomm* matrices the number of iterations did not improve when using a Krylov subspace accelerator, see Table 2.10. For other problems the Krylov subspace method showed a better convergence, and the number of iterations increased more slowly than for the iterative version as the number of subdomains increased. Also note that in any case the stationary version seems to be more sensitive to a smaller choice of overlap. See for example for the first reliability model (Tables 2.12 and 2.13). For the second reliability model the stationary methods failed to converge within 500 iterations (see Tables 2.14, 2.15, 2.23).

Finally, we comment on the results obtained with the two-level method. The number of iterations could be reduced (except for the *tcomm* examples) with our two-level approach. Unfortunately, the increased construction time in the two-level method almost outweighs the time saved during the iterative solve. In most cases we

could only observe a very small reduction in the overall time needed, if any. In general, a better choice of overlap or domain decomposition seems to be more effective than a two-level method. We should mention that a number of attempts were made to find more cost-effective coarsening strategies, but we were unable to obtain better results.

Table 2.5: Complete LU factorization vs. incomplete LU factorization.

Matrix	K	LU					ILU				
		GMRES precond. with					GMRES precond. with				
		constr	AS		RAS		constr	AS		RAS	
			solve	it	solve	it		solve	it	solve	it
ncd(07)	2	27.40	0.92	8	0.80	7	0.17	0.14	12	0.14	12
	4	16.05	0.73	8	0.65	7	**0.16**	**0.14**	**12**	0.15	12
	8	9.43	0.60	8	0.53	7	0.18	0.14	12	0.15	12
	16	5.50	0.62	10	0.63	10	0.18	0.18	14	0.17	13
	32	3.09	0.47	10	0.48	10	0.18	0.18	13	0.18	13
	64	1.95	0.56	15	0.57	15	0.19	0.30	19	0.27	18
	80	1.69	0.51	14	0.44	12	0.20	0.28	18	0.26	16
twod(06)	2	6.09	0.41	6	0.36	5	1.80	0.15	8	0.13	7
	4	2.59	0.31	7	0.30	7	0.46	0.12	9	0.12	9
	8	1.55	0.28	9	0.28	9	**0.36**	**0.14**	**11**	**0.14**	**11**
	16	1.04	0.34	13	0.35	13	0.31	0.19	14	0.20	14
	32	0.64	0.35	16	0.35	16	0.25	0.24	17	0.23	16
	64	0.45	0.49	23	0.51	22	0.22	0.35	23	0.39	23
	80	0.38	0.50	25	0.62	25	0.19	0.40	26	0.48	26
tcomm(46)	2	28.69	1.49	3	1.42	3	3.26	0.43	4	0.43	4
	4	33.41	1.34	3	1.31	3	2.55	0.35	4	0.32	4
	8	29.83	1.20	3	1.20	3	2.32	0.33	4	0.32	4
	16	19.36	0.98	3	0.94	3	2.01	0.35	4	0.33	4
	32	9.01	0.75	3	0.74	3	1.56	0.34	4	0.34	4
	64	6.13	0.63	3	0.59	3	1.42	0.40	4	0.35	4
	80	5.00	0.59	3	0.55	3	**1.30**	**0.37**	**4**	**0.37**	**4**
reliab1(300)	2	5.49	0.75	11	0.76	11	0.76	0.28	15	0.28	15
	4	3.13	0.56	11	0.57	11	**0.64**	**0.28**	**15**	**0.28**	**15**
	8	1.87	0.85	21	0.84	20	0.53	0.56	26	0.63	26
	16	1.18	0.85	25	0.91	24	0.39	0.62	30	0.76	30
	32	0.80	0.85	27	0.92	27	0.37	0.72	33	0.93	33
	64	0.58	0.87	29	0.95	29	0.32	0.81	35	1.01	34
	80	0.55	0.84	30	1.06	30	0.28	0.85	36	1.04	35
reliab2(300)	2	6.60	0.81	10	0.89	10	**0.52**	**0.35**	**18**	**0.35**	**18**
	4	3.24	0.75	14	0.76	14	0.61	0.41	20	0.41	20
	8	1.90	0.89	22	0.98	22	0.40	0.81	38	1.08	38
	16	1.21	1.38	38	1.66	38	0.34	0.95	42	1.27	42
	32	0.85	1.01	32	1.11	31	0.31	1.98	84	2.45	82
	64	0.58	1.17	39	1.50	39	0.27	2.21	90	2.83	90
	80	0.53	1.08	36	1.32	36	0.27	2.19	87	2.69	85

2.10.1.1 Results with incomplete LU factorization and small amount of overlap

Table 2.6: Results for the *ncd* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A small amount of overlap was chosen.

Overlap	K	constr.	GMRES precond. with				without Krylov accel.			
			AS		RAS		AS		RAS	
			solve	it	solve	it	solve	it	solve	it
ncd(07)										
	2	0.17	0.14	12	0.15	12	0.21	29	0.25	29
	4	**0.17**	0.14	12	0.15	12	**0.13**	**18**	0.16	19
	8	0.19	0.14	12	0.15	12	0.14	18	0.15	18
no restr.	16	0.18	0.19	14	0.17	13	0.21	24	0.21	23
	32	0.18	0.18	13	0.19	13	0.22	25	0.24	24
	64	0.19	0.32	19	0.31	18	0.49	44	0.48	43
	80	0.19	0.28	18	0.27	16	0.41	34	0.32	25
	2	0.17	0.14	12	0.15	12	0.22	29	0.23	29
	4	0.17	0.29	22	0.29	22	1.72	225	2.10	254
	8	0.20	0.33	26	0.35	26	>3	>500	>4	>500
consec.	16	0.17	0.35	27	0.36	26	2.26	284	2.57	293
	32	0.15	0.46	31	0.48	30	>4	>500	>4	>500
	64	0.16	0.79	46	0.88	46	>4	>500	>5	>500
	80	0.18	0.86	49	0.99	48	>4	>500	>5	>500
ncd(10)										
	2	0.56	0.37	11	0.40	11	0.34	17	0.38	17
	4	**0.52**	0.35	10	0.38	10	**0.25**	**12**	0.28	12
	8	0.49	0.40	11	0.43	11	0.54	25	0.59	25
no restr.	16	0.48	0.41	12	0.43	12	0.45	21	0.46	20
	32	0.50	0.51	14	0.54	14	0.47	21	0.50	20
	64	0.50	0.65	17	0.61	15	0.61	25	0.67	25
	80	0.50	0.65	16	0.61	15	0.61	24	0.65	24
	2	0.57	0.37	11	0.40	11	0.35	17	0.39	17
	4	0.51	0.50	14	0.53	14	1.39	72	1.55	72
	8	0.49	0.53	15	0.60	15	1.34	68	1.52	68
consec.	16	0.44	1.03	27	1.11	27	>9	>500	>10	>500
	32	0.44	1.34	32	1.41	32	>10	>500	>11	>500
	64	0.42	2.42	48	2.53	48	>10	>500	>12	>500
	80	0.42	1.96	40	2.23	39	>11	>500	>12	>500

Table 2.7: Results for the *ncd* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A small amount of overlap was chosen.

Overlap	K	constr.	GMRES precond. with				without Krylov accel.			
			AS		RAS		AS		RAS	
			solve	it	solve	it	solve	it	solve	it
ncd(15)										
	2	2.88	1.63	9	1.75	9	2.28	24	2.48	24
	4	1.68	1.57	9	1.63	9	1.19	13	1.27	13
	8	1.69	2.06	11	1.86	10	1.29	14	1.35	14
no restr.	16	1.80	1.94	11	2.04	11	1.48	16	1.59	16
	32	1.57	2.42	13	2.50	13	1.66	18	1.69	17
	64	1.61	3.13	16	2.70	14	1.96	20	2.10	20
	80	1.78	3.13	16	3.02	15	1.88	20	1.90	19
	2	2.94	1.62	9	1.75	9	2.34	24	2.50	24
	4	**1.74**	1.68	9	1.70	9	**0.97**	**10**	0.99	10
	8	1.58	2.93	15	3.09	15	>47	>500	>51	>500
consec.	16	1.82	4.01	19	3.86	18	>47	>500	>51	>500
	32	1.55	7.26	29	7.44	29	>47	>500	>50	>500
	64	1.53	6.11	26	6.36	26	>47	>500	>51	>500
	80	1.45	13.07	42	13.39	42	>48	>500	>52	>500
ncd(20)										
	2	5.69	5.17	10	5.72	10	3.12	11	3.14	10
	4	4.80	4.84	10	4.64	9	3.00	12	2.99	11
	8	**4.84**	4.49	10	5.14	10	**2.72**	**12**	2.88	12
no restr.	16	4.53	5.49	12	4.94	11	3.06	14	3.33	14
	32	4.58	5.88	13	6.20	13	3.75	17	3.78	16
	64	4.12	7.78	16	7.65	15	4.04	18	4.40	18
	80	4.07	7.57	16	7.01	15	4.47	20	4.25	18
	2	5.87	4.88	10	5.35	10	2.92	11	3.05	10
	4	5.00	8.82	17	8.99	17	>122	>500	>127	>500
	8	4.13	7.93	17	8.57	17	>110	>500	>118	>500
consec.	16	4.08	6.08	14	6.49	14	17.32	81	20.44	88
	32	3.89	12.32	24	12.96	24	>107	>500	>116	>500
	64	3.80	15.28	27	16.46	27	>114	>500	>123	>500
	80	3.82	16.71	28	17.35	28	110.21	473	114.41	452

Table 2.8: Results for the *twod* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A small amount of overlap was chosen.

Overlap	K	constr.	GMRES precond. with				without Krylov accel.			
			AS		RAS		AS		RAS	
			solve	it	solve	it	solve	it	solve	it
twod(06)										
	2	1.89	0.16	8	0.16	7	0.11	8	0.12	8
	4	0.48	0.13	9	0.14	9	0.47	56	0.58	57
	8	0.38	0.14	11	0.16	11	0.47	58	0.54	59
no restr.	16	0.36	0.19	14	0.23	14	0.32	38	0.37	39
	32	**0.25**	**0.23**	**17**	0.23	16	0.50	52	0.52	53
	64	0.21	0.35	23	0.38	23	5.72	443	6.45	450
	80	0.25	0.42	26	0.49	26	1.23	117	1.39	118
	2	1.93	0.15	8	0.15	7	0.11	8	0.12	8
	4	0.32	0.18	15	0.19	15	0.42	56	0.47	57
	8	0.37	0.23	17	0.24	17	1.48	162	1.62	162
consec.	16	0.31	0.26	19	0.27	19	0.80	95	0.93	95
	32	0.24	0.31	22	0.34	22	0.78	86	0.82	87
	64	0.20	0.50	30	0.52	30	5.23	460	5.96	466
	80	0.20	0.51	31	0.59	31	1.76	178	1.95	180
twod(10)										
	2	2.99	0.42	7	0.48	7	0.37	10	0.38	9
	4	4.59	0.60	9	0.65	9	0.52	11	0.53	11
	8	4.59	0.78	11	0.82	11	1.18	25	1.32	25
no restr.	16	3.30	0.88	12	0.92	12	1.31	26	1.31	26
	32	**1.82**	**1.41**	**20**	1.43	20	2.45	54	2.58	54
	64	1.64	1.93	26	2.02	26	5.48	118	6.15	120
	80	1.62	2.47	31	2.45	30	19.31	343	20.75	348
	2	2.95	0.40	7	0.47	7	0.37	10	0.39	9
	4	4.43	0.62	9	0.65	9	0.48	11	0.51	11
	8	4.39	1.14	16	1.20	16	1.30	29	1.49	29
consec.	16	3.01	1.54	20	1.62	20	3.30	67	3.50	67
	32	1.61	1.97	28	2.12	28	3.76	92	4.14	92
	64	1.49	2.91	37	3.01	37	14.46	324	15.72	324
	80	1.41	3.45	41	3.63	41	16.42	352	18.34	356

Table 2.9: Results for the *twod* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A small amount of overlap was chosen.

Overlap	K	constr.	GMRES precond. with				without Krylov accel.			
			AS		RAS		AS		RAS	
			solve	it	solve	it	solve	it	solve	it
twod(12)										
	2	17.24	2.11	9	2.11	9	1.41	11	1.50	11
	4	10.42	1.98	10	2.04	10	1.89	18	2.07	18
	8	11.86	2.98	13	2.90	12	2.51	20	2.66	20
no restr.	16	12.19	3.85	16	4.05	16	4.80	37	5.05	37
	32	**6.88**	**4.89**	**20**	5.16	20	4.90	43	5.21	43
	64	5.40	7.25	27	7.32	27	8.78	78	9.16	77
	80	4.43	7.96	30	8.26	30	8.83	82	9.30	80
	2	17.24	1.92	9	2.12	9	1.33	11	1.45	11
	4	8.62	3.46	17	3.62	17	2.18	23	2.29	22
	8	11.28	4.83	20	5.02	20	5.48	48	5.81	48
consec.	16	11.05	4.57	19	5.09	20	4.54	38	4.84	38
	32	6.56	10.06	34	10.53	34	12.02	108	12.84	108
	64	4.84	14.97	45	15.25	45	20.94	193	22.20	193
	80	4.12	14.78	45	15.08	45	22.49	199	24.86	199
twod(14)										
	2	30.00	3.85	9	4.20	9	3.22	13	3.60	13
	4	27.91	4.44	12	4.70	12	3.31	16	3.45	16
	8	31.33	5.50	13	5.79	13	4.18	17	4.38	17
no restr.	16	18.22	6.28	16	6.54	16	9.00	43	9.41	43
	32	**14.00**	**9.49**	**22**	9.83	22	11.82	57	12.51	56
	64	12.98	17.83	34	18.28	34	20.77	93	21.89	92
	80	9.82	15.65	32	15.36	31	22.34	111	24.03	112
	2	29.85	3.70	9	4.13	9	3.23	13	3.61	13
	4	23.97	8.81	21	9.24	21	5.38	28	6.04	28
	8	31.84	11.24	23	11.81	23	9.68	39	10.04	39
consec.	16	17.38	13.55	28	13.88	28	24.22	114	26.25	114
	32	13.98	16.87	33	17.61	33	19.70	97	21.31	97
	64	12.35	36.55	61	37.01	60	30.42	140	32.61	140
	80	9.15	27.38	47	28.13	47	27.04	141	29.07	143

Table 2.10: Results for the *tcomm* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A small amount of overlap was chosen.

Overlap	K	constr.	GMRES precond. with				without Krylov accel.			
			AS		RAS		AS		RAS	
			solve	it	solve	it	solve	it	solve	it
tcomm(47)										
	2	6.76	0.80	4	0.90	4	0.43	4	0.53	4
	4	4.77	0.66	4	0.68	4	0.28	4	0.33	4
	8	4.73	0.66	4	0.72	4	0.30	4	0.39	4
no restr.	16	4.02	0.65	4	0.69	4	0.32	4	0.33	4
	32	2.69	0.64	4	0.67	4	0.29	4	0.35	4
	64	2.38	0.65	4	0.72	4	0.29	4	0.32	4
	80	2.14	0.68	4	0.68	4	0.29	4	0.33	4
	2	6.72	0.84	4	0.87	4	0.43	4	0.48	4
	4	4.49	0.65	4	0.72	4	0.31	4	0.34	4
	8	4.34	0.71	4	0.68	4	0.31	4	0.34	4
consec.	16	3.69	0.64	4	0.71	4	0.29	4	0.32	4
	32	2.55	0.69	4	0.69	4	0.29	4	0.33	4
	64	2.23	0.63	4	0.68	4	0.32	4	0.38	4
	80	**2.03**	0.64	4	0.71	4	**0.29**	**4**	0.34	4
tcomm(49)										
	2	12.20	1.46	4	1.63	4	0.72	4	0.83	4
	4	12.89	1.42	4	1.50	4	0.65	4	0.72	4
	8	13.21	1.33	4	1.40	4	0.57	4	0.66	4
no restr.	16	10.85	1.33	4	1.40	4	0.57	4	0.64	4
	32	7.59	1.45	4	1.38	4	0.58	4	0.71	4
	64	5.79	1.43	4	1.50	4	0.57	4	0.75	4
	80	5.40	1.32	4	1.51	4	0.61	4	0.72	4
	2	12.41	1.47	4	1.78	4	0.71	4	0.90	4
	4	11.94	1.38	4	1.52	4	0.69	4	0.73	4
	8	12.53	1.36	4	1.39	4	0.57	4	0.72	4
consec.	16	10.37	1.35	4	1.70	4	0.59	4	0.71	4
	32	6.75	1.36	4	1.61	4	0.64	4	0.67	4
	64	5.22	1.28	4	1.51	4	0.60	4	0.64	4
	80	**4.72**	1.33	4	1.42	4	**0.57**	**4**	0.67	4

Table 2.11: Results for the *mutex* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-3} was used. A small amount of overlap was chosen.

Overlap	K	constr.	GMRES precond. with				without Krylov accel.			
			AS		RAS		AS		RAS	
			solve	it	solve	it	solve	it	solve	it
mutex(09)										
	2	2.08	0.11	6	0.12	6	0.06	5	0.09	7
	4	1.06	0.17	10	0.14	8	0.18	17	0.16	14
	8	0.71	0.20	12	0.17	10	0.25	24	0.33	30
no restr.	16	0.88	0.23	13	0.22	12	0.26	23	0.21	17
	32	0.62	0.25	14	0.22	12	0.28	24	0.24	19
	64	0.50	0.30	16	0.25	13	0.38	31	0.32	24
	80	**0.45**	0.30	16	**0.26**	**13**	0.42	34	0.43	32
	2	2.09	0.11	6	0.12	6	0.06	5	0.09	7
	4	0.93	0.19	12	0.21	12	2.49	254	1.45	137
	8	0.62	0.22	14	0.23	14	4.22	451	4.28	422
consec.	16	0.66	0.28	17	0.30	17	4.67	467	5.38	500
	32	0.52	0.25	15	0.27	15	3.16	283	3.25	285
	64	0.40	0.31	18	0.34	18	4.11	376	4.88	406
	80	0.39	0.35	19	0.35	18	3.64	326	5.60	456
mutex(12)										
	2	44.80	0.83	7	0.94	7	1.54	18	0.62	7
	4	26.24	1.02	10	0.94	9	1.72	24	0.89	12
	8	13.84	1.00	11	0.95	10	1.35	21	0.94	14
no restr.	16	8.53	1.19	14	1.05	12	1.29	23	1.18	20
	32	5.01	1.03	13	0.99	12	1.43	27	1.15	21
	64	3.23	1.23	16	1.14	14	1.92	39	1.86	36
	80	2.84	1.23	16	1.02	13	1.73	36	1.39	26
	2	44.01	0.84	7	0.92	7	1.55	18	0.65	7
	4	18.96	1.14	12	1.20	12	2.41	38	2.48	38
	8	9.56	1.24	14	1.29	14	4.16	74	7.65	129
consec.	16	5.26	1.47	18	1.63	19	3.72	78	10.29	201
	32	3.28	1.23	17	1.28	17	10.93	254	11.45	250
	64	2.07	1.45	20	1.50	20	17.11	430	20.99	482
	80	**1.91**	**1.35**	**19**	1.41	19	5.60	139	12.90	295

Table 2.12: Results for the *reliab1* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-3} was used. A small amount of overlap was chosen.

Overlap	K	constr.	GMRES precond. with				without Krylov accel.			
			AS		RAS		AS		RAS	
			solve	it	solve	it	solve	it	solve	it
reliab1(300)										
	2	0.70	0.26	15	0.28	15	0.23	21	0.26	21
	4	0.60	0.26	15	0.28	15	0.23	21	0.26	21
	8	0.50	0.51	26	0.54	26	0.59	50	0.63	49
no restr.	16	0.36	0.58	30	0.61	30	0.81	75	0.87	72
	32	0.34	0.68	33	0.73	33	1.14	96	1.19	92
	64	0.29	0.77	35	0.79	34	1.29	103	1.36	98
	80	0.27	0.80	36	0.83	35	1.48	116	1.57	111
	2	0.70	0.26	15	0.28	15	0.23	21	0.26	21
	4	**0.59**	0.26	15	0.28	15	**0.23**	**21**	0.26	21
	8	0.42	0.57	29	0.60	29	0.61	54	0.65	53
consec.	16	0.34	0.98	45	1.03	45	2.02	192	2.23	192
	32	0.32	1.29	56	1.53	56	2.40	211	2.68	212
	64	0.27	1.34	57	1.41	55	2.69	224	2.99	223
	80	0.24	1.53	67	1.79	67	3.34	274	3.74	274
reliab1(600)										
	2	5.10	1.38	15	1.63	15	1.81	34	2.09	34
	4	3.95	1.28	15	1.48	15	1.79	34	1.94	34
	8	3.18	4.14	33	4.30	33	3.50	59	3.63	57
no restr.	16	**2.23**	**1.92**	**22**	2.02	22	2.91	57	3.09	56
	32	1.84	2.80	26	2.73	26	4.45	78	4.70	76
	64	1.62	5.27	41	4.77	41	8.11	142	8.51	137
	80	1.46	4.96	40	4.31	39	7.13	130	7.45	125
	2	5.03	1.45	15	1.55	15	1.81	34	2.10	34
	4	3.26	1.42	15	1.51	15	1.75	34	1.90	34
	8	3.12	3.58	33	4.23	33	3.39	59	3.52	57
consec.	16	2.14	3.69	36	3.87	36	7.69	152	8.41	152
	32	1.77	4.75	38	4.58	38	5.37	96	5.62	94
	64	1.52	10.90	76	9.16	73	15.01	277	16.19	274
	80	1.38	11.62	82	10.13	82	16.35	314	17.78	314

Table 2.13: Results for the *reliab1* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-3} was used. A small amount of overlap was chosen.

Overlap	K	constr.	GMRES precond. with				without Krylov accel.			
			AS		RAS		AS		RAS	
			solve	it	solve	it	solve	it	solve	it
reliab1(900)										
	2	17.29	5.15	17	5.41	17	7.05	45	7.38	45
	4	11.84	4.71	17	4.91	17	5.90	45	6.35	45
	8	8.53	8.76	27	8.51	26	14.00	107	14.60	104
no restr.	16	6.87	10.59	31	10.91	31	14.05	109	14.70	106
	32	**5.52**	**10.21**	**30**	10.70	30	11.57	84	11.91	81
	64	4.15	18.80	45	19.21	45	22.64	175	23.49	169
	80	4.11	17.78	43	18.26	43	18.42	135	19.01	131
	2	17.39	5.23	17	5.49	17	7.11	45	7.46	45
	4	11.58	4.79	17	4.93	17	5.92	45	6.31	45
	8	9.47	8.75	27	8.47	26	14.14	107	14.82	104
consec.	16	6.78	17.08	42	17.78	42	14.64	114	15.39	111
	32	5.12	10.94	31	11.27	31	10.65	79	11.11	77
	64	4.01	30.24	75	30.34	74	36.83	290	39.03	286
	80	3.90	27.51	68	28.23	67	39.36	299	42.20	298
reliab1(1200)										
	2	39.92	13.03	22	14.06	22	17.61	61	19.71	61
	4	30.28	15.97	26	16.43	26	22.23	83	22.67	81
	8	23.89	15.77	26	16.17	26	21.87	83	22.83	81
no restr.	16	14.78	18.08	30	18.84	30	22.90	97	23.99	94
	32	10.84	15.81	28	16.45	28	13.09	61	14.18	61
	64	8.64	35.97	47	35.36	46	46.06	197	48.05	191
	80	8.36	28.59	41	28.40	40	30.27	131	31.55	127
	2	40.09	13.06	22	14.13	22	17.58	61	19.69	61
	4	29.92	15.60	26	16.18	26	21.94	83	22.76	81
	8	22.65	28.31	39	28.91	39	53.30	206	56.98	206
consec.	16	14.64	30.90	43	31.71	43	49.72	212	53.69	212
	32	**10.56**	28.31	41	28.86	41	**21.65**	**101**	23.50	101
	64	8.61	122.76	106	124.85	106	>116	>500	>125	>500
	80	8.02	68.97	65	70.63	66	53.33	232	56.41	228

Table 2.14: Results for the *reliab2* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-3} was used. A small amount of overlap was chosen.

Overlap	K	constr.	GMRES precond. with				without Krylov accel.			
			AS		RAS		AS		RAS	
			solve	it	solve	it	solve	it	solve	it
reliab2(300)										
	2	**0.50**	**0.33**	**18**	0.37	18	>6.33	>500	>6.89	>500
	4	0.61	0.39	21	0.42	21	>6.61	>500	>7.10	>500
	8	0.34	0.78	38	0.81	38	>6.04	>500	>6.48	>500
no restr.	16	0.32	0.94	44	1.00	44	>5.92	>500	>6.40	>500
	32	0.29	1.80	82	1.83	80	>6.11	>500	>6.83	>500
	64	0.26	2.11	91	2.33	90	>6.52	>500	>7.34	>500
	80	0.26	2.14	89	2.24	88	>6.78	>500	>7.54	>500
	2	0.51	0.33	18	0.34	18	>6.38	>500	>6.87	>500
	4	0.58	0.39	21	0.42	21	>6.95	>500	>7.11	>500
	8	0.36	1.16	51	1.21	51	>5.70	>500	>6.30	>500
consec.	16	0.32	1.79	82	1.87	82	>5.56	>500	>6.14	>500
	32	0.28	4.53	200	4.80	200	>5.73	>500	>6.37	>500
	64	0.25	>5.94	>250	>6.27	>250	>6.13	>500	>6.77	>500
	80	0.23	>5.93	>250	>6.27	>250	>6.26	>500	>7.03	>500
reliab2(600)										
	2	4.16	2.59	26	3.12	26	>30.90	>500	>34.16	>500
	4	**3.65**	**2.60**	**28**	3.12	28	>29.82	>500	>31.93	>500
	8	2.42	3.85	34	3.83	33	>28.56	>500	>31.09	>500
no restr.	16	1.95	5.70	50	5.86	50	>28.03	>500	>30.56	>500
	32	1.60	7.49	52	6.55	51	>27.95	>500	>30.44	>500
	64	1.44	12.96	92	10.96	91	>27.26	>500	>29.88	>500
	80	1.48	14.15	99	12.18	98	>27.88	>500	>30.10	>500
	2	4.17	2.70	26	2.97	26	>30.74	>500	>34.52	>500
	4	3.38	3.84	34	4.09	34	>28.66	>500	>30.92	>500
	8	2.62	3.67	37	4.58	37	>26.76	>500	>29.18	>500
consec.	16	1.86	9.75	90	10.16	90	>25.84	>500	>27.95	>500
	32	1.57	12.70	92	11.58	91	>24.36	>500	>26.65	>500
	64	1.44	14.27	101	12.91	100	>24.49	>500	>26.82	>500
	80	1.42	>33.55	>250	>32.76	>250	>24.84	>500	>27.17	>500

Table 2.15: Results for the *reliab2* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-3} was used. A small amount of overlap was chosen.

Overlap	K	constr.	GMRES precond. with				without Krylov accel.			
			AS		RAS		AS		RAS	
			solve	it	solve	it	solve	it	solve	it
reliab2(900)										
	2	11.11	11.06	32	11.64	32	>78.24	>500	>85.96	>500
	4	10.40	11.40	33	11.77	33	>72.60	>500	>78.44	>500
	8	**6.95**	**11.97**	**34**	12.21	34	>69.92	>500	>74.66	>500
no restr.	16	5.78	20.01	48	20.44	48	>68.34	>500	>73.48	>500
	32	4.50	18.72	46	19.13	46	>65.56	>500	>71.59	>500
	64	3.86	40.92	97	41.39	96	>63.35	>500	>67.58	>500
	80	3.52	39.58	95	39.14	93	>65.47	>500	>70.82	>500
	2	11.15	11.23	32	11.56	32	>79.25	>500	>86.08	>500
	4	10.62	19.06	45	19.46	45	>73.43	>500	>78.43	>500
	8	7.83	21.11	49	21.74	49	>64.13	>500	>68.70	>500
consec.	16	5.93	42.42	94	40.36	94	>59.95	>500	>65.02	>500
	32	4.35	40.05	96	40.44	95	>59.65	>500	>64.62	>500
	64	3.75	64.88	151	65.72	150	>58.37	>500	>63.11	>500
	80	3.46	>107.62	>250	>109.35	>250	>66.16	>500	>147.90	>500
reliab2(1200)										
	2	25.34	19.23	31	20.56	31	>129.18	>500	>150.70	>500
	4	22.32	23.57	36	24.53	36	>136.55	>500	>150.71	>500
	8	**16.14**	**24.70**	**38**	25.25	38	>130.15	>500	>140.54	>500
no restr.	16	12.51	38.55	50	40.19	50	>122.01	>500	>132.14	>500
	32	9.75	82.21	80	80.93	78	>114.82	>500	>123.98	>500
	64	7.80	101.59	92	101.10	91	>117.12	>500	>128.00	>500
	80	7.29	171.90	148	120.43	102	>118.15	>500	>129.21	>500
	2	25.13	18.81	31	20.21	31	>128.09	>500	>149.61	>500
	4	24.95	33.49	45	34.36	45	>138.80	>500	>150.80	>500
	8	16.18	36.75	49	38.19	49	>108.72	>500	>118.54	>500
consec.	16	12.96	62.35	88	63.64	88	>106.32	>500	>115.47	>500
	32	9.87	116.08	101	119.58	100	>103.49	>500	>112.64	>500
	64	7.85	>290.01	>250	>296.18	>250	>104.75	>500	>112.95	>500
	80	7.29	>295.30	>250	>300.85	>250	>103.81	>500	>111.98	>500

2.10.1.2 Results with incomplete LU factorization and large amount of overlap

Table 2.16: Results for the *ncd* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A large amount of overlap was chosen.

Overlap	K	constr.	GMRES precond. with AS		GMRES precond. with RAS		without Krylov accel. AS		without Krylov accel. RAS	
			solve	it	solve	it	solve	it	solve	it
ncd(07)										
no restr.	2	**0.24**	**0.16**	**12**	**0.16**	**12**	0.25	29	0.26	28
	4	0.32	0.19	12	0.17	11	0.17	16	0.17	15
	8	0.46	0.26	13	0.23	12	0.42	29	0.23	16
	16	0.59	0.32	13	0.27	12	0.31	17	0.30	16
	32	0.82	0.45	14	0.34	11	0.47	18	0.32	13
	64	1.18	0.67	15	0.58	14	0.88	23	0.71	20
	80	1.37	0.86	17	0.66	14	1.36	31	0.73	18
consec.	2	0.25	0.16	12	0.17	12	0.25	29	0.26	28
	4	0.24	0.30	22	0.31	22	2.00	225	2.48	260
	8	0.28	0.39	26	0.40	26	>4.86	>500	>5.08	>500
	16	0.32	0.43	26	0.43	26	3.12	283	3.36	297
	32	0.30	0.56	31	0.55	30	>6.15	>500	>6.39	>500
	64	0.33	1.03	47	1.02	46	>7.21	>500	>8.11	>500
	80	0.38	1.28	47	1.29	47	>8.71	>500	>8.83	>500
ncd(10)										
no restr.	2	**0.84**	**0.44**	**11**	0.48	11	0.45	18	0.50	18
	4	0.97	0.53	10	0.51	10	0.59	19	0.60	18
	8	1.27	0.62	11	0.65	11	0.71	18	0.71	19
	16	1.33	0.79	12	0.85	12	0.96	21	0.90	21
	32	1.88	1.07	13	1.09	14	1.04	18	0.98	16
	64	2.58	1.63	15	1.41	14	1.58	19	1.34	17
	80	2.83	1.82	17	1.52	15	3.32	38	2.04	25
consec.	2	0.93	0.44	11	0.50	11	0.45	18	0.52	18
	4	0.75	0.64	14	0.63	14	1.95	73	2.15	73
	8	0.80	0.72	15	0.69	15	2.00	70	2.12	69
	16	0.73	1.33	26	1.33	25	>14.54	>500	>15.17	>500
	32	0.81	1.78	32	1.86	32	>15.90	>500	>16.38	>500
	64	0.92	3.11	46	3.02	45	>18.12	>500	>18.23	>500
	80	0.90	2.49	39	2.44	38	>18.07	>500	>18.71	>500

Table 2.17: Results for the *ncd* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A large amount of overlap was chosen.

Overlap	K	constr.	GMRES precond. with				without Krylov accel.			
			AS		RAS		AS		RAS	
			solve	it	solve	it	solve	it	solve	it
ncd(15)										
	2	**2.59**	1.87	9	2.03	9	**1.13**	**10**	1.16	10
	4	2.63	1.85	9	1.97	9	1.14	10	1.19	10
	8	3.23	2.25	10	2.32	10	1.81	14	1.85	14
no restr.	16	3.91	2.75	11	2.44	10	2.37	16	2.36	16
	32	4.54	3.30	12	3.28	12	2.57	15	2.35	14
	64	5.71	5.01	15	4.46	14	3.94	18	4.22	20
	80	6.00	5.90	17	4.51	14	7.00	30	4.34	20
	2	2.72	1.82	9	1.86	9	1.09	10	1.17	10
	4	2.30	1.68	9	1.73	9	1.00	10	1.04	10
	8	2.47	3.17	15	3.26	15	>54.43	>500	>59.82	>500
consec.	16	2.76	4.48	18	4.49	18	>60.92	>500	>62.67	>500
	32	2.36	8.08	29	8.11	29	>58.82	>500	>61.24	>500
	64	2.54	7.15	26	7.28	26	>63.25	>500	>65.85	>500
	80	2.53	14.88	42	12.58	38	>64.06	>500	>66.08	>500
ncd(20)										
	2	6.49	4.52	9	4.86	9	2.91	10	3.21	10
	4	7.59	5.09	10	4.69	9	3.52	12	3.76	12
	8	7.08	5.17	10	4.84	9	3.63	12	3.74	12
no restr.	16	8.55	7.47	13	6.36	11	9.01	27	4.70	14
	32	9.85	7.79	13	7.87	13	5.77	16	5.70	16
	64	11.62	10.88	15	10.71	15	8.57	19	7.36	17
	80	11.08	10.11	15	9.69	15	7.71	18	7.36	18
	2	**6.05**	4.24	9	4.63	9	**2.66**	**10**	3.02	10
	4	5.51	8.79	17	9.00	17	>131.52	>500	>136.49	>500
	8	4.70	8.45	17	8.70	17	>126.14	>500	>128.84	>500
consec.	16	5.19	6.55	14	6.76	14	16.88	68	22.44	86
	32	5.49	13.38	24	12.88	23	>130.28	>500	>135.82	>500
	64	5.04	16.02	27	15.20	26	>133.58	>500	>139.25	>500
	80	5.10	15.91	27	16.39	27	130.00	475	128.38	451

Table 2.18: Results for the *twod* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A large amount of overlap was chosen.

Overlap	K	constr.	GMRES precond. with				without Krylov accel.			
			AS		RAS		AS		RAS	
			solve	it	solve	it	solve	it	solve	it
twod(06)										
	2	1.66	0.15	7	0.14	7	0.13	8	0.14	8
	4	0.60	0.13	8	0.13	8	0.12	12	0.12	11
	8	0.53	0.16	10	0.14	9	0.15	14	0.15	13
no restr.	16	0.63	0.30	13	0.23	12	0.23	16	0.21	15
	32	0.64	0.32	16	0.35	14	0.27	19	0.23	16
	64	0.57	0.53	21	0.47	20	1.14	65	1.22	64
	80	0.57	0.65	25	0.57	22	0.68	34	0.61	30
	2	1.76	0.14	7	0.14	7	0.11	8	0.11	8
	4	**0.46**	0.17	13	0.17	13	**0.14**	**17**	0.15	16
	8	0.47	0.24	17	0.27	17	1.63	162	1.77	162
consec.	16	0.41	0.27	18	0.28	18	0.93	95	1.07	95
	32	0.35	0.32	20	0.33	20	0.87	80	0.95	85
	64	0.30	0.54	29	0.68	29	2.43	201	2.77	207
	80	0.31	0.55	29	0.66	28	1.75	140	2.21	162
twod(10)										
	2	**3.06**	0.41	7	0.49	7	**0.28**	**8**	0.31	8
	4	4.96	0.58	8	0.64	8	0.41	9	0.45	9
	8	5.80	0.82	10	0.82	10	0.68	12	0.72	12
no restr.	16	4.12	0.94	11	0.89	11	0.72	13	0.69	12
	32	2.52	1.30	17	1.27	16	1.14	21	1.16	19
	64	2.90	1.96	22	1.91	21	1.82	29	2.00	30
	80	2.79	2.37	25	2.51	24	4.61	67	4.71	65
	2	3.17	0.41	7	0.43	7	0.32	8	0.32	8
	4	4.75	0.61	8	0.57	8	0.41	9	0.43	9
	8	4.36	1.12	15	1.25	16	1.34	27	1.34	26
consec.	16	3.41	1.54	20	1.60	20	3.41	67	3.66	67
	32	1.86	1.91	26	2.39	27	4.12	88	4.42	89
	64	1.83	2.98	35	3.65	35	17.53	327	18.88	327
	80	1.73	3.23	37	4.17	38	7.04	140	8.15	144

Table 2.19: Results for the *twod* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A large amount of overlap was chosen.

Overlap	K	constr.	GMRES precond. with				without Krylov accel.			
			AS		RAS		AS		RAS	
			solve	it	solve	it	solve	it	solve	it
twod(12)										
	2	13.22	1.91	9	2.09	9	1.41	11	1.57	11
	4	**9.31**	1.97	10	1.85	9	**1.27**	**12**	1.33	12
	8	10.95	2.66	12	2.45	11	1.81	14	1.85	14
no restr.	16	12.10	3.25	14	3.28	14	2.35	17	2.49	17
	32	8.76	4.17	17	4.57	17	2.79	21	2.87	20
	64	7.90	6.37	23	6.14	22	4.18	29	4.03	27
	80	6.55	6.61	24	6.44	23	4.42	31	4.16	29
	2	13.25	1.87	9	1.89	9	1.33	11	1.40	11
	4	7.82	3.63	17	3.60	17	2.12	22	2.30	22
	8	11.81	4.73	20	4.89	20	5.92	50	6.23	50
consec.	16	11.39	4.24	18	4.30	18	3.42	27	3.63	27
	32	7.01	9.63	33	11.54	34	13.05	108	13.75	108
	64	5.63	14.70	45	18.41	45	23.27	199	25.40	197
	80	4.85	13.90	43	16.45	43	23.45	197	25.07	199
twod(14)										
	2	31.42	3.83	9	3.61	8	3.02	12	3.40	12
	4	24.53	4.21	11	3.71	10	2.89	14	3.00	14
	8	33.34	4.69	11	4.84	11	3.63	14	3.75	14
no restr.	16	20.00	5.65	14	5.19	13	4.09	18	3.99	17
	32	**16.62**	8.54	19	8.06	18	**5.74**	**24**	6.09	24
	64	16.56	12.79	25	11.99	23	9.18	35	8.87	32
	80	13.99	12.73	25	13.20	25	9.85	39	9.81	36
	2	31.95	3.93	9	3.37	8	3.07	12	3.38	12
	4	20.64	8.70	21	9.31	21	5.59	28	5.64	26
	8	35.12	11.02	23	12.22	23	9.86	39	10.30	39
consec.	16	17.62	13.22	28	14.47	28	24.30	114	25.78	114
	32	14.54	16.26	32	18.12	32	21.07	96	21.39	93
	80	10.13	24.96	44	24.75	43	26.06	129	27.33	128

Table 2.20: Results for the *tcomm* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A large amount of overlap was chosen.

Overlap	K	constr.	GMRES precond. with				without Krylov accel.			
			AS		RAS		AS		RAS	
			solve	it	solve	it	solve	it	solve	it
tcomm(47)										
	2	6.35	0.78	4	0.81	4	0.41	4	0.46	4
	4	4.58	0.62	4	0.66	4	0.28	4	0.31	4
	8	4.70	0.62	4	0.66	4	0.29	4	0.33	4
no restr.	16	4.28	0.63	4	0.67	4	0.29	4	0.32	4
	32	3.13	0.66	4	0.69	4	0.31	4	0.34	4
	64	2.95	0.67	4	0.70	4	0.33	4	0.35	4
	80	3.01	0.72	4	0.83	4	0.36	4	0.39	4
	2	6.81	0.81	4	0.92	4	0.43	4	0.51	4
	4	4.57	0.68	4	0.68	4	0.29	4	0.33	4
	8	4.58	0.68	4	0.72	4	0.31	4	0.38	4
consec.	16	4.01	0.65	4	0.69	4	0.31	4	0.35	4
	32	2.72	0.69	4	0.72	4	0.35	4	0.36	4
	64	2.57	0.66	4	0.70	4	0.31	4	0.38	4
	80	**2.46**	0.68	4	0.75	4	**0.33**	**4**	0.38	4
tcomm(49)										
	2	12.36	1.48	4	1.76	4	0.70	4	0.82	4
	4	13.06	1.44	4	1.64	4	0.70	4	0.76	4
	8	13.72	1.36	4	1.40	4	0.58	4	0.68	4
no restr.	16	12.03	1.33	4	1.56	4	0.65	4	0.68	4
	32	8.70	1.36	4	1.66	4	0.67	4	0.69	4
	64	7.27	1.49	4	1.69	4	0.65	4	0.73	4
	80	6.69	1.50	4	1.66	4	0.71	4	0.72	4
	2	12.22	1.49	4	1.76	4	0.67	4	0.84	4
	4	11.76	1.42	4	1.71	4	0.67	4	0.71	4
	8	12.21	1.30	4	1.41	4	0.59	4	0.64	4
consec.	16	10.98	1.39	4	1.62	4	0.61	4	0.68	4
	32	6.87	1.36	4	1.58	4	0.58	4	0.70	4
	64	5.76	1.33	4	1.37	4	0.58	4	0.67	4
	80	**5.12**	1.33	4	1.44	4	**0.61**	**4**	0.67	4

Table 2.21: Results for the *reliab1* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-3} was used. A large amount of overlap was chosen.

Overlap	K	constr.	GMRES precond. with				without Krylov accel.			
			AS		RAS		AS		RAS	
			solve	it	solve	it	solve	it	solve	it
reliab1(300)										
	2	0.80	0.24	12	0.25	12	0.26	20	0.28	20
	4	**0.74**	**0.24**	**12**	0.25	12	0.26	19	0.27	19
	8	0.82	0.33	15	0.30	13	0.38	25	0.35	22
no restr.	16	0.64	0.37	16	0.30	13	0.42	27	0.37	23
	32	0.69	0.45	18	0.37	14	0.58	32	0.45	24
	64	0.71	0.50	17	0.43	14	0.87	39	0.67	29
	80	0.67	0.65	19	0.50	16	0.99	40	0.77	29
	2	0.84	0.24	12	0.25	12	0.26	20	0.30	20
	4	0.72	0.26	12	0.26	11	0.26	19	0.27	19
	8	0.55	0.61	27	0.69	26	0.64	49	0.66	47
consec.	16	0.45	1.00	40	1.03	40	2.94	222	3.13	220
	32	0.46	1.43	50	1.54	51	3.41	219	4.59	279
	64	0.41	1.72	55	1.61	49	4.96	285	5.05	284
	80	0.37	1.84	65	2.17	66	5.13	318	6.09	350
reliab1(600)										
	2	5.85	1.55	15	1.80	15	1.93	32	2.14	32
	4	4.49	1.66	15	1.75	15	2.06	32	2.26	32
	8	4.04	1.85	16	1.72	15	2.61	37	2.54	33
no restr.	16	3.05	1.83	16	1.81	15	2.59	37	2.35	34
	32	**2.93**	2.00	17	**1.86**	**16**	3.17	43	2.97	38
	64	2.67	2.74	21	2.21	18	4.20	52	3.82	45
	80	2.63	2.59	20	2.26	18	4.62	55	3.64	43
	2	6.27	1.73	15	1.77	15	1.90	32	2.18	32
	4	4.03	1.60	15	1.73	15	1.87	32	2.01	32
	8	3.52	1.72	16	1.54	14	2.43	37	2.26	33
consec.	16	2.50	4.45	36	4.58	36	9.28	153	10.00	153
	32	2.11	5.25	38	4.93	37	5.85	92	5.93	87
	64	2.11	12.56	76	11.03	75	18.83	282	19.86	280
	80	1.85	13.79	84	12.48	86	25.17	389	26.51	383

Table 2.22: Results for the *reliab1* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-3} was used. A large amount of overlap was chosen.

Overlap	K	constr.	GMRES precond. with				without Krylov accel.			
			AS		RAS		AS		RAS	
			solve	it	solve	it	solve	it	solve	it
reliab1(900)										
	2	19.68	5.49	17	5.79	17	7.75	45	8.38	45
	4	13.35	5.15	17	5.28	17	6.76	45	7.26	45
	8	10.50	6.13	19	5.77	18	8.56	56	8.24	51
no restr.	16	8.19	6.38	20	5.72	18	8.31	55	8.34	52
	32	7.45	6.71	20	6.57	19	9.69	56	9.70	53
	64	**6.01**	8.56	24	**7.38**	**21**	12.25	72	10.72	59
	80	6.16	8.41	24	7.88	22	11.77	67	11.21	59
	2	19.30	5.55	17	5.80	17	7.64	45	8.01	45
	4	12.59	5.18	17	5.24	17	6.69	45	7.44	45
	8	11.35	6.28	19	5.96	18	8.84	56	8.07	51
consec.	16	7.64	18.04	41	17.63	40	14.78	100	15.03	97
	32	6.00	10.85	29	10.61	28	11.39	73	11.26	68
	64	4.54	25.83	60	25.19	56	37.24	262	37.95	250
	80	4.49	26.11	60	26.78	60	51.45	349	54.37	345
reliab1(1200)										
	2	41.59	17.53	22	18.44	22	13.39	45	15.30	45
	4	31.35	14.92	20	14.49	19	12.96	48	13.29	44
	8	26.24	14.88	20	14.40	19	13.64	48	13.08	44
no restr.	16	16.72	15.37	21	14.81	20	13.34	52	12.52	46
	32	12.45	16.03	22	15.31	21	**11.13**	**46**	11.50	45
	64	10.86	20.05	25	18.28	23	19.31	70	17.12	58
	80	11.01	21.41	26	19.58	24	17.05	62	14.97	52
	2	41.65	17.09	22	18.21	22	13.39	45	15.37	45
	4	30.26	14.67	20	14.25	19	12.95	48	12.79	44
	8	23.89	39.75	38	40.79	38	51.03	187	54.59	186
consec.	16	15.45	43.87	41	44.62	41	48.03	195	51.97	195
	32	11.20	37.71	38	38.35	38	20.74	92	22.22	92
	64	9.52	124.58	106	126.29	106	>123.32	>500	132.01	498
	80	8.94	68.48	64	68.43	63	57.32	233	58.16	220

Table 2.23: Results for the *reliab2* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-3} was used. A large amount of overlap was chosen.

Overlap	K	constr.	GMRES precond. with				without Krylov accel.			
			AS		RAS		AS		RAS	
			solve	it	solve	it	solve	it	solve	it
reliab2(300)										
	2	0.60	0.38	18	0.40	18	>6.71	>500	>7.36	>500
	4	0.69	0.43	19	0.45	19	>7.23	>500	>7.64	>500
	8	**0.44**	0.52	23	**0.45**	**20**	>6.95	>500	>7.43	>500
no restr.	16	0.42	0.58	24	0.49	21	>7.32	>500	>7.59	>500
	32	0.39	0.75	29	0.71	27	>7.49	>500	>8.04	>500
	64	0.33	1.15	40	1.16	39	>7.64	>500	>8.40	>500
	80	0.32	1.25	42	1.21	40	>7.76	>500	>8.56	>500
	2	0.58	0.38	18	0.40	18	>6.73	>500	>7.37	>500
	4	0.62	0.41	19	0.43	19	>6.86	>500	>7.40	>500
	8	0.42	1.31	46	1.30	45	>6.32	>500	>6.74	>500
consec.	16	0.37	1.28	46	1.34	46	>6.11	>500	>6.64	>500
	32	0.31	3.63	105	3.66	102	>6.31	>500	>7.00	>500
	64	0.27	5.85	151	5.76	149	>6.65	>500	>7.24	>500
	80	0.25	>9.58	>250	>10.21	>250	>6.62	>500	>7.37	>500
reliab2(600)										
	2	4.48	3.04	24	3.87	24	>31.87	>500	>35.40	>500
	4	3.86	3.35	24	3.43	24	>31.51	>500	>33.92	>500
	8	2.75	3.25	24	3.34	24	>31.67	>500	>34.06	>500
no restr.	16	2.31	3.50	28	3.49	27	>32.84	>500	>35.00	>500
	64	**1.81**	5.67	32	**2.42**	**19**	>33.40	>500	>36.05	>500
	80	1.89	6.52	38	5.09	36	>33.87	>500	>36.19	>500
	2	4.44	3.35	24	3.67	24	>32.05	>500	>35.44	>500
	4	3.78	4.86	31	4.96	31	>30.80	>500	>32.54	>500
	8	2.83	3.80	30	3.92	30	>30.53	>500	>32.75	>500
consec.	16	2.03	12.24	80	12.20	78	>29.23	>500	>31.03	>500
	32	1.71	18.21	86	16.06	85	>27.75	>500	>29.87	>500
	64	1.54	19.52	89	17.61	89	>27.19	>500	>29.50	>500
	80	1.56	>57.24	>250	>54.43	>250	>26.81	>500	>28.90	>500

2.10.1.3 Results with incomplete LU factorization, small amount of overlap, and edge weights during partitioning

Table 2.24: Results for the *ncd* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A small amount of overlap was chosen, and we used weighted edges in Metis.

Overlap	K	constr.	GMRES precond. with				without Krylov accel.			
			AS		RAS		AS		RAS	
			solve	it	solve	it	solve	it	solve	it
ncd(07)										
	2	**0.16**	**0.12**	**11**	0.13	11	0.20	29	0.22	29
	4	0.18	0.14	12	0.13	11	0.12	17	0.13	16
	8	0.18	0.13	11	0.14	11	0.13	17	0.14	16
no restr.	16	0.18	0.17	14	0.16	12	0.28	34	0.29	33
	32	0.19	0.23	17	0.21	15	0.34	37	0.33	33
	64	0.20	0.32	21	0.31	19	0.50	47	0.39	34
	80	0.20	0.32	20	0.27	16	0.43	39	0.28	23
	2	0.17	0.12	11	0.13	11	0.20	29	0.22	29
	4	0.17	0.13	12	0.14	12	0.39	57	0.38	50
	8	0.17	0.14	12	0.15	12	0.37	51	0.41	51
consec.	16	0.16	0.19	16	0.20	16	0.56	75	0.62	75
	32	0.15	0.24	19	0.27	19	0.84	104	0.80	89
	64	0.15	0.32	23	0.34	23	0.67	76	0.76	76
	80	0.15	0.31	22	0.32	21	0.58	64	0.65	64
ncd(10)										
	2	0.51	0.26	8	0.29	8	0.45	23	0.50	23
	4	**0.42**	**0.34**	**10**	0.37	10	0.51	25	0.56	25
	8	0.48	0.43	12	0.42	11	0.46	22	0.50	22
no restr.	16	0.46	0.52	15	0.52	14	0.60	28	0.65	28
	32	0.49	0.50	14	0.53	13	0.43	19	0.41	16
	64	0.51	0.68	17	0.62	15	0.78	31	0.72	25
	80	0.52	0.74	18	0.75	16	0.99	38	0.84	30
	2	0.52	0.28	8	0.29	8	0.46	23	0.50	23
	4	0.42	0.38	10	0.38	10	0.51	25	0.55	25
	8	0.47	0.42	12	0.46	12	0.62	30	0.68	30
consec.	16	0.42	0.55	15	0.57	15	0.76	37	0.82	37
	32	0.44	0.52	15	0.56	15	0.87	41	0.93	41
	64	0.41	0.61	17	0.62	16	0.99	46	0.94	39
	80	0.41	0.70	19	0.77	18	1.25	57	1.11	45

Table 2.25: Results for the *ncd* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A small amount of overlap was chosen, and we used weighted edges in Metis.

Overlap	K	constr.	GMRES precond. with				without Krylov accel.			
			AS		RAS		AS		RAS	
			solve	it	solve	it	solve	it	solve	it
ncd(15)										
	2	1.85	1.45	8	1.52	8	2.17	23	2.34	23
	4	1.63	1.59	9	1.67	9	1.62	18	1.75	18
	8	**1.61**	1.96	11	2.04	11	**1.46**	**16**	1.45	15
no restr.	16	1.68	2.50	13	2.11	11	1.65	18	1.57	16
	32	1.62	2.95	15	2.83	14	1.96	21	1.89	19
	64	1.57	3.14	16	2.95	15	2.07	22	2.11	21
	80	1.71	3.42	17	3.34	16	2.22	23	2.23	22
	2	1.91	1.45	8	1.54	8	2.24	23	2.36	23
	4	1.62	1.64	9	1.66	9	1.62	18	1.79	18
	8	1.66	1.97	11	2.20	11	1.33	15	1.44	15
consec.	16	1.63	2.22	12	2.33	12	1.80	20	1.73	18
	32	1.43	2.97	15	2.76	14	2.20	25	2.29	24
	64	1.34	3.17	17	3.05	16	2.33	27	2.55	27
	80	1.33	3.67	19	3.82	19	3.06	35	3.01	32
ncd(20)										
	2	**3.70**	**3.63**	**8**	3.99	8	3.75	15	4.27	15
	4	4.89	3.87	9	4.13	9	2.76	12	3.01	12
	8	4.71	3.70	9	3.93	9	2.59	12	2.77	12
no restr.	16	4.18	5.67	13	5.32	12	3.85	18	4.15	18
	32	3.81	5.04	12	4.95	11	3.39	16	3.21	14
	64	3.74	6.66	15	6.54	14	4.79	22	4.39	19
	80	3.64	8.71	18	7.63	16	5.30	24	4.94	21
	2	3.77	3.61	8	3.97	8	3.73	15	4.25	15
	4	4.51	3.86	9	4.03	9	2.72	12	2.96	12
	8	4.56	3.73	9	3.88	9	2.62	12	2.77	12
consec.	16	4.02	6.10	14	6.56	14	4.62	22	5.00	22
	32	3.45	4.86	12	5.27	12	4.50	22	4.44	20
	64	3.25	6.93	16	6.77	15	5.93	29	5.51	25
	80	3.21	8.12	18	8.78	18	6.83	33	6.48	29

Table 2.26: Results for the *twod* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A small amount of overlap was chosen, and we used weighted edges in Metis. For each matrix, the best overall timings are in boldface.

Overlap	K	constr.	GMRES precond. with				without Krylov accel.			
			AS		RAS		AS		RAS	
			solve	it	solve	it	solve	it	solve	it
twod(06)										
	2	0.35	0.10	9	0.10	8	0.12	17	0.13	17
	4	**0.24**	0.14	12	**0.13**	**11**	0.37	54	0.42	55
	8	0.25	0.14	12	0.15	12	0.26	38	0.29	38
no restr.	16	0.19	0.17	14	0.19	14	1.55	175	1.71	178
	32	0.16	0.26	20	0.28	20	5.07	500	5.43	500
	64	0.15	0.32	23	0.35	23	5.12	475	5.73	484
	80	0.15	0.42	28	0.46	28	5.70	500	6.24	500
	2	0.35	0.11	9	0.10	8	0.12	17	0.13	17
	4	0.24	0.14	12	0.14	11	0.37	54	0.42	55
	8	0.22	0.20	17	0.21	17	0.38	53	0.43	54
consec.	16	0.19	0.25	20	0.27	20	1.40	181	1.59	183
	32	0.16	0.32	24	0.35	24	>4.65	>500	>5.03	>500
	64	0.14	0.39	27	0.43	27	>4.97	>500	>5.58	>500
	80	0.14	0.55	34	0.56	33	>5.13	>500	>5.69	>500
twod(10)										
	2	2.70	0.43	8	0.50	8	0.53	16	0.67	17
	4	1.64	0.61	11	0.65	11	1.71	52	1.87	52
	8	1.94	0.64	12	0.68	12	1.04	31	1.14	31
no restr.	16	1.42	0.82	15	0.85	15	1.97	59	2.16	59
	32	**1.05**	**1.17**	**21**	1.23	21	8.63	212	9.62	216
	64	1.00	1.40	24	1.41	23	7.83	192	8.84	197
	80	0.93	1.66	27	1.76	27	>21.72	>500	>23.48	>500
	2	2.71	0.43	8	0.49	8	0.53	16	0.62	17
	4	1.66	0.61	11	0.66	11	1.71	52	1.87	52
	8	1.93	0.84	15	0.91	15	1.03	31	1.17	32
consec.	16	1.36	1.39	23	1.44	23	3.15	95	3.67	97
	32	1.00	1.92	31	2.06	31	8.35	230	9.25	233
	64	0.91	2.01	32	2.12	32	>19.92	>500	>21.65	>500
	80	0.88	2.70	39	2.85	39	>20.56	>500	>22.39	>500

Table 2.27: Results for the *twod* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A small amount of overlap was chosen, and we used weighted edges in Metis. For each matrix, the best overall timings are in boldface.

Overlap	K	constr.	GMRES precond. with				without Krylov accel.			
			AS		RAS		AS		RAS	
			solve	it	solve	it	solve	it	solve	it
twod(12)										
	2	9.41	1.63	9	1.88	9	1.56	16	1.64	16
	4	7.48	2.20	12	2.38	12	2.62	29	2.84	29
	8	6.78	2.41	13	2.54	13	3.77	41	4.04	41
no restr.	16	**4.83**	**2.81**	**15**	3.01	15	3.27	36	3.67	37
	32	3.64	4.96	23	5.16	23	5.95	67	6.36	66
	64	2.92	6.95	29	7.19	29	12.50	136	13.96	139
	80	2.98	9.02	34	9.19	34	9.00	98	9.62	96
	2	9.49	1.65	9	1.78	9	1.52	16	1.62	16
	4	7.53	3.13	16	3.35	16	4.38	48	4.88	49
	8	6.93	4.24	20	4.48	20	10.72	110	11.44	110
consec.	16	4.90	5.92	25	6.14	25	5.21	54	5.69	55
	32	3.37	6.77	29	6.93	29	7.88	90	8.58	90
	64	2.74	13.58	45	13.60	45	15.97	180	17.29	180
	80	2.73	19.81	69	20.29	69	20.87	231	22.67	231
twod(14)										
	2	22.04	3.65	10	3.97	10	3.10	15	3.33	14
	4	23.94	3.93	11	3.81	10	2.85	15	3.10	15
	8	16.12	4.91	14	5.12	14	7.93	44	8.51	44
no restr.	16	11.22	7.05	19	7.28	19	5.72	33	6.13	33
	32	**9.04**	**7.40**	**20**	7.70	20	8.66	52	9.11	51
	64	7.18	12.50	29	13.07	29	15.83	92	16.66	90
	80	6.89	13.72	31	14.29	31	13.99	82	14.62	80
	2	22.14	3.67	10	3.99	10	3.10	15	3.22	14
	4	24.37	6.20	16	6.66	16	7.02	36	7.47	36
	8	15.87	7.34	19	7.22	18	8.66	47	9.17	47
consec.	16	11.18	15.55	33	16.10	33	15.80	88	16.87	88
	32	8.51	15.03	33	15.81	33	21.63	130	23.61	130
	64	6.74	24.61	45	25.18	45	40.98	238	44.27	238
	80	6.59	27.10	48	27.72	48	35.27	206	37.92	206

Table 2.28: Results for the *twod* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used. A small amount of overlap was chosen, and we used weighted edges in Metis. For each matrix, the best overall timings are in boldface.

Overlap	K	constr.	GMRES precond. with				without Krylov accel.			
			AS		RAS		AS		RAS	
			solve	it	solve	it	solve	it	solve	it
tcomm(47)										
	2	1.58	0.90	6	0.95	6	0.43	6	0.48	6
	4	**1.47**	0.85	6	0.91	6	**0.39**	**6**	0.44	6
	8	1.49	0.85	6	0.91	6	0.39	6	0.44	6
no restr.	16	1.74	0.93	6	0.94	6	0.41	6	0.46	6
	32	1.78	0.94	6	1.04	6	0.43	6	0.48	6
	64	1.94	0.97	6	1.06	6	0.47	6	0.51	6
	80	2.04	0.98	6	1.08	6	0.49	6	0.53	6
	2	1.63	0.90	6	0.96	6	0.43	6	0.48	6
	4	1.99	0.85	6	0.90	6	0.38	6	0.44	6
	8	2.00	0.84	6	0.91	6	0.38	6	0.44	6
consec.	16	1.95	0.88	6	1.00	6	0.40	6	0.45	6
	32	1.78	0.88	6	0.98	6	0.41	6	0.46	6
	64	1.67	0.90	6	0.95	6	0.43	6	0.48	6
	80	1.66	0.91	6	0.94	6	0.44	6	0.48	6
tcomm(49)										
	2	8.34	2.01	6	2.20	6	1.01	6	1.20	6
	4	2.66	1.79	6	1.91	6	0.83	6	0.95	6
	8	**2.68**	1.68	6	1.78	6	**0.75**	**6**	0.85	6
no restr.	16	3.80	1.71	6	1.81	6	0.77	6	0.87	6
	32	3.03	1.78	6	1.86	6	0.82	6	0.91	6
	64	3.63	1.90	6	2.12	6	0.88	6	0.97	6
	80	3.76	1.91	6	1.99	6	0.91	6	0.99	6
	2	8.51	2.05	6	2.22	6	1.02	6	1.21	6
	4	4.14	1.81	6	1.91	6	0.83	6	0.95	6
	8	4.31	1.74	6	1.88	6	0.75	6	0.85	6
consec.	16	4.73	1.69	6	1.80	6	0.75	6	0.86	6
	32	3.50	1.79	6	1.98	6	0.78	6	0.88	6
	64	3.38	1.77	6	1.86	6	0.81	6	0.90	6
	80	3.36	1.88	6	2.00	6	0.82	6	0.92	6

Table 2.29: Results for the *mutex* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-3} was used. A small amount of overlap was chosen, and we used weighted edges in Metis. For each matrix, the best overall timings are in boldface.

Overlap	K	constr.	GMRES precond. with				without Krylov accel.			
			AS		RAS		AS		RAS	
			solve	it	solve	it	solve	it	solve	it
mutex(09)										
	2	3.13	0.14	6	0.16	7	0.11	7	0.13	8
	4	3.43	0.32	11	0.26	9	0.44	20	1.01	46
	8	3.46	0.43	12	0.38	11	0.82	29	1.28	47
no restr.	16	3.66	0.57	13	0.50	12	1.28	37	1.65	49
	32	3.32	0.67	14	0.63	14	1.86	47	3.63	98
	64	3.45	0.82	15	0.72	14	2.71	59	2.06	48
	80	4.59	0.96	16	0.79	14	3.07	60	3.11	65
	2	3.11	0.14	6	0.16	7	0.11	7	0.13	8
	4	2.46	0.27	12	0.29	12	8.12	500	8.55	500
	8	2.24	0.33	13	0.36	14	5.45	291	5.42	290
consec.	16	1.88	0.37	15	0.37	15	8.43	461	8.08	451
	32	1.43	0.42	17	0.42	17	5.37	299	4.99	277
	64	**0.99**	**0.44**	**18**	0.45	18	8.71	500	8.95	500
	80	1.01	0.44	18	0.45	18	6.43	349	6.24	337
mutex(12)										
	2	55.34	0.99	7	1.07	7	0.88	9	1.11	11
	4	59.98	1.55	9	1.92	11	2.05	16	6.87	54
	8	52.16	2.36	12	2.39	12	4.57	29	9.32	59
no restr.	16	40.78	2.88	13	2.47	12	6.11	37	5.77	36
	32	34.01	3.14	14	2.90	13	9.84	54	13.04	74
	64	27.57	3.60	15	3.19	13	12.35	62	13.71	73
	80	25.39	3.60	15	3.32	14	12.55	63	13.56	71
	2	55.05	0.99	7	1.00	7	0.88	9	1.11	11
	4	31.61	1.62	12	1.66	13	6.24	69	12.32	136
	8	19.88	1.70	14	1.73	14	15.80	187	16.99	192
consec.	16	11.16	1.79	16	1.78	16	19.65	262	23.49	308
	32	7.74	1.63	16	1.64	16	8.16	118	16.76	236
	64	5.27	1.64	17	1.66	17	9.85	150	18.96	280
	80	**4.64**	**1.71**	**18**	1.80	18	20.58	317	23.85	356

Table 2.30: Results for the *reliab1* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-3} was used. A small amount of overlap was chosen, and we used weighted edges in Metis. For each matrix, the best overall timings are in boldface.

Overlap	K	constr.	GMRES precond. with				without Krylov accel.			
			AS		RAS		AS		RAS	
			solve	it	solve	it	solve	it	solve	it
reliab1(300)										
	2	0.78	0.20	11	0.20	11	0.20	18	0.22	18
	4	0.63	0.34	19	0.35	19	0.26	24	0.28	23
	8	0.49	0.57	29	0.68	29	1.24	112	1.37	114
no restr.	16	0.45	0.58	28	0.69	28	0.65	53	0.70	52
	32	**0.34**	**0.45**	**23**	0.46	22	0.97	82	1.08	83
	64	0.28	0.91	40	1.23	40	1.23	98	1.38	100
	80	0.27	0.74	33	0.92	33	1.55	118	1.64	114
	2	0.80	0.19	11	0.20	11	0.20	18	0.22	18
	4	0.61	0.34	19	0.35	19	0.27	24	0.31	23
	8	0.49	1.26	52	1.72	52	4.22	378	4.57	380
consec.	16	0.44	1.19	49	1.67	49	1.62	138	1.76	138
	32	0.30	0.68	33	0.86	33	3.06	270	3.38	270
	64	0.26	1.12	47	1.55	47	4.34	360	4.85	362
	80	0.25	1.31	53	1.83	53	3.98	323	4.43	323
reliab1(600)										
	2	5.43	1.54	15	1.73	15	2.04	32	2.32	32
	4	3.63	2.51	27	3.13	26	2.29	46	2.43	45
	8	2.52	3.99	38	5.66	38	6.13	124	6.47	121
no restr.	16	**2.29**	**2.44**	**26**	2.75	26	4.02	78	4.21	76
	32	2.04	4.48	36	5.06	36	5.66	97	5.89	94
	64	1.59	4.49	37	4.63	36	8.75	155	9.18	151
	80	1.52	4.71	38	4.82	37	7.30	129	7.58	124
	2	5.33	1.60	15	1.65	15	2.01	32	2.27	32
	4	3.58	2.55	27	3.07	26	2.29	46	2.44	45
	8	2.52	4.57	40	5.79	40	6.33	129	6.66	125
consec.	16	2.09	3.91	37	4.92	37	8.65	165	9.69	165
	32	2.02	8.86	57	10.31	57	11.40	205	12.30	203
	64	1.50	8.33	53	8.97	53	15.08	283	16.28	283
	80	1.40	8.47	56	8.93	56	16.86	316	18.24	315

Table 2.31: Results for the *reliab1* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-3} was used. A small amount of overlap was chosen, and we used weighted edges in Metis.

Overlap	K	constr.	GMRES precond. with				without Krylov accel.			
			AS		RAS		AS		RAS	
			solve	it	solve	it	solve	it	solve	it
reliab1(900)										
	2	16.68	5.32	17	5.35	17	7.16	45	7.53	45
	4	11.93	5.41	18	5.61	18	6.66	45	7.12	45
	8	8.10	5.78	20	5.48	19	6.93	53	7.31	52
no restr.	16	7.07	11.34	32	13.06	32	10.21	80	10.82	79
	32	5.31	11.37	32	13.26	32	14.66	109	15.24	106
	64	4.15	17.97	43	21.54	43	25.29	186	26.38	181
	80	3.87	17.05	42	19.79	41	18.66	145	19.66	142
	2	16.72	5.25	17	5.37	17	7.20	45	7.57	45
	4	11.97	5.48	18	5.56	18	6.69	45	7.21	45
	8	**7.53**	**6.15**	**21**	6.39	21	6.88	53	7.30	52
consec.	16	6.47	24.23	56	28.16	55	26.43	210	28.52	210
	32	5.23	19.37	45	22.64	45	37.60	284	40.35	284
	64	3.99	36.92	87	41.73	87	66.43	500	71.10	500
	80	4.01	27.48	67	29.56	71	42.33	329	45.29	328
reliab1(1200)										
	2	37.74	10.73	19	11.74	19	16.01	57	18.05	57
	4	33.47	10.71	19	11.25	19	16.18	57	17.09	57
	8	22.89	17.00	28	17.59	28	32.37	123	33.64	120
no restr.	16	13.32	27.60	40	28.39	40	29.68	133	30.94	130
	32	**12.06**	18.20	30	**17.84**	**29**	19.32	80	20.12	78
	64	8.65	30.20	42	30.83	42	42.58	185	44.76	180
	2	37.52	10.69	19	11.96	19	16.33	57	18.50	57
	4	32.61	10.64	19	11.16	19	15.70	57	17.13	57
	8	17.91	17.15	28	17.41	28	30.61	123	31.86	120
consec.	16	13.55	32.08	44	32.88	44	32.25	147	33.98	144
	32	11.36	36.85	48	38.00	48	46.76	198	50.80	198

2.10.1.4 Results with the two-level method

Table 2.32: Results with the 2-level method for the *ncd* and *twod* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-4} was used.

Matrix	K	small overlap			large overlap		
		constr	solve	it	constr	solve	it
ncd(10)	2	1.94	0.56	9	2.05	0.59	9
	4	**1.90**	**0.50**	**8**	2.26	0.56	8
	8	1.86	0.56	9	2.52	0.68	9
	16	1.85	0.63	10	2.61	0.85	10
	32	1.88	0.70	11	3.05	1.07	11
	64	1.86	0.87	13	3.63	1.40	12
	80	1.89	0.88	13	3.88	1.55	13
ncd(15)	2	8.21	2.62	8	7.94	2.36	7
	4	**7.01**	**2.66**	**8**	7.89	4.37	12
	8	7.09	2.97	9	8.46	2.85	8
	16	7.21	2.98	9	9.03	3.41	9
	32	7.02	3.75	11	9.59	4.06	10
	64	7.03	4.56	13	10.77	5.52	12
twod(10)	2	**4.69**	**0.84**	**8**	4.87	0.74	7
	4	6.09	0.87	8	6.50	0.89	8
	8	6.09	1.08	10	7.17	1.16	10
	16	4.93	1.18	11	5.64	1.28	11
	32	3.51	1.76	17	4.24	1.73	15
	64	3.42	2.26	21	4.61	2.38	19
	80	3.35	2.53	23	4.50	2.72	21
twod(12)	2	22.16	2.90	9	18.38	2.90	9
	4	15.34	2.98	10	**14.62**	**3.07**	**10**
	8	16.77	3.73	12	16.07	3.50	11
	16	16.67	4.59	14	17.41	4.68	14
	32	12.31	5.56	17	14.28	5.55	16
	64	11.00	7.08	21	13.37	7.28	20
	80	10.12	7.41	22	12.19	7.29	20

Table 2.33: Results with the 2-level method for the *reliability* matrices. K is the number of domains, 'constr.' the time (in seconds) needed to construct the RAS preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For local solves the incomplete LU factorization with drop tolerance 10^{-3} was used.

Matrix	K	small overlap			large overlap		
		constr	solve	it	constr	solve	it
reliab1(600)	2	10.37	2.65	14	10.65	2.67	14
	4	9.26	2.49	14	9.53	2.52	14
	8	8.54	3.87	20	9.04	2.65	14
	16	**7.59**	**2.66**	**16**	8.10	2.56	14
	32	7.21	3.52	18	7.85	2.78	14
	64	6.93	4.31	21	7.74	3.36	16
	80	6.78	4.52	22	7.62	3.35	16
reliab1(900)	2	32.86	7.45	16	33.17	7.45	16
	4	27.17	7.25	16	27.60	7.31	16
	8	23.81	8.46	19	24.78	7.04	16
	16	22.06	9.60	21	23.00	7.58	17
	32	**20.62**	**8.54**	**19**	21.97	7.97	17
	64	19.23	14.10	24	20.69	10.81	19
	80	19.37	13.19	23	20.71	11.07	19
reliab2(600)	2	10.18	4.43	21	10.40	4.48	21
	4	9.62	4.07	21	9.91	4.16	21
	8	**8.44**	**4.44**	**23**	8.82	4.11	21
	16	8.02	5.41	29	8.43	4.59	23
	32	7.63	6.53	29	8.17	5.04	23
	64	7.57	7.57	32	8.24	5.80	25
	80	7.49	7.96	33	8.34	6.17	26
reliab2(900)	2	26.72	14.84	25	27.22	14.16	24
	4	26.09	14.87	25	26.38	14.16	24
	8	**22.70**	**14.39**	**25**	23.63	14.45	25
	16	21.57	16.76	28	22.12	14.51	25
	32	20.14	17.67	29	21.13	15.50	26
	64	19.45	22.25	34	20.73	17.52	28
	80	19.19	21.10	33	20.46	16.44	27

2.10.2 Parallel results

Here we present the results of our numerical experiments on a high performance cluster. The experiments were run on the *puma* cluster at Emory University. *Puma* has 32 nodes and 128 processor cores. Each node has two dual core AMD 2214 2.2 GHz Opteron CPUs, 4 GB RAM and an 80 GB drive. We compare GMRES preconditioned with RAS and AS with various restrictions on the chosen overlap. As in the serial experiments we compare small and large amount of overlap, overlap chosen from all domains, or only immediate neighbors, and domain decomposition with and without weights on the edges of the underlying graph of $P + P^t$.

The matrices are distributed using the parallel graph partitioner ParMETIS [65]. We are using 1, 2, 4, 8, 16, and 20 nodes with 4 processor cores each in our experiments. That is, the number of domains used are 4, 8, 16, 32, 64, and 80. For some of the larger example no results were obtained with 4 or 8 domains as the memory requirements were too large.

The software library used is PETSc [7]. Due to the lack of a serial ILUT in PETSc we use an ILU that depends on the levels of fill-in for the local solves. Similar to the serial experiments, GMRES was restarted after 50 iterations and we used a tolerance of 10^{-12}.

Our results for the *ncd* matrices are given in Tables 2.34, and 2.35. The results for the *twod* matrices are given in Tables 2.36, and 2.37. The results for the first reliability model are given in Tables 2.38, 2.39, and 2.40, and for the second reliability model in Tables 2.41, 2.42, and 2.43. For better comparison, each table shows all different chosen parameters for one specific example.

First of all, in the parallel case the restricted additive Schwarz preconditioner outperforms the additive Schwarz preconditioner. The rather high difference in number of iterations (see for example Table 2.34) may be due to a different handling of the results on the overlap in the implementation of AS in PETSc and our serial

implementation of it.

Another observation is that in parallel mode a large choice of overlap is beneficial. While the construction time and the time for each local solve increase, the number of iterations usually decreases and so does the overall time needed, that is, the time to construct the preconditioner and the time needed to solve the linear system. For example for the *reliab1(2000)* matrix, we need 82 iterations to solve the system on 80 subdomains with a small choice of overlap, while for a large choice of overlap we only need 34 iterations. The time to construct the preconditioner increases from 0.41 seconds to 1.22 seconds, but the time needed to solve the linear system decreases from 24.15 seconds to 11.13 seconds (see Table 2.39). An exception are the matrices from the *ncd* family. Here, the number of iterations decreases only slightly while the time needed to construct the preconditioner and the time needed to solve the linear system increase as we increase the amount of overlap (see Table 2.34 and 2.35).

In our parallel implementation it sometimes seems to be an advantageous to use a weighted graph during the domain decomposition. This is most noticeable for the *ncd* matrices. The best case for this matrix was to use weights during the partitioning, and choose a small amount of overlap from consecutive domains, see Table 2.35. This is also the only class of examples were the best results were achieved when overlap was chosen from consecutive subdomains. For the other examples it was not of benefit to choose overlap only from neighboring subdomains. The increase in the number of iterations was too large, so that the benefit in reduced communication (if any) was out-weighted. Also note that for the cases were a weighted graph was used during the partitioning an increased cost per iteration could be observed. This may be due to higher communication requirements and/or an overall higher amount of overlap (so, larger subdomain solves). For example, for the *reliab1(2000)* matrix the same number of iterations is needed for the weighted and unweighted case to solve the system on 80 subdomains with a large choice of overlap. But both the time needed to construct

the preconditioner and the time needed to solve the linear system are smaller for the unweighted case (see Table 2.38).

Lastly, we comment on the choice of the number of subdomains. The best case for every tested matrix was always achieved with 80 subdomains (the largest number of subdomains tested). Overall, we usually achieved a reduction in both the time needed to construct the preconditioner and the time needed to solve the linear system. However, for some examples there was a rapid jump in the number of iterations needed when increasing the number of subdomains, so that the time needed to solve the linear system also increased; see for example Table 2.39 for the cases when overlap is chosen from consecutive subdomains.

Table 2.34: Results for the *ncd(20)* matrix. K is the number of subdomains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For the incomplete LU 3 levels of fill-in were used.

K	unweighted					weighted				
	constr.	AS		RAS		constr.	AS		RAS	
		solve	it	solve	it		solve	it	solve	it
small overlap chosen from all subdomains										
4	1.90	22.33	18	22.33	18	1.93	18.07	14	17.23	13
8	0.99	11.30	20	9.46	16	1.03	9.09	16	7.93	13
16	0.56	11.55	40	8.70	31	0.74	6.17	24	4.90	18
32	0.32	6.64	42	4.12	28	0.34	4.77	34	3.59	26
64	0.23	4.50	43	2.79	29	0.83	3.78	30	2.47	21
80	0.21	3.71	40	2.69	31	0.28	7.55	70	4.18	37
large overlap chosen from all subdomains										
4	4.33	27.79	19	25.47	17	4.66	24.12	15	21.05	12
8	3.69	14.38	19	11.22	13	3.29	15.22	18	11.66	11
16	2.28	16.86	40	10.69	25	2.67	15.43	33	9.97	18
32	1.55	8.88	34	4.91	16	2.17	10.73	33	6.60	17
64	2.42	27.59	91	8.65	27	3.85	16.38	39	7.04	17
80	2.45	29.18	94	8.74	25	3.20	34.99	92	7.37	17
small overlap chosen from consecutive subdomains										
4	1.85	24.48	19	23.41	18	1.66	17.97	14	17.09	13
8	0.90	15.82	28	14.06	25	0.79	12.23	25	11.73	24
16	0.47	53.31	197	12.79	44	0.42	5.08	20	4.85	19
32	0.26	70.78	>500	27.83	197	0.24	3.67	28	3.42	26
64	0.17	45.60	>500	13.32	146	0.16	3.23	35	3.18	33
80	0.17	45.01	>500	44.57	>500	**0.16**	2.87	32	**2.66**	**30**
large overlap chosen from consecutive subdomains										
4	2.09	26.02	19	24.90	18	2.29	24.73	16	21.63	13
8	1.04	18.16	30	14.53	24	1.28	10.68	17	9.41	14
16	0.62	14.67	47	12.26	41	0.82	8.20	25	7.10	21
32	0.38	75.89	>500	23.14	150	0.47	5.24	28	4.13	21
64	0.19	43.65	>500	13.01	146	0.27	4.50	39	3.49	31
80	0.18	45.75	>500	45.70	>500	0.25	4.18	37	3.80	34

Table 2.35: Results for the *ncd(25)* matrix. K is the number of subdomains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For the incomplete LU 3 levels of fill-in were used.

K	unweighted					weighted				
	constr.	AS		RAS		constr.	AS		RAS	
		solve	it	solve	it		solve	it	solve	it
small overlap chosen from all subdomains										
4	3.77	38.87	15	36.96	14	3.81	31.02	10	29.38	9
8	1.97	20.11	15	18.15	13	2.01	18.69	13	16.62	11
16	1.07	14.47	23	11.97	19	1.08	12.11	20	10.01	16
32	0.59	9.74	30	7.91	25	0.81	11.27	35	8.29	27
64	0.38	9.39	44	7.19	33	0.44	7.53	35	5.73	25
80	0.74	7.21	44	5.00	32	0.47	7.05	33	5.30	26
large overlap chosen from all subdomains										
4	8.00	45.20	15	38.98	12	8.89	40.85	12	38.71	11
8	6.16	32.34	20	22.76	12	6.69	25.01	12	20.54	8
16	3.94	21.43	26	14.99	17	4.96	24.94	25	16.42	14
32	2.69	26.01	50	9.72	18	3.62	29.70	50	13.37	22
64	3.86	16.50	33	12.07	25	6.58	22.14	38	12.26	21
80	4.80	38.05	93	11.13	27	5.19	24.30	43	11.25	20
small overlap chosen from consecutive subdomains										
4	3.56	42.49	16	38.52	14	3.30	30.61	10	28.91	9
8	2.43	128.35	96	110.10	85	1.63	16.22	11	15.34	10
16	1.21	99.19	147	59.28	89	1.16	14.48	25	13.27	23
32	0.65	20.38	59	17.64	48	0.45	13.68	42	11.98	38
64	0.37	94.14	>500	93.93	>500	0.29	8.07	39	8.25	38
80	0.32	69.54	>500	70.84	>500	**0.27**	6.01	35	**4.89**	**34**
large overlap chosen from consecutive subdomains										
4	3.69	53.90	20	47.24	17	4.46	82.39	30	68.92	25
8	2.62	125.63	92	109.55	82	2.73	22.98	15	20.75	13
16	1.29	38.72	51	29.81	42	1.33	16.17	23	13.10	18
32	0.69	22.43	65	18.65	49	0.67	18.03	48	13.63	39
64	0.46	97.82	>500	96.85	>500	0.48	8.99	42	7.67	37
80	0.39	71.36	>500	70.04	495	0.56	6.21	39	5.69	34

Table 2.36: Results for the *twod(14)* matrix. K is the number of subdomains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For the incomplete LU 20 levels of fill-in were used.

K	unweighted					weighted				
	constr.	AS		RAS		constr.	AS		RAS	
		solve	it	solve	it		solve	it	solve	it
small overlap chosen from all subdomains										
4	1.10	19.22	21	18.58	20	1.13	15.84	17	15.28	16
8	0.57	12.82	28	11.19	24	0.58	13.65	25	12.59	22
16	0.31	9.17	35	7.80	30	0.32	10.66	42	9.60	38
32	0.20	5.03	40	4.20	34	0.23	7.37	41	6.20	35
64	0.14	4.56	49	3.89	43	0.15	4.89	51	4.23	45
80	0.14	3.97	49	3.24	42	0.15	4.57	58	3.63	45
large overlap chosen from all subdomains										
4	2.11	19.24	20	16.67	16	2.17	15.79	16	14.77	14
8	1.23	10.79	21	9.40	17	1.23	12.25	20	10.56	15
16	0.74	6.75	24	5.39	18	0.77	9.21	32	7.58	25
32	0.50	4.53	33	3.76	27	0.58	7.55	37	5.17	24
64	0.52	3.84	38	2.74	27	0.44	4.51	41	3.19	29
80	0.40	3.13	38	2.30	28	**0.39**	3.16	36	**2.20**	**25**
small overlap chosen from consecutive subdomains										
4	0.92	19.06	21	18.34	20	0.98	17.91	21	17.86	21
8	0.49	17.09	38	16.55	37	0.49	23.51	48	22.51	46
16	0.26	14.83	54	14.58	52	0.25	18.15	82	17.75	80
32	0.18	7.64	67	7.50	66	0.17	14.44	88	13.74	85
64	0.11	6.93	84	6.65	81	0.11	7.54	88	7.13	83
80	0.12	6.29	89	5.88	84	0.11	6.76	93	6.36	89
large overlap chosen from consecutive subdomains										
4	0.94	19.53	21	17.05	17	0.96	18.26	21	17.66	20
8	0.50	18.83	41	16.81	37	0.52	20.70	40	18.39	35
16	0.26	16.54	63	14.87	52	0.27	20.22	87	18.39	80
32	0.19	11.40	85	8.60	64	0.17	12.99	96	11.30	86
64	0.12	7.53	91	6.63	81	0.12	8.10	93	7.19	84
80	0.12	6.67	93	5.91	84	0.12	8.63	118	6.68	89

Table 2.37: Results for the *twod(17)* matrix. K is the number of subdomains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For the incomplete LU 20 levels of fill-in were used.

K	unweighted					weighted				
	constr.	AS		RAS		constr.	AS		RAS	
		solve	it	solve	it		solve	it	solve	it
small overlap chosen from all subdomains										
4	2.70	66.68	30	66.61	30	2.97	67.55	28	65.71	27
8	1.42	35.57	31	32.39	28	1.40	42.73	35	40.23	33
16	0.78	35.21	60	29.19	47	0.79	30.45	47	26.99	42
32	0.45	21.45	77	17.93	62	0.50	23.34	70	21.05	61
64	0.30	13.63	78	11.53	65	0.30	17.86	93	14.95	80
80	0.28	13.72	84	12.11	75	0.27	17.45	96	15.35	87
large overlap chosen from all subdomains										
4	5.13	70.61	31	66.79	29	5.29	66.31	27	59.13	23
8	3.44	37.01	31	28.65	23	2.93	54.04	42	46.20	36
16	1.75	25.36	40	20.18	32	1.72	29.06	43	23.18	34
32	2.37	15.17	46	11.97	37	1.67	16.99	43	12.39	31
64	0.96	12.71	65	8.72	39	0.85	11.64	53	9.52	42
80	**0.85**	11.44	65	**7.77**	**42**	0.79	14.01	72	8.82	43
small overlap chosen from consecutive subdomains										
4	2.37	66.26	30	65.96	30	2.36	93.65	41	93.60	41
8	1.15	61.36	51	58.59	49	1.25	73.53	61	73.65	61
16	0.67	52.02	92	50.61	90	0.66	41.57	73	40.18	70
32	0.36	31.92	110	30.98	105	0.42	34.99	100	33.83	98
64	0.24	22.94	134	22.73	132	0.24	25.31	141	22.44	129
80	0.21	24.80	148	23.55	145	0.22	26.35	147	24.58	140
large overlap chosen from consecutive subdomains										
4	2.37	59.29	26	55.51	24	2.56	99.17	43	92.24	40
8	1.22	64.75	56	58.90	49	1.68	76.35	63	73.27	59
16	0.67	54.74	95	51.62	91	0.66	46.38	81	40.68	70
32	0.42	40.87	116	38.01	104	0.38	33.22	112	30.06	98
64	0.24	24.36	143	21.88	130	0.24	27.78	159	23.14	133
80	0.23	26.81	167	23.73	145	0.22	31.69	183	24.34	139

Table 2.38: Results for the *reliab1(1200)* matrix. K is the number of subdomains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For the incomplete LU 20 levels of fill-in were used.

K	unweighted					weighted				
	constr.	AS		RAS		constr.	AS		RAS	
		solve	it	solve	it		solve	it	solve	it
small overlap chosen from all subdomains										
4	1.69	45.70	36	45.66	36	1.65	31.92	23	31.90	23
8	0.82	16.75	24	17.34	25	0.85	14.17	20	14.61	21
16	0.46	15.55	36	13.31	32	0.47	18.52	48	16.23	43
32	0.27	14.03	68	13.17	63	0.27	14.40	69	12.64	59
64	0.20	8.35	79	7.49	71	0.22	10.24	91	8.56	78
80	0.19	9.75	72	9.05	67	0.18	8.46	66	7.42	57
large overlap chosen from all subdomains										
4	3.21	29.09	20	28.06	19	3.15	31.59	22	26.60	17
8	1.77	16.86	23	15.10	20	1.78	12.04	14	10.65	11
16	1.06	13.16	28	9.52	21	1.06	10.36	26	7.98	19
32	0.81	7.88	34	5.86	26	0.80	7.76	34	6.12	27
64	0.56	4.85	38	3.68	29	0.64	6.20	46	4.38	34
80	**0.52**	5.16	35	**3.64**	**26**	0.73	5.66	37	4.44	26
small overlap chosen from consecutive subdomains										
4	1.38	46.05	36	46.02	36	1.43	53.19	40	53.21	40
8	0.70	35.96	51	35.13	50	0.70	35.03	53	34.71	52
16	0.37	30.84	66	30.22	63	0.38	25.76	73	25.48	72
32	0.21	27.60	132	28.55	135	0.21	24.76	121	23.89	116
64	0.15	17.03	153	16.11	146	0.15	13.29	140	12.68	136
80	0.14	17.99	129	17.49	125	0.14	17.30	138	16.47	133
large overlap chosen from consecutive subdomains										
4	1.43	28.76	20	27.78	19	1.45	49.29	37	49.32	37
8	0.70	38.10	56	36.36	52	0.72	32.51	48	30.24	45
16	0.38	34.74	75	32.65	67	0.39	28.09	79	25.32	71
32	0.22	32.91	150	29.26	137	0.35	27.50	133	25.33	123
64	0.15	16.71	156	15.82	146	0.15	14.79	153	12.10	131
80	0.14	19.11	135	17.19	124	0.14	20.66	148	17.20	131

Table 2.39: Results for the *reliab1(2000)* matrix. K is the number of subdomains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For the incomplete LU 20 levels of fill-in were used.

K	unweighted					weighted				
	constr.	AS		RAS		constr.	AS		RAS	
		solve	it	solve	it		solve	it	solve	it
small overlap chosen from all subdomains										
8	2.47	63.10	35	59.79	33	2.38	81.59	47	76.11	44
16	1.36	71.70	71	69.96	69	1.31	79.01	79	74.20	76
32	0.75	43.40	85	38.92	77	0.75	37.71	57	34.38	49
64	0.49	36.39	111	29.59	92	0.47	25.61	80	21.86	68
80	0.41	27.38	91	24.15	82	0.41	25.05	89	21.06	77
large overlap chosen from all subdomains										
8	4.92	68.93	37	58.42	31	6.95	67.12	38	55.43	31
16	3.26	43.13	39	33.82	31	3.50	46.83	41	35.90	33
32	2.05	25.48	44	20.16	36	2.01	23.58	40	21.51	33
64	1.29	20.81	59	15.52	43	1.32	16.48	47	12.87	38
80	1.22	14.58	43	11.13	34	**1.63**	13.74	42	**9.55**	**32**
small overlap chosen from consecutive subdomains										
8	2.00	74.91	41	74.93	41	1.99	83.41	48	81.63	47
16	1.10	122.11	122	121.12	120	1.10	139.29	133	136.82	131
32	0.59	85.68	171	81.41	163	0.61	58.78	91	56.27	88
64	0.36	43.12	158	42.67	155	0.36	37.90	143	35.86	136
80	0.34	57.23	155	55.51	149	0.32	40.84	146	39.51	142
large overlap chosen from consecutive subdomains										
8	2.10	82.22	44	76.16	41	2.12	79.39	45	73.04	42
16	1.55	126.60	127	122.12	120	1.14	147.40	138	133.27	126
32	0.77	88.35	177	83.25	165	0.77	67.67	100	57.27	89
64	0.38	47.17	172	42.64	151	0.36	40.88	153	35.53	135
80	0.34	60.69	167	54.91	147	0.32	44.26	156	40.55	143

Table 2.40: Results for the *reliab1(2800)* matrix. K is the number of subdomains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For the incomplete LU 20 levels of fill-in were used.

K	unweighted					weighted				
	constr.	AS		RAS		constr.	AS		RAS	
		solve	it	solve	it		solve	it	solve	it
small overlap chosen from all subdomains										
16	2.58	110.84	61	101.75	53	2.55	86.26	45	79.76	42
32	1.46	77.96	81	72.84	76	1.48	86.43	98	78.73	91
64	1.11	36.65	71	32.90	62	1.16	67.67	146	64.38	139
80	0.82	43.44	83	40.62	78	0.86	54.87	93	44.39	78
large overlap chosen from all subdomains										
16	6.54	88.72	45	71.61	37	6.63	72.54	37	60.54	31
32	4.08	51.06	47	39.99	38	4.05	49.00	49	39.05	40
64	3.02	33.93	51	24.24	41	3.23	42.50	81	31.32	56
80	**2.83**	35.39	63	**24.13**	**42**	2.78	45.16	74	26.19	42
small overlap chosen from consecutive subdomains										
16	2.23	203.07	117	203.81	117	2.19	184.70	100	182.96	99
32	1.16	154.65	158	148.57	150	1.20	113.67	133	109.10	128
64	0.88	60.78	131	57.31	124	0.72	87.77	198	87.78	198
80	0.63	74.46	139	74.32	139	0.64	90.03	171	91.63	173
large overlap chosen from consecutive subdomains										
16	2.18	207.48	117	204.68	117	2.79	199.68	111	184.35	99
32	1.16	176.02	181	150.18	150	1.21	121.78	142	108.59	128
64	0.85	65.56	140	55.24	119	0.71	133.13	300	105.17	238
80	0.63	80.83	148	74.56	139	0.68	117.05	199	96.91	167

Table 2.41: Results for the *reliab2(1200)* matrix. K is the number of subdomains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For the incomplete LU 20 levels of fill-in were used.

K	unweighted					weighted				
	constr.	AS		RAS		constr.	AS		RAS	
		solve	it	solve	it		solve	it	solve	it
small overlap chosen from all subdomains										
4	1.69	29.29	21	29.29	21	1.73	30.63	25	30.60	25
8	0.83	16.53	23	16.50	23	0.83	15.80	25	15.76	25
16	0.46	16.79	40	15.30	37	0.46	13.05	32	12.44	30
32	0.27	15.70	67	13.24	51	0.28	10.20	50	8.87	45
64	0.20	8.03	80	7.13	71	0.20	8.22	83	7.21	73
80	0.18	8.81	70	6.87	52	0.19	12.08	91	9.96	79
large overlap chosen from all subdomains										
4	3.22	24.44	15	22.64	13	3.18	26.78	19	24.30	16
8	1.76	14.57	18	12.95	15	1.79	13.92	19	12.15	15
16	1.06	10.55	24	8.27	18	1.08	8.77	22	7.73	17
32	0.81	9.76	38	6.33	26	0.72	7.29	34	5.77	27
64	0.64	5.27	43	4.06	34	0.58	4.93	40	4.08	32
80	**0.55**	4.17	29	**2.79**	**21**	0.55	5.26	35	3.61	25
small overlap chosen from consecutive subdomains										
4	1.37	43.59	34	43.59	34	1.40	30.61	25	30.62	25
8	0.70	21.48	31	20.76	30	0.71	29.95	49	29.96	49
16	0.36	30.20	76	29.70	75	0.38	26.27	71	26.65	70
32	0.21	26.34	106	25.71	101	0.22	16.24	91	15.32	87
64	0.14	16.27	166	16.04	161	0.15	11.64	127	11.16	120
80	0.15	15.91	129	15.50	123	0.14	21.28	164	20.30	155
large overlap chosen from consecutive subdomains										
4	1.44	44.10	34	44.06	34	1.46	26.72	19	24.23	16
8	0.71	23.40	33	21.99	31	0.75	31.99	50	31.02	49
16	0.38	30.08	76	26.51	66	0.55	26.34	71	26.99	70
32	0.22	29.87	125	25.60	101	0.24	17.80	96	16.00	88
64	0.15	17.50	178	15.61	152	0.17	12.69	135	11.12	120
80	0.15	17.23	137	14.52	116	0.15	24.47	187	20.44	154

Table 2.42: Results for the *reliab2(2000)* matrix. K is the number of subdomains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For the incomplete LU 20 levels of fill-in were used.

K	unweighted					weighted				
	constr.	AS		RAS		constr.	AS		RAS	
		solve	it	solve	it		solve	it	solve	it
small overlap chosen from all subdomains										
8	2.48	41.64	20	40.15	19	2.48	46.69	26	46.66	26
16	1.32	78.78	75	76.73	72	1.34	32.13	32	30.12	30
32	0.72	57.76	121	51.05	104	0.75	33.94	88	30.83	81
64	0.47	23.50	91	22.25	87	0.47	22.89	95	20.70	87
80	0.41	23.04	77	20.88	70	0.41	12.16	42	10.77	38
large overlap chosen from all subdomains										
8	4.90	43.28	20	40.18	18	4.96	42.68	22	38.66	19
16	3.34	28.96	25	26.43	23	3.41	28.69	27	22.23	20
32	1.85	28.96	52	24.27	44	1.92	18.96	41	16.20	35
64	1.34	14.62	47	12.25	41	1.47	14.51	49	11.83	41
80	1.33	13.09	39	10.30	31	**1.48**	10.54	35	**7.46**	**25**
small overlap chosen from consecutive subdomains										
8	2.03	53.46	27	53.39	27	2.08	76.95	45	76.90	45
16	1.09	69.17	68	69.73	68	1.11	59.36	61	58.05	60
32	0.60	53.61	109	50.85	101	0.62	50.57	138	50.00	137
64	0.36	36.13	144	34.94	141	0.36	28.21	123	27.07	118
80	0.32	34.18	113	32.94	107	0.33	23.29	84	21.27	78
large overlap chosen from consecutive subdomains										
8	2.94	54.59	27	54.42	27	2.15	78.45	45	77.97	45
16	1.56	52.06	47	51.30	46	1.14	64.42	68	59.01	61
32	0.61	52.60	104	45.95	92	0.89	54.32	142	49.36	132
64	0.39	43.82	177	35.51	141	0.40	26.56	112	24.01	98
80	0.33	31.14	99	27.05	89	0.32	24.63	88	20.26	75

Table 2.43: Results for the *reliab2(2800)* matrix. K is the number of subdomains, 'constr.' the time (in seconds) needed to construct the preconditioner, 'it' the number of iterations needed to reduce the 2-norm of the residual below 10^{-12}, 'solve' the time (in seconds). For the incomplete LU 20 levels of fill-in were used.

K	unweighted					weighted				
	constr.	AS		RAS		constr.	AS		RAS	
		solve	it	solve	it		solve	it	solve	it
small overlap chosen from all subdomains										
16	2.67	69.19	35	67.33	34	2.58	61.20	34	57.69	32
32	1.47	82.66	88	74.66	82	1.51	64.28	82	54.54	69
64	1.08	45.94	99	40.46	89	1.17	58.06	123	44.81	92
80	0.85	32.60	54	28.64	47	0.85	35.64	71	31.49	61
large overlap chosen from all subdomains										
16	6.43	58.95	29	53.30	26	9.78	56.44	31	48.68	26
32	3.98	68.23	71	50.51	48	4.10	63.53	79	43.68	48
64	3.04	37.02	69	28.37	46	3.14	49.98	96	33.45	63
80	2.63	26.21	42	18.79	31	**2.57**	21.93	38	**16.84**	**30**
small overlap chosen from consecutive subdomains										
16	2.14	149.26	80	149.73	80	2.23	83.79	46	80.90	45
32	1.19	116.65	129	115.44	129	1.20	65.86	84	61.21	78
64	0.92	102.64	199	100.36	197	0.85	63.51	138	61.12	133
80	0.64	55.59	98	53.16	95	0.62	58.51	119	58.82	119
large overlap chosen from consecutive subdomains										
16	2.15	150.34	80	150.37	80	3.06	84.89	47	81.69	45
32	1.19	133.78	145	118.33	131	1.64	67.54	84	62.28	78
64	0.95	108.69	215	108.18	214	0.95	75.26	161	64.40	137
80	0.64	59.07	105	53.30	95	0.65	58.58	119	59.46	119

2.11 Summary and conclusions

We have extended the RAS method to the computation of the stationary vector of large, sparse Markov chains. Our results suggest that when combined with GMRES acceleration and inexact solves, RAS is a promising approach for the solution of Markov chains with large, sparse transition matrices. Although primarily designed with parallel computing in mind, for sufficiently large problems the proposed technique is found to be superior to standard approaches (like ILU preconditioning) even in a sequential implementation. In the parallel case the method also shows promising results.

Chapter 3

Iterative solvers for the GeneRank problem

3.1 Introduction

Advances in biotechnology make it possible to collect a vast amount of genomic data. For example, using gene microarrays, it is now possible to probe a person's gene expression profile over the more than 30,000 genes of the human genome. Biomedical researchers try to link signals extracted from these gene microarray experiments to genetic factors underlying disease. One of the most important problems is to identify key genes that play a role in a particular disease.

A gene microarray consists of a large number of known DNA probe sequences that are put in distinct locations on a slide. Gene microarrays can be used for gene expression profiling. The DNA in a cell does not change, but certain derivatives of the DNA, the mRNA and tRNA, are produced as stimulation occurs through environmental conditions, and treatments. For more details, see [10, 22].

In 2005 Morrison et al. proposed a new model called GeneRank [81]. It is a modification of Google's PageRank algorithm. The model shares many of the mathematical

properties of PageRank, and the ranking of genes is reduced to the solution of a large linear system. Besides information gained from microarray experiments, GeneRank also considers known connection between genes. Thus, errors in the microarray experiments are less likely to influence the results than in methods which are based on expression levels alone.

3.2 Definitions and auxiliary results

Connection between genes can be constructed via the Gene Ontology (GO) database (http://geneontology.org). Let the set $G = \{g_1, g_2, \ldots, g_n\}$ consist of n genes in a microarray. Two genes g_i and g_j are connected if they share an annotation in GO. Similar to PageRank, the idea of GeneRank is that a gene is significant if it is connected to manly highly ranked genes. In contrast to PakeRank, the connection are not directed. Thus, instead of an un-symmetric hyperlink matrix, GeneRank considers the symmetric adjacency matrix W of the gene network. W is given by

$$w_{ij} = \begin{cases} 1 & \text{if } g_i \text{ and } g_j \ (i \neq j) \text{ share an annotation in GO,} \\ 0 & \text{otherwise.} \end{cases}$$

Note that W is unweighted, while the hyperlink matrix in PageRank is weighted so that each row sums up to one. A diagonal matrix D is constructed to provide such a scaling. Since a gene might not be connected to any of the other genes, W may have zero rows. We let deg_i denote the sum of the ith row (or column) of W, that is,

$$deg_i = \sum_{j=1}^{n} w_{ij} = \sum_{j=1}^{n} w_{ji}.$$

The diagonal matrix D is defined by $D = \mathrm{diag}(d_1, \ldots, d_n)$, where

$$d_i = \begin{cases} deg_i & \text{if } deg_i > 0 \\ 1 & \text{otherwise.} \end{cases}$$

Note that D is nonsingular and nonnegative by construction. Now, $(D^{-1}W)^t$ corresponds to the weighted hyperlink matrix in PageRank. In the case of GeneRank we do not need to modify the matrix further, since irreducibility is not needed.

So far only the connection between genes are considered, but not the results from the gene microarray experiment. Let $\boldsymbol{ex} = \begin{bmatrix} ex_1 & ex_2 & \ldots & ex_n \end{bmatrix}^t$ be a vector that is obtained from such an experiment. The entry $ex_i \geq 0$ is the absolute value of the expression change of gene g_i. In addition, a *damping factor* α with $0 \leq \alpha \leq 1$ that controls the weighting of expression change to connectivity is introduced. Then, the GeneRank algorithm can be written as a large linear system:

$$(I - \alpha W D^{-1})x^* = (1 - \alpha)\boldsymbol{ex}. \tag{3.1}$$

The vector x^* is called *GeneRank vector*, and the entries of it give some information about the significance of a gene. Note that for $\alpha = 0$ genes are ranked solely on the basis of the microarray experiment. If $\alpha = 1$, then a ranking is constructed using only the connectivity information, and the results from the microarray experiments are ignored.

3.3 Symmetric formulation of GeneRank

The matrix W is symmetric, but $I - \alpha W D^{-1}$ is not symmetric. Thus, for the solution of the linear system (3.1) non-symmetric method have to be used. The information about the symmetry of W cannot be exploited. In 2011 Wu et al. recognized that the

GeneRank problem can be rewritten as a symmetric linear system [117]. The main idea is simply to write the linear system (3.1) as

$$(D - \alpha W)D^{-1}x^* = (1 - \alpha)\boldsymbol{ex}, \tag{3.2}$$

or equivalently, as

$$(D - \alpha W)\hat{x} = (1 - \alpha)\boldsymbol{ex}, \tag{3.3}$$

with $\hat{x} = D^{-1}x^*$. The matrix $D - \alpha W$ is symmetric. With this modification, methods that are suitable for symmetric systems can be used for the GeneRanke problem. In the next section we will see that the symmetric GeneRank matrix has some additional nice properties.

3.4 Properties of the symmetric GeneRank matrix

Wu et al. also showed that the matrix $D - \alpha W$ has some nice properties besides symmetry. First of all, they showed that $D - \alpha W$ is positive definite. Thus, the conjugate gradient (CG) method [57] can be used to solve the linear system (3.3).

Wu et al. investigated the efficiency of the Jacobi preconditioner for $D - \alpha W$. In that case, the preconditioned matrix is given by

$$D^{-1/2}(D - \alpha W)D^{-1/2} = I - \alpha D^{-1/2}WD^{-1/2}.$$

Thus, the preconditioned linear system looks as follows:

$$(I - \alpha D^{-1/2}WD^{-1/2})\bar{x} = (1 - \alpha)D^{-1/2}\boldsymbol{ex}, \tag{3.4}$$

with $\bar{x} = D^{1/2}\hat{x} = D^{1/2}(D^{-1}x^*) = D^{-1/2}x^*$.

Wu et al. also showed that the eigenvalues of the preconditioned matrix are bounded as follows:

$$\begin{aligned} \lambda_{\max}(I - \alpha D^{-1/2} W D^{-1/2}) \leq & \quad 1 + \alpha \\ \lambda_{\min}(I - \alpha D^{-1/2} W D^{-1/2}) \geq & \quad 1 - \alpha \end{aligned} \tag{3.5}$$

The range of eigenvalues increases as α increases. Thus, the rate of convergence of CG decreases as α increases. Using Gershgorin's Circle Theorem, we can also bound the eigenvalues of $D - \alpha W$ as follows:

$$\begin{aligned} \lambda_{\max}(D - \alpha W) \leq & \quad (1 + \alpha) \max_{i=1,\ldots,n} \{d_i\} \\ \lambda_{\min}(D - \alpha W) \geq & \quad (1 - \alpha) \min_{i=1,\ldots,n} \{d_i\} \end{aligned} \tag{3.6}$$

In addition, we can show that $D - \alpha W$ and $I - \alpha D^{-1/2} W D^{-1/2}$ are nonsingular M-matrices.

Proposition 3.4.1.

1. *$D - \alpha W$ is a nonsingular M-matrix, for $0 \leq \alpha < 1$.*

2. *$I - \alpha D^{-1/2} W D^{-1/2}$ is a nonsingular M-matrix, for $0 \leq \alpha < 1$.*

Proof. We can rewrite the matrix as $D - \alpha W = \Delta I - (\tilde{D} + \alpha W)$, where Δ is the maximum degree of the underlying graph of W. That is, $\Delta = \max\{d_i \mid i = 1, \ldots, n\}$. The matrix $\tilde{D}$ is diagonal with $\tilde{d}_i = \Delta - d_i$ on the diagonal. Thus, $\tilde{D}$ is nonnegative, and with $\alpha > 0$ and $W \geq 0$ it follows that $\tilde{D} + \alpha W \geq 0$. Note that $\Delta I - (\tilde{D} + \alpha W)$ is a nonsingular M-matrix if $\rho(\tilde{D} + \alpha W) < \Delta$. The spectral radius is bounded above by the 1-norm. Thus, $\rho(\tilde{D} + \alpha W) \leq ||\tilde{D} + \alpha W||_1 = \max_{d_i}\{\Delta - d_i + \alpha d_i\} < \Delta$, for $0 < \alpha < 1$.

Wu et al. showed that $\rho(D^{-1/2} W D^{-1/2}) \leq 1$. It follows that $\rho(\alpha D^{-1/2} W D^{-1/2}) \leq \alpha < 1$. Since $D^{-1/2} W D^{-1/2}$ is nonnegative, $I - \alpha D^{-1/2} W D^{-1/2}$ is a nonsingular

M-matrix. □

3.5 Methods tested

Wu et al. successfully employed the Jacobi preconditioner together with CG for the solution of the linear system (3.3). They compared the method with the original GeneRank, the Jacobi and (Modified) Arnoldi iterations. CG preconditioned with Jacobi was faster for every example tested. However, the number of iterations increases as α increases.

We investigated the efficiency of the (restricted) additive Schwarz method for the solution of the linear system (3.4). Note that the restricted additive Schwarz preconditioner is not symmetric and cannot be used for CG. We compared CG preconditioned with additive Schwarz with the iterative additive and restricted additive Schwarz method. We found that the convergence rates for CG with AS preconditioning are encouraging. In particular, the number of iterations does not increase as α increases. However, the construction time for the preconditioner is quite high. In addition, each iteration of CG is more expensive due to the application of AS. Some results can be seen in Appendix B.

As an alternative, we propose to use the Chebyshev iteration. The advantage of the Chebyshev iteration is that the cost per iteration is very low. A disadvantage is that we need bounds on the eigenvalues. But, as we saw earlier, here we do have bounds on the largest and smallest eigenvalue of $I-\alpha D^{-1/2}WD^{-1/2}$, that can be used for the Chebyshev iteration. Details about the algorithm are given in Algorithm 3.1. The parameters $l_{\min}$ and $l_{\max}$ are bounds on the smallest and largest eigenvalue of the preconditioned matrix. In our case, $l_{\min} = 1-\alpha$ and $l_{\max} = 1+\alpha$.

In the next section we describe the test problems and afterwards we give the results of our numerical experiments.

Algorithm 3.1 Chebyshev iteration

$d = (l_{\max} + l_{\min})/2$, $c = (l_{\max} - l_{\min})/2$
$x = x_0$, $r = b - Ax$
for $i = 1, 2, \ldots, m$ **do**
    $z = M^{-1}r$
    if $i = 1$ **then**
        $p = z$
        $\alpha = 2/d$
    else
        $\beta = (c \cdot \alpha/2)^2$
        $\alpha = 1/(d - \beta)$
        $p = z + \beta \cdot p$
    end if
    $x = x + \alpha \cdot p$
    $r = b - Ax$
    if $norm(r) < tol$ **then**
        break;
    end if
end for

3.6 Description of test problems

We use two different types of matrices. The first matrix is the SNPa matrix (single-nucleotide polymorphism matrix). This matrix was kindly provided to us by Professor Yimin Wei of Fudan University. It has $n = 152,520$ rows and columns, and $639,248$ nonzeros. The nonzero structure of the SNPa matrix can be seen in Figure 3.1.

The second type is a class of matrices that comes from a range-dependent random graph model called RENGA. Two vertices i and j are connected with probability $\beta\lambda^{|j-i|-1}$, where $0 < \lambda < 1$ and $\beta > 0$ are given parameters. These networks capture the connectivity structure seen in proteome interaction data [52, 51]. A MATLAB code is available from [107]. In our experiments we set $\lambda = 0.9$ and $\beta = 1$, the default values in RENGA. Note that with $\beta = 1$ node i is connected to node $i+1$ for $i = 1, \ldots, n-1$.

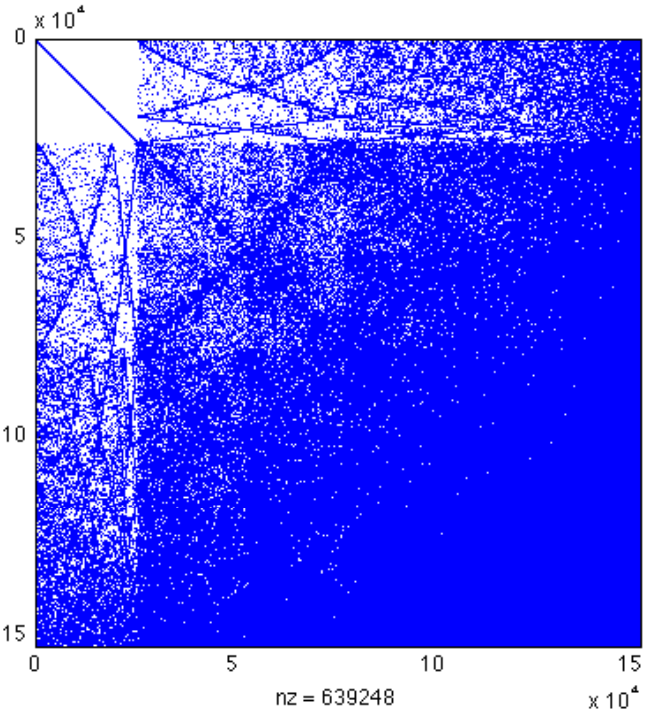


Figure 3.1: Nonzero pattern of the SNPa matrix.

3.7 Numerical experiments

We compare the Chebyshev method with the conjugate gradient method and a Jacobi preconditioned conjugate gradient. The Chebyshev iteration is used to solve the system $(I - \alpha D^{-1/2} W D^{-1/2})\hat{x} = (1-\alpha) D^{-1/2} \boldsymbol{ex}$ using the bounds on the largest and smallest eigenvalue given in (3.5). The stopping criteria for each of the methods depends on the 1-norm of the residual. That is, we stop once $\|r\|_1 < tol$. The initial guess is the zero vector. We use two different choices for $\boldsymbol{ex}$: $\boldsymbol{ex} = (1/n)\boldsymbol{e}$, where $\boldsymbol{e}$ is the vector of all ones, and $\boldsymbol{ex} = p$, where p is a random probability vector.

The results for the SNPa matrix are given in Tables 3.1 and 3.2, and the results for the RENGA matrices are given in Tables 3.3 and 3.4.

Note that the rate of convergence depends only on α and not on the size of the matrix. The eigenvalues of $I - \alpha D^{-1/2} W D^{-1/2}$ are fairly uniformly distributed in the interval $[1-\alpha, 1+\alpha]$. In such a case $\kappa_2(I - \alpha D^{-1/2} W D^{-1/2}) = \frac{\lambda_{\max}}{\lambda_{\min}} \leq \frac{1+\alpha}{1-\alpha}$ gives a

Table 3.1: Results for the SNPa matrix. The matrix has $n = 152,520$ rows and columns. The tolerance used is 10^{-10}. Here $\boldsymbol{ex} = (1/n)\boldsymbol{e}$, where $\boldsymbol{e}$ is the vector of all ones. The number of iterations and the CPU time in seconds (in brackets) are given.

α	0.50	0.75	0.80	0.99
CG	86 (1.06)	107 (1.38)	145 (1.79)	470 (6.11)
PCG	17 (0.26)	24 (0.35)	35 (0.51)	91 (1.28)
Chebyshev	17 (0.10)	25 (0.14)	37 (0.21)	130 (0.73)

Table 3.2: Results for the SNPa matrix. The matrix has $n = 152,520$ rows and columns. The tolerance used is 10^{-10}. Here $\boldsymbol{ex} = p$, where p is a random probability vector. The number of iterations and the CPU time in seconds (in brackets) are given.

α	0.50	0.75	0.80	0.99
CG	87 (1.07)	109 (1.40)	148 (1.80)	484 (5.54)
PCG	17 (0.23)	24 (0.32)	35 (0.48)	90 (1.19)
Chebyshev	17 (0.10)	25 (0.14)	37 (0.21)	131 (0.73)

Table 3.3: Results for the RENGA matrices. The tolerance used is 10^{-10}. Here $\boldsymbol{ex} = (1/n)\boldsymbol{e}$, where $\boldsymbol{e}$ is the vector of all ones. The number of iterations and the CPU time in seconds (in brackets) are given.

α	0.50	0.75	0.80	0.99
		n=100,000		
CG	22 (0.23)	26 (0.27)	33 (0.34)	105 (1.07)
PCG	13 (0.15)	19 (0.22)	27 (0.31)	92 (1.06)
Chebyshev	17 (0.11)	24 (0.16)	35 (0.23)	125 (0.81)
		n=500,000		
CG	23 (1.46)	25 (1.59)	33 (2.11)	106 (9.29)
PCG	13 (0.92)	19 (1.37)	27 (1.92)	92 (6.56)
Chebyshev	17 (0.59)	24 (0.87)	35 (1.27)	125 (4.47)

good estimate of the rate of convergence of CG and Chebyshev. Also note that we tried unsuccessfully to use a direct solver (sparse Cholesky) for the SNPa matrix, due to excessive fill-in.

Table 3.4: Results for the RENGA matrices. The tolerance used is 10^{-10}. Here $\boldsymbol{ex} = p$, where p is a random probability vector. The number of iterations and the CPU time in seconds (in brackets) are given.

α	0.50	0.75	0.80	0.99
		n=100,000		
CG	23 (0.24)	26 (0.27)	36 (0.37)	116 (1.19)
PCG	14 (0.16)	20 (0.23)	29 (0.34)	98 (1.13)
Chebyshev	17 (0.11)	24 (0.16)	35 (0.23)	125 (0.81)
		n=500,000		
CG	23 (1.45)	26 (1.64)	35 (2.23)	117 (7.42)
PCG	14 (0.99)	20 (1.42)	29 (2.06)	99 (7.03)
Chebyshev	17 (0.59)	24 (0.87)	35 (1.27)	125 (4.48)

3.8 Summary

We investigated two methods for the solution of the linear system arising from the gene ranking problem. While additive Schwarz as a preconditioner for CG leads to a method with a small number of iterations that does not depend on the value of the damping parameter α, a high construction time makes this method less desirable. Good results were obtained with the Chebyshev iteration. While the number of iteration is higher than for CG with Jacobi preconditioner, the cost per iteration is much cheaper and leads to a faster convergence in terms of CPU time. Also, the Chebyshev iteration is more desirable in a parallel setting as it avoids computing dot products.

Chapter 4

Disaggregation techniques for large scale-free graphs

4.1 Indroduction

One faces many challenges when analyzing scale-free networks. The networks of interest are usually huge and are constantly increasing in size. For example, a Google search on the number "2" in October 2010 returned about 16,250,000,000 results while the same search in January 2012 returned about 25,270,000,000 results. Thus, it is required to use distributed memory systems for computations on large scale-free graphs.

In network analysis one is often interested in finding a large number of eigenvalues and eigenvectors of the graph Laplacian. The computation of eigenvalues and eigenvectors of scale-free graphs is very expensive. Any Krylov method-based eigensolver (the methods of choice for large-scale problems) spends the majority of the time in the matrix-vector product. Yoo et al. identified the increased communication overhead in the matrix-vector product as a significant performance bottleneck for parallel algorithms for scale-free graphs [118]. A common technique to improve the commu-

nication requirements is to re-partition the matrix before starting any computations. However, state of the art parallel graph partitioners such as ParMETIS [65] and PT-Scotch [30] were designed for graphs with a more regular or uniform degree distribution. They employ multilevel partitioning algorithms [24, 55, 62, 63, 59, 60, 64] which depend on coarsening the graph until it is small enough to be efficiently partitioned. In the case of scale-free graphs these partitioners produce partitions that only slightly improve the communication behavior and require a high amount of time and memory [1]. The aforementioned graph partitioners attempt to partition the nodes of a graph. Edge or 2D partitioning has been successfully used to improve the scalability of the matrix-vector product [118]. While matrix-vector products based on 2D partitioning can be easily used for matrix-free eigensolvers, available multilevel methods require a row-(or edge-)wise distribution of the matrix.

Here we present a method that embeds the original irregular graph into a more regular one by disaggregating (splitting up) vertices in the original graph. The matrix-vector operations for the original graph are performed via a factored triple matrix-vector product involving the embedding graph. Even though the latter graph is larger, we are able to decrease the communication requirements considerably and improve the performance of the matrix-vector product.

4.2 Disaggregation

Scale-free graphs have a very irregular degree distribution. The existence of a few very high degree nodes together with many small degree nodes make computations with these types of graph particularly challenging. We attempt to tackle these challenges by disaggregation, or splitting up nodes in the graph. This results in a larger graph with a more favorable structure. Enlarging a graph of a sparse matrix with unfavorable structure to a graph with a more desirable structure has been used before, for example,

in the context of matrix stretching, where dense rows (columns) in a sparse matrix are split into several more sparse rows (columns) [2, 49]. For other methods that use matrix enlarging, see [4]. In the finite element literature, the popular FETI (finite element tearing and interconnecting) technique [43] can be viewed as a specialized disaggregation method.

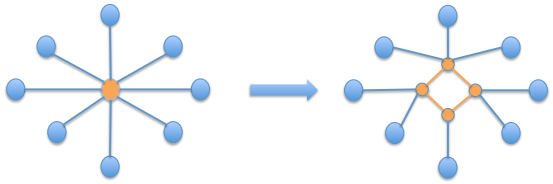


Figure 4.1: Disaggregating a node and using a circle as connections between the new nodes.

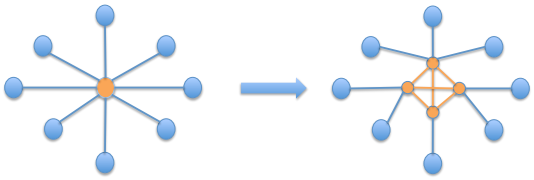


Figure 4.2: Disaggregating a node and using a complete graph as connections between the new nodes.

If a node i is disaggregated, it is split up into several new nodes $i_1, \ldots, i_k$ and every neighbor of node i in the graph is connected to exactly one of the nodes $i_1, \ldots, i_k$. Usually, we connect the nodes $i_1, \ldots, i_k$, called internal nodes, with a connected structure such as a cycle as seen in Figure 4.1 or a complete graph as seen in Figure 4.2. We can represent the disaggregated graph $\mathcal{G}_f$ as a combination of a graph that contains the same number of edges as the original graph $\mathcal{G}$ and a graph that contains only the new internal edges, called *internal graph*. A visualization can be seen in Figure 4.3. The matrix corresponding to the disaggregated graph $\mathcal{G}_f$ is denoted by A_f, the matrix with the same number of edges as $\mathcal{G}$, but nodes from $\mathcal{G}_f$ is denoted by B, and the matrix corresponding to the internal graph is denoted by C. We can

write A_f as $A_f = B + sC$, where s is the weight given to the internal edges. Under certain conditions the eigenvalues of A_f approximate the eigenvalues of A provided that the weight s on the internal edges is chosen large enough.

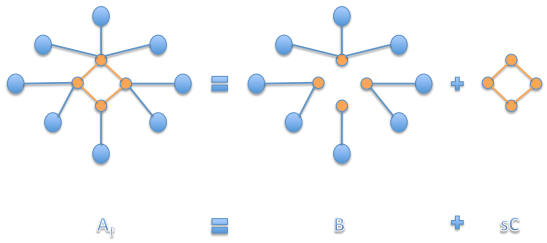


Figure 4.3: Presenting the disaggregated graph as a combination of the graph with internal and external edges.

Next, we construct a nonnegative intergraph-transfer operator Q. Let n denote the number of nodes in the original graph $\mathcal{G}$ and $n_{\mathcal{G}_f}$ the number of nodes in the disaggregated graph $\mathcal{G}_f$. The operator Q is used to prolongate a vector from $\mathbb{R}^n$ to $\mathbb{R}^{n_{\mathcal{G}_f}}$,

$$Q : \mathbb{R}^n \to \mathbb{R}^{n_{\mathcal{G}_f}}$$

The matrix Q is constructed as follows. If $i = 1, \ldots, n_{\mathcal{G}_f}$ are the nodes in $\mathcal{G}_f$ and $j = 1, \ldots, n$ are the nodes in $\mathcal{G}$, then $Q_{ij} = 1$ whenever node i in $\mathcal{G}_f$ represents node j in $\mathcal{G}$. Thus, if node j in $\mathcal{G}$ is not disaggregated, the jth column in Q has exactly one entry. If j is disaggregated into k nodes, the jth column contains exactly k entries. Now, the original matrix A can be written in terms of the disaggregated matrix A_f by

$$A = Q^t A_f Q.$$

That is, we can replace the matrix-vector product Ax by the factored triple matrix-vector product $Q^t(A_f(Qx))$. Note that $C \cdot Q = 0$ and the internal graph is not necessary for the factored triple matrix-vector product. That is, one can use $Q^t(B(Qx))$ for the matrix-vector product. The transfer operator Q can be scaled such that its

columns are orthonormal. Let $D \in \mathbb{R}^{n\times n}$ be a diagonal matrix such that $D_{i,i} = \frac{1}{\sqrt{k}}$ if node i is disaggregated into k nodes. Then, $\hat{Q} = Q \cdot D$ has the following properties:

- $\hat{Q}^t\hat{Q} = I_n$ and
- $\hat{Q}\hat{Q}^t$ is a projection onto the range of $\hat{Q}$.

In matrix form, the disaggregation can be described as follows. Consider the following matrix

$$\mathcal{A} = \begin{bmatrix} A_{11} & A_{12} & 0 \\ A_{21} & A_{22} & \mathbf{a} \\ 0 & \mathbf{a}^t & \alpha \end{bmatrix}.$$

Here $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ is a $n \times n$ matrix, $\mathbf{a} \in \mathbb{R}^m$, $m \leq n$, and $\alpha \in \mathbb{R}$. In our application, the last row (and column) will correspond to a vertex of the associated sparse graph for $\mathcal{A}$ with large degree m that we want to disaggregate.

We are interested in the following "matrix embedding". Let $\mathbf{e} = (1) \in \mathbb{R}^m$ be the constant vector of all ones. Form the $m \times m$ diagonal matrix $D = \mathrm{diag}(d_i)_{i=1}^m$, where $\mathbf{d} = (d_i)_{i=1}^m$ is to be determined later on. For example, let Λ be the $m \times m$ graph Laplacian matrix corresponding to a cycle:

$$\Lambda = \begin{bmatrix} 2 & -1 & 0 & \dots & -1 \\ -1 & 2 & -1 & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & -1 & 2 & -1 \\ -1 & 0 & \dots & -1 & 2 \end{bmatrix}. \tag{4.1}$$

In what follows, Λ can be any graph Laplacian matrix corresponding to a graph defined by the sparsity structure imposed on the additionally introduced nodes. This graph sometimes (in what follows) will be referred to as "*internal graph.*" For any

such internal graph Laplacian, we have $\Lambda \mathbf{e} = 0$.

Given a parameter $s \geq 0$ and a given vector $\mathbf{c}_2 \in \mathbb{R}^m$, we form the $m \times m$ diagonal matrix $C_2 = \text{diag}(\mathbf{c}_2)$ (i.e., $C_2\mathbf{e} = \mathbf{c}_2$), and consider

$$T = -DC_2 + s\Lambda.$$

We are interested in the following $(n+m) \times (n+m)$ embedding matrix:

$$\mathcal{A}_f = \begin{bmatrix} A_{11} & A_{12} & 0 \\ A_{21} & A_{22} & D \\ 0 & D & T \end{bmatrix}. \tag{4.2}$$

From now on we assume that the original matrix $\mathcal{A}$ corresponds to the graph Laplacian. For the considerations below it is sufficient to assume that $\mathcal{A}\mathbf{c} \geq 0$ for a positive vector

$$\mathbf{c} = \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \\ \sigma \end{bmatrix},$$

and that $\mathcal{A}$ has nonpositive off-diagonal entries. In that case, we choose the diagonal matrix $D = \text{diag } (d_i)_{i=1}^m$ as $d_i = \sigma a_i$, where $\mathbf{a} = (a_i)$. It is clear that $d_i < 0$ and $D\mathbf{e} = \mathbf{a}\sigma$.

We have the following result.

Lemma 4.2.1. *The embedding matrix $\mathcal{A}_f$ has nonpositive off-diagonal entries and its action on the positive vector $\mathbf{c}_f = \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \\ \mathbf{e} \end{bmatrix}$ results in a nonnegative vector that has the same values as $\mathcal{A}\mathbf{c}$ for the common rows of $\mathcal{A}$ and $\mathcal{A}_f$. For the embedding rows this action is zero. That is, if $\mathbf{c}$ is a null-vector of $\mathcal{A}$, then $\mathbf{c}_f$ is a null-vector of $\mathcal{A}_f$.*

In the latter case, consider

$$Q = \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & \frac{1}{\sigma}\mathbf{e} \end{bmatrix}.$$

Then, the following Galerkin relation holds:

$$\mathcal{A} = Q^t \mathcal{A}_f Q. \tag{4.3}$$

Proof. We first notice that all off–diagonal entries of $\mathcal{A}_f$ are nonpositive. What remains to show is that its action on the positive vector $\begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \\ \mathbf{e} \end{bmatrix} \in \mathbb{R}^{n+m}$ results in a nonnegative vector. In fact, that action for the common rows of $\mathcal{A}$ and $\mathcal{A}_f$ is the same as the action of the original matrix $\mathcal{A}$ (which is nonnegative by assumption). We have

$$\mathcal{A}_{21}\mathbf{c}_1 + A_{22}\mathbf{c}_2 + D\mathbf{e} = \mathcal{A}_{21}\mathbf{c}_1 + A_{22}\mathbf{c}_2 + \mathbf{a}\sigma = (\mathcal{A}\mathbf{c})_2 .$$

For the embedding rows, the action is zero due to the choice of Λ and T. Indeed, we have

$$D\mathbf{c}_2 + T\mathbf{e} = D\mathbf{c}_2 + (-DC_2)\mathbf{e} + \Lambda\mathbf{e} = D\mathbf{c}_2 + (-D\mathbf{c}_2) = 0.$$

To prove the Galerkin relation (4.3), from the third component of $\mathcal{A}\mathbf{c} = 0$, we obtain

$$\mathbf{a}^t\mathbf{c}_2 + \alpha\sigma = 0. \tag{4.4}$$

Then, by direct computation, we have (using $C_2\mathbf{e} = \mathbf{c}_2$, $D\mathbf{e} = \mathbf{a}\sigma$, and hence $\mathbf{e}^t D = \sigma\mathbf{a}^t$)

$$
\begin{aligned}
Q^t \mathcal{A}_f Q &= \begin{bmatrix} A_{11} & A_{12} & 0 \\ A_{21} & A_{22} & \frac{1}{\sigma} D\mathbf{e} \\ 0 & \frac{1}{\sigma}\mathbf{e}^t D & \frac{1}{\sigma}\mathbf{e}^t \left(-DC_2 + s\Lambda\right) \frac{1}{\sigma}\mathbf{e} \end{bmatrix} \\
&= \begin{bmatrix} A_{11} & A_{12} & 0 \\ A_{21} & A_{22} & \mathbf{a} \\ 0 & \mathbf{a}^t & \frac{1}{\sigma^2}\mathbf{e}^t(-D\mathbf{c}_2) \end{bmatrix} \\
&= \begin{bmatrix} A_{11} & A_{12} & 0 \\ A_{21} & A_{22} & \mathbf{a} \\ 0 & \mathbf{a}^t & -\frac{1}{\sigma}\mathbf{a}^t\mathbf{c}_2 \end{bmatrix} \\
&= \begin{bmatrix} A_{11} & A_{12} & 0 \\ A_{21} & A_{22} & \mathbf{a} \\ 0 & \mathbf{a}^t & \alpha \end{bmatrix} = \mathcal{A}.
\end{aligned}
$$

In the last equality, we used (4.4). This completes the proof. □

In particular, it follows from Lemma 4.4 that for $\sigma = 1$ and c_2 vector of all ones, $\mathcal{A}_f\mathbf{e} = 0$. Since $\mathcal{A}_f$ has nonpositive off-diagonal entries, it is a graph Laplacian. In conclusion, we have the following result.

Theorem 4.2.2. *The graph Laplacian matrix $\mathcal{A}$ can be embedded into a larger matrix $\mathcal{A}_f$ that is also a graph Laplacian matrix and has a smaller number of non-zeros per row. The two graph Laplacians are related via a Galerkin relation, $\mathcal{A} = Q^t\mathcal{A}_f Q$ for a block–diagonal aggregation type matrix Q, where each column of Q contains non-zero constant entries in its rows that arise from a disaggregated vertex.*

4.3 Parallel Disaggregation

The communication overhead during the matrix-vector product has been identified as the performance bottleneck for parallel eigensolvers for scale-free graphs. In our experience, large scale-free graphs require communication between all processors during the matrix-vector product. Even after re-partitioning with a graph partitioner such as ParMETIS or PT-Scotch, communication is needed between all of the processors. Abou-Rjeili and Karypis explained the inability of these partitioners to find a suitable partitioning for scale-free graphs as follows [1]. Most of the available partitioners rely on multilevel methods where the graph is coarsened until it is small enough to employ partitioning directly. The coarsening methods use vertex matching to reduce the number of nodes, and this method depends on finding large enough matchings to coarsen the graph efficiently. Scale-free graphs have a very irregular degree distribution and a large number of low degree nodes are connected to a small number of very high degree nodes. This property leads to relatively small matchings and the partitioner needs a high number of levels to coarsen the graph enough to be able to partition it. In addition, this leads to a very high memory demand, and is not suitable for large graphs. Furthermore, scale-free graphs usually also have the small-world property, that is, the graph has a small diameter and high average clustering. This property also makes partitioning more challenging.

A good vertex-based partitioning tries to balance the computational work by assigning roughly the same number of vertices and edges to each processor while at the same time minimizing the communication requirements. Banded matrices are optimal for communication pattern, as every processor only needs to communicate with a few close neighbors. We use disaggregation of nodes to embed the original graph in a larger graph whose matrix representation has a more banded structure. We strive to achieve a given inter-processor communication structure. Any node that violates this communication pattern is disaggregated and copies of this node

are distributed among the processors in such a way that the desired communication pattern is not violated. We usually restrict the communication to a percentage or fraction of the other processors.

The communication pattern that we are enforcing depends on the"distance" between two processors. We define the distance of two processors as follows.

Definition 4.3.1. Assume that the number of processors is np. For two processors P and Q, we define the *distance* between the two processors as

$$dist(P, Q) = min(|P - Q|, np - |P - Q|),$$

where P and Q are the indices (or ranks) of the processors. That is, $0 \leq P,\ Q \leq np-1$.

In particular, the distance between processor 0 and processor $np - 1$ is one. If communication is restricted to a fraction $p \in (0, 1)$ of the number of processors np, we say that node v violates the communication pattern if for some neighbor u

$$dist(PROC(v), PROC(u)) > \lfloor \frac{p \cdot np}{2} \rfloor,$$

where $PROC(v)$ and $PROC(u)$ denote the processors that hold v and u, respectively.

The following proposition and its proof show how a node is split up and the neighbors are connected to the new nodes.

Proposition 4.3.2. *Assume that communication should be restricted to a fraction $p \in (0, 1)$ of the number of processors np. This restriction can be fulfilled by disaggregating any node in the given graph $\mathcal{G} = (V, E)$ that violates the communication pattern into f nodes, where $f = \lceil \frac{np}{l} \rceil$, with $l = \lfloor \frac{p \cdot np}{2} \rfloor$.*

Proof. Node $i \in V$ violates the communication pattern if there exist $j \in V$ with $(i, j) \in E$ such that $dist(PROC(i), PROC(j)) > l$. We disaggregate node i into f nodes $i_0, \ldots, i_{f-1}$. That is, in the disaggregated graph $\mathcal{G}_f$ node i is represented by

the nodes $i_0, \ldots, i_{f-1}$, where node i_k, $k = 0, \ldots, f-1$ lies on processor $(PROC(i) + k \cdot l) \bmod np$. The neighbors of i in graph $\mathcal{G}$ are connected to the nodes $i_0, \ldots, i_{f-1}$ in the disaggregated graph $\mathcal{G}_f$ as follows. If u is a neighbor of i in $\mathcal{G}$, that is $(i, u) \in E$, then u is connected to i_k in $\mathcal{G}_f$, where

$$k = \begin{cases} \lfloor \frac{PROC(u) - PROC(i)}{l} \rfloor, & \text{if } PROC(u) \geq PROC(i) \\ \lfloor \frac{np + PROC(u) - PROC(i)}{l} \rfloor, & \text{if } PROC(u) < PROC(i). \end{cases}$$

Since i_k lies on processor $proc = (PROC(i) + k \cdot l) \bmod np$, and $dist(PROC(u), proc) \leq l$, the desired communication pattern is not violated by the edge between u and i_k. $\square$

The details of the parallel disaggregation method are summarized in Algorithm 4.1. The input matrix A can be either a (weighted) incidence matrix or a matrix corresponding to a graph Laplacian. In both cases the output matrix A_f will be a Laplacian matrix. To derive A_f we use the representation of a Laplacian matrix in terms of its *edge-vertex incidence matrix* EV and *weight matrix* W [17, 19, 53],

$$A_f = (EV)^t \cdot W \cdot (EV)$$

For a graph $\mathcal{G} = (V, E)$ the edge-vertex matrix EV is a matrix of size $|E| \times |V|$ that is set up as follows. Each edge $e \in E$ is arbitrarily directed as $e = (u, v)$. The row e in EV has two entries, $EV(e, u) = 1$ and $EV(e, v) = -1$. The weight matrix W of size $|E| \times |E|$ is a diagonal matrix with positive entries on its main diagonal equal to the edge weights from the original graph $\mathcal{G}$.

In our algorithm, we first count the number of nodes that violate the desired communication pattern on every processor. Those are the nodes that need to be disaggregated. Each of these nodes i is disaggregated into f nodes $i_0, \ldots, i_{f-1}$, where i_0 replaces i on $PROC(i)$ and i_k is placed on processor $(PROC(i) + k \cdot l) \bmod np$. Thus, if n_d is the number of nodes in $\mathcal{G}$ that need to be disaggregated, the number

Algorithm 4.1 Parallel Disaggregation

Input symmetric matrix $A \in \mathbb{R}^{n \times n}$, fraction $p \in (0, 1)$
Output disaggregated matrix A_f, transfer operator Q

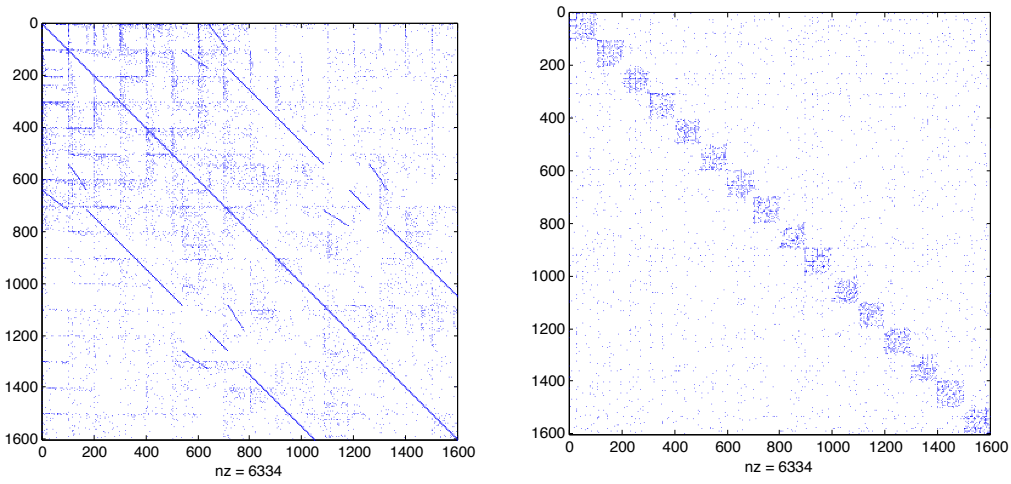
$np \leftarrow$ number of processors, $l \leftarrow \lfloor \frac{prop \cdot np}{2} \rfloor$, $f \leftarrow \lceil \frac{np}{l} \rceil$
for each processor *proc* **do**
  $count \leftarrow 0$
  for each row i that lies on processor *proc* **do**
    if $\exists j$ with $A(i,j) \neq 0$ AND $dist(proc, PROC(j)) > l$ **then**
      $count \leftarrow count + 1$ ▷ disaggregate i
    end if
  end for
  send *count* to the processors $(proc + k \cdot l) \bmod np$, $k = 1, \ldots, f-1$
    ▷ these processors receive copies of the disaggregated nodes from processor *proc*
  receive the corresponding *count* values from processors $(proc + k \cdot l) \bmod np$, $k = 1, \ldots, f-1$,
    ▷ determine the additional number of rows needed on this processor
end for
$e_{\mathcal{G}_f} \leftarrow$ number of edges in $\mathcal{G}_f$ ▷ this value depends on the internal graph
$n_{\mathcal{G}_f} \leftarrow$ number of nodes in $\mathcal{G}_f$
set up edge-vertex matrix EV of size $e_{\mathcal{G}_f} \times n_{\mathcal{G}_f}$
set up matrix Q of size $n_{\mathcal{G}_f} \times n$, diagonal matrix W of size $e_{\mathcal{G}_f} \times e_{\mathcal{G}_f}$
for each processor *proc* **do**
  for each row i that lies on processor *proc* **do**
    if $\exists j$ with $A(i,j) \neq 0$ AND $dist(proc, PROC(j)) > l$ **then**
      disaggregate i: represent row i by f rows $i_0, \ldots, i_{f-1}$
      set $Q(i_k, i) \leftarrow 1$ for $k = 0, \ldots, f-1$
      set edges between nodes $i_0, \ldots, i_{f-1}$ ▷ depends on internal graph
      for all j with $A(i,j) \neq 0$ **do**
        if $PROC(j) \geq proc$ **then** $k \leftarrow \lfloor \frac{PROC(j) - proc}{l} \rfloor$
        else $k \leftarrow \lfloor \frac{np + PROC(j) - proc}{l} \rfloor$
        $j_{\mathcal{G}_f} =$ index of node j in $\mathcal{G}_f$, $e \leftarrow (i_k, j_{\mathcal{G}_f})$
        $EV(e, i_k) \leftarrow 1$, $EV(e, j_{\mathcal{G}_f}) \leftarrow -1$, $W(e,e) \leftarrow |A(i,j)|$
      end for
    else
      $i_{\mathcal{G}_f} =$ index of node i in $\mathcal{G}_f$, $Q(i_{\mathcal{G}_f}, i) \leftarrow 1$
      for all j with $A(i,j) \neq 0$ **do**
        $j_{\mathcal{G}_f} =$ index of node j in $\mathcal{G}_f$, $e \leftarrow (i_{\mathcal{G}_f}, j_{\mathcal{G}_f})$,
        $EV(e, i_{\mathcal{G}_f}) \leftarrow 1$, $EV(e, j_{\mathcal{G}_f}) \leftarrow -1$, $W(e,e) \leftarrow |A(i,j)|$
      end for
    end if
  end for
end for
$A_f \leftarrow (EV)^t \cdot W \cdot EV$

of nodes in the disaggregated graph $\mathcal{G}_f$ is given by $n_{\mathcal{G}_f} = |V| + n_d \cdot (f-1)$.

The number of edges in the disaggregated graph $\mathcal{G}_f$ depends on the chosen internal graph that describes the connections between $i_0, \ldots, i_{f-1}$. If the internal graph is a cycle (see matrix Λ in (4.1)), one edge is added for every i_k, that is, the number of edges in $\mathcal{G}_f$ is given by $e_{\mathcal{G}_f} = |E| + n_d \cdot f$. After the number of nodes and edges of $\mathcal{G}_f$ are determined, we set up the required matrices $EV \in \mathbb{R}^{e_{\mathcal{G}_f} \times n_{\mathcal{G}_f}}$, $W \in \mathbb{R}^{e_{\mathcal{G}_f} \times e_{\mathcal{G}_f}}$, and $Q \in \mathbb{R}^{n_{\mathcal{G}_f} \times n}$.

In a second sweep through the original graph $\mathcal{G}$ the values in EV, W, and Q are being set. For every node i the process depends on if i is to be disaggregated. First, we describe the procedure if i needs to be disaggregated. For every i_k, $k = 0, \ldots, f-1$, set $Q(i_k, i) = 1$. In addition we need to connect the nodes $i_0, \ldots, i_{f-1}$. This depends on the chosen internal graph. If a cycle is used, we connect i_k with i_{k+1}, $k = 0, \ldots, f-2$ and i_{f-1} with i_0. That is, for every $e = (i_k, i_{k+1})$, $k = 0, \ldots, f-2$, we set $E(e, i_k) = 1$, $E(e, i_{k+1}) = -1$, and $W(e, e) = s$. We also set $E((i_{f-1}, i_0), i_{f-1}) = 1$, $E((i_{f-1}, i_0), i_0) = -1$, and $W((i_{f-1}, i_0), (i_{f-1}, i_0)) = s$. Next, the neighbors of i in $\mathcal{G}$ have to be connected to the appropriate i_k. For $i \in V$ such that $(i, j) \in E$, k is chosen as in the proof of proposition 4.3.2. Let $j_{\mathcal{G}_f}$ denote the index of j in $\mathcal{G}_f$. For $e = (i_k, j_{\mathcal{G}_f})$ we set $E(e, i_k) = 1$, $E(e, j_{\mathcal{G}_f}) = -1$, and $W(e, e) = |A(i, j)|$. Note that if j lies on the same processor as i, then $k = 0$ and the edge $e = (i_k, j_{\mathcal{G}_f})$ lies on the same processor as the edge (i, j) in $\mathcal{G}$. Next, we describe the procedure if node i does not need to be disaggregated. In this case we simply set $Q(i_{\mathcal{G}_f}, i) = 1$, where $i_{\mathcal{G}_f}$ denotes the index of i in $\mathcal{G}_f$. For every $e = (i, j) \in E$, we determine the corresponding edge $e_{\mathcal{G}_f} = (i_{\mathcal{G}_f}, j_{\mathcal{G}_f}) \in E_{\mathcal{G}_f}$ and set $E(e_{\mathcal{G}_f}, i_{\mathcal{G}_f}) = 1$, $E(e_{\mathcal{G}_f}, j_{\mathcal{G}_f}) = -1$, and $W(e_{\mathcal{G}_f}, e_{\mathcal{G}_f}) = |A(i, j)|$. After all values in EV and W have been set, A_f can be determined by $A_f = (EV)^t \cdot W \cdot (EV)$.

As described in the previous section, the original matrix A is given by $A = Q^t A_f Q$, where $A_f = B + s \cdot C$ is the disaggregated matrix and Q is an intergraph-transfer

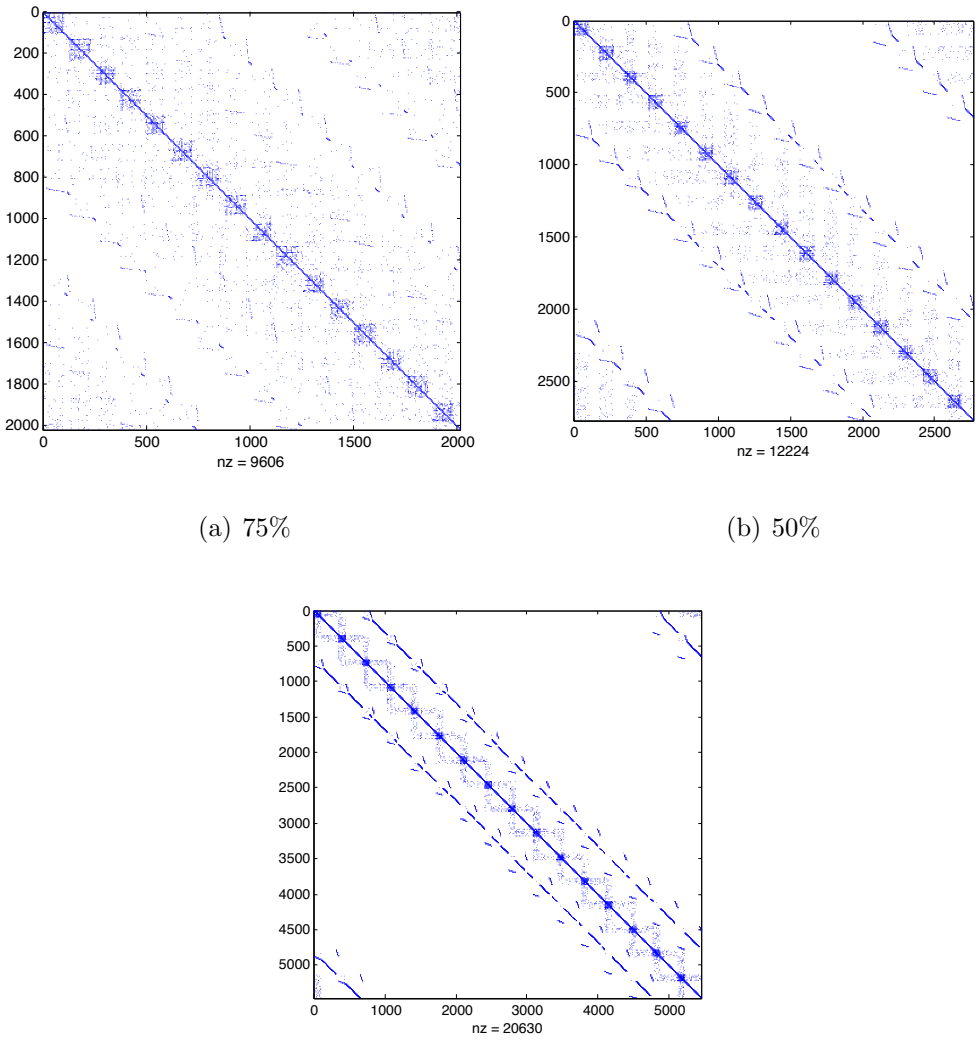


(a) Original (b) ParMETIS

Figure 4.4: Non-zero pattern of the original matrix, and the original matrix after redistributing with ParMETIS.

operator. For the matrix-vector product it is not necessary to connect the disaggregated nodes, that is, the matrix C can be omitted. However, if the disaggregated matrix is to be used in a different context, for example as an auxiliary preconditioner, the connectivity of the graph might be desirable. Proposition 4.3.2 only shows that the graph corresponding to matrix B does not violate the communication pattern. However, if a cycle is used to connect the internal nodes $i_0, \ldots, i_{f-1}$ the graph $\mathcal{G}_f$ corresponding to the disaggregated matrix A_f does not violate this pattern either. This can easily be seen as subsequent nodes i_k and i_{k+1} are assigned to processors $proc_1 = (proc + k \cdot l) \text{mod } np$ and $proc_2 = (proc + (k+1) \cdot l) \text{mod } np$, respectively, and $dist(proc_1, proc_2) \leq l$. In the following, we will use a cycle for the connection between internal (disaggregated) nodes and set the weights on these internal edges to one, that is $s = 1$.

In Figure 4.4 and 4.5 we present the non-zero structure of a scale-free graph before and after disaggregation. The scale-free graph was generated by the parallel scale-free graph generator [119]. The matrix is split up on 16 processors and has 100 nodes per processor. In Figure 4.4 on the left-hand side (a) the non-zero structure



(a) 75%

(b) 50%

(c) 25%

Figure 4.5: Non-zero pattern after using disaggregation to limit communication to 75%, 50%, or 25% of the other processors.

of the original matrix is shown. It turns out that this matrix structure leads to communication between every single processor during the matrix-vector product. In Figure 4.4 on the top right-hand side (b) the structure of the same matrix is given after it is reordered with ParMETIS. The matrix is denser on the block diagonal compared to the original matrix, but communication between all processors during the matrix-vector product is still required. ParMETIS mainly reduces the size of the

messages but not the number of messages that needs to be send between processors. Thus, we have a large number of small messages which is a particularly unfavorable setting for distributed systems. In Figure 4.5 the non-zero structure of the same matrix is shown after it is disaggregated, that is the non-zero structure of A_f is shown. We restricted the communication to 75%, 50% and 25% of the processors. Recall that every processor can communicate to processors that have a distance of at most $l = \lfloor \frac{p \cdot np}{2} \rfloor$. Thus, for the particular example with 16 processors every processor is only allowed to communicate with the closest 12, 8, or 4 processors.

While the communication volume is not reduced or only slightly reduced with disaggregation, the number of messages that are being sent during a matrix-vector product is significantly reduced. Thus, we have a relatively small number of large messages. In addition, each processor communicates with the same number of processors. This is a very favorable communication behavior for a distributed setting. Also, since the distribution of the disaggregated nodes is done in a very structured way, load balancing can be preserved provided that roughly the same number of nodes are disaggregated on every processor. The matrix-vector product will be implemented by the factored triple matrix-vector product $Q^t(A_f(Qx))$. Note that Q is very sparse. If a node is not disaggregated Q has exactly one entry in the corresponding column. If a node is disaggregated the number of entries in the corresponding column depends on the number of nodes that this node is split up into. Next, we show that this number does not increase as the number of processors increases.

Proposition 4.3.3. *If communication is restricted to a fraction $p \in (0, 1)$ of the number of processors np with $p \cdot np > 2$, then every processor can communicate solely with its $2 \cdot l = 2 \cdot \lfloor \frac{p \cdot np}{2} \rfloor$ nearest neighbors. Every node that violates the desired communication pattern is disaggregated into $f = \lceil \frac{np}{l} \rceil$ nodes, and f is bounded by*

$$\frac{2}{p} \leq f \leq \frac{2}{p - 2/np}.$$

Proof. From $l = \lfloor \frac{p \cdot np}{2} \rfloor$ it follows that

$$\frac{p \cdot np}{2} - 1 \leq l \leq \frac{p \cdot np}{2}.$$

Thus, f is bounded below by

$$f = \lceil \frac{np}{l} \rceil \geq \frac{np}{\frac{p \cdot np}{2}} = \frac{2}{p},$$

and above by

$$f \leq \frac{np}{\frac{p \cdot np}{2} - 1} = \frac{2}{p - \frac{2}{np}}.$$

□

That is, the communication requirement, meaning the number of messages, for a multiplication with Q does not increase even if the number of used processors increases.

4.4 Numerical Results

Our experiments were conducted on Hera, a large parallel system at Lawrence Livermore National Laboratory. Hera is a multicore Linux cluster with 864 nodes. Each node has 32 GB memory and 4 sockets with AMD Quadcore 2.3 GHz processors. The nodes are connected by Infiniband network.

In our implementation we use the PETSc library [7] for the matrix-vector product. We use a parallel scale-free graph generator [119] to test our method. The graph generator generates scale-free graphs using the preferential attachment method [9]. In addition, we used a real-world example from the WebGraph library [18]. This social graph, called Hollywood-2011, represents working relationships between actors. Nodes are actors, and two actors are joined by an edge whenever they appeared in a

movie together. In Table 4.1 the increase in matrix size of the disaggregated matrix is given as ratio between the size of the disaggregated and original matrix. The first two matrices are generated with the scale-free graph generator and have average degree two and five. The hollywood matrix has 2,180,759 nodes, 228,985,632 edges, and average degree 105.003. As expected, the more we restrict the communication, the more the size of the disaggregated matrix increases. While the matrix sizes increase considerably we will later see that the time saved during communication is large enough to compensate for the additional computational requirement.

Table 4.1: Ratio of the number of nodes of the disaggregated matrix and the original matrix.

	75%	50%	25%	10%
avg=2	1.7	2.6	5.8	15.4
avg=5	2.2	3.4	7.1	18.3
hollywood	1.7	2.4	5.0	12.8

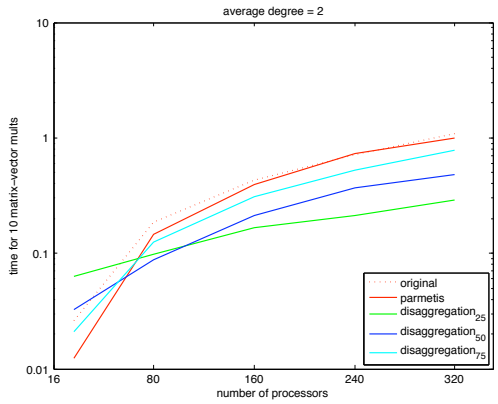
Table 4.2: Ratio of the number of nodes of the disaggregated matrix and the original matrix. The matrix was re-partitioned with ParMETIS before applying disaggregation.

	75%	50%	25%	10%
avg=2	1.4	1.9	3.9	9.7
avg=5	2.0	3.0	6.4	16.4
hollywood	1.66	2.4	4.77	12.72

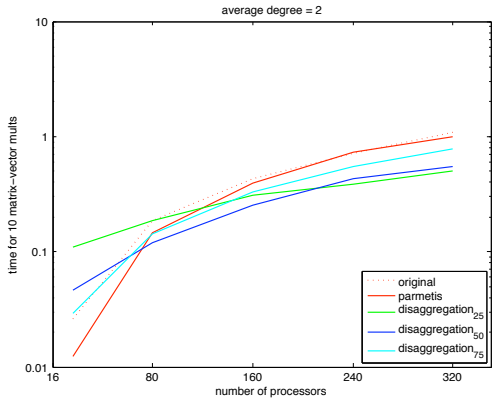
Table 4.3: Ratio of the number of edges of the disaggregated matrix and the original matrix.

	75%	50%	25%	10%
avg=2	1.5	2.2	4.2	10
avg=5	1.4	1.8	2.8	5.9
hollywood	1.03	1.06	1.13	1.36

We compared the size of the disaggregated matrix if re-partitioning with ParMETIS is applied before using disaggregation. The results are given in Table 4.2. Note that re-partitioning the matrix leads to slightly smaller disaggregated matrices. In Table 4.3 we provide the ratio of the number of non-zeros of the disaggregated matrix to the number of non-zeros in the original matrix. The increase in number of non-zeros results purely from adding internal edges.

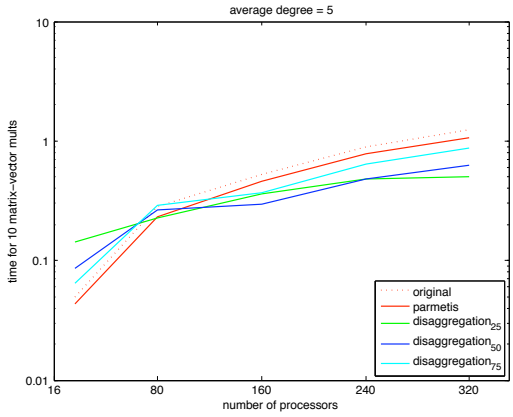


(a) MatVec: $A_f x$

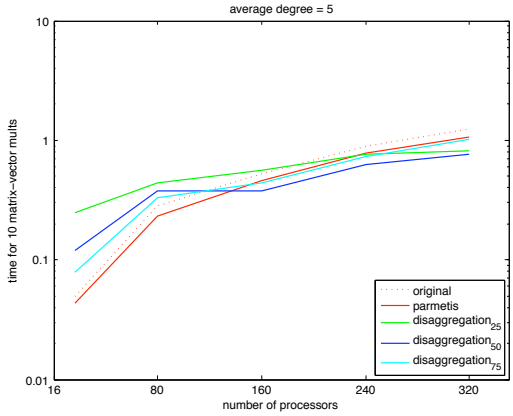


(b) MatVec: $Q^t A_f Q x$

Figure 4.6: Numerical results for matrices with average degree two. The matrices have 10,000 nodes per processor. Before disaggregating the matrices, we first repartitioned them with ParMETIS. The time needed for 10 matrix-vector products is given in seconds.



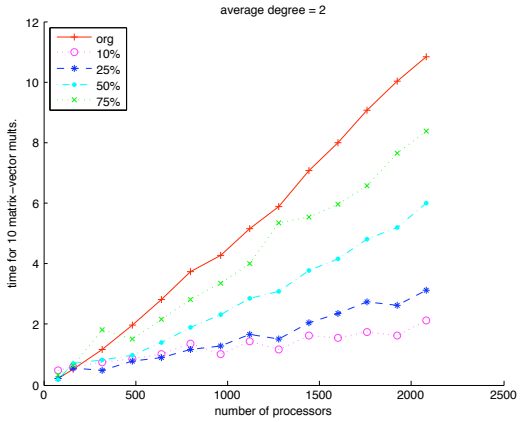
(a) MatVec: $A_f x$



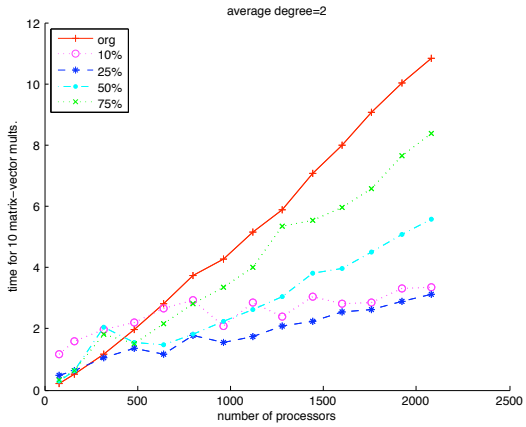
(b) MatVec: $Q^t A_f Q x$

Figure 4.7: Numerical results for matrices with average degree five. The matrices have 10,000 nodes per processor. Before disaggregating the matrices, we first repartitioned them with ParMETIS. The time needed for 10 matrix-vector products is given in seconds.

In Figure 4.6 and Figure 4.7 the time needed to perform 10 matrix-vector products with the matrices generated by [119] are given. We consider two different matrix-vector products, the factored triple matrix-vector product $Q^t(A_f(Qx))$ (right column) and $A_f x$ (left column). The matrices have 10,000 nodes per processor, and have average degree 2 (Figure 4.6) and 5 (Figure 4.7). During disaggregation, communication is restricted to 75%, 50%, and 25%. In both cases the matrix was re-partitioned with



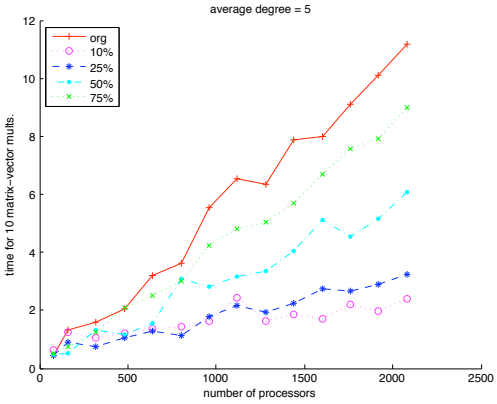
(a) MatVec: $A_f x$



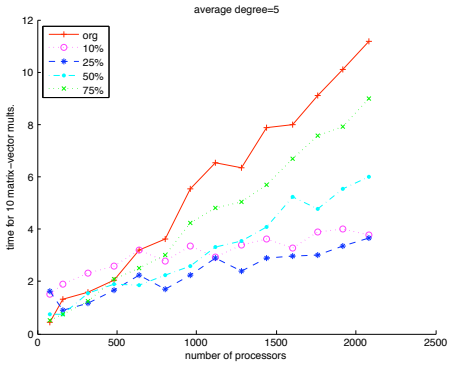
(b) MatVec: $Q^t A_f Q x$

Figure 4.8: Numerical results for matrices with average degree two. The matrices have 10,000 nodes per processor. No repartitioning is used. The time needed for 10 matrix-vector products is given in seconds.

ParMETIS before applying disaggregation. First, we can note that re-partitioning with ParMETIS (solid red line) only gives a small advantage over the matrix-vector product with the original matrix (dotted red line) if a small number of processors are used. For a very small number of processors disaggregation does not give an advantage. The time saved during communication does not offset the increased workload generated by working with a larger matrix. However, as the number of processors increases the reduced communication becomes more prevalent. As the number of processors increases it becomes evident that restricting communication brings a large



(a) MatVec: $A_f x$

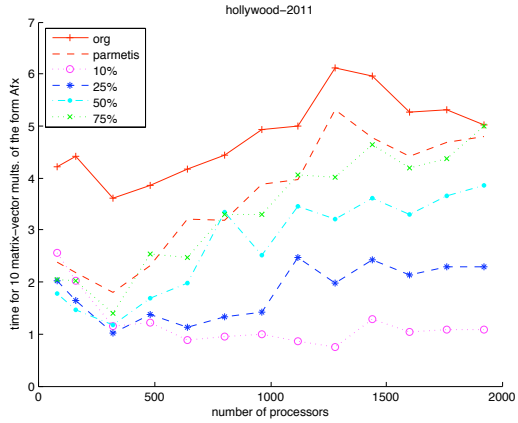


(b) MatVec: $Q^t A_f Q x$

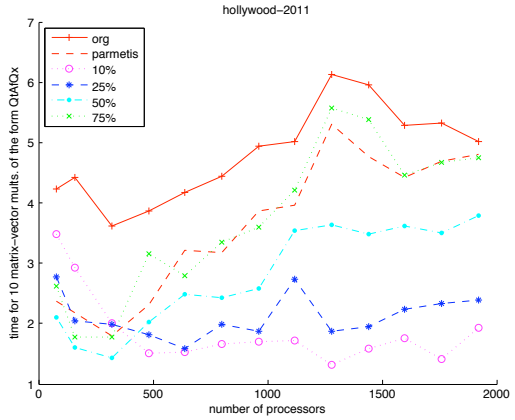
Figure 4.9: Numerical results for matrices with average degree five. The matrices have 10,000 nodes per processor. No repartitioning is used. The time needed for 10 matrix-vector products is given in seconds.

advantage even though the matrix size increases considerably. Note that we used a rather small number of processors. For a larger number of processors the memory requirement for ParMETIS became too large.

In our next experiment we omitted re-partitioning the matrix. The results are given in Figure 4.8 and Figure 4.9. The matrices used are of the same type as in the previous experiment. That is, the matrices are generated with [119] and have 10,000 nodes per processor and average degree two or five. Besides restricting the communication to 75%, 50%, and 25%, we included results were communication was restricted even further. These experiments, similar as the ones before, are weak



(a) MatVec: $A_f x$



(b) MatVec: $Q^t A_f Q x$

Figure 4.10: Numerical results for the hollywood-2011 matrix. Before disaggregating the matrix, we first repartitioned it with ParMETIS. The time needed for 10 matrix-vector products is given in seconds.

scaling experiments. That is, ideally the time should stay constant as the number of processors increases and the problem size per processor stays the same. For the matrix-vector product with the original matrix (red line) this is clearly not the case. Instead, the time needed for 10 matrix-vector products increases very rapidly as the number of processors increases. While the timings for the 75% and 50% case also increase at a fairly large rate, the 25% and 10% case bring a clear advantage. Note that for multiplication with the disaggregated matrix A_f restricting the communication

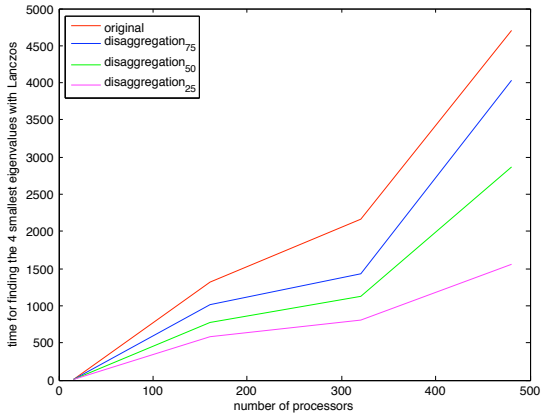


Figure 4.11: Time in seconds needed to find the 4 smallest eigenvalues with Lanczos algorithm. The matrix has 100 nodes per processor and average degree 2.

to 10% eventually outperforms disaggregation with 25% restriction. However, for the factored triple matrix-vector product $Q^t(A_f(Qx))$ the increased communication and computation requirements for the multiplication with Q and Q^t offset this advantage and 25% restriction usually outperform the 10% case. This observation suggests that restricting the communication even further does not give an additional advantage.

We also give the results of an experiment with a real world graph in Figure 4.10. Note that this is a strong scaling experiment and ideally the time needed should decrease as the number of processors increases. We partitioned the matrix with ParMETIS before the disaggregation mainly to balance the work load. The partitioning with ParMETIS is very cost intensive and not suitable in practice. We note that for this real-world problem the behavior is similar to the synthetic problem and restricting the communication brings a huge advantage over the performance of the matrix-vector product with the original matrix.

Lastly, we provide a small experiment in Figure 4.11 to demonstrate the effect of the improved matrix-vector product on the performance of an eigensolver. We used the Lanczos eigensolver from the SLEPc library [56] for our experiment. The matrix

was generated with [119] and has 100 nodes per processor and average degree 2. Matrix-vector multiplication was done via the factored triple matrix-vector product $Q^t(A_f(Qx))$. A considerable time reduction when using the factored triple matrix-vector product can be observed.

4.5 Conclusions

The matrix-vector product is the bottleneck for the parallel computation of eigenvalues and eigenvectors of large scale-free graphs. Currently, no parallel method is available to partition a scale-free graph in such a way that matrix-vector product can be completed in a sufficient way. The lack of good partitioners for scale-free graphs arises mainly from the irregular degree distribution and the existence of very large degree nodes. We provided a method to embed a scale-free graph into a more regular graph. The structure of the resulting graph is favorable for distributed environments. Even though the resulting graph is larger, we are able to improve the regular matrix-vector product with a factored triple matrix-vector product using the disaggregated matrix and a transfer operator. While we focused on disaggregating scale-free graphs, we expect that the method described could also be used for other graphs with irregular structure that cannot be successfully partitioned.

Chapter 5

Conclusions and suggestions for future work

In this thesis we have studied several problems arising in computational network analysis. We have given the first theoretical analysis of the restricted additive Schwarz method for singular M-matrices. Thus, we have extended the RAS method to the computation of the stationary vector of large, sparse Markov chains. Our results suggest that when combined with GMRES acceleration and inexact solves, RAS is a promising approach for the solution of Markov chains with large, sparse transition matrices. Although primarily designed with parallel computing in mind, for sufficiently large problems the proposed technique is found to be superior to standard approaches (like ILU preconditioning) even in a sequential implementation. In the parallel case RAS also shows promising results. The speed of convergence of GMRES with RAS preconditioner is influenced by the choice of domain decomposition. In some instances we have observed that the stationary RAS iteration (without Krylov acceleration) is quite competitive. Future work may include the investigation of a more suitable domain decomposition than a graph partitioner that minimizes the edge cut.

We demonstrated experimentally that the Chebyshev iteration is a competitive alternative in solving large linear systems that arise from a gene ranking problem. While the number of iterations is higher than for CG with Jacobi preconditioner, the previously best available method, the cost per iteration is much lower and leads to a faster solution in terms of cpu time. Also, the Chebyshev iteration is more desirable in a parallel setting as it avoids computing dot product. Future work may include further efforts to find a preconditioner for CG so that the number of iteration does not depend on the damping factor.

We provided a method to reduce the communication requirements during parallel matrix-vector multiplication in the case when the underlying graph of the matrix is scale-free. The main idea is to embed the scale-free graph into a more regular graph that has a more favorable structure for distributed environments. Even though the resulting graph is larger, we are able to improve the regular matrix-vector product with a factored triple matrix-vector product using the disaggregated matrix and a transfer operator. While we focused on disaggregating scale-free graphs, we expect that the method described could also be used for other graphs with irregular structure that cannot be successfully partitioned. For future work we are interested in using the disaggregated matrix with large internal weight s to compute its spectrum. For large s the spectrum of the disaggregated matrix approximates the spectrum of the original matrix. For this to be feasible, a scalable (such as AMG) preconditioner is needed. In addition, we are working to use the disaggregated matrix with $s = O(1)$, to construct an "auxiliary space" (AMG) preconditioner for the original matrix. It has the form $B^{-1} = M^{-1} + Q^t B^{-1}_{\text{disaggr.}} Q$ where M is a standard smoother for the original matrix and $B^{-1}_{\text{disaggr.}}$ is a preconditioner for the embedding matrix $A_f = B + sC$. This approach can be used within the effective 2D matrix storage [118] of the original matrix. In either case, these preconditioners can be used in the Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG) eigensolver [68].

Appendix A

Algorithms

Algorithm A.1 Arnoldi iteration (modified Gram-Schmidt variant)

$v_1 = r_0/\|r_0\|_2$
for $j = 1, 2, \ldots, m$ **do**
    Compute $w_j = Av_j$
    for $i = 1, 2, \ldots, j$ **do**
        Compute $h_{ij} = (w_j, v_i)$
        $w_j = w_j - h_{ij}v_i$
    end for
    $h_{j+1,j} = \|w_j\|_2$
    if $h_{j+1,j} = 0$ **then** Stop
    $v_{j+1} = w_j/h_{j+1,j}$
end for

Algorithm A.2 Lanczos iteration

$\beta_1 = 0$, $v_0 = 0$
$v_1 = r_0/\|r_0\|_2$
for $j = 1, 2, \ldots, m$ **do**
    Compute $w_j = Av_j - \beta_j v_{j-1}$
    Compute $\alpha_j = (w_j, v_j)$
    Set $w_j = w_j - \alpha_j v_j$
    $\beta_{j+1} = \|w_j\|_2$
    if $h_{j+1,j} = 0$ **then** Stop
    $v_{j+1} = w_j/\beta_{j+1,j}$
end for

Algorithm A.3 Gaussian Elimination

for $i = 2, \ldots, n$ **do**
    for $k = 1, \ldots, i-1$ **do**
        $a_{ik} := a_{ik}/a_{kk}$
        for $j = k+1, \ldots, n$ **do**
            $a_{ij} := a_{ij} - a_{ik} \cdot a_{kj}$
        end for
    end for
end for

Algorithm A.4 GMRES (basic form)

Compute $r_0 = b - Ax_0$, $\beta = \|r_0\|_2$, and $v_1 = r_0/\beta$
Define the $(m+1) \times m$ matrix $\bar{H}_m$ and set $h_{ij} = 0$
for $j = 1, 2, \ldots,$ until convergence **do**
    Compute $w_j = Av_j$
    for $i = 1, 2, \ldots, j$ **do**
        Compute $h_{ij} = (w_j, v_i)$
        $w_j = w_j - h_{ij} v_i$
    end for
    $h_{j+1,j} = \|w_j\|_2$
    if $h_{j+1,j} = 0$ **then**
        set $m = j$ and **go to** 15
    end if
    $v_{j+1} = w_j / h_{j+1,j}$
    Compute u_j as the minimizer of $\|\beta e_1 - \bar{H}_j u_j\|_2$
    $x = x_0 + V_j u_j$
end for

Algorithm A.5 GMRES(m)

Compute $r_0 = b - Ax_0$, $\beta = \|r_0\|_2$, and $v_1 = r_0/\beta$
Define the $(m+1) \times m$ matrix $\bar{H}_m$ and set $h_{ij} = 0$
for $j = 1, 2, \ldots, m$ **do**
    Compute $w_j = Av_j$
    for $i = 1, 2, \ldots, j$ **do**
        Compute $h_{ij} = (w_j, v_i)$
        $w_j = w_j - h_{ij} v_i$
    end for
    $h_{j+1,j} = \|w_j\|_2$
    if $h_{j+1,j} = 0$ **then**
        set $m = j$ and **go to** 15
    end if
    $v_{j+1} = w_j / h_{j+1,j}$
end for
Compute u_m as the minimizer of $\|\beta e_1 - \bar{H}_m u_m\|_2$
$x = x_0 + V_m u_m$

Algorithm A.6 CG

Compute $r_0 = b - Ax_0$, and $p_1 = r_0$
for $j = 1, 2, \ldots,$ until convergence **do**
    $\alpha_{j-1} = (r_{j-1}, r_{j-1}) / (Ap_{j-1}, p_{j-1})$
    $x_j = x_{j-1} + \alpha_{j-1} p_{j-1}$
    $r_j = r_{j-1} + \alpha_{j-1} Ap_{j-1}$
    $\beta_{j-1} = (r_j, r_j) / (r_{j-1}, r_{j-1})$
    $p_j = r_j + \beta_{j-1} p_{j-1}$
end for

Appendix B

Additional numerical experiments

Here we present some results for the additive Schwarz method applied to the GeneRank problem. The experiments were run on a 2.13 GHz Intel Core 2 Duo processor with 2 GB memory.

renga matrix, n=500000

α	0.5	0.7	0.99
PCG-AS(2) 1.0e-04	6 (15.18, 7.53, 22.71)	6 (16.24, 6.81, 23.05)	6 (16.93, 7.35, 24.28)
AS(2) 1.0e-04	6 (15.53, 6.68, 22.21)	6 (16.53, 5.60, 22.13)	7 (17.04, 7.16, 24.20)
RAS(2) 1.0e-04	6 (15.95, 6.70, 22.65)	6 (16.46, 6.27, 22.73)	8 (17.30, 8.90, 26.20)
PCG-AS(16) 1.0e-04	9 (36.75, 22.91, 59.66)	10 (36.08, 26.42, 62.50)	12 (38.89, 34.32, 73.21)
AS(16) 1.0e-04	10 (37.20, 24.03, 61.23)	12 (38.05, 28.44, 66.49)	19 (37.42, 50.88, 88.30)
RAS(16) 1.0e-04	6 (34.80, 12.72, 47.52)	6 (36.81, 12.54, 49.35)	8 (38.64, 18.11, 56.75)
PCG-AS(32) 1.0e-04	9 (34.28, 22.04, 56.32)	10 (36.51, 25.00, 61.51)	12 (38.27, 30.17, 68.44)
AS(32) 1.0e-04	10 (36.67, 22.05, 58.72)	12 (37.99, 27.20, 65.19)	19 (38.60, 46.27, 84.87)
RAS(32) 1.0e-04	6 (35.19, 12.03, 47.22)	6 (38.13, 13.56, 51.69)	8 (39.08, 17.48, 56.56)
PCG-AS(64) 1.0e-04	11 (33.93, 22.57, 56.50)	11 (35.83, 23.70, 59.53)	14 (36.19, 29.66, 65.85)
AS(64) 1.0e-04	10 (36.04, 18.85, 54.89)	12 (37.33, 23.90, 61.23)	20 (38.01, 42.17, 80.18)
RAS(64) 1.0e-04	6 (36.16, 10.65, 46.81)	6 (38.07, 10.52, 48.59)	8 (38.91, 14.54, 53.45)

renga matrix, n=100000

α	0.5	0.7	0.85	0.99
PCG-AS(2) 1.0e-04	6 (2.75, 1.42, 4.17)	6 (2.77, 1.72, 4.49)	6 (2.87, 1.54, 4.41)	6 (2.98, 1.46, 4.44)
AS(2) 1.0e-04	6 (2.98, 1.21, 4.19)	6 (2.94, 1.24, 4.18)	6 (3.15, 1.31, 4.46)	8 (3.19, 1.71, 4.90)
RAS(2) 1.0e-04	6 (2.87, 1.14, 4.01)	6 (3.05, 1.13, 4.18)	6 (3.02, 1.24, 4.26)	7 (3.11, 1.40, 4.51)
PCG-AS(16) 1.0e-04	10 (6.73, 4.48, 11.21)	10 (7.29, 4.70, 11.99)	11 (7.26, 4.81, 12.07)	12 (7.30, 5.57, 12.87)
AS(16) 1.0e-04	10 (7.18, 4.10, 11.28)	12 (7.32, 5.23, 12.55)	14 (7.69, 6.07, 13.76)	20 (7.87, 8.87, 16.74)
RAS(16) 1.0e-04	6 (7.16, 2.06, 9.22)	6 (7.42, 2.09, 9.51)	6 (7.59, 2.44, 10.03)	7 (7.84, 2.92, 10.76)
PCG-AS(32) 1.0e-04	10 (7.12, 4.20, 11.32)	10 (6.98, 4.63, 11.61)	11 (7.30, 4.67, 11.97)	12 (7.17, 5.56, 12.73)
AS(32) 1.0e-04	10 (6.74, 3.71, 10.45)	12 (7.45, 4.90, 12.35)	14 (7.42, 5.34, 12.76)	20 (7.48, 8.68, 16.16)
RAS(32) 1.0e-04	6 (6.82, 2.00, 8.82)	6 (7.22, 2.14, 9.36)	6 (7.18, 2.09, 9.27)	7 (7.41, 2.47, 9.88)
PCG-AS(64) 1.0e-04	11 (6.51, 3.85, 10.36)	12 (6.75, 4.76, 11.51)	12 (6.83, 4.53, 11.36)	14 (6.89, 5.39, 12.28)
AS(64) 1.0e-04	10 (6.33, 3.18, 9.51)	12 (6.94, 4.11, 11.05)	14 (7.12, 4.96, 12.08)	19 (7.31, 7.00, 14.31)
RAS(64) 1.0e-04	6 (6.62, 1.80, 8.42)	6 (7.12, 1.93, 9.05)	6 (7.25, 2.04, 9.29)	7 (7.28, 2.40, 9.68)

SNPa matrix, $n = 152520$

α	0.5	0.7	0.99
PCG-AS(2) 1.0e-02	17 (0.88, 1.86, 2.74)	23 (1.24, 2.22, 3.46)	112 (2.10, 11.58, 13.68)
AS(2) 1.0e-02	31 (0.97, 2.55, 3.52)	57 (1.04, 5.06, 6.10)	1589 (2.00, 159.15, 161.15)
RAS(2) 1.0e-02	31 (0.92, 2.84, 3.76)	57 (1.11, 5.20, 6.31)	1955 (1.98, 200.65, 202.63)
PCG-AS(16) 1.0e-02	22 (0.70, 2.43, 3.13)	27 (0.71, 2.99, 3.70)	150 (0.69, 16.27, 16.96)
AS(16) 1.0e-02	33 (0.75, 3.90, 4.65)	60 (0.77, 7.28, 8.05)	1697 (0.77, 205.88, 206.65)
RAS(16) 1.0e-02	33 (0.66, 3.38, 4.04)	61 (0.69, 6.18, 6.87)	2000 (0.77, 210.13, 210.90)
PCG-AS(32) 1.0e-02	22 (0.44, 2.18, 2.62)	27 (0.48, 2.68, 3.16)	148 (0.51, 14.26, 14.77)
AS(32) 1.0e-02	34 (0.53, 3.64, 4.17)	62 (0.52, 6.69, 7.21)	1730 (0.56, 189.16, 189.72)
RAS(32) 1.0e-02	34 (0.53, 2.88, 3.41)	61 (0.48, 5.55, 6.03)	2000 (0.55, 191.96, 192.51)
PCG-AS(64) 1.0e-02	22 (0.59, 2.07, 2.66)	27 (0.42, 2.38, 2.80)	148 (0.48, 13.71, 14.19)
AS(64) 1.0e-02	35 (0.38, 3.34, 3.72)	64 (0.58, 6.38, 6.96)	1820 (0.50, 172.98, 173.48)
RAS(64) 1.0e-02	34 (0.46, 2.38, 2.84)	61 (0.43, 5.05, 5.48)	2000 (0.48, 172.71, 173.19)

Bibliography

[1] A. Abou-Rjeili and G. Karypis. Multilevel algorithms for partitioning power-law graphs. In *Proceedings of the 20th international conference on Parallel and distributed processing*, IPDPS'06, Washington, DC, USA, 2006. IEEE Computer Society.

[2] M. Adlers and Å. Björck. Matrix stretching for sparse least squares problems. *Numerical Linear Algebra with Applications*, 7(2):51–65, 2000.

[3] G. Alefeld and N. Schneider. On square roots of M-matrices. *Linear Algebra and its Applications*, 42:119–132, 1982.

[4] F. L. Alvarado. Matrix enlarging methods and their application. *BIT*, 37(3):473–505, 1997.

[5] W. E. Arnoldi. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of Applied Mathematics*, 9(17):17–29, 1951.

[6] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In L. Ungar, M. Craven, D. Gunopulos, and T. Eliassi-Rad, editors, *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, pages 44–54, New York, 2006. ACM.

[7] S. Balay, K. Buschelman, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, L. C. Mcinnes, B. Smith, and H. Zhang. PETSc Users Manual. Technical Report ANL-95/11, Revision 2.1.1, Argonne National Laboratory, 2001.

[8] A. L. Barabási. *Linked - How Everything is Connected to Everything Else and What It Means for Business, Science, and Everyday Life.* Plume, New York, 2003.

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

[10] D. E. Bassett, M. B. Eisen, and M. S. Boguski. Gene Expression Informatics–it's all in your mine. *Nature Genetics Supplement*, 21:51–55, January 1999.

[11] M. Benzi. Preconditioning techniques for large linear systems: A survey. *Journal of Computational Physics*, 182(2):418 – 477, 2002.

[12] M. Benzi, A. Frommer, R. Nabben, and D. B. Szyld. Algebraic theory of multiplicative Schwarz methods. *Numerische Mathematik*, 89:605–639, 2001.

[13] M. Benzi and V. Kuhlemann. Restricted additive Schwarz methods for Markov chains. *Numerical Linear Algebra with Applications*, 18(6):1011–1029, November 2011.

[14] M. Benzi and D. B. Szyld. Existence and uniqueness of splittings for stationary iterative methods with applications to alternating methods. *Numerische Mathematik*, 76:309–321, 1997.

[15] M. Benzi and M. Tůma. A parallel solver for large-scale Markov chains. *Applied Numerical Mathematics*, 41:135–153, 2002.

[16] A. Berman and R. J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 1994.

[17] N. Biggs. *Algebraic Graph Theory*. Cambridge Mathematical Library. Cambridge University Press, 1994.

[18] P. Boldi and S. Vigna. The webgraph framework I: compression techniques. In *Proceedings of the 13th international conference on World Wide Web*, WWW '04, pages 595–602, New York, 2004. ACM.

[19] B. Bollobás. *Graph Theory: An Introductory Course*. Springer-Verlag, New York, 1979.

[20] B. Bollobás. *Modern Graph Theory*. Graduate Texts in Mathematics. Springer-Verlag, New York, 1998.

[21] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web: experiments and models. In *Proceedings of the Ninth International World-Wide Web Conference (WWW9, Amsterdam, May 15 - 19, 2000 - Best Paper)*, Reston, VA, 2000. Foretec Seminars, Inc. (CD-ROM).

[22] P. O. Brown and D. Botstein. Exploring the new world of the genome with DNA microarrays. *Nature Genetics*, 21:33–37, 1999.

[23] R. Bru, F. Pedroche, and D. B. Szyld. Additive Schwarz iterations for Markov chains. *SIAM Journal on Matrix Analysis and Applications*, 27(2):445–458, 2005.

[24] H. Brunst, H.-C. Hoppe, W. E. Nagel, and M. Winkler. Performance optimization for large scale computing: The scalable VAMPIR approach. In *Proceedings*

of the International Conference on Computational Science-Part II, ICCS '01, pages 751–760, London, 2001. Springer-Verlag.

[25] J. J. Buoni. Incomplete factorization of singular M-matrices. *SIAM Journal on Algebraic and Discrete Methods*, 7(2):193–198, April 1986.

[26] R. G. Busacker and T. L. Saaty. *Finite Graphs and Networks: An Introduction with Applications*. International Series in Pure and Applied Mathematics. McGraw-Hill, New York, 1965.

[27] X.-C. Cai and Y. Saad. Overlapping domain decomposition algorithms for general sparse matrices. *Numerical Linear Algebra with Applications*, 3:221–237, 1996.

[28] X.-C. Cai and M. Sarkis. A restricted additive Schwarz preconditioner for general sparse linear systems. *SIAM Journal on Scientific Computing*, 21:792–797, 1999.

[29] T. F. Chan and T. P. Mathew. Domain decomposition algorithms. In A. Iserles, editor, *Acta Numerica 1994*, pages 61–143. Cambridge University Press, 1994.

[30] C. Chevalier and F. Pellegrini. PT-Scotch: A tool for efficient parallel graph ordering. *Parallel Computing*, 34(6-8):318–331, July 2008.

[31] A. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. *Physical Review E*, 70(6):066111, December 2004.

[32] R. Cooley, B. Mobasher, and J. Srivastava. Web Mining: Information and Pattern Discovery on the World Wide Web. *IEEE International Conference on Tools with Artificial Intelligence*, pages 558–567, 1997.

[33] P. Crucitti, V. Latora, M. Marchiori, and A. Rapisarda. Efficiency of scale-free networks: Error and attack tolerance. *Physica A*, 320(cond-mat/0205601):642. 23 p, May 2002.

[34] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th national conference*, ACM '69, pages 157–172, New York, 1969. ACM.

[35] H. De Sterck, T. A. Manteuffel, S. F. McCormick, K. Miller, J. Pearson, J. Ruge, and G. Sanders. Smoothed aggregation multigrid for Markov chains. *SIAM Journal on Scientific Computing*, 32(1):40–61, February 2010.

[36] K. D. Devine, E. G. Boman, L. A. Riesen, Ü. V. Çatalyürek, and C. Chevalier. Getting started with Zoltan: A short tutorial. In *Proc. of 2009 Dagstuhl Seminar on Combinatorial Scientific Computing*, 2009. Also available as Sandia National Labs Tech Report SAND2009-0578C.

[37] R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, Heidelberg, third edition, 2005.

[38] M. Dryja. An additive Schwarz algorithm for two and three dimensional finite element problems. In G. A. Meurant J. Périaux O. B. Widlund T. F. Chan, R. Glowinski, editor, *Second International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 168–172, Philadelphia, 1989. SIAM.

[39] M. Dryja and O. B. Widlund. An additive variant of the Schwarz alternating method for the case of many subregions. Technical Report 339, also Ultracomputer Note 131, Department of Computer Science, Courant Institute, New York University, 1987.

[40] J. Duch and A. Arenas. Community detection in complex networks using extremal optimization. *Physical Review E*, 72(2):027104, August 2005.

[41] A. E. Dunlop and B. W. Kernighan. A procedure for placement of standard-cell VLSI circuits. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 4(1):92–98, 1985.

[42] E. Estrada. *The Structure of Complex Networks: Theory and Applications*. Oxford University Press, Oxford, 2011.

[43] C. Farhat and F.-X. Roux. A method of finite element tearing and interconnecting and its parallel solution algorithm. *International Journal for Numerical Methods in Engineering*, 32(6):1205–1227, 1991.

[44] P. Fernandes, B. Plateau, and W. J. Stewart. Efficient descriptor-vector multiplications in stochastic automata networks. *Journal of the ACM*, 45:381–414, 1998.

[45] A. Frommer and D. B. Szyld. Weighted max norms, splittings, and overlapping additive Schwarz iterations. *Numerische Mathematik*, 83:259–278, 1999.

[46] A. Frommer and D. B. Szyld. An algebraic convergence theory for restricted additive Schwarz methods using weighted max norms. *SIAM Journal on Numerical Analysis*, 39:463–479, 2001.

[47] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237 – 267, 1976.

[48] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, 3rd edition, 1996.

[49] J. F. Grcar. Matrix stretching for linear equations. Technical Report SAND90-8723, Sandia National Laboratories, November 1990.

[50] M. Griebel and P. Oswald. On the abstract theory of additive and multiplicative Schwarz algorithms. *Numerische Mathematik*, 70(2):163–180, April 1995.

[51] P. Grindrod. Range-dependent random graphs and their application to modeling large small-world proteome datasets. *Physical Review E: Statistical, Nonlinear, and Soft Matter Physics*, 66(6 Pt 2):066702, 2002.

[52] P. Grindrod. Modeling proteome networks with range-dependent graphs. *American Journal of PharmacoGenomics*, 3(1):1–4, 2003.

[53] S. Guattery and G. L. Miller. Graph embeddings and Laplacian eigenvalues. *SIAM Journal on Matrix Analysis and Applications*, 21(3):703–723, Feb.-March 2000.

[54] B. Hendrickson and R. Leland. The Chaco User's Guide: Version 2.0. Technical Report SAND94–2692, Sandia National Lab, 1994.

[55] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. In *Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '95, page 28, New York, NY, USA, 1995. ACM.

[56] V. Hernández, J. E. Román, and V. Vidal. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Transactions on Mathematical Software*, 31(3):351–362, 2005.

[57] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49:409–436, 1952.

[58] Y. Ji, X. Xu, and G. D. Stormo. A graph theoretical approach for predicting common RNA secondary structure motifs including pseudoknots in unaligned sequences. *Bioinformatics*, 20(10):1591–1602, July 2004.

[59] G. Karypis and V. Kumar. Parallel multilevel k-way partitioning scheme for irregular graphs. In *Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '96, Washington, DC, USA, 1996. IEEE Computer Society.

[60] G. Karypis and V. Kumar. A coarse-grain parallel formulation of multilevel k-way graph partitioning algorithm. In *8th SIAM Conference on Parallel Processing for Scientific Computing*, Philadelphia, 1997. SIAM.

[61] G. Karypis and V. Kumar. Metis 3.0: Unstructured graph partitioning and sparse matrix ordering system. Technical Report 97-061, Dept. Computer Science, University of Minnesota, 1997.

[62] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, December 1998.

[63] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48:96–129, 1998.

[64] G. Karypis and V. Kumar. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *Journal of Parallel and Distributed Computing*, 48(1):71–95, 1998.

[65] G. Karypis, K. Schloegel, and V. Kumar. ParMETIS: Parallel graph partitioning and sparse matrix ordering library. Technical Report 97-060, Dept. Computer Science, University of Minnesota, 2003.

[66] J. P. Kavanagh and M. Neumann. Consistency and convergence of the parallel multisplitting method for singular M-matrices. *SIAM Journal on Matrix Analysis and Applications*, 10:210–218, 1989.

[67] H. B. Keller. On the solution of singular and semidefinite linear systems by iteration. *Journal of the Society for Industrial and Applied Mathematics: Series B, Numerical Analysis*, 2(2):281–290, 1965.

[68] A. V. Knyazev. Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. *SIAM Journal on Scientific Computing*, 23(2):517–541, 2001.

[69] R. Kosala and H. Blockeel. Web mining research: A survey. *SIGKDD Explorations*, 2(1):1–15, June 2000.

[70] V. Krebs. Uncloaking Terrorist Networks. *First Monday*, 7(4), April 2002.

[71] A. N. Langville and C. D. Meyer. *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton University Press, Princeton, NJ, USA, 2006.

[72] P. J. Lanzkron, D. J. Rose, and D. B. Szyld. Convergence of nested classical iterative methods for linear systems. *Numerische Mathematik*, 58:685–702, 1991.

[73] Y.-J. Lee, J. Wu, J. Xu, and L. Zikatanov. On the convergence of iterative methods for semidefinite linear systems. *SIAM Journal on Matrix Analysis and Applications*, 28(3):634–641, August 2006.

[74] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15(4):1036–1055, November 1986.

[75] I. Marek and D. B. Szyld. Comparison theorems for weak splittings of bounded operators. *Numerische Mathematik*, 58:387–397, 1990.

[76] I. Marek and D. B. Szyld. Algebraic Schwarz methods for the numerical solution of Markov chains. *Linear Algebra and its Applications*, 386:67–81, 2004.

[77] J. A. Meijerink and H. A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Mathematics of Computation*, 31:148–162, 1977.

[78] C. D. Meyer. *Matrix Analysis and Applied Linear Algebra*. Society for Industrial and Applied Mathematics, Philadelphia, 2000.

[79] B. Mohar. The Laplacian spectrum of graphs. In Y. Alavi, G. Chartrand, O.R. Oellermann, and A.J. Schwenk, editors, *Graph Theory, Combinatorics, and Applications*, pages 871–898. Wiley, 1991.

[80] B. Mohar and M. Juvan. Some applications of Laplace eigenvalues of graphs. In G. Hahn and G. Sabidussi, editors, *Graph Symmetry: Algebraic Methods and Applications, NATO ASI Series C*, volume 497, pages 227–275, 1997.

[81] J. L. Morrison, R. Breitling, D. J. Higham, and D. R. Gilbert. GeneRank: Using search engine technology for the analysis of microarray experiments. *BMC Bioinformatics*, 6(233), 2005.

[82] E. Nabieva, K. Jim, A. Agarwal, B. Chazelle, and M. Singh. Whole-proteome prediction of protein function via graph-theoretic analysis of interaction maps. *Bioinformatics*, 21(1):302–310, January 2005.

[83] M. Neumann and R. J. Plemmons. Convergent nonnegative matrices and iterative methods for consistent linear systems. *Numerische Mathematik*, 31:265–279, 1978.

[84] M. E. J. Newman. Spread of epidemic disease on networks. *Physical Review E*, 66(016128), July 2002.

[85] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, 2003.

[86] M. E. J. Newman. Detecting community structure in networks. *The European Physical Journal B - Condensed Matter and Complex Systems*, 38(2):321–330, March 2004.

[87] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, 69(6):066133, June 2004.

[88] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(2):026113, February 2004.

[89] J. M. Ortega. *Numerical Analysis: A Second Course*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, Philadelphia, 1990.

[90] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999.

[91] F. Pellegrini and J. Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In H. M. Liddell, A. C., L. O. Hertzberger, and P. M. A. Sloot, editors, *HPCN Europe*, volume 1067 of *Lecture Notes in Computer Science*, pages 493–498. Springer, 1996.

[92] B. Philippe, Y. Saad, and W. J. Stewart. Numerical methods in Markov chain modelling. *Operations Research*, 40:1156–1179, 1996.

[93] G. Poole and T. Boullion. A survey on M-matrices. *SIAM Review*, 16(4):419–427, 1974.

[94] R. Preis and R. Diekmann. PARTY - A software library for graph partitioning. In B. H. V. Topping, editor, *Advances in Computational Mechanics with Parallel and Distributed Processing*, pages 63–71, Edinburgh, 1997. Civil-Comp Press.

[95] A. Quarteroni and A. Valli. *Domain Decomposition Methods for Partial Differential Equations*. Numerical Mathematics and Scientific Computation. Oxford University Press, New York, 1999.

[96] R. Albert V. Latora R. Kinney, P. Crucitti. Modeling cascading failures in the north american power grid. *The European Physical Journal B*, 46:101–107, 2005.

[97] Y. Saad. ILUT: A dual threshold incomplete LU factorization. *Numerical Linear Algebra with Applications*, 4:387–402, 1994.

[98] Y. Saad. *Iterative Methods for Sparse Linear Systems, Second Edition*. Society for Industrial and Applied Mathematics, Philadelphia, 2nd edition, April 2003.

[99] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, July 1986.

[100] A. Schenker. *Graph-theoretic techniques for web content mining*. PhD thesis, Department of Computer Science and Engineering, College of Engineering, University of South Florida, Tampa, FL, USA, 2003.

[101] H. Schneider. Theorems on M-splittings of a singular M-matrix which depend on graph structure. *Linear Algebra and its Applications*, 58:407 – 424, 1984.

[102] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:888–905, 1997.

[103] B. Smith, P. Bjørstad, and W. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, New York, 2004.

[104] Y. Z. Song. Comparisons of nonnegative splittings of matrices. *Linear Algebra and its Applications*, 154/156:433–455, 1991.

[105] W. J. Stewart. MARCA Models: A collection of Markov chain models. `http://www4.ncsu.edu/~billy/MARCA_Models/MARCA_Models.html`.

[106] W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, Princeton, NJ, 1994.

[107] A. Taylor and D. J. Higham. CONTEST: A Controllable Test Matrix Toolbox for MATLAB. *ACM Transactions on Mathematical Software*, 35(4), 2009.

[108] A. Toselli and O. Widlund. *Domain Decomposition Methods - Algorithms and Theory*. Springer Series in Computational Mathematics. Springer, 2005.

[109] R. S. Varga. Factorization and normalized iterative methods. In R. E. Langer, editor, *Boundary Problems in Differential Equations*, pages 121–142, Madison, 1960. University of Wisconsin Press.

[110] R. S. Varga. *Matrix Iterative Analysis*. Prentice Hall, Englewood Cliffs, NJ, 1962.

[111] E. Vecharynski, Y. Saad, and M. Sosonkina. Graph partitioning with matrix coefficients for symmetric positive definite linear systems. Technical Report umsi-2011-143, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, 2011.

[112] E. Virnik. An algebraic multigrid preconditioner for a class of singular M-matrices. *SIAM Journal on Scientific Computing*, 29(5):1982–1991, September 2007.

[113] C. Walshaw and M. Cross. JOSTLE: Parallel multilevel graph partitioning software - an overview. In F. Magoules, editor, *Mesh Partitioning Techniques and Domain Decomposition Techniques*, pages 27–58, Stirling, 2007. Saxe-Coburg Publications.

[114] D. J. Watts. *Six Degrees: The Science of a Connected Age*. W. W. Norton, New York, 2004.

[115] D. J. Watts and S. H. Strogatz. Collective dynamics of "small-world" networks. *Nature*, 393(6684):440–442, June 1998.

[116] Z. Woźnicki. Nonnegative splitting theory. *Japan Journal of Industrial and Applied Mathematics*, 11:289–342, 1994.

[117] G. Wu, W. Xu, Y. Zhang, and Y. Wei. A preconditioned conjugate gradient algorithm for GeneRank with application to microarray data mining. *Data Mining and Knowledge Discovery*, pages 1–30, November 2011.

[118] A. Yoo, A. H. Baker, R. Pearce, and V. E. Henson. A scalable eigensolver for large scale-free graphs using 2d graph partitioning. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 63:1–63:11, New York, NY, USA, 2011. ACM.

[119] A. Yoo and K. Henderson. Parallel massive scale-free graph generators. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, SC '06, New York, NY, USA, 2006. ACM.

[120] M. Youssef, C. Scoglio, and S. Pahwa. Robustness measure for power grids with respect to cascading failures. In Y. Qian K. Tutschku P. Van Mieghem, U. R. Kriegerl, editor, *Proceedings of the 2011 International Workshop on Modeling, Analysis, and Control of Complex Networks*, Cnet '11, pages 45–49. ITCP, 2011.