STEADY-STATE ANALYSIS OF GOOGLE-LIKE STOCHASTIC MATRICES

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Gökçe Nil Noyan

September, 2007

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

Assoc. Prof. Dr. Tuğrul Dayar (Advisor)

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

Assoc. Prof. Dr. Uğur Güdükbay

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

Assoc. Prof. Dr. Ezhan Karaşan

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute

ABSTRACT

STEADY-STATE ANALYSIS OF GOOGLE-LIKE STOCHASTIC MATRICES

Gökçe Nil Noyan
M.S. in Computer Engineering
Supervisor: Assoc. Prof. Dr. Tuğrul Dayar
September, 2007

Many search engines use a two-step process to retrieve from the web pages related to a user's query. In the first step, traditional text processing is performed to find all pages matching the given query terms. Due to the massive size of the web, this step can result in thousands of retrieved pages. In the second step, many search engines sort the list of retrieved pages according to some ranking criterion to make it manageable for the user. One popular way to create this ranking is to exploit additional information inherent in the web due to its hyperlink structure. One successful and well publicized link-based ranking system is PageRank, the ranking system used by the Google search engine. The dynamically changing matrices reflecting the hyperlink structure of the web and used by Google in ranking pages are not only very large, but they are also sparse, reducible, stochastic matrices with some zero rows. Ranking pages amounts to solving for the steady-state vectors of linear combinations of these matrices with appropriately chosen rank-1 matrices. The most suitable method of choice for this task appears to be the power method. Certain improvements have been obtained using techniques such as quadratic extrapolation and iterative aggregation. In this thesis, we propose iterative methods based on various block partitionings, including those with triangular diagonal blocks obtained using cutsets, for the computation of the steady-state vector of such stochastic matrices. The proposed iterative methods together with power and quadratically extrapolated power methods are coded into a software tool. Experimental results on benchmark matrices show that it is possible to recommend Gauss-Seidel for easier web problems and block Gauss-Seidel with partitionings based on a block upper triangular form in the remaining problems, although it takes about twice as much memory as quadratically extrapolated power method.

Keywords: Google, PageRank, stochastic matrices, power method, quadratic extrapolation, block iterative methods, aggregation, partitionings, cutsets, triangular blocks.

ÖZET

GOOGLE-BENZERİ RASSAL MATRİSLERİN UZUN VADELİ ÇÖZÜMLEMESİ

Gökçe Nil Noyan
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Doç. Dr. Tuğrul Dayar
Eylül, 2007

Birçok arama motoru, ağ üzerinden kullanıcının sorgusuyla ilgili sayfaları bulabilmek için iki aşamalı bir süreç kullanır. Birinci aşamada, verilen sorgulama terimlerini içeren tüm sayfaları bulabilmek için alışılagelen metin işleme yapılır. Ağın devasa büyüklüğü nedeniyle bu aşama binlerce sayfanın elde edilmesiyle sonuçlanabilir. İkinci aşamada, birçok arama motoru, elde edilen sayfa listesinin kullanıcı açısından idare edilebilir olması için onu bir sınıflandırma kriterine göre sıralar. Bu sıralamayı oluşturmanın yollarından popüler olan biri, hiperbağlantı yapısından kaynaklanan, ağdaki ek bilgilerden yararlanmaktır. Google arama motoru tarafından kullanılan SayfaDeğeri, bağa dayalı sınıflandırma sistemlerinden başarılı olmuş ve çok tanınan biridir. Ağın hiperbağlantı yapısını yansıtan sürekli değişmekte olan ve Google tarafından sayfaları sıralamak için kullanılan matrisler sadece çok büyük değil, fakat aynı zamanda bazı sıraları sıfır olan, seyrek, indiregenebilir rassal matrislerdir. Doküman sıralamalarını bulabilmek, bu tip rassal matrislerin, özel rank-1 matrisleriyle konveks kombinasyonlarının uzun vadeli vektörlerini elde etmekten geçer. Bu iş için seçilecek en uygun yöntem kuvvet yöntemi olarak gözükmektedir. Bu konuda, ikinci dereceden ekstrapolasyon ve dolaylı birleştirme gibi tekniklerle belirli ilerlemeler sağlanmıştır. Bu tezde, böyle rassal matrislerin uzun vadeli vektörlerini hesap etmek için, kesenkümeler kullanılarak elde edilmiş üçgen köşegen blokları olanlar da dahil olmak üzere, blok bölünmelere dayalı dolaylı yöntemler önerilmektedir. Önerilen dolaylı yöntemler, kuvvet ve ikinci dereceden ektrapolasyonlu kuvvet yöntemleri ile birlikte bir yazılım paketine kodlanmıştır. Denektaş matrisler üzerinde deneysel sonuçlar daha kolay olan ağ problemleri için Gauss-Seidel diğer problemler için ise, ikinci dereceden ekstrapolasyonlu güç yönteminin neredeyse iki katı bellek kullanmasına rağmen, blok üst üçgen biçime dayalı bölünmeler ile blok Gauss-Seidel önerilebileceğini göstermektedir.

Anahtar sözcükler: Google, SayfaDeğeri, rassal matrisler, güç yöntemi, ikinci dereceden ekstrapolasyon, blok dolaylı yöntemler, birleştirme, bölünmeler, kesenkümeler, üçgen bloklar.

Acknowledgement

I would like thank my supervisor Assoc. Prof. Dr. Tuğrul Dayar for his guidance, support, and patience throughout this study. I would like to thank my thesis committe members Assoc. Prof. Dr. Uğur Güdükbay and Assoc. Prof. Dr. Ezhan Karaşan for their constructive remarks and suggestions. I would also like to thank Bilkent University and the Computer Engineering Department for exemption of tuition and fees of three years and the scholarship of one year and also TÜBİTAK for their scholarship of one year. Finally, I would like to thank my family and friends, especially, Tayfun Küçükyılmaz, Sengör Altıngövde, Engin Demir, Ata Türk, and Ali Cevahir for their help, encouragement and moral support throughout this study.

Contents

List of Tables

Chapter 1

Introduction

The World Wide Web (WWW) is a dynamic global library which includes vast amount of information. It is a collection of completely uncontrolled, heterogeneous documents. There are billions of web pages in this library with a doubling life less than a year. Web information retrieval is a challenging task due to the size of its environment and diversity of the web pages. Search queries may result with thousands of web pages, most of which may be irrelevant to the query or less important. Since users cannot extract relevant information within thousands of pages, state-of-the-art search engines, such as Google, use ranking techniques to list query results in the order of relevance to the query.

Google's search engine uses a link-based, query-independent ranking system called PageRank. PageRank is first introduced by Google's founders Page and Brin at Stanford University [7]. PageRank computation is one of the most effective and widely used approach for ranking pages. PageRank uses the *hyperlink* structure existing within the web, which is referred to as the *web graph*.

PageRank uses the web graph to build a Markov Chain (MC) [37, p.4] and iteratively computes its steady-state vector. For steady-state analysis, one needs

to solve the systems of linear equations

$$\pi P = \pi, \ \sum_{j \in \mathcal{S}} \pi_j = 1, \tag{1.1}$$

where P is the transition probability matrix of the irreducible MC describing transition probabilities among states (that is, pages of the web in this case), $\mathcal{S}$ is the set of states of P, and π is its steady-state distribution (row) vector. When the transition probability matrix P is irreducible, then π in (1.1) exists, is unique, and is positive [37, p.15].

The PageRank algorithm iteratively computes the steady-state vector using the power method. Unfortunately there are a few problems with using the hyperlink structure of the web. There are web pages called *dangling nodes*, which have no hyperlinks. Dangling nodes exists in many forms: a data sheet, a postscript graph, a page with a JPEG picture, a PDF document. All rows of the MC associated with the web graph do not sum up to one due to the existence of dangling nodes, and therefore, the MC at hand is reducible. The MC associated with the web graph is not stochastic, due to existence of dangling nodes, and reducible. However, the PageRank algorithm handles these difficulties elegantly.

PageRank should be computed repeatedly for the changing web. This computation is challenging since it is expensive both in time and space. Different acceleration techniques have been considered after the proposal of the basic model. Some techniques aim to reduce the work incurred at each iteration of the power method, while others aim to reduce the number of iterations required by the power method. These goals are often at odds with one another. For example, reducing the number of iterations usually comes at the expense of a slight increase in the work per iteration. As long as this overhead is minimal, the proposed acceleration is considered beneficial [21, p.348]. In this context, certain improvements have been obtained using techniques such as quadratic extrapolation [19] and iterative aggregation [37, pp.307–331].

In this thesis, we propose iterative methods based on various block partitionings, including those with triangular diagonal blocks obtained using cutsets

[35, 30], for the computation of the steady-state vector of such stochastic matrices. The motivation is to decrease the iteration counts and solution times with respect to power and quadratically extrapolated power methods without increasing the space requirements too much. To this end, the proposed iterative methods together with power and quadratically extrapolated power methods are coded into a software tool and experiments are conducted on a variety of benchmark matrices.

In Chapter 2, we provide background information on MCs and numerical methods for their steady-state analysis. Details of the PageRank algorithm and important techniques proposed for accelerating PageRank computation are presented in Chapter 3. In Chapter 4, we define the proposed block partitionings and discuss implementation issues.

Properties of the benchmark matrices, experimental results with the solvers in the software tool [10] on the benchmark matrices, and evaluation of the results are presented in Chapter 5. The conclusion is given in Chapter 6.

Chapter 2

Background

Mathematical models are used to represent the behavior of various systems and are widely used in the natural sciences and engineering disciplines but also in social sciences. In many cases, problems such as population growth, disease transmission and decision making can be represented by mathematical models. One type of widely used representation is the class of Markov processes. Markov processes are used when the system has well-defined states (possibly an infinite number), which it will occupy over time. The defining property of a Markovian system is that the future evolution of the system depends only on the current state and not on its past history; this is called the memoryless property. When the state space of a Markov process is discrete, the process is referred to as a MC. If the MC may change state at any point in time, then the process is said to be a continuous-time Markov chain (CTMC). If the MC may change state at discrete points in time, then the process is called a discrete-time Markov chain (DTMC).

In the next section we define MCs formally. Discrete- and continuous-time MCs are discussed in more detail in the following two subsections. In the second section we give a formal definition of the steady-state vector. Finally, we provide an overview of numerical methods for the steady-state analysis of MCs in the last section.

2.1 Markov Chains

A MC is a type of a stochastic process, and therefore, we first give the definition of a stochastic process.

Definition 1. *[37, p.4] A stochastic process is a family of random variables* $\{X(t), t \in \mathcal{T}\}$ *on a given probability space* $\mathcal{S}$ *and indexed by parameter t, where t varies over some index set (parameter space)* $\mathcal{T}$*.*

$\mathcal{T}$ is a subset of $(-\infty, +\infty)$ and usually thought of as the time parameter set. As such, $\mathcal{T}$ is sometimes called the time range, and $X(t) \in \mathcal{S}$ denotes the value of the observation at time $t \in \mathcal{T}$.

Next, we give the definition of a Markov process and then that of a MC. In the discussion that follows, $\aleph$ denotes the set of natural numbers.

Definition 2. *A Markov process is a stochastic process that has no memory. That is, only the current state of the process can affect where it goes next. Thus, if* $t_0 < t_1 < \ldots < t_k$ $(k \in \aleph)$ *represent instants in the time parameter set* $\mathcal{T}$*, a stochastic process* $\{X(t), t \in \mathcal{T}\}$ *becomes a Markov process if it possesses the memoryless property [37, p.4]:*

$$Prob\{X(t) \leq x \mid X(t_0) = x_0, X(t_1) = x_1, \ldots, X(t_k) = x_k\}$$

$$= Prob\{X(t) \leq x \mid X(t_k) = x_k\}.$$

Definition 3. *[37, p.4] A MC is a Markov process whose state space is discrete.*

The state space $\mathcal{S}$ of a MC is usually taken to be the set $\aleph$ or a subset of it. The state $X(t_k)$ contains all the relevant information concerning the history of the process at time $t_k \in \mathcal{T}$. This does not imply that transitions are not allowed to depend on the actual time instant at which they occur. When the transitions out of state $X(t_k)$ depend on time t_k, the MC is said to be nonhomogeneous. If transitions are independent of time, the Markov process is said to be homogeneous [37, p.4].

2.1.1 Discrete-Time Markov Chains

If the state space $\mathcal{S}$ of a Markov process is discrete and the process may change state at discrete points in time, we say that the process is a DTMC. In this case, the discrete parameter space $\mathcal{T}$ may be represented by the set $\aleph$ [37, p.4] and the evolution of the DTMC by the sequence of random variables $X_0, X_1, X_2, \ldots$.

A DTMC satisfies the memoryless property for all $n \in \aleph$ and all states $x_n \in \mathcal{S}$ [37, p.5] as in

$$Prob\{X_{k+1} = x_{k+1} \mid X_0 = x_0, X_1 = x_1, \ldots, X_k = x_k\}$$

$$= Prob\{X_{k+1} = x_{k+1} \mid X_k = x_k\}.$$

The conditional probability of making a transition from state i to state j when the time parameter increases from k to $k+1$, is called the transition probability of a MC and denoted shortly as [37, p.5]

$$p_{ij}(k) = Prob\{X_{k+1} = j \mid X_k = i\}.$$

For a homogeneous DTMC these probabilities are independent of k and denoted by

$$p_{ij} = Prob\{X_{k+1} = j \mid X_k = i\} \text{ for } k \in \aleph.$$

A transition probability matrix is used for representing the behavior of a homogeneous DTMC. The transition probability matrix is formed by placing p_{ij} in row i and column j for all i and j, and is denoted by P. Since the total probability of making transitions from a state to all states in $\mathcal{S}$ is one, the sum of elements in any row of P must be one. That is, P is a stochastic matrix, in which $p_{ij} \geq 0$ and $\sum_{j \in \mathcal{S}} p_{ij} = 1$ for $i \in \mathcal{S}$. When the MC is nonhomogeneous, the elements p_{ij} are replaced with $p_{ij}(k)$ and the matrix P with $P(k)$.

2.1.2 Continuous-Time Markov Chains

When a MC may change state at any point in time, we say that the process is a CTMC. In DTMCs we only address time steps; with CTMCs we interpret time instants as nonnegative numbers, that is, elements of the set $\mathcal{T} = [0, \infty)$. In a CTMC, for any sequence $t_0 < t_1 < \ldots < t_k < t_{k+1}$ and for random variables $X(t_0), X(t_1), \ldots$ the memoryless property holds [37, p.7] as in

$$Prob\{X(t_{k+1}) = x_{k+1} \mid X(t_0) = x_0, X(t_1) = x_1, \ldots, X(t_k) = x_k\}$$
$$= Prob\{X(t_{k+1}) = x_{k+1} \mid X(t_n) = x_n\}.$$

Transition probabilities for a nonhomogeneous CTMC are defined as

$$p_{ij}(s,t) = Prob\{X(t) = j \mid X(s) = i\} \text{ for } t > s.$$

If the CTMC is homogeneous, these probabilities depend on the difference between t and s, and are described as

$$p_{ij}(\tau) = Prob\{X(s+\tau) = j \mid X(s) = i\} \text{ for } \tau = t - s.$$

A CTMC is represented by the transition rate matrix $Q(t)$. Although this matrix is formed in a similar way to P, its off-diagonal elements are the instantaneous transition rates among different states when τ is sufficiently small. When the CTMC is homogeneous, the transition rates q_{ij} are independent of time, and the transition rate matrix is simply written as Q. Note that, the diagonal element in each row is equal to the negated sum of the off-diagonal elements in that row, i.e.,

$$q_{ii} = -\sum_{j \neq i} q_{ij} \text{ for all } i \in \mathcal{S}.$$

The next section introduces an important measure we are interested in computing in this work.

2.2 Steady-State Vector

In analyzing MCs, one may be interested in studying the behavior of the modeled system over a short period of time. Another study, however, would involve the long-run behavior of the system, that is, when number of transitions tends to infinity. In such a case, a systematic procedure which will predict the long-run behavior of the system becomes necessary. Next we give some definitions regarding the classification of states in a MC which are useful in studying the long-run behavior of the system.

Definition 4. *[31, pp.143–144] Two states that are accessible from each other by following transitions in the MC are said to communicate. If all states communicate, the MC is said to be irreducible.*

Definition 5. *[37, p.10] A state in a MC is periodic if the probability of returning to the state is zero except at regular intervals. If a state is not periodic, it is aperiodic. If all states are aperiodic, then the MC is said to be aperiodic.*

We remark that the concept of periodicity applies only to DTMCs. If a state has transitions to itself but has no transitions to other states, it is called an absorbing state. That is, once the system is in an absorbing state, it will remain in that state indefinitely. Of interest to us in this work are the probabilities of being in states in the long-run.

Let the probability that a DTMC is in state i at time k be denoted by $\pi_i^{(k)}$, where $\pi_i^{(k)}$ is element i of the probability (row) vector $\pi^{(k)}$. Then we have the following definitions.

Definition 6. *[37, p.15] Given an initial probability distribution $\pi^{(0)}$, if the limit*

$$\lim_{k\to\infty} \pi^{(k)}$$

exists, then this limit is unique and called the steady-state (long-run, limiting, equilibrium) distribution vector; we write

$$\pi = \lim_{k\to\infty} \pi^{(k)}.$$

A similar definition can be made for a MC with a continuous parameter space.

Definition 7. *[37, p.15] Let P be the transition probability matrix of a DTMC and let vector z (whose element z_j denotes the probability of being in state j) be a probability distribution; i.e.,*

$$0 \leq z_j \leq 1, \ \sum_{j \in \mathcal{S}} z_j = 1.$$

Then z is said to be a stationary distribution vector of P if and only if $zP = z$.

Definition 8. *[37, p.18] Let Q be the transition rate matrix of a CTMC and let vector z (whose element z_j denotes the probability of being in state j) be a probability distribution; i.e.,*

$$0 \leq z_j \leq 1, \ \sum_{j \in \mathcal{S}} z_j = 1.$$

Then z is said to be a stationary distribution vector of Q if and only if $zQ = 0$.

A stationary distribution is not necessarily the steady-state probability distribution of a MC, but for irreducible and aperiodic finite DTMCs it is [37, p.5]. Note that the steady-state distribution is independent of the initial distribution.

If the steady-state distribution exists for a homogeneous DTMC, it satisfies

$$\pi P = \pi, \ \sum_{j \in \mathcal{S}} \pi_j = 1,$$

and for a homogeneous CTMC, it satisfies

$$\pi Q = 0, \ \sum_{j \in \mathcal{S}} \pi_j = 1.$$

A CTMC Q can be uniformized to obtain a DTMC P as

$$P = I + \Delta t Q, \text{ where } \ \Delta t \leq \frac{1}{\max\limits_{i \in \mathcal{S}} |q_{ii}|} \ .$$

When it exists, the steady-state vector of the CTMC (obtained from $\pi Q = 0$) is

identical to that of the uniformized DTMC (obtained from $\pi P = \pi$) [37, p.24]. In this work, we are interested in computing π for Google-like stochastic matrices introduced in Chapter 3.

In the next section, we briefly review iterative methods for the steady-state analysis of large, sparse MCs.

2.3 Steady-State Analysis of Markov Chains

Numerical methods that compute the steady-state vector of a MC in a predetermined number of operations are classified as direct methods [37, p.61]. On the other hand, iterative methods begin from some initial approximation and compute a new approximation at each iteration, with the expectation that the approximation converges to the steady-state vector [37, p.61].

Gaussian elimination [37, p.63] and QR factorization [16] are two of the direct methods used in MC problems. Iterative methods can be classified into three categories. Stationary iterative methods include the power method, (block) Jacobi overrelaxation ((B)JOR), and (block) successive overrelaxation ((B)SOR). Krylov subspace (or projection) methods include Arnoldi, generalized minimum residual (GMRES), full orthogonalization (FOM), biconjugate gradient (BCG), biconjugate gradient stabilized (BiCGStab), quasi-minimal residual (QMR) and conjugate gradient squared (CGS) methods [37, pp.117–230],[32]. Decompositional methods include iterative aggregation-disaggregation (IAD) [37, pp.307–331] and Schwarz methods [26]. In this study, we do not consider approximation methods (see, for instance, [27]) and bounding methods (see, for instance, [33]).

As the number of states in the MC increases, computing its steady-state vector becomes challenging. In order to handle very large state spaces, sparse storage schemes are used. For large problems, direct methods become inefficient, because they introduce new nonzero elements during factorization. On the other hand, iterative methods involve matrix-vector multiplications or equivalent operations, which maintain the nonzero structure of the matrix. Although they may exhibit

slow convergence rates, they enable the computation of the steady-state vector up to some user defined accuracy. In the rest of this work, we concentrate on stationary iterative methods and IAD method, since Krylov subspace methods require more storage, and tend to be less robust, especially in the absence of preconditioners [37, pp.222–225], [32, Ch.9–10].

The following sections introduce the power method, the (B)JOR and (B)SOR iterative methods based on splittings, and the IAD method.

2.3.1 Power Method

The power method may be used to determine the left eigenvector corresponding to the dominant eigenvalue of a matrix. If the computation of the steady-state vector of a MC is formulated as an eigenvalue problem, i.e., $\pi P = \pi$, the power method may be used to solve the problem. Let the probability transition matrix be P and let the initial probability distribution vector of P be $\pi^{(0)}$. The probability distribution vector after one transition, $\pi^{(1)}$, may be obtained from the product $\pi^{(0)}P$. Likewise, the probability distribution vector after two transitions may be obtained from the product $\pi^{(1)}P$. For $k \in \aleph$, the probability distribution vector of the system after k transitions is obtained by multiplying the probability vector obtained after $(k-1)$ transitions by P. Thus, we have

$$\pi^{(k)} = \pi^{(k-1)}P \quad \text{for} \quad k = 1, 2, \ldots$$

which leads to

$$\pi^{(k)} = \pi^{(k-1)}P = \pi^{(k-2)}P^2 = \cdots = \pi^{(0)}P^k \quad \text{for} \quad k = 1, 2, \ldots \ .$$

Recall that for a finite, aperiodic, and irreducible DTMC, $\pi^{(k)}$ converges to the steady-state vector regardless of the choice of initial vector [37, pp.121–125]. Since the dominant eigenvalue of P is 1, the convergence rate of the power method depends on the magnitude of the subdominant eigenvalue of P. Hence, the rate

increases if the magnitude of the subdominant eigenvalue becomes smaller. Typically, the convergence rate of the power method is slow.

2.3.2 Iterative Methods Based on Splittings

Stationary iterative methods based on splittings can be also used for solving the system of linear equations given by

$$Ax = b,$$

where $A \in \mathbb{R}^{n\times n}$ is the coefficient matrix, $b \in \mathbb{R}^{n\times 1}$ is the right-hand side vector, and $x \in \mathbb{R}^{n\times 1}$ is the vector of unknowns. In our case, $A = Q^T$ or $A = P^T - I$, $b = 0$, and $x = \pi^T$.

Now, consider the partitioning of A and x given by

$$A = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1K} \\ A_{21} & A_{22} & \cdots & A_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ A_{K1} & A_{K2} & \cdots & A_{KK} \end{pmatrix}_{n\times n}, \; x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_K \end{pmatrix}_{n\times 1}$$

for some $K \in \{1, 2, \ldots, n\}$, and write A as the sum of three terms as in

$$A = D - L - U,$$

where D is the block diagonal of A, $-L$ is its strictly block lower-triangular part, and $-U$ is its strictly block upper-triangular part. That is,

$$D = \begin{pmatrix} A_{11} & 0 & \cdots & 0 \\ 0 & A_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & A_{KK} \end{pmatrix},$$

$$L = -\begin{pmatrix} 0 & 0 & \cdots & 0 \\ A_{21} & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ A_{K1} & \cdots & A_{KK-1} & 0 \end{pmatrix},\ U = -\begin{pmatrix} 0 & A_{12} & \cdots & A_{1K} \\ 0 & 0 & \ddots & \vdots \\ \vdots & \vdots & \ddots & A_{K-1K} \\ 0 & 0 & \cdots & 0 \end{pmatrix}.$$

The methods of (B)JOR and (B)SOR for MCs are iterative methods of the form

$$Mx^{(k+1)} = Nx^{(k)} \text{ for } k = 0, 1, \ldots,$$

where M^{-1} is nonsingular in the splitting $A = M - N$. In the next two subsections, we define M and N for (B)JOR and (B)SOR.

2.3.2.1 (B)JOR Method

For BJOR, M and N are given by [32, p.96]

$$M^{BJOR} = \frac{D}{\omega}, \quad N^{BJOR} = \frac{1-\omega}{\omega}D + L + U \ ,$$

where $\omega \in (0, 2)$ is the relaxation parameter. BJOR reduces to block Jacobi (BJ) for $\omega = 1$ and becomes point JOR when $K = n$.

Writing this in detail, we obtain

$$D_{ii}x_i^{(k+1)} = (1-\omega)D_{ii}x_i^{(k)} + \omega\left(\sum_{j=1}^{i-1} L_{ij}x_j^{(k)} + \sum_{j=i+1}^{K} U_{ij}x_j^{(k)}\right) \quad \text{for} \quad i = 1, 2, \ldots, K.$$

2.3.2.2 (B)SOR Method

For BSOR, M and N are given by

$$M^{BSOR} = \frac{D}{\omega} - U \ , \quad N^{BSOR} = \frac{1-\omega}{\omega}D + L \ ,$$

where $\omega \in (0,2)$ is the relaxation parameter. BSOR reduces to block Gauss-Seidel (BGS) for $\omega = 1$ and becomes point SOR when $K = n$.

Writing this in detail, we obtain

$$D_{ii}x_i^{(k+1)} = (1-\omega)D_{ii}x_i^{(k)} + \omega\left(\sum_{j=1}^{i-1} L_{ij}x_j^{(k)} + \sum_{j=i+1}^{K} U_{ij}x_j^{(k+1)}\right) \text{ for } i = 1, 2, \ldots, K.$$

The convergence rate of iterative methods based on splittings depends on the magnitude of the subdominant eigenvalue of the iteration matrix $T = M^{-1}N$. In general, block iterative methods require more computation per iteration than point iterative methods based on splittings, but this is offset by a faster rate of convergence [37, p.138]. Among the methods discussed so far, power method converges the slowest, BSOR typically outperforms BJOR.

2.3.3 IAD Method

The transpose of the $(K \times K)$ matrix H whose ijth element is given by

$$h_{ij} = e^T A_{ij}\phi_j,$$

where

$$\phi_j = x_j/\|x_j\|_1$$

is referred to as the *exactly aggregated matrix* corresponding to A.

Now, let $\xi = (\xi_1, \xi_2, \ldots, \xi_K)^T$ be the unique positive vector satisfying $H\xi = 0$, $\sum_{i=1}^{K} \xi_i = 1$. Then the IAD method with a BGS disaggregation step (IADBGS) [37, p.311] is equivalent to the iterative formula

$$x^{(k+1)} = (D-U)^{-1}LD^{(k)}x^{(k)},$$

where

$$h_{ij}^{(k)} = e^T A_{ij}\phi_j^{(k)},$$

$$H^{(k)}\xi^{(k)} = 0, \xi^{(k)} > 0, \sum_{i=1}^{K} \xi_i^{(k)} = 1,$$

$$\text{diag}(\xi_1^{(k)} I/\|x_1^{(k)}\|, \xi_2^{(k)} I/\|x_2^{(k)}\|, \ldots, \xi_K^{(k)} I/\|x_K^{(k)}\|).$$

Writing this in detail, we obtain

$$D_{ii} x_i^{(k+1)} = \sum_{j=1}^{i-1} L_{ij} z_j^{(k+1)} + \sum_{j=i+1}^{K} U_{ij} x_j^{(k+1)} \quad \text{for} \quad i = 1, 2, \ldots, K,$$

where

$$z^{(k+1)} = (\xi_1^{(k)} (\phi_1^{(k)})^T, \xi_2^{(k)} (\phi_2^{(k)})^T, \ldots, \xi_K^{(k)} (\phi_K^{(k)})^T)^T.$$

A similar formula may be written for BJ (IADBJ), hence, for BJOR (IADBJOR) and BSOR (IADBSOR) disaggregation steps.

Definition 9. *[37, p.285] Let the state space of a MC be partitioned into disjoint subsets but with strong interactions among the states of a subset but with weak interactions among the subsets themselves. Such MCs are referred to as nearly completely decomposable (NCD).*

For NCD MCs the convergence of IAD methods will be rapid [37, p.342]. Let $P = \text{diag}(P_{11}, P_{22}, \cdots, P_{KK}) + \epsilon$, then $\|\epsilon\|_\infty$, is referred as the *degree of coupling*. The error in the approximate solution is reduced by a factor of order ϵ at each iteration. In two-level solvers, direct methods may be used for the solution of the aggregated matrix or individual blocks when there is sufficient space to factorize them.

In the next chapter, we discuss specialized iterative methods for the steady-state analysis of MCs obtained from Google-like stochastic matrices.

Chapter 3

Steady-State Analysis of Google-like Stochastic Matrices

One of the recent areas in which MCs appear is the analysis of the hyperlink structure of the web. The web has a massive size and search engines have to consider all web pages in which a query term exists. In order to make the list of pages returned to a query manageable for a user, search engines use some kind of ranking in which link analysis is used. Hypertext Induced Topic Search (HITS) and PageRank are two of the most popular link analysis algorithms.

The HITS algorithm was developed by John Kleinberg in 1998. HITS defines *authorities* and *hubs*, where an authority is a web page with several inlinks and a hub is a web page with several outlinks. HITS assigns an authority score and a hub score to each web page. The idea behind this approach is that good authorities are pointed by good hubs and good hubs point to good authorities [23, p.136]. The outcome of the HITS algorithm is two ranked lists. The first list contains the most authoritative pages related to a query and the second one contains the most "hubby" pages [23, pp.142–143]. A user may be interested in one ranked list rather than the other depending on the application. The HITS algorithm is query-dependent; that is, documents containing references to the query terms are determined at query time and at least one matrix eigenvector

problem is solved [23, p.143]. An extension of the HITS algorithm is used by the search engine TEOMA [23, p.144].

One other successful link-based ranking algorithm is PageRank, the ranking algorithm used by Google's search engine [21, p.1]. The PageRank algorithm was developed by Sergey Brin and Larry Page in 1998. The idea was that a page is important if it is pointed by other important pages [22, p.2]. Each web page is assigned a PageRank score that measures its importance. The PageRank score of a particular web page is the sum of the PageRank scores of all pages that point to that page [23, p.137]. Moreover, when an important page points to several pages, its PageRank score is distributed proportionally [22, p.2]. Unlike HITS, PageRank is query-independent.

In 2000, a third link analysis algorithm called Stochastic Approach for Link Structure Analysis (SALSA) is developed by Lempel and Moran [25]. SALSA combines some of the best features of PageRank and HITS. SALSA uses MCs and like HITS, it assigns an authority score and a hub score to each web page [23, p.154]. One major drawback of SALSA, like HITS, is its query-dependence [23, p.157].

This work concentrates on the PageRank algorithm, which is presented in the following section in detail.

3.1 PageRank Algorithm

The PageRank model of Google uses the hyperlink structure of the web to build a DTMC, P. The PageRank model forces P to be stochastic, irreducible, and aperiodic, to ensure that its steady-state vector exists.

Let us assume there are n pages on the web and consider the hyperlink structure of the web as a directed graph, where nodes represents web pages and directed arcs represent hyperlinks. The PageRank model first represents this graph with a square matrix P of order n, whose element p_{ij} is the probability of moving from

state i (page i) to state j (page j) in one time step.

Furthermore, the probability distribution vector v denotes visit probabilities of the user for web pages at steady-state and satisfies $v^T e = 1$. If it is equally likely for a user to visit any of the n web pages, then $v = e/n$. Of course, the column vector v can differ from the uniform distribution and be biased according to preference for certain types of pages. For this reason, v is called the *personalization* vector.

It has been already mentioned that there are pages on the web for which there are no outgoing hyperlinks and such pages are called *dangling*. The first modification is to artificially add appropriate links to all zero rows in P so that their sums become one and there are no dangling nodes remaining. That is, all zero rows in P are replaced with v^T, and the resulting matrix is given by

$$\overline{P} = P + av^T,$$

where a is the column vector whose element i is equal to one if page i has no outlinks and zero otherwise for $i = 1, 2, \ldots, n$ [21, p.339]. Note that a can be written as

$$a = e - Pe.$$

Although this modification makes P stochastic, and thus it has one as its dominant eigenvalue, it still may be reducible.

Hence, another modification is made and the resulting matrix is given by

$$\overline{\overline{P}} = \alpha\overline{P} + (1 - \alpha)E,$$

where $E = ev^T$ and α is a constant *scaling factor* such that $0 < \alpha < 1$.

The convex combination of the stochastic matrix $\overline{P}$ with the full perturbation

matrix E ensures that $\overline{\overline{P}}$ is stochastic, irreducible, and aperiodic. This corresponds to adding each page a new set of outgoing transitions, and we can write

$$\overline{\overline{P}} = \alpha P + (\alpha a + (1-\alpha)e)v^T. \tag{3.1}$$

Observe that $\overline{\overline{P}}$ is a rank-2 modification of P.

The steady-state vector, π, of $\overline{\overline{P}}$ is called the PageRank vector and computing the PageRank vector can be viewed as solving the eigenvalue problem $\pi\overline{\overline{P}} = \pi$. Power iterations on matrix $\overline{\overline{P}}$ now converge to the unique PageRank vector from which web pages can be ranked according to their relative importance. Note that, small change in the value of α gives a dramatic change in the steady-state vector [34, p.310].

The next section covers details of the method proposed for computing the steady-state vector of $\overline{\overline{P}}$ in the literature.

3.2 Power Method

The basic PageRank algorithm uses the power method to compute the steady-state vector of $\overline{\overline{P}}$. For the starting vector $\pi^{(0)}$ such that $\pi^{(0)} \geq 0$ and $\pi^{(0)}e = 1$, we have

$$\begin{aligned}\pi^{(k+1)} = \pi^{(k)}\overline{\overline{P}} &= \alpha\pi^{(k)}\overline{P} + (1-\alpha)\pi^{(k)}ev^T = \alpha\pi^{(k)}\overline{P} + (1-\alpha)v^T \\ &= \alpha\pi^{(k)}P + (\alpha\pi^{(k)}a + (1-\alpha))v^T \text{ for } k = 0, 1, \ldots \ .\end{aligned}$$

Since, $\overline{\overline{P}} > 0$ and $\overline{\overline{P}}e = e$, $\pi^{(k+1)}$ satisfies $\pi^{(k+1)} > 0$ and $\pi^{(k+1)}e = 1$. Hence, the power method applied to $\overline{\overline{P}}$ can be implemented with a vector-matrix multiplication using αP and a few level-1 operations (such as dot product and saxpy); $\overline{P}$ and $\overline{\overline{P}}$ are never formed or stored. When P is sparse, each vector-matrix multiplication can be computed in $nz(P)$ flops, where $nz(P)$ is the number of nonzeros in P, and if P is sparse, $O(nz(P)) \approx O(n)$. Moreover, at each iteration, the power method requires the storage of three vectors, the current iterate, αa and v^T [21,

p.344].

It has already been mentioned that the rate of convergence of the power method depends on the subdominant eigenvalue of the iteration matrix. For $\overline{\overline{P}}$, the subdominant eigenvalue is equal to α for a reducible matrix αP, and is strictly less than α for an irreducible matrix αP. For $\overline{\overline{P}}$ obtained from the web graph, convergence of the power method takes place at the rate by which α^k goes to 0. Thus, as α becomes smaller, convergence becomes faster. However, the smaller α is, to a lesser extent the hyperlink structure of the web is used to determine web page rankings. Slightly different α values can produce very different PageRank vectors, and as α goes to 1, sensitivity issues begin to arise. Brin and Page, the founders of Google, use $\alpha = 0.85$, and for tolerance levels measured by residual norms ranging from 10^{-3} to 10^{-7}, they report convergence within 50 to 100 power iterations [21, p.345].

Although, the PageRank algorithm aims to solve an essentially old problem (that is, computing the steady-state vector of a MC), the enormous size of the problem makes it challenging. In the following section, we discuss an improvement proposed in the context of the power method for accelerating the computation of the PageRank vector.

3.3 Quadratic Extrapolation on Power Method

An approach aiming to reduce the number of iterations of the power method is proposed by Kamvar et al. [19] and is known as quadratic extrapolation (QE). It is reported that by periodically applying QE for values of α close to 1, the convergence of PageRank can be accelerated by a factor of over 3 [19, p.265]. QE computes the extrapolated solution $\pi^{(k)}$ using the last four approximate solution vectors, $\pi^{(k-3)}$, $\pi^{(k-2)}$, $\pi^{(k-1)}$ and $\pi^{(k)}$. In QE, it is assumed that the matrix $\overline{\overline{P}}$ has only two eigenvectors and the extrapolated solution $\pi^{(k)}$ can be expressed as a linear combination of the last three approximate solution vectors.

Let us define the three successive approximations as

$$\pi^{(k-2)} = \overline{\overline{P}}\pi^{(k-3)},\ \pi^{(k-1)} = \overline{\overline{P}}\pi^{(k-2)},\ \pi^{(k)} = \overline{\overline{P}}\pi^{(k-1)}.$$

and introduce

$$y^{(k-2)} = \pi^{(k-2)} - \pi^{(k-3)},\ y^{(k-1)} = \pi^{(k-1)} - \pi^{(k-3)},\ y^{(k)} = \pi^{(k)} - \pi^{(k-3)}.$$

Under the given assumptions, the characteristic polynomial of $\overline{\overline{P}}$ is of the form

$$p_{\overline{\overline{P}}}(\lambda) = \gamma_0 + \gamma_1\lambda + \gamma_2\lambda^2 + \gamma_3\lambda^3.$$

We know that one is an eigenvalue of $\overline{\overline{P}}$ and eigenvalues of $\overline{\overline{P}}$ are also the zeros of the characteristic polynomial, so $p_{\overline{\overline{P}}}(1) = \gamma_0 + \gamma_1 + \gamma_2 + \gamma_3 = 0$. By using some algebra and the Cayley-Hamilton theorem [28, p.509], we have

$$\left(y^{(k-2)}\ y^{(k-1)}\ y^{(k)}\right)\gamma = 0,$$

where $\gamma = (\gamma_1\ \gamma_2\ \gamma_3)^T$. We are interested in the value of γ other than the trivial solution $\gamma = 0$. By letting $\gamma_3 = 1$, we obtain

$$\left(y^{(k-2)}\ y^{(k-1)}\right)\begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix} = -y^{(k)} \text{ and } \begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix} = -Y^{+}y^{(k)},$$

where Y^+ is the pseudoinverse of the matrix $Y = \left(y^{(k-2)}\ y^{(k-1)}\right)$ defined in [19, p.266]. In order to find the values γ_1 and γ_2, the truncated QR factorization $Y = QR$ can be used by executing two steps of the Gram-Schmidt algorithm since Y is an $(n \times 2)$ matrix. Then $-Q^Ty^{(T)}$ is computed, and finally the upper-triangular system

$$R\begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix} = -Q^Ty^{(k)}$$

is solved for $(\gamma_1\ \gamma_2)^T$ by using back substitution [19, p.266]. Since $-Q^Ty^{(k)}$ can be formed in $O(n)$ operations and R is (2×2), γ_1 and γ_2 are effectively computed

in $O(n)$ flops.

With QE, the approximations obtained by the power method are modified periodically. The approximate solution vector $\pi^{(k)}$ of the power method is modified by QE as [19, p.266]

$$\pi^{(k)} = \beta_0 \pi^{(k-2)} + \beta_1 \pi^{(k-1)} + \beta_2 \pi^{(k)},$$

where $\beta_0 = \gamma_1 + \gamma_2 + \gamma_3$, $\beta_1 = \gamma_2 + \gamma_3$, and $\beta_2 = \gamma_3$.

Since QE decreases the error in $\pi^{(k)}$ along the direction of the second and third eigenvectors, it enhances the convergence of future iterations of the power method [19, p.267]. Brezinski et al. generalized QE and interpreted it on the basis of Krylov subspace method [6].

In the next section, we discuss an approach for updating the steady-state vector of Google-like matrices.

3.4 Updating the Steady-State Vector

Web is a highly dynamical environment and computing the steady-state vector is not a one time job. As hyperlinks and web pages are added and deleted, P, and therefore, the steady-state vector changes. Instead of computing the new steady-state vector of $\overline{\overline{P}}$ in (3.1) from scratch, one may use components of the previously computed steady-state vector; this is called *updating*. When only the hyperlink structure of the web graph changes, the problem is called *element-updating*; if states are added or deleted, the problem is called *state-updating*. The state-updating problem is clearly more difficult, and it includes the element-updating problem as a special case [24, pp.1–2]. In [24], a general purpose algorithm is presented, which is based on aggregation-disaggregation principals and simultaneously handles both kinds of updating problems.

Assume that the steady-state vector $\phi = (\phi_1 \ \phi_2 \ \dots \ \phi_m)$ for an irreducible MC is already known and suppose that this MC needs to be updated. Let the

updated transition probability matrix be S and the updated steady-state vector be $\pi = (\pi_1\ \pi_2\ \ldots\ \pi_n)$. Note that n is not necessarily equal to m because the updating process allows for addition and deletion of states as well as the alteration of transition probabilities.

The idea behind approximate aggregation is to use ϕ and S to build an aggregated MC having a transition probability matrix A, that is smaller in size than S, so that the steady-state vector of A can be used to generate an estimate of π [24, p.4]. The state space $\mathcal{S}$ of S is first partitioned as $\mathcal{S} = \mathcal{G} \cup \overline{\mathcal{G}}$, where $\mathcal{G}$ consists of states that are likely to be most affected by the updates. Newly added states are automatically included in $\mathcal{G}$ and deleted states are accounted for by changing the affected transition probabilities to zero. The complement $\overline{\mathcal{G}}$ contains all other states. The effect of perturbations involving only a few states in large, sparse MCs is primarily local, and most steady-state probabilities are not significantly affected [24, p.4].

After partitioning and reordering, let us have

$$S = \begin{array}{c} \\ \mathcal{G} \\ \overline{\mathcal{G}} \end{array} \begin{array}{c} \begin{array}{cc} \mathcal{G} & \overline{\mathcal{G}} \end{array} \\ \begin{pmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{pmatrix} \end{array} \tag{3.2}$$

and

$$\pi = (\pi_1\ \pi_2\ \ldots\ \pi_g \mid \pi_{g+1}\ \ldots\ \pi_n) = (\pi_1\ \pi_2\ \ldots\ \pi_g \mid \overline{\pi}),$$

where S_{11} is $(g \times g)$. The steady-state probabilities from the original distribution ϕ that correspond to the states in $\overline{\mathcal{G}}$ are placed in a row vector $\overline{\phi}$ and states in $\overline{\mathcal{G}}$ are lumped into one superstate to create a smaller aggregated MC whose transition matrix is given by [24, p.5]

$$A = \begin{pmatrix} S_{11} & S_{12}e \\ sS_{21} & 1 - sS_{21}e \end{pmatrix},$$

where $s = \phi/(\phi e)$. Now

$$\pi_i \approx \begin{cases} \alpha_i & \text{if state } i \text{ belongs to } \mathcal{G}, \\ \alpha_{g+1} s_i & \text{if state } i \text{ belongs to } \overline{\mathcal{G}} \end{cases}$$

where α is the steady-state vector of A.

The iterative aggregation updating algorithm proposed in [24] starts by partitioning and reordering the states as described above. Steps of this algorithm are as follows [24]:

Initialize

1. Partition the states of the updated chain as $\mathcal{S} = \mathcal{G} \cup \overline{\mathcal{G}}$ and reorder S as described in (3.2)
2. $\overline{\phi} \longleftarrow$ components of ϕ that correspond to states in $\overline{\mathcal{G}}$
3. $s \longleftarrow \overline{\phi}\ /\ \overline{\phi} e$

Iterate until convergence

1. $A \longleftarrow \begin{pmatrix} S_{11} & S_{12}e \\ sS_{21} & 1 - sS_{21}e \end{pmatrix}$
2. $\alpha \longleftarrow (\alpha_1, \alpha_2, \ldots, \alpha_g, \alpha_{g+1})$ (steady-state vector of A)
3. $\chi \longleftarrow (\alpha_1, \alpha_2, \ldots, \alpha_g | \alpha_{g+1} s)$
4. $\psi \longleftarrow \chi P$
5. If $\|\psi - \chi\| < \tau$ for a given tolerance τ, then quit;
   else $s \longleftarrow \psi/\psi e$ and go to step 1

The iterative aggregation updating algorithm converges to the steady-state vector π for all partitions $\mathcal{S} = \mathcal{G} \cup \overline{\mathcal{G}}$. The rate at which the approximations converge to π is exactly at which the powers S^k converge, where the only significant stochastic

complement of S is that of S_{22} and defined as $\overline{S}_{22} = \overline{\overline{P}}_{22} + \overline{\overline{P}}_{21}(I - \overline{\overline{P}}_{11})^{-1}\overline{\overline{P}}_{12}$ [24, p.10]. If the subdominant eigenvalue $\lambda_2(\overline{S}_{22})$ of S is real and simple, then the asymptotic rate of convergence is $-\log_{10}|\lambda_2(\overline{S}_{22})|$ [24, p.10]. More information on the iterative aggregation updating algorithm and convergence analysis can be found in [24, 17].

Chapter 4

Proposed Methods

It is known that Google matrices can be permuted to block upper-triangular form in which the diagonal blocks are irreducible [29]. An algorithm for computing a (2×2) block partition of an irreducible matrix with zero-free diagonal in which one of the diagonals block is triangular is presented in [9]. In this work, these results are used to show that the solution process can be improved by computing a (2×2) block partition for each irreducible diagonal block in this permuted matrix, where one of the diagonal blocks is triangular. Solution of linear systems which have triangular coefficient matrices can be performed directly by using forward or backward substitutions depending on the shape of triangularity. Thus, for a block upper-triangular matrix with irreducible diagonal blocks, such a (2×2) block partition can be computed for each diagonal block and substitution can be used for solving the triangular diagonal blocks at each (outer) block iteration, while solution of the remaining diagonal blocks can be approximated with some kind of (inner) point iteration. This approach circumvents the fill-in problem associated with factorizing the diagonal blocks.

In the following sections, we present three methods for obtaining such block partitionings for Google-like stochastic matrices after we discuss how one can obtain triangular diagonal blocks in a given matrix.

4.1 Obtaining Triangular Blocks Using Cutsets

4.1.1 Partitioning an Irreducible Matrix

The generator matrix of an irreducible CTMC can be symmetrically permuted and partitioned into four blocks as in

$$Q = \begin{array}{c} \\ \mathcal{C} \\ \mathcal{T} \end{array} \begin{array}{c} \begin{array}{cc} \mathcal{C} & \mathcal{T} \end{array} \\ \begin{pmatrix} C & Y \\ Z & T \end{pmatrix} \end{array}$$

by partitioning its state space $\mathcal{S}$ into two subsets such that $\mathcal{S}_\mathcal{C} \cup \mathcal{S}_\mathcal{T} = \mathcal{S}$, and $\mathcal{S}_\mathcal{C} \cap \mathcal{S}_\mathcal{T} = \emptyset$, where C and T are square submatrices of order $n_\mathcal{C}$ and $n_\mathcal{T}$, respectively, and $\mathcal{T}$ is triangular [9].

Partitioning π accordingly as $(\pi_\mathcal{C}, \pi_\mathcal{T})$, we have

$$(\pi_\mathcal{C}\ \pi_\mathcal{T}) \begin{pmatrix} C & Y \\ Z & T \end{pmatrix} = 0,$$

where $(\pi_\mathcal{C}, \pi_\mathcal{T})e = 1$. Note that T is necessarily nonsingular, $n_\mathcal{T} = |\mathcal{T}|$, $n_\mathcal{C} = |\mathcal{C}|$,

and $n = n_{\mathcal{T}} + n_{\mathcal{C}}$. Next we discuss how such a partitioning can be obtained.

Let the *directed graph (digraph)* associated with the off-diagonal part of Q be $G(\mathcal{V}, \mathcal{E})$ and let $\mathcal{V} = \{1, 2, \ldots, n\}$ be the *node* (or *vertex*) *set* and $\mathcal{E} = \{(i, j) \mid q_{ij} \neq 0,\ i \neq j,\ \text{and}\ i, j \in \mathcal{V}\}$ be the *edge* (or *arc*) *set*. The following definition is used for computing this partition.

Definition 10. *[35, p.647] A vertex v cuts a path if it is an endpoint of one of the edges in this path. A set of vertices in a graph is a cutset (or feedback vertex set) if any cycle in the graph is cut by at least one vertex from this set.*

The smallest set of vertices which cuts all cycles in a directed graph $G(\mathcal{V}, \mathcal{E})$ is called *minimum cutset* [14] of $G(\mathcal{V}, \mathcal{E})$. The minimum cutset of a graph need not be unique [35, p.646].

Lemma 1. *[15, p.227] If $\mathcal{C}$ is a cutset of $G(\mathcal{V}, \mathcal{E})$ associated with the off-diagonal part of Q and $\mathcal{T} = \mathcal{S} - \mathcal{C}$, then*

i. there exists an ordering of states in $\mathcal{T}$ such that T is triangular;

ii. T is nonsingular.

Note that the second part of the lemma follows from the fact that Q has a zero-free diagonal.

Now it is clear that one has to find a small cutset of Q, since smaller the order of submatrix C is, the larger the order of submatrix T becomes. Since a triangular block can be solved exactly by using substitution it is useful to obtain a larger triangular block. The minimum cutset problem is an NP-complete problem for general graphs [20]; therefore, non-optimal solutions need to be considered.

Fortunately, for certain classes of graphs, polynomial time algorithms exist, such as Shamir's algorithm [35] whose complexity is linear in the number of edges in $\mathcal{E}$. Shamir's algorithm works on (quasi-) reducible graphs and aborts in the case of nonreducibility. These concepts are explained in the following paragraphs.

Reducibility in graphs is defined for *rooted graphs*, where the root is a node from which every other node in $\mathcal{V}$ is reachable by following a sequence of edges. If for a rooted graph different depth first search (DFS) orders of $G(\mathcal{V}, \mathcal{E})$ starting from the root result in a unique *directed acyclic graph* (*dag*), then the graph is called *reducible* [9, p.4]. It is easy to confuse reducibility in graphs with reducibility in matrices. To avoid this confusion, note that an irreducible matrix has a *strongly connected* graph $G(\mathcal{V}, \mathcal{E})$ [12, p.114]; that is, every node is reachable from every other node in $\mathcal{V}$. However, a reducible graph can be strongly connected and a nonreducible graph need not be strongly connected [35, p.648].

Shamir's algorithm computes the minimum cutset for a reducible graph by using DFS. Unfortunately, it may abort in the case of nonreducibility. However, it may also not abort and find the minimum cutset [35, p.654]. Such graphs are called *quasi-reducible* [30, p.206]. For more detail about Shamir's algorithm one can refer [35, 9].

Another algorithm, Rosen's algorithm called Cutfind, which is a modification of Shamir's algorithm, also runs in linear time and space and finds cutsets of not (quasi-)reducible graphs. Cutsets computed for not (quasi-)reducible graphs may not be minimum; however, Dayar [9] showed that Cutfind is a fast algorithm for large graphs compared to other approximation algorithms and the size of the cutset computed is generally satisfying.

Note that the root node is essential for using the Cutfind (also Shamir's) algorithm. In a strongly-connected graph, every node can be the root node. However, the web graph is not strongly connected; that is, Q is reducible. The aim is to obtain irreducible diagonal blocks of Q, which ensures the existence of a root node in each diagonal block, and use the Cutfind algorithm on the diagonal blocks to obtain triangular blocks. In the following section we cover Tarjan's algorithm which can be used to permute a matrix to block-triangular form in which the diagonal blocks are irreducible.

4.1.2 Partitioning a Reducible Matrix

Tarjan's algorithm finds all strongly connected components of a graph. Thus, Tarjan's algorithm can be used to find a symmetric permutation of a square matrix resulting in block triangular form with irreducible diagonal blocks [13, pp.137–138]. Without loss of generality, let Q be symmetrically permuted to block-lower triangular form using Tarjan's algorithm as in

$$Q = \begin{pmatrix} Q_{11} & & & \\ Q_{21} & Q_{22} & & \\ \vdots & \vdots & \ddots & \\ Q_{K1} & Q_{K2} & \cdots & Q_{KK} \end{pmatrix},$$

where the blocks Q_{ii} are square and irreducible [13, p.137]. Tarjan's algorithm has a complexity of $O(n) + O(\tau)$, where n is the order of Q and τ is the number of its off-diagonal nonzeros. The steps of Tarjan's algorithm and a detailed discussion of a Fortran implementation for sparse matrices, mc13dd in Harwell subroutine

library [1], can be found in [13].

Tarjan's algorithm returns the permutation indicating the irreducible diagonal blocks to which the Cutfind algorithm can be applied. Note that if Q is irreducible, there is no need to use Tarjan's algorithm, since there is only one diagonal block, the matrix itself.

In the next section, we discuss the details of this approach with an eye on the PageRank model and present an algorithm of the proposed method with implementation details.

4.2 Partitioning Google-like Stochastic Matrices

As shown in (3.1), the PageRank model of the web can be written as

$$\overline{\overline{P}} = \alpha P + (\alpha (a - e) + e) v^T,$$

where v is the personalization vector, e is the vector of ones, and $a = e - Pe$. Letting $Q^T = \overline{\overline{P}}^T - I$, we obtain

$$Q^T = \alpha P^T - I + \alpha v (a - e)^T + v e^T. \tag{4.1}$$

Now, let $A = \alpha P^T - I$. In order to exploit the sparse structure of P in $\overline{\overline{P}}$, computations are carried out on A by using the vectors a, v, and e. The full matrix Q^T in (4.1) is never explicitly formed or stored.

For the PageRank model, we apply Tarjan's algorithm to A and symmetrically permute A using the returned permutation. Without loss of generality, let us assume A has the following nonzero structure:

$$A = \begin{array}{c} \\ n_1 \\ n_2 \\ \vdots \\ n_K \end{array} \begin{array}{c} \begin{array}{cccc} n_1 & n_2 & \cdots & n_K \end{array} \\ \begin{pmatrix} A_{11} & & & \\ A_{21} & A_{22} & & \\ \vdots & \vdots & \ddots & \\ A_{K1} & \cdots & \cdots & A_{KK} \end{pmatrix} \end{array}. \tag{4.2}$$

Since all the diagonal blocks are irreducible and have zero-free diagonals, we can use the Cutfind algorithm on each of them without having any problem associated with determining a root node. Each of the nodes in an irreducible block can become the root.

After applying the Cutfind algorithm to each of the irreducible diagonal blocks, we obtain states in the cutsets defining the submatices C_{ii} for $i = 1, 2, \ldots, K$ as in

$$\begin{array}{c} \\ n_{C_{11}} \\ n_{T_{11}} \\ n_{C_{22}} \\ n_{T_{22}} \\ \vdots \\ \vdots \\ n_{C_{KK}} \\ n_{T_{KK}} \end{array} \begin{array}{c} \begin{array}{cccccccc} n_{C_{11}} & n_{T_{11}} & n_{C_{22}} & n_{T_{22}} & \cdots & \cdots & n_{C_{KK}} & n_{T_{KK}} \end{array} \\ \left(\begin{array}{cc|cc|ccc|cc} C_{11} & Y_{11} & & & & & & & \\ \mathrm{Z}_{11} & T_{11} & & & & & & & \\ \hline C_{21} & Y_{21} & C_{22} & Y_{22} & & & & & \\ Z_{21} & T_{21} & Z_{22} & T_{22} & & & & & \\ \hline \vdots & \vdots & \vdots & \vdots & \ddots & & & & \\ \vdots & \vdots & \vdots & \vdots & & \ddots & & & \\ \hline C_{K1} & Y_{K1} & C_{K2} & Y_{K2} & \ldots & \ldots & C_{KK} & Y_{KK} \\ Z_{K1} & T_{K1} & Z_{K2} & T_{K2} & \ldots & \ldots & Z_{KK} & T_{KK} \end{array}\right) \end{array}, \tag{4.3}$$

where $n_{\mathcal{T}} = \sum_{i=1}^{K} n_{T_{ii}}$ and $n_{\mathcal{C}} = \sum_{i=1}^{K} n_{C_{ii}}$. We know that for states in $\mathcal{T}_{ii}$ there exists an ordering such that T_{ii} is triangular [15, p.227]. To find this ordering, one practical way is to use Tarjan's algorithm. Tarjan's algorithm, as implemented in mc13dd and used for block triangularization purposes, returns $n_{\mathcal{T}_{ii}}$ irreducible diagonal blocks for the submatrix T_{ii}, and hence, the permutation for a lower-triangular ordering. If an upper-triangular ordering is desired, the returned permutation should be reversed.

It is also possible to obtain an ordering for triangular diagonal blocks, again using Tarjan's algorithm and the Cutfind algorithm. To achieve such an ordering the Cutfind algorithm is applied to A_{ii} in (4.2) and A is symmetrically permuted to (2×2) block form, in which the first diagonal block is triangular. Applying the same idea recursively to the second diagonal block and symmetrically permuting the matrix, we eventually obtain the following nonzero structure

$$\begin{array}{c} \\ n_{T_{11}} \\ n_{T_{22}} \\ \vdots \\ n_{T_{M-1M-1}} \\ n_{T_{MM}} \end{array} \begin{array}{c} \begin{array}{ccccc} n_{T_{11}} & n_{T_{22}} & \cdots & n_{T_{M-1M-1}} & n_{T_{MM}} \end{array} \\ \begin{pmatrix} T_{11} & X_{12} & \cdots & X_{1M-1} & X_{1M} \\ X_{21} & T_{22} & \cdots & X_{2M-1} & X_{2M} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ X_{M-11} & X_{M-12} & \cdots & T_{M-1M-1} & X_{M-1M} \\ X_{M1} & X_{M2} & \cdots & X_{MM-1} & T_{MM} \end{pmatrix} \end{array}, \tag{4.3}$$

where $\sum_{i=1}^{M} n_{T_{ii}} = n$. The Q^T matrix in (4.1) now can be solved for π^T by some kind of iterative method based on a block partitioning. Note that the triangular diagonal blocks in A become full in Q^T; however, we can still solve them with substitution using the Sherman-Morrison (S-M) formula given next.

Definition 11. *[28, p.124] If $A_{n\times n}$ is nonsingular and if c and d are $(n \times 1)$ column vectors such that $1 + d^T A^{-1} c \neq 0$, then the sum $\left(A + cd^T\right)$ is nonsingular and its inverse is given by*

$$\left(A + cd^T\right)^{-1} = A^{-1} - \frac{A^{-1}cd^T A^{-1}}{1 + d^T A^{-1} c}.$$

Let us write

$$Q^T = A + v\left(\alpha(a-e)^T + e^T\right),$$

then

$$Q^T = A + cd^T,$$

where $c = v$ and $d^T = \alpha\,(a-e) + e^T$ is a row vector of length n. Note that now we can apply the S-M formula to the diagonal blocks T_{ii} in (4.3) since they are easily invertible. The S-M formula for these blocks may be written as

$$(T_{ii} + c_{T_{ii}} d_{T_{ii}}^T)^{-1} = T_{ii}^{-1} - \frac{T_{ii}^{-1} c_{T_{ii}} d_{T_{ii}}^T T_{ii}^{-1}}{1 + d_{T_{ii}}^T T_{ii}^{-1} c_{T_{ii}}} \quad \text{for all} \quad i = 1, 2, \ldots, K\ ,$$

where $c_{T_{ii}}$ and $d_{T_{ii}}^T$ are the corresponding subvectors of c and d^T. Then the solution to a linear system of the form $(T_{ii} + c_{T_{ii}} d_{T_{ii}}^T)x_i = b_i$ becomes

$$x_i = T_{ii}^{-1} b_i - \frac{T_{ii}^{-1} c_{T_{ii}} d_{T_{ii}}^T \cdot T_{ii}^{-1} b_i}{1 + d_{T_{ii}}^T \cdot T_{ii}^{-1} c_{T_{ii}}}.$$

Now, $T_{ii}^{-1} b_i$ and $T_{ii}^{-1} c_{T_{ii}}$ can be easily solved by using substitutions. The remaining computations can be performed by a dot product and a saxpy.

To sum up, now Q^T can be solved with an iterative method based on a block partitioning. At each outer iteration, the triangular diagonal blocks can be solved directly with the help of the S-M formula, while the solution of non-triangular diagonal blocks can be approximated using some kind of point iteration. Now we present our algorithms and discuss implementation details.

Algorithm 1 shows how one can compute the steady-state vector of a Google-like stochastic matrix using iterative methods based on block partitionings with some triangular diagonal blocks. We remark that an irreducible MC with α set to 1 is a special case of Algorithm 1. Our implementation is designed to solve $Q^T\pi^T = 0$ by using P, P^T or Q as input. For Google-like stochastic matrices, the implementation takes the nonzero pattern of P as input using two integer arrays. The first array stores its column indices and is of length nz_P, where nz_P is the number of nonzero elements in P. The second array, whose length is $(n + 1)$, stores the beginning of row indices in the first array. When there are nonzero values, they are stored in a third array of length nz_P. For a Google-like stochastic matrix the nonzero values in each row of P are uniformly distributed. As preprocessing, we scale P with α and obtain $A = \alpha P^T - I$, then prepare the vectors v and a.

Algorithm 1. *Computes the steady-state vector of* $\overline{\overline{P}}$.

Ensure: π *is the steady-state vector of* $\overline{\overline{P}}$

Require: P *is a Google-like stochastic matrix,* v *is the personalization vector,* a *is the vector representing dangling nodes (i.e.,* $a = e - Pe$*),* $0 < \alpha < 1$

1: Compute the permutation that block triangularizes P *so that it has irreducible diagonal blocks.*

2: For each irreducible diagonal block, compute the cutset of the graph associated with the off-diagonal part of the diagonal block.

3: For each irreducible diagonal block, compute the permutation that triangularizes the submatrix associated with the nodes in cutset's complement.

4: Order and group the resulting submatrices along the diagonal to determine the overall permutation.

5: Compute steady-state vector π *by using an iterative method based on a block partitionings in which the triangular diagonal blocks are solved exactly using substitutions with the help of S-M formula and the remaining diagonal blocks are*

solved approximately by a point iterative method.

For step 1, we consider an implementation of Tarjan's algorithm, mc13dd [13]. We converted this Fortran routine to C for ease of portability. This routine uses the nonzero structure of the input matrix A and returns a permutation vector. Before moving to the next step, states in the irreducible diagonal blocks are ordered lexicographically. Then A is symmetrically permuted and transformed to a block form in which irreducible diagonal blocks are stored separately as rowwise sparse matrices and its off-diagonal part is also stored as a rowwise sparse matrix. Thus, the nonzero patterns of diagonal blocks become readily available.

Step 2 of Algorithm 1 involves finding the cutsets of graphs associated with the off-diagonal parts of irreducible diagonal blocks using their nonzero patterns. For this step Rosen's algorithm, Cutfind, is implemented [9]. The node with the largest outdegree in each irreducible diagonal block is chosen as the root node, and when there are ties, the node with the smallest lexicographical index is chosen. In our implementation, diagonal elements appear as the first element in their corresponding rows. This enables diagonal elements to be skipped and the graph associated with the off-diagonal part of A to be considered. The implementation of the Cutfind algorithm uses five integer work arrays of length n and a character work array of length n, besides the previously discussed two integer arrays for representing the nonzero structure of A. Two of the integer work arrays are used for implementing a stack of edges, one is used for labeling nodes, one is used for keeping track of the next unprocessed edge to consider for each node, and one is used for recording preorders of nodes. The character work array is used for marking processed nodes. The time complexity of the Cutfind algorithm is $O(nz_A)$ and the operations that are carried out are integer comparisons.

Tarjan's algorithm may return irreducible diagonal blocks of orders one and two. It is clear that the application of the Cutfind algorithm to these diagonal blocks is unnecessary. For blocks of order two, either of the states can be placed in the cutset, and for blocks of order 1 it is meaningless to find a cutset. Before running the Cutfind algorithm such diagonal blocks are identified and the algorithm is not applied to them. However, blocks of order one are checked whether they have zero off-diagonal elements because the ones with zero off-diagonal elements can be grouped to form a triangular block. The Cutfind algorithm is applied to diagonal blocks that have more than two states.

In step 3, the permutation that triangularizes the submatrix associated with the nodes in T_{ii}, that is, for states that are in cutset's complement, are obtained again by using Tarjan's algorithm. Different strategies can be considered when forming the overall permutation in step 4 of Algorithm 1. Let us refer to these strategies as partitionings 1 and 2 and explain their details in the next two subsections.

4.2.1 Partitioning 1

In partitioning 1, the triangular blocks and the diagonal blocks of order one with zero off-diagonal elements are grouped to form a larger triangular block. That is, blocks of order one with zero off-diagonal elements are placed consecutively in the first block and T_{ii} are placed after them. Note that, one state of blocks of order two (in our implementation the first state) are placed in the first block. Ordering of the remaining diagonal blocks which consist of the states in the cutset, that is C_{ii} (note that, for blocks of order two one state, in our implementation the last state, is placed in the cutset) are not changed. Diagonal blocks of order one which do not have zero off-diagonal elements are grouped into the last block.

After symmetrically permuting A according to the permutation obtained in this manner, we obtain the following nonzero structure:

$$
\begin{array}{c}
\\
n_{T_{00}} \\
n_{T_{11}} \\
\vdots \\
n_{T_{KK}} \\
n_{C_{11}} \\
\vdots \\
n_{C_{KK}} \\
n_{C_{K+1K+1}}
\end{array}
\begin{array}{c}
\begin{array}{cccccccc}
n_{T_{00}} & n_{T_{11}} & \cdots & n_{T_{KK}} & n_{C_{11}} & \cdots & n_{C_{KK}} & n_{C_{K+1K+1}}
\end{array} \\
\left(\begin{array}{cccccccc}
T_{00} & & & & & & & \\
\hline
T_{10} & T_{11} & & & Z_{11} & & & \\
\vdots & \vdots & \ddots & & \vdots & \ddots & & \\
T_{K0} & T_{K1} & \cdots & T_{KK} & Z_{K1} & \cdots & Z_{KK} & \\
\hline
Y_{10} & Y_{11} & & & C_{11} & & & \\
\vdots & \vdots & \ddots & & \vdots & \ddots & & \\
\hline
Y_{K0} & Y_{K1} & \cdots & Y_{KK} & C_{K1} & \cdots & C_{KK} & \\
\hline
Y_{K+10} & Y_{K+11} & \cdots & Y_{K+1K} & C_{K+11} & \cdots & C_{K+1K} & C_{K+1K+1}
\end{array}\right)
\end{array},
$$

where the order of A is given by $n = K + 1_{n_{T_{00}>1}} + 1_{n_{C_{K+1K+1}>1}}$ and its number of diagonal blocks can be written as $nb = n_{T_{00}} + n_{C_{K+1K+1}} + K$. Ipsen et al. showed that lumping dangling nodes into a single node increases the efficiency of power method. The efficiency increases as the number of dangling nodes increases [18]. Moving from this point, diagonal blocks of order one with zero off-diagonal are grouped in T_{00} and diagonal elements with some off-diagonal elements are grouped in C_{K+1K+1}. A remark must be made at this point about the triangularity of T_{ii}. If Tarjan's algorithm is applied to obtain a block-lower triangular matrix, then T_{ii} should also permuted to lower-triangular form, and one state diagonal blocks must be checked whether they have zero off-diagonal elements in rows. If upper-triangular T_{ii} are aimed, then one state diagonal blocks must be checked for zero off-diagonal elements in columns. This remark is valid for partitioning 1 and partitioning 2 which we discuss next. An option for partitioning 1 is restricting the number of diagonal blocks so that we have a (2×2) block partition in which the first diagonal block is triangular.

4.2.2 Partitioning 2

Partitioning 2 is somewhat simpler than partitioning 1. The diagonal blocks of order one are handled as in partitioning 1. The ordering of remaining diagonal blocks is not changed except for those of order two and C_{ii}. For blocks of order two, the first state is moved to the first block, which is triangular, and the second state is moved to the last block. While generating the overall permutation vector, consecutive processing of diagonal blocks is essential to ensure the triangularity of the first block. After symmetrically permuting A with the permutation obtained in this manner, the nonzero structure of the matrix becomes

$$
\begin{array}{c}
\\
n_{T_{00}} \\
n_{C_{11}} \\
n_{T_{11}} \\
n_{C_{22}} \\
n_{T_{22}} \\
\vdots \\
n_{C_{KK}} \\
n_{T_{KK}} \\
n_{C_{K+1K+1}}
\end{array}
\begin{array}{c}
\begin{array}{ccccccccc}
n_{T_{00}} & n_{C_{11}} & n_{T_{11}} & n_{C_{22}} & n_{T_{22}} & \cdots & n_{C_{KK}} & n_{T_{KK}} & n_{C_{K+1K+1}}
\end{array} \\
\left(\begin{array}{ccccccccc}
T_{00} & & & & & & & & \\
\hline
Y_{10} & C_{11} & Y_{11} & C_{12} & Y_{12} & \cdots & C_{1K} & Y_{1K} & C_{1K+1} \\
T_{10} & Z_{11} & T_{11} & Z_{12} & T_{12} & \cdots & Z_{1K} & T_{1K} & Z_{1K+1} \\
\hline
Y_{20} & C_{21} & Y_{21} & C_{22} & Y_{22} & \cdots & C_{2K} & Y_{2K} & C_{2K+1} \\
T_{20} & Z_{21} & T_{21} & Z_{22} & T_{22} & \cdots & Z_{2K} & T_{2K} & Z_{2K+1} \\
\hline
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
\hline
Y_{K0} & C_{K1} & Y_{K1} & C_{K2} & Y_{K2} & \cdots & C_{KK} & Y_{KK} & C_{KK+1} \\
T_{K0} & Z_{K1} & T_{K1} & Z_{K2} & T_{K2} & \cdots & Z_{KK} & T_{KK} & Z_{KK+1} \\
\hline
Y_{K+10} & C_{K+11} & Y_{K+11} & C_{K+12} & Y_{K+12} & \cdots & C_{K+1K} & Y_{K+1K} & C_{K+1K+1}
\end{array}\right)
\end{array},
$$

where $n_{\mathcal{T}} = \sum_{i=1}^{K} n_{T_{ii}}$ and $n_{\mathcal{C}} = \sum_{i=1}^{K+1} n_{C_{ii}}$.

We remark that partitionings 1 and 2 do not guarantee that the overall permutation yields diagonal blocks that are all triangular. The next subsection shows how one can obtain such an overall permutation.

4.2.3 Partitioning 3

This partitioning replaces the first four steps of Algorithm 1 with a recursive procedure. The matrix considered at the particular recursive call (which is the matrix A in the initial call) is block triangularized as in step 1 of Algorithm 1 so that it has irreducible diagonal blocks. Then cutsets are computed on its irreducible diagonal blocks as in step 2 of Algorithm 1 and a permutation is formed in which the cutset's complements' are triangularized and grouped together into a larger triangular block as in steps 3 and 4 of Algorithm 1. Hence, one obtains a (2×2) block partitioning in which one of the diagonal blocks is triangular and the other diagonal block can be perceived just like the input matrix to this recursive call, thereby generating a recursive call. The recursion ends when this other block is of order 0 or 1.

The implementation of partitioning 3 uses the same data structures and routines as in partitionings 1 and 2 except that it requires three additional work arrays having lengths of nz, $(n+1)$, and n. The first two of these are used for storing the nonzero pattern of the submatrix to be considered at that recursive call and the last one to record the permutation in between recursive calls.

We symmetrically permute the matrix according to the partitionings 1, 2, or 3. The matrix Q^T is scaled to have a maximum value of 1.0 in each row to prevent overflow or underflow in the computations and then transformed to block form. In step 5, the steady-state vector is computed by using a block iterative method. The triangular diagonal blocks are solved exactly at each outer iteration with the help of the S-M formula. The remaining diagonal blocks are solved approximately using a point iterative method.

Next we present an example through which we illustrate the computation of partitionings 1, 2, and 3.

4.3 An Illustrative Example

Let us have a probability matrix representing the hyperlink structure of six web pages. We only consider the nonzero structure of the matrix, rather than its nonzero values. Let X represent a nonzero value and the probability matrix P in our example be as follows:

$$P = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} & \mathrm{X} & \mathrm{X} & & & \\ & & & & & \\ \mathrm{X} & \mathrm{X} & & & \mathrm{X} & \\ & & & & \mathrm{X} & \mathrm{X} \\ & & & \mathrm{X} & & \mathrm{X} \\ & & & \mathrm{X} & & \end{pmatrix} \end{array}.$$

Note that state 2 represents a dangling node. Recall that $A = \alpha P^T - I$. However, scaling with α does not change the nonzero structure of P and A becomes,

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} \mathrm{X} & \mathrm{X} & \mathrm{X} & & & \\ & \mathrm{X} & & & & \\ \mathrm{X} & \mathrm{X} & \mathrm{X} & & & \\ & & & \mathrm{X} & \mathrm{X} & \mathrm{X} \\ & & \mathrm{X} & \mathrm{X} & \mathrm{X} & \\ & & & \mathrm{X} & \mathrm{X} & \mathrm{X} \end{pmatrix} \end{array}.$$

Since A is a reducible matrix, the next step is to use Tarjan's algorithm. After applying Tarjan's algorithm to A to obtain an upper-triangular block structure, we obtain the permutation vector (4 5 6 2 1 3). If we symmetrically permute A accordingly, we get

$$\begin{array}{c} \\ 4 \\ 5 \\ 6 \\ 2 \\ 1 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccccc} 4 & 5 & 6 & 2 & 1 & 3 \end{array} \\ \left(\begin{array}{ccc|c|cc} \mathrm{X} & \mathrm{X} & \mathrm{X} & & & \\ \mathrm{X} & \mathrm{X} & & & & \mathrm{X} \\ \mathrm{X} & \mathrm{X} & \mathrm{X} & & & \\ \hline & & & \mathrm{X} & \mathrm{X} & \mathrm{X} \\ \hline & & & & \mathrm{X} & \mathrm{X} \\ & & & & \mathrm{X} & \mathrm{X} \end{array}\right) \end{array}.$$

The irreducible diagonal blocks are given by the subset of states $\{4, 5, 6\}$, $\{2\}$, and $\{1, 3\}$. Note that there is one irreducible diagonal block of order two and one irreducible diagonal block of order 1.

Now, let us consider partitioning 1. Since state 2 has zero off-diagonal elements in its column, it will be at the beginning of the permutation. After applying the Cutfind algorithm to the first irreducible diagonal block, we obtain its cutset

as $\mathcal{C}_1 = \{4\}$ and therefore the cutset's complement as $\mathcal{T}_1 = \{5, 6\}$. Upper triangularization of the submatrix induced by $\mathcal{T}_1$ using Tarjan's algorithm yields the permutation vector (6 5). For the last irreducible diagonal block, state 1 is an element of the cutset and state 3, being in the cutset's complement, is placed in the first triangular block. So the permutation vector becomes, (2 6 5 3 1 4) and the order of the triangular block is four. The permuted matrix has the following nonzero structure

$$\begin{array}{c} \\ 2 \\ 6 \\ 5 \\ 3 \\ 1 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccccc} 2 & 6 & 5 & 3 & 1 & 4 \end{array} \\ \left(\begin{array}{cccc|c|c} \mathrm{X} & & & \mathrm{X} & \mathrm{X} & \\ & \mathrm{X} & \mathrm{X} & & & \mathrm{X} \\ & & \mathrm{X} & \mathrm{X} & & \mathrm{X} \\ & & & \mathrm{X} & \mathrm{X} & \\ \hline & & & \mathrm{X} & \mathrm{X} & \\ \hline & \mathrm{X} & \mathrm{X} & & & \mathrm{X} \end{array}\right) \end{array}.$$

Now, we have three diagonal blocks, the first one is upper-triangular and of order four, and the others are both of order one.

Restricting the number of diagonal blocks so that we have a 2×2 block partitioning in which the first diagonal block is triangular, we obtain

$$\begin{array}{c} \\ 2 \\ 6 \\ 5 \\ 3 \\ 1 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccccc} 2 & 6 & 5 & 3 & 1 & 4 \end{array} \\ \left(\begin{array}{cccc|c|c} \mathrm{X} & & & \mathrm{X} & \mathrm{X} & \\ & \mathrm{X} & \mathrm{X} & & & \mathrm{X} \\ & & \mathrm{X} & \mathrm{X} & & \mathrm{X} \\ & & & \mathrm{X} & \mathrm{X} & \\ \hline & & & \mathrm{X} & \mathrm{X} & \\ & \mathrm{X} & \mathrm{X} & & & \mathrm{X} \end{array}\right) \end{array}.$$

Now, let us consider partitioning 2. We again place state 2 in the first diagonal block. For the irreducible diagonal block of order two, the first state is placed in the first block and the second one is placed in the last block. Since the result of the Cutfind algorithm on the diagonal block of order three is the same, state 4 is placed in the cutset and states 5 and 6 are permuted as (6 5) to obtain an upper-triangular block. The permutation vector becomes (2 1 4 6 5 3) and we have the four diagonal blocks given in

$$
\begin{array}{c} \\ 2 \\ 1 \\ 4 \\ 6 \\ 5 \\ 3 \end{array}
\begin{array}{c}
\begin{array}{cccccc} 2 & 1 & 4 & 6 & 5 & 3 \end{array} \\
\left(\begin{array}{cc|c|cc|c}
\mathrm{X} & \mathrm{X} & & & & \mathrm{X} \\
 & \mathrm{X} & & & & \mathrm{X} \\
\hline
 & & \mathrm{X} & \mathrm{X} & \mathrm{X} & \\
\hline
 & & \mathrm{X} & \mathrm{X} & \mathrm{X} & \\
 & & \mathrm{X} & & \mathrm{X} & \mathrm{X} \\
\hline
 & \mathrm{X} & & & & \mathrm{X}
\end{array}\right)
\end{array}.
$$

Note that the first and the third blocks are triangular.

For partitioning 3, we again apply Tarjan's algorithm to A and obtain the permutation vector (4 5 6 2 1 3). After symmetrically permuting A accordingly and applying the Cutfind algorithm to the first irreducible diagonal block, we obtain its cutset as $\mathcal{C}_1 = \{4\}$, and therefore the cutset's complement as $\mathcal{T}_1 = \{5, 6\}$. Upper-triangularization of the submatrix induced by $\mathcal{T}_1$ using Tarjan's algorithm yields the permutation vector (6 5 4 2 1 3). All remaining diagonal blocks are of order 1 or 2. Therefore, $\{2, 1\}$ are moved to cutset's complement. Restricting the number of diagonal blocks so that we have (2×2) block partitioning, we obtain

$$
\begin{array}{c}
 \\
6 \\
5 \\
2 \\
1 \\
4 \\
3
\end{array}
\begin{array}{c}
\begin{array}{cccccc} 6 & 5 & 2 & 1 & 4 & 3 \end{array} \\
\left(\begin{array}{cccc|cc}
\mathrm{X} & \mathrm{X} & & & \mathrm{X} & \\
 & \mathrm{X} & & & \mathrm{X} & \mathrm{X} \\
 & & \mathrm{X} & \mathrm{X} & & \mathrm{X} \\
 & & & \mathrm{X} & & \mathrm{X} \\
\hline
\mathrm{X} & \mathrm{X} & & & \mathrm{X} & \\
 & & & \mathrm{X} & & \mathrm{X}
\end{array}\right)
\end{array}.
$$

Applying Rosen's algorithm to the second diagonal block, we obtain the cutset as empty set, therefore, cutset's complement becomes $\mathcal{T}_2 = \{4, 3\}$. Note that both diagonal blocks are triangular.

In the next section, we present the results of numerical experiments with Google-like stochastic matrices and other irreducible MCs from the literature.

Chapter 5

Experimental Results

In the previous chapter, we presented three block partitionings to be used with iterative methods in the computation of the steady-state vector of a Google-like stochastic matrix. In this chapter, after discussing the properties of benchmark matrices used in experiments and the experimental framework, we provide the results of numerical experiments and compare the performance of solvers in the software tool with an emphasis on number of iterations taken, accuracy achieved, time for preprocessing and solution, and space used.

5.1 Properties of Benchmark Matrices

Five web matrices are employed in the experiments. Two of these come from the University of Florida Sparse Matrix Collection [8] and referred to as *Stanford* and *StanfordBerkeley*. Two are crawled by UbiCrawler [2] and uncompressed using WebGraph [3] software; they are named *Eu2005* and *In2004*. The fifth one is

Matrix	n	nz	# of dangling nodes	# of non-existent diagonal elements
Stanford	281,903	2,312,497	20,315	281,903
StanfordBerkeley	683,446	7,583,376	68,062	683,446
Eu2005	862,664	19,235,140	71,675	361,237
In2004	1,382,908	16,917,053	86	1,005,498
Webbase	2,662,002	44,843,770	574,863	2,662,002
2D	16,641	66,049	0	0
Easy	20,301	140,504	0	0
Telecom	20,491	101,041	0	0
Ncd	23,426	156,026	0	0
Mutex	39,203	563,491	0	0
Qnatm	104,625	593,115	0	0

Table 5.1: Properties of benchmark matrices.

named *Webbase* and is obtained from the Stanford Webbase Project [4]. All of these matrices are reducible, have orders ranging between 281,903 and 2,662,02, and possess dangling nodes.

Although the work in this thesis is geared towards reducible matrices, we have also experimented with irreducible matrices. The matrices we considered are previously used, for instance, in [11], and referred to as *2D*, *Easy*, *Telecom*, *Ncd*, *Mutex*, *Qnatm*. These matrices come from six different applications, such as biological modeling (*2D*), computer performance evaluation (*Ncd*, *Mutex*), and telecommunication networks (*Easy*, *Telecom*, *Qnatm*). The orders of these matrices range between 16,000 and 105,000, and they do not possess any zero rows.

Detailed information regarding the reducible and irreducible matrices used in experiments appears in Table 5.1. The first column provides the matrix name. Columns n and nz give the order of the matrix and its number of nonzeros. Columns four and five provide the number of dangling nodes and number of nonexistent diagonal elements in the matrices. Results of numerical experiments on these matrices follow in the next sections.

5.2 Experimental Setup

Experiments are performed on a 3 GHz Pentium IV processor with 2 Gigabytes main memory under the Linux operating system using the o3 level optimization in compiling the code [10]. There are currently eight solvers in the code: POWER, quadratically extrapolated (QPOWER), J, GS, BJ, BGS, IADBJ, and IADBGS. Of these, the last four are able to utilize block partitionings.

Two other straightforward block partitioning techniques are considered. The *equal* partitioning forms (approximately) equal order blocks and the second partitioning, *other*, uses blocks of order respectively $1, 2, 3, \ldots$ [11, p.1692]. The *other* partitioning has about $\sqrt{2n}$ blocks with the largest block of order roughly $\sqrt{2n}$. "The *equal* partitioning has $\sqrt{n}$ blocks of order $\sqrt{n}$ if n is a perfect square. If $n \neq \lfloor\sqrt{n}\rfloor^2$, there is an extra block of order $n - \lfloor\sqrt{n}\rfloor^2$." [11, p.1693]. In the rest of this thesis we name the former of these partitioning 4 and the latter partitioning 5.

The partitioning parameters of experiments performed using block partitionings can be expressed as a Cartesian product of four sets. Let set $\mathcal{B} = \{\mathrm{y}, \mathrm{n}\}$ denote whether Tarjan's algorithm is used or not, set $\mathcal{C} = \{\mathrm{y}, \mathrm{n}\}$ denote whether Rosen's algorithm is used or not, set $\mathcal{R} = \{\mathrm{y}, \mathrm{n}\}$ denote whether the number of diagonal blocks is restricted to two or not, and set $\mathcal{O} = \{\mathrm{u}, \mathrm{l}\}$ denote whether triangularized diagonal blocks are upper- or lower-triangular. Then experiments with block partitionings take as partitioning parameters elements from proper subsets of $\mathcal{B} \times \mathcal{C} \times \mathcal{R} \times \mathcal{O}$. Experiments performed on web matrices using partitionings 1 and 2 can utilize elements of $\{\mathrm{y}\} \times \mathcal{C} \times \mathcal{R} \times \mathcal{O}$, those using partitioning 3 can utilize $\{\mathrm{y}\} \times \{\mathrm{y}\} \times \{\mathrm{n}\} \times \mathcal{O}$, and those using partitionings 4 and 5 can utilize

$\{\mathrm{n}\} \times \{\mathrm{n}\} \times \{\mathrm{n}\} \times \{\mathrm{u}\}$. On the other hand, experiments performed on irreducible matrices using partitionings 1 and 2 can utilize elements of $\{\mathrm{n}\} \times \{\mathrm{y}\} \times \mathcal{R} \times \mathcal{O}$ and those using partitioning 3 can utilize elements of $\{\mathrm{n}\} \times \{\mathrm{y}\} \times \{\mathrm{n}\} \times \mathcal{O}$, and those using partitionings 4 and 5 can utilize the corresponding ones for web matrices. Note that for experiments performed on web matrices using IAD solvers based on partitioning 1 elements of $\{\mathrm{y}\} \times \{\mathrm{n}\} \times \mathcal{R} \times \mathcal{O}$ are utilized and this set of experiments are not performed on irreducible matrices.

Different problems resulting from web matrices are considered by letting $\alpha \in \{0.85, 0.90, 0.95, 0.97, 0.99\}$. The value of α for irreducible matrices is 1.0. Hence, there are altogether 31 test problems.

5.3 Performance Evaluation

Experimental results are presented in the following two subsections respectively for the 25 reducible matrices and 6 irreducible matrices after data pertaining to the resulting nonzero structures under the assumed block partitionings are given at the beginning of each subsection. The results are discussed in the third subsection.

5.3.1 Experiments with Reducible Matrices

Data pertaining to the resulting nonzero structures under the assumed block partitionings are provided in Tables 5.3–5.10. The number and the location of nonzeros in these sparse matrices have a significant effect on the number of iterations performed and solution time. Note however that the nonzero structure of

a web matrix does not change for different values of α. Two tables are presented for each problem; thus, we have 10 tables.

The first of the two tables provides summary information about the nonzero structure of the block partitionings for the five partitionings used. Again the first column (Part) indicates the partitioning used and the second column (Para) lists the parameters of the particular partitioning. Number of diagonal blocks, maximum, minimum, and average order of a diagonal block in the particular partitioning appear in columns three (nb), four (Max_n), five (Min_n), and six (Ave_n). Maximum, minimum, and average number of nonzeros of a diagonal block in the particular partitioning appear in columns seven (Max_{nz}), eight (Min_{nz}), and nine (Ave_{nz}). Column ten (nz_{off}) provides the number of nonzeros in off-diagonal blocks. Note that, for partitioning 3, partitioning 4, and partitioning 5 nonzero structure is given only for orientation parameter u, since given information on nonzero structure is the same for orientation parameter l.

The second of the two tables for each problem gives detailed information about the nonzero structure of the block partitionings when partitionings 1 and 2 are used. The first column (Part) indicates the partitioning used and the second column (Para) lists the parameters of the particular partitioning. Columns three ($n_{T_{00}}$) and four ($nz_{T_{00}}$) correspond to the order and the number of nonzeros of the first diagonal block in the employed partitioning, respectively. Length of the final cutset is given in column five (n_T). Column six indicates the total number of nonzeros in diagonal blocks, except the first diagonal block for partitioning 1, and except the first and last diagonal blocks for partitioning 2.

The best timing results for each problem with each of the available solvers (under the five block partitionings wheneever applicable), is presented in Tables

Part	Para	nb	Max_n	Min_n	Ave_n	Max_{nz}	Min_{nz}	Ave_{nz}	nz_{off}
1	y,y,y,l	2	122,866	122,866	140,951.5	848,844	452,823	650,833.5	1,292,733
1	y,y,y,u	2	102,723	102,723	140,951.5	699,421	517,281	608,351.0	1,377,698
1	y,y,n,l	3,520	50,370	1	34.9	494,987	1	331.9	1,425,977
1	y,y,n,u	3,520	50,370	1	29.2	517,281	1	338.9	1,401,385
1	y,n,n,l	3,520	150,532	2	80.1	1,726,846	4	673.2	224,891
1	y,n,n,u	3,520	150,532	2	74.3	1,726,846	4	667.6	244,614
2	y,y,n,l	4,776	100,162	1	53.1	494,987	1	241.4	1,441,346
2	y,y,n,u	4,776	100,162	1	53.1	494,987	1	237.3	1,461,074
2	y,n,n,l	3,520	150,532	1	80.1	1,726,846	1	672.6	226,848
2	y,n,n,u	3,520	150,532	1	74.3	1,726,846	1	667.0	246,646
3	y,y,n,u	42	185,332	1	6,712.0	555,177	1	16,844.8	1,886,918
4	n,n,n,u	531	1,003	530	530.9	1,036	530	538.2	2,308,602
5	n,n,n,u	751	750	1	375.4	775	1	380.4	2,308,739

Table 5.2: Nonzero structure of the *Stanford* problem for all partitionings.

Part	Para	$n_{T_{00}}$	$nz_{T_{00}}$	n_T	nz_{rem}
1	y,y,y,l	159,037	452,823	122,866	848,844
1	y,y,y,u	179,180	517,281	102,723	699,421
1	y,y,n,l	159,037	452,823	122,866	715,600
1	y,y,n,u	179,180	517,281	102,723	675,734
1	y,n,n,l	172	172		2,369,337
1	y,n,n,u	20,315	20,315		2,329,471
2	y,y,n,l	1,303	1,356	122,866	1,151,698
2	y,y,n,u	21,446	21,569	102,723	1,111,757
2	y,n,n,l	172	172		2,367,380
2	y,n,n,u	20,315	20,315		2,327,439

Table 5.3: Other information on nonzero structure of the *Stanford* problem with partitionings 1 and 2.

Part	Para	nb	Max_n	Min_n	Ave_n	Max_{nz}	Min_{nz}	Ave_{nz}	nz_{off}
1	y,y,y,l	2	322,658	322,658	341,723.0	3,683,242	1,013,364	2,348,303.0	3,570,216
1	y,y,y,u	2	259,331	259,331	341,723.0	2,754,582	1,195,425	1,975,003.5	4,316,815
1	y,y,n,l	6,750	106,595	1	47.8	1,665,952	1	601.3	4,208,017
1	y,y,n,u	6,750	106,595	1	38.4	1,665,952	1	553.0	4,534,222
1	y,n,n,l	6,750	333,752	2	100.6	4,843,536	4	1,087.4	926,824
1	y,n,n,u	6,750	333,752	2	91.2	4,843,536	4	1,021.5	1,371,763
2	y,y,n,l	10,116	227,157	1	57.2	1,665,952	1	398.1	4,239,810
2	y,y,n,u	10,116	227,157	1	57.2	1,665,952	1	354.1	4,685,016
2	y,n,n,l	6,750	333,752	1	100.6	4,843,536	1	1,087.2	928,567
2	y,n,n,u	6,750	333,752	1	91.2	4,843,536	1	1,021.2	1,373,832
3	y,y,n,u	163	1,170	826	826.4	1,719,076	2	12,946.2	6,156,597
4	n,n,n,u	827	1,168	1	584.6	50,507	826	4,931.4	4,188,520
5	n,n,n,u	1,169	458,614	2	4,192.9	59,110	1	3,464.8	4,216,462

Table 5.4: Nonzero structure of the *StanfordBerkeley* problem for all partitionings.

Part	Para	$n_{T_{00}}$	$nz_{T_{00}}$	n_T	nz_{rem}
1	y,y,y,l	360,788	1,013,364	322,658	3,683,242
1	y,y,y,u	424,115	1,195,425	259,331	2,754,582
1	y,y,n,l	360,788	1,013,364	322,658	3,045,441
1	y,y,n,u	424,115	1,195,425	259,331	2,537,175
1	y,n,n,l	4,735	4,735		7,335,263
1	y,n,n,u	68,062	68,062		6,826,997
2	y,y,n,l	6,426	6,484	322,658	4,020,528
2	y,y,n,u	69,753	69,870	259,331	3,511,936
2	y,n,n,l	4,735	4,735		7,333,520
2	y,n,n,u	68,062	68,062		6,824,928

Table 5.5: Other information on nonzero structure of the *StanfordBerkeley* problem with partitionings 1 and 2.

Part	Para	nb	Max_n	Min_n	Ave_n	Max_{nz}	Min_{nz}	Ave_{nz}	nz_{off}
1	y,y,y,l	2	397,231	397,231	198,615.5	9,882,355	1,354,607	5,618,481.0	1,158,774
1	y,y,y,u	2	315,790	315,790	157,895.0	9,157,873	1,875,258	5,516,566.0	1,173,377
1	y,y,n,l	1,163	300,962	1	341.6	9,045,432	1	9,086.9	1,158,774
1	y,y,n,u	1,163	300,962	1	271.5	9,045,432	1	9,452.0	1,173,377
1	y,n,n,l	1,162	89,607	2	94.6	18,201,086	4	15,867.1	1,158,774
1	y,n,n,u	1,163	752,725	2	671.7	18,201,086	4	15,840.9	1,173,377
2	y,y,n,l	1,588	451,763	1	486.4	9,045,432	1	6,654.2	9,029,473
2	y,y,n,u	1,588	451,763	1	486.4	9,045,432	1	6,645.1	9,043,989
2	y,n,n,u	1,163	89,607	2	94.6	18,201,086	1	15,840.6	1,173,734
2	y,n,n,l	1,162	752,725	2	671.7	18,201,086	1	15,867.0	1,158,974
3	y,y,n,u	378	986,091	1	2,917.5	1,942,183	1	6,713.3	17,058,750
4	n,n,n,u	929	1,480	928	928.6	113,337	928	7,829.3	12,322,997
5	n,n,n,u	1,314	1,313	1	656.5	153,043	1	5,423.7	12,469,660

Table 5.6: Nonzero structure of the *Eu2005* problem for all partitionings.

Part	Para	$n_{T_{00}}$	$nz_{T_{00}}$	n_T	nz_{rem}
1	y,y,y,l	0	18,201,086	397,231	236,517
1	y,y,y,u	81,441	81,441	315,790	18,341,559
1	y,y,n,l	0	18,201,086	397,231	236,517
1	y,y,n,u	81,441	81,441	315,790	18,341,559
1	y,n,n,l	0	18,201,086		236,517
1	y,n,n,u	81,441	81,441		18,341,559
2	y,y,n,l	0	368	397,231	10,566,522
2	y,y,n,u	81,441	82,067	315,790	10,470,321
2	y,n,n,u	0	18,201,086		18,341,202
2	y,n,n,l	81,441	81,441		236,317

Table 5.7: Other information on nonzero structure of the *Eu2005* problem with partitionings 1 and 2.

Part	Para	nb	Max_n	Min_n	Ave_n	Max_{nz}	Min_{nz}	Ave_{nz}	nz_{off}
1	y,y,y,l	2	453,130	453,130	691,454.0	8,280,317	2,573,755	5,427,036.0	7,068,479
1	y,y,y,u	2	747,816	747,816	691,454.0	9,070,706	1,745,311	5,408,008.5	7,106,534
1	y,y,n,l	16,644	209,076	1	27.2	3,180,612	1	640.7	7,259,360
1	y,y,n,u	16,644	350,939	1	44.9	3,180,612	1	619.1	7,618,442
1	y,n,n,l	16,644	593,687	2	65.4	8,234,607	4	991.9	1,413,736
1	y,n,n,u	16,644	593,687	2	83.1	8,234,607	4	1,002.4	1,239,060
2	y,y,n,l	26,978	384,611	1	40.2	3,180,612	1	374.9	7,809,139
2	y,y,n,u	26,978	384,611	1	51.1	3,180,612	1	381.4	7,633,801
2	y,n,n,l	16,644	593,687	1	65.4	8,234,607	1	991.9	1,413,651
2	y,n,n,u	16,644	593,687	1	83.1	8,234,607	1	1,002.9	1,230,381
3	y,y,n,u	474	986,091	1	2,917.5	2,965,885	1	7,583.6	14,327,904
4	n,n,n,u	1,176	2,283	1,175	1,175.9	244,197	1,175	9,404.2	6,863,176
5	n,n,n,u	1,663	1,662	1	831.6	397,360	1	6,614.8	6,922,088

Table 5.8: Nonzero structure of the *In2004* problem for all partitionings.

Part	Para	$n_{T_{00}}$	$nz_{T_{00}}$	n_T	nz_{rem}
1	y,y,y,l	929,778	2,573,755	453,130	8,280,317
1	y,y,y,u	635,092	1,745,311	747,816	9,070,706
1	y,y,n,l	929,778	2,573,755	453,130	8,089,436
1	y,y,n,u	635,092	1,745,311	747,816	8,558,798
1	y,n,n,l	294,780	294,780		16,214,035
1	y,n,n,u	94	94		16,683,397
2	y,y,n,l	297,934	305,915	453,130	9,807,497
2	y,y,n,u	3,248	3,297	747,816	10,285,453
2	y,n,n,l	294,780	294,780		16,214,120
2	y,n,n,u	94	94		16,692,076

Table 5.9: Other information on nonzero structure of the *In2004* problem with partitionings 1 and 2.

Part	Para	nb	Max_n	Min_n	Ave_n	Max_{nz}	Min_{nz}	Ave_{nz}	nz_{off}
1	y,y,y,l	2	1,026,409	1,026,409	513,204.5	13,999,537	5,085,941	9,542,739.0	28,420,294
1	y,y,y,u	2	807,006	807,006	403,503.0	17,107,790	6,049,826	11,578,808.0	24,348,156
1	y,y,n,l	8,584	590,095	1	119.6	8,888,544	1	2,030.1	30,079,285
1	y,y,n,u	8,584	370,692	1	94.0	8,888,544	1	2,103.9	29,445,974
1	y,n,n,l	8,584	1,390,621	2	268.7	31,714,016	4	4,541.2	8,523,737
1	y,n,n,u	8,584	1,390,621	2	243.1	31,714,016	4	4,528.3	8,634,908
2	y,y,n,l	10,188	1,054,569	1	167.8	8,888,544	1	1,654.8	30,647,129
2	y,y,n,u	10,188	1,054,569	1	167.8	8,888,544	1	1,644.1	30,755,389
2	y,n,n,l	8,584	1,390,621	2	268.7	31,714,016	1	4,541.1	8,524,636
2	y,n,n,u	8,584	1,390,621	2	243.1	31,714,016	1	4,528.1	8,636,363
3	y,y,n,u	404	2,225,698	1	6,589.1	8,197,002	1	21,961.9	38,633,180
4	n,n,n,u	1,632	1,841	1,631	1,631.1	401,126	1,631	11,113.7	29,368,144
5	n,n,n,u	2,307	2,306	1	1,153.9	347,195	1	7,675.2	29,799,021

Table 5.10: Nonzero structure of the *Webbase* problem for all partitionings.

Part	Para	$n_{T_{00}}$	$nz_{T_{00}}$	n_T	nz_{rem}
1	y,y,y,l	355,460	358,949	1,026,409	13,999,537
1	y,y,y,u	574,863	578,352	807,006	17,107,790
1	y,y,n,l	355,460	358,949	1,026,409	12,340,546
1	y,y,n,u	574,863	578,352	807,006	12,009,972
1	y,n,n,l	355,460	358,949		38,626,575
1	y,n,n,u	574,863	578,352		38,296,001
2	y,y,n,l	355,460	358,949	1,026,409	16,496,313
2	y,y,n,u	574,863	578,352	807,006	16,165,183
2	y,n,n,l	355,460	358,949		38,625,676
2	y,n,n,u	574,863	578,352		38,294,546

Table 5.11: Other information on nonzero structure of the *Webbase* problem with partitionings 1 and 2.

5.12–5.36. Bold numbers in the tables show the best timing result for each problem. The stopping criteria used in POWER, QPOWER, JOR, and SOR are

$$\textbf{stop if } k \geq \ maxit \textbf{ or } \left\|x^{(k)} - x^{(k-1)}\right\|_\infty \leq stoptol$$

and in BJOR, BSOR, IADBJOR, and IADBSOR are

$$\textbf{stop if } k \geq \ maxit \textbf{ or } \left\|x^{(k)} - x^{(k-1)}\right\|_\infty \leq stoptol$$

and

$$\textbf{stop if } k \geq \ maxit \quad \textbf{or } \left\|x^{(k)} - x^{(k-1)}\right\|_\infty \leq stoptol \textbf{ or}$$

$$\left(\left\|x^{(k)} - x^{(k-1)}\right\|_\infty \leq stoptol_1 \ \textbf{\textit{and}}\right.$$

$$\left.\left\|x^{(k)} - x^{(k-1)}\right\|_\infty - \left\|x^{(k-1)} - x^{(k-2)}\right\|_\infty \leq \ stoptol_2,\right.$$

where k is the iteration number, *maxit* is the maximum number of iterations to

be performed, and *stoptol* is the stopping tolerance. "The use of $stoptol_1$ and $stoptol_2$ forces the solver to terminate when the norm residual is decreasing too slowly, while the differences between two successive iterates is small enough." [11, p.1697]. We set *stoptol*, $stoptol_1$ and $stoptol_2$ to $1e-10$, $1e-6$ and $1e-12$ respectively, whereas *maxit* is set to 1,000. BJOR, BSOR, IADBJOR, and IADBSOR have two additional parameters indicating inner tolerance and maximum number of inner iterations to be performed when using the corresponding point iterative method for the solution of diagonal blocks. The values of these parameters are respectively elements of the sets $\{10^{-3}, 10^{-5}, 10^{-10}\}$ and $\{3, 5\}$. In the IAD solvers, if the order of the aggregated matrix is larger than 500, the corresponding point iterative method is employed, otherwise the GTH method [37, pp.84–86] (a more robust version of GE for MCs) is used for its solution. If the corresponding point iterative method is used for the solution of the aggregated matrix, the two parameters take the values 10^{-15} and 100,000. The relaxation parameter ω is set to 1.0 for all solvers but QPOWER and POWER; thus, we have the solvers (B)J, (B)GS, IADBJ, and IADBGS.

Each table consists of thirty rows and nine columns except those of the *Webbase* problem in which space limitations are exceeded with the IAD solvers based on partitioning 3. The first two columns (Solver and Part) indicate the solver name and the block partitioning used. Column three (Iter) and four (Res) give the number of iterations performed and the infinity norm of the residual vector upon stopping. Preprocessing, Pe, solution, and total times in seconds appear in columns five (Prep), six (Pe), seven (Solu), and eight (Total), respectively. The preprocessing time includes time for allocating and setting the necessary data structures, and whereever applicable, scaling the coefficient matrix, computing the block partitioning and transforming the sparse point representation of the

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		103	8e−11	2.3		20.2	22.5	38
QPOWER		86	9e−11	2.4		17.5	19.8	53
J		104	7e−11	2.6		23.0	25.5	50
GS		45	6e−11	2.5		9.6	12.1	50
BJ(n,n,n,u,1e−3,3)	4	103	8e−11	3.3		22.2	25.5	86
BJ(n,n,n,u,1e−3,3)	5	103	8e−11	3.2		21.4	24.6	86
BJ(y,n,n,l,1e−5,3)	1	34	2e−11	4.6		13.8	18.3	86
BJ(y,n,y,l,1e−10,3)	1	34	2e−11	4.5		13.8	18.3	86
BJ(y,y,n,l,1e−10,3)	1	153	8e−11	5.4		23.7	29.0	108
BJ(y,y,y,l,1e−5,3)	1	277	9e−11	5.4		45.1	50.5	108
BJ(y,n,n,u,1e−5,3)	2	67	8e−11	4.6		24.8	29.5	86
BJ(y,y,n,l,1e−10,3)	2	122	8e−11	5.4		18.0	23.4	110
BJ(y,y,n,l)	3	169	8e−11	5.7		13.8	19.5	122
BGS(n,n,n,u,1e−3,5)	4	45	6e−11	3.3		10.0	13.3	86
BGS(n,n,n,u,1e−3,3)	5	45	6e−11	3.3		9.9	13.2	86
BGS(y,n,n,l,1e−5,3)	1	17	2e−11	4.6		7.1	11.6	86
BGS(y,n,y,l,1e−5,3)	1	17	2e−11	4.6		7.0	11.6	86
BGS(y,y,n,l,1e−3,3)	1	43	1e−11	5.3		7.0	12.3	108
BGS(y,y,y,l,1e−5,3)	1	43	1e−11	5.4		7.3	12.7	108
BGS(y,n,n,l,1e−3,3)	2	35	5e−11	4.6		14.0	18.6	86
BGS(y,y,n,l,1e−10,3)	2	42	6e−11	5.4		6.7	12.0	110
BGS(y,y,n,l)	3	44	5e−11	5.7		4.1	**9.8**	122
IADBJ(n,n,n,u,1e−3,5)	4	86	8e−11	3.4	11.4	41.5	56.3	149
IADBJ(n,n,n,u,1e−3,3)	5	85	7e−11	3.4	10.7	40.5	54.6	149
IADBJ(y,n,n,u,1e−10,5)	1	22	8e−11	4.9	3.1	12.5	20.5	149
IADBJ(y,y,n,u)	3	93	8e−11	5.6	182.1	11.7	199.5	151
IADBGS(n,n,n,u,1e−3,5)	4	43	6e−11	3.3	11.4	19.1	33.8	149
IADBGS(n,n,n,u,1e−3,3)	5	43	6e−11	3.4	10.7	20.5	34.6	149
IADBGS(y,n,n,l,1e−10,5)	1	11	4e−11	4.7	2.5	6.7	14.0	149
IADBGS(y,y,n,u)	3	44	4e−11	5.5	184.5	6.1	196.1	151

Table 5.12: Solver statistics for the *Stanford* problem when $\alpha = 0.85$.

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		154	8e−11	2.4		29.0	31.4	38
QPOWER		112	1e−10	2.6		23.8	26.4	53
J		154	8e−11	2.5		32.8	35.2	50
GS		65	8e−11	2.7		14.0	16.6	50
BJ(n,n,n,u,1e−5,5)	4	154	8e−11	3.3		32.9	36.1	86
BJ(n,n,n,u,1e−3,3)	5	154	8e−11	3.2		32.8	36.0	86
BJ(y,n,n,l,1e−3,3)	1	49	5e−11	4.6		19.7	24.3	86
BJ(y,n,y,u,1e−5,3)	1	49	5e−11	4.8		19.5	24.4	86
BJ(y,y,n,l,1e−3,3)	1	210	9e−11	5.3		31.7	37.0	108
BJ(y,y,y,l,1e−3,3)	1	338	9e−11	5.4		55.0	60.4	108
BJ(y,n,n,u,1e−3,3)	2	103	9e−11	4.8		39.3	44.1	86
BJ(y,y,n,l,1e−5,3)	2	176	8e−11	5.3		25.8	31.1	110
BJ(y,y,n,l)	3	230	9e−11	5.6		17.8	23.4	122
BGS(n,n,n,u,1e−5,5)	4	65	8e−11	3.4		14.3	17.7	86
BGS(n,n,n,u,1e−3,3)	5	65	8e−11	3.2		14.2	17.4	86
BGS(y,n,n,l,1e−3,3)	1	25	2e−11	4.6		10.2	14.8	86
BGS(y,n,y,l,1e−3,3)	1	25	2e−11	4.5		10.1	14.6	86
BGS(y,y,n,l,1e−5,3)	1	64	1e−11	5.4		10.2	15.5	108
BGS(y,y,y,l,1e−10,3)	1	64	1e−11	5.5		10.8	16.2	108
BGS(y,n,n,l,1e−3,3)	2	53	8e−11	4.6		21.2	25.7	86
BGS(y,y,n,u,1e−10,3)	2	62	8e−11	5.5		9.5	15.0	110
BGS(y,y,n,l)	3	63	7e−11	5.7		5.7	**11.4**	122
IADBJ(n,n,n,u,1e−5,3)	4	132	8e−11	3.3	11.4	56.5	71.2	149
IADBJ(n,n,n,u,1e−5,5)	5	128	8e−11	3.3	10.7	60.3	74.4	149
IADBJ(y,n,n,u,1e−10,5)	1	33	8e−11	4.7	3.1	19.2	27.0	149
IADBJ(y,y,n,u)	3	142	9e−11	5.6	182.1	17.6	205.3	151
IADBGS(n,n,n,u,1e−5,5)	4	63	6e−11	3.3	11.4	27.8	42.5	149
IADBGS(n,n,n,u,1e−3,5)	5	63	6e−11	3.2	10.8	29.6	43.6	149
IADBGS(y,n,n,l,1e−10,5)	1	15	7e−11	4.7	2.5	9.2	16.4	149
IADBGS(y,y,n,u)	3	65	4e−11	5.6	182.2	8.6	196.4	151

Table 5.13: Solver statistics for the *Stanford* problem when α=0.90.

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		288	9e−11	2.5		54.5	57.0	38
QPOWER		218	2e−10	2.4		43.2	45.6	53
J		288	9e−11	2.5		61.1	63.5	50
GS		124	9e−11	2.5		26.7	29.2	50
BJ(n,n,n,u,1e−3,5)	4	288	9e−11	3.3		61.4	64.7	86
BJ(n,n,n,u,1e−3,3)	5	288	9e−11	3.3		61.0	64.3	86
BJ(y,n,n,l,1e−5,5)	1	57	8e−11	4.6		35.6	40.2	86
BJ(y,n,y,l,1e−3,5)	1	57	8e−11	4.6		35.7	40.2	86
BJ(y,y,n,l,1e−3,3)	1	367	9e−11	5.3		55.4	60.7	108
BJ(y,y,y,l,1e−10,3)	1	512	1e−10	5.4		83.4	88.8	108
BJ(y,n,n,l,1e−3,3)	2	213	9e−11	4.7		80.9	85.6	86
BJ(y,y,n,l,1e−5,3)	2	319	9e−11	5.3		46.3	51.6	110
BJ(y,y,n,l)	3	406	9e−11	5.6		32.7	38.3	122
BGS(n,n,n,u,1e−3,3)	4	124	9e−11	3.2		27.1	30.4	86
BGS(n,n,n,u,1e−3,3)	5	124	9e−11	3.2		27.0	30.3	86
BGS(y,n,n,l,1e−3,5)	1	30	1e−11	4.6		18.9	23.5	86
BGS(y,n,y,l,1e−10,5)	1	30	1e−11	4.5		19.0	23.5	86
BGS(y,y,n,l,1e−5,3)	1	121	2e−11	5.5		18.9	24.4	108
BGS(y,y,y,u,1e−10,3)	1	123	2e−11	5.8		19.4	25.2	108
BGS(y,n,n,l,1e−5,3)	2	108	9e−11	4.5		42.0	46.5	86
BGS(y,y,n,l,1e−5,3)	2	117	9e−11	5.3		17.8	23.1	110
BGS(y,y,n,l)	3	117	9e−11	5.7		10.2	**15.8**	122
IADBJ(n,n,n,u,1e−3,3)	4	267	8e−11	3.4	11.4	117.6	132.4	149
IADBJ(n,n,n,u,1e−5,5)	5	257	9e−11	3.3	10.7	123.1	137.1	149
IADBJ(y,n,n,u,1e−10,5)	1	61	9e−11	4.7	3.1	36.5	44.3	149
IADBJ(y,y,n,u)	3	289	9e−11	5.5	182.5	35.6	223.6	151
IADBGS(n,n,n,u,1e−5,3)	4	117	9e−11	3.4	11.4	51.3	66.2	149
IADBGS(n,n,n,u,1e−3,3)	5	118	7e−11	3.4	10.7	56.2	70.3	149
IADBGS(y,n,n,u,1e−10,5)	1	29	7e−11	4.8	3.1	17.3	25.2	149
IADBGS(y,y,n,u)	3	124	4e−11	5.6	183.5	15.9	205.0	151

Table 5.14: Solver statistics for the *Stanford* problem when α=0.95.

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		475	1e−12	2.5		90.7	93.2	38
QPOWER		394	2e−10	2.3		76.7	79.0	53
J		477	9e−11	2.6		280.6	283.2	50
GS		201	9e−11	2.6		42.5	45.1	50
BJ(n,n,n,u,1e−3,5)	4	475	1e−10	3.3		101.1	104.3	86
BJ(n,n,n,u,1e−3,3)	5	475	1e−10	3.3		101.1	104.3	86
BJ(y,n,n,l,1e−10,5)	1	94	9e−11	4.6		58.5	63.1	86
BJ(y,n,y,l,1e−10,5)	1	94	9e−11	4.6		59.2	63.7	86
BJ(y,y,n,l,1e−3,3)	1	555	9e−11	5.5		83.6	89.1	108
BJ(y,y,y,l,1e−5,3)	1	733	1e−10	5.4		118.8	124.2	108
BJ(y,n,n,u,1e−3,3)	2	360	9e−11	4.7		136.1	140.7	86
BJ(y,y,n,l,1e−10,3)	2	498	9e−11	5.5		71.9	77.4	110
BJ(y,y,n,u)	3	628	9e−11	5.7		50.3	56.0	122
BGS(n,n,n,u,1e−5,3)	4	201	9e−11	3.3		43.8	47.1	86
BGS(n,n,n,u,1e−3,3)	5	201	9e−11	3.3		43.9	47.2	86
BGS(y,n,n,l,1e−3,3)	1	76	3e−11	4.7		29.6	34.3	86
BGS(y,n,y,l,1e−3,3)	1	76	3e−11	4.5		29.6	34.1	86
BGS(y,y,n,u,1e−10,3)	1	198	4e−11	5.5		30.1	35.6	108
BGS(y,y,y,u,1e−10,3)	1	198	4e−11	5.7		31.7	37.4	108
BGS(y,n,n,u,1e−5,3)	2	181	9e−11	4.7		70.1	74.8	86
BGS(y,y,n,l,1e−10,3)	2	192	9e−11	5.3		29.0	34.3	110
BGS(y,y,n,l)	3	191	9e−11	5.6		16.4	**21.9**	122
IADBJ(n,n,n,u,1e−3,5)	4	441	9e−11	3.4	11.4	195.1	210.0	149
IADBJ(n,n,n,u,1e−5,5)	5	425	9e−11	3.4	10.7	204.8	218.9	149
IADBJ(y,n,n,u,1e−10,5)	1	104	1e−10	4.8	3.1	62.3	70.2	149
IADBJ(y,y,n,u)	3	479	1e−10	5.5	180.9	58.2	244.6	151
IADBGS(n,n,n,u,1e−5,5)	4	188	7e−11	3.3	11.4	82.5	97.2	149
IADBGS(n,n,n,u,1e−3,3)	5	189	7e−11	3.2	10.8	88.3	102.3	149
IADBGS(y,n,n,l,1e−10,5)	1	45	9e−11	4.6	2.5	28.0	35.0	149
IADBGS(y,y,n,u)	3	200	5e−11	6.1	180.6	25.6	212.3	151

Table 5.15: Solver statistics for the *Stanford* problem when α=0.97.

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		1,000*	4e−09	2.4		193.1	195.4	38
QPOWER		1,000*	5e−10	2.6		200.4	203.0	53
J		1,000*	4e−09	4.3		220.3	224.5	50
GS		574	6e−11	2.6		121.2	123.8	50
BJ(n,n,n,u,1e−3,3)	4	1,000*	4e−09	3.3		212.7	216.0	86
BJ(n,n,n,u,1e−5,5)	5	1,000*	4e−09	3.3		212.2	215.4	86
BJ(y,n,n,u,1e−3,5)	1	236	3e−10	4.7		150.0	154.7	86
BJ(y,n,y,u,1e−3,5)	1	236	3e−10	4.7		149.9	154.6	86
BJ(y,y,n,u,1e−3,3)	1	1,000*	3e−08	5.5		145.6	151.1	108
BJ(y,y,y,u,1e−5,3)	1	1,000*	1e−07	5.7		150.4	156.1	108
BJ(y,n,n,u,1e−10,3)	2	1,000*	3e−10	4.8		377.4	382.2	86
BJ(y,y,n,l,1e−5,3)	2	1,000*	9e−09	5.3		144.2	149.5	110
BJ(y,y,n,u)	3	1,000*	3e−8	5.6		75.6	81.2	122
BGS(n,n,n,u,1e−3,5)	4	571	1e−10	3.3		123.1	126.4	86
BGS(n,n,n,u,1e−5,5)	5	574	6e−11	3.3		124.6	127.9	86
BGS(y,n,n,l,1e−3,3)	1	208	3e−11	4.5		80.6	85.1	86
BGS(y,n,y,l,1e−5,3)	1	208	3e−11	4.7		80.4	85.1	86
BGS(y,y,n,u,1e−3,3)	1	557	5e−11	5.5		83.7	89.2	108
BGS(y,y,y,u,1e−5,3)	1	555	6e−11	5.8		85.7	91.5	108
BGS(y,n,n,l,1e−5,3)	2	520	1e−10	4.6		201.6	206.1	86
BGS(y,y,n,l,1e−5,3)	2	585	1e−10	5.3		85.8	91.1	110
BGS(y,y,n,u)	3	552	7e−11	5.6		43.7	**49.3**	122
IADBJ(n,n,n,u,1e−3,3)	4	1,000*	9e−11	3.3	11.4	517.7	532.5	149
IADBJ(n,n,n,u,1e−3,5)	5	1,000*	7e−10	3.7	10.7	490.7	505.0	149
IADBJ(y,n,n,u,1e−10,5)	1	322	1e−10	4.7	3.2	172.4	180.3	149
IADBJ(y,y,n,u)	3	1,000*	9e−9	5.3	195.7	123.2	324.4	151
IADBGS(n,n,n,u,1e−5,3)	4	576	5e−11	3.4	11.4	251.3	266.1	149
IADBGS(n,n,n,u,1e−5,5)	5	556	5e−11	3.3	10.7	262.3	276.3	149
IADBGS(y,n,n,u,1e−10,5)	1	123	1e−10	4.7	3.1	76.9	84.7	149
IADBGS(y,y,n,u)	3	566	5e−11	5.5	185.8	70.7	262.0	151

Table 5.16: Solver statistics for the *Stanford* problem when α=0.99.

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		93	8e−11	5.7		13.1	18.8	115
QPOWER		76	1e−10	6.0		12.3	18.2	152
J		93	8e−11	6.6		14.7	21.3	152
GS		55	6e−11	6.7		8.8	**15.6**	152
BJ(n,n,n,u,1e−5,3)	4	81	8e−11	6.6		13.7	20.3	254
BJ(n,n,n,u,1e−5,5)	5	82	8e−11	6.7		13.8	20.5	254
BJ (y,n,n,l,1e−5,5)	1	20	2e−11	7.6		13.7	21.3	254
BJ (y,n,y,l,1e−10,3)	1	31	7e−11	7.5		14.0	21.4	254
BJ (y,y,n,l,1e−10,3)	1	119	8e−11	6.8		37.7	44.5	315
BJ (y,y,y,l,1e−10,3)	1	174	9e−11	7.0		57.5	64.5	315
BJ (y,n,n,u,1e−5,3)	2	82	8e−11	7.5		33.6	41.1	254
BJ (y,y,n,l,1e−10,3)	2	98	8e−11	8.2		25.1	33.2	320
BJ(y,y,n,u)	3	123	8e−11	12.3		27.3	39.6	357
BGS(n,n,n,u,1e−5,5)	4	55	7e−11	6.6		9.4	16.0	254
BGS(n,n,n,u,1e−10,3)	5	52	6e−11	6.6		10.9	17.5	254
BGS (y,n,n,l,1e−10,5)	1	13	5e−12	7.6		8.8	16.4	254
BGS (y,n,y,l,1e−3,5)	1	13	5e−12	7.5		9.0	16.4	254
BGS (y,y,n,u,1e−10,3)	1	42	1e−11	7.2		12.3	19.5	315
BGS (y,y,y,u,1e−10,3)	1	42	3e−11	7.0		13.1	20.1	315
BGS (y,n,n,u,1e−3,3)	2	40	6e−11	7.9		16.3	24.2	254
BGS (y,y,n,l,1e−10,3)	2	40	7e−11	8.1		10.7	18.8	320
BGS(y,y,n,u)	3	42	7e−11	12.2		10.0	22.2	357
IADBJ(n,n,n,u,1e−5,5)	4	75	9e−11	6.7	3.9	19.7	30.3	425
IADBJ(n,n,n,u,1e−3,3)	5	92	8e−11	6.8	3.4	23.9	34.0	425
IADBJ(y,n,n,u,1e−10,5)	1	19	8e−11	8.0	26.1	12.1	46.2	425
IADBJ(y,y,n,u)	3	92	9e−11	12.1	1,487.1	100.3	1,599.5	438
IADBGS(n,n,n,u,1e−10,3)	4	33	5e−11	6.7	3.9	10.0	20.6	425
IADBGS(n,n,n,u,1e−10,3)	5	35	3e−11	7.1	3.4	10.6	21.1	425
IADBGS(y,n,n,l,1e−10,5)	1	12	4e−11	8.0	19.7	7.3	35.0	425
IADBGS(y,y,n,u)	3	42	7e−11	11.8	1,463.3	46.0	1,521.1	438

Table 5.17: Solver statistics for the *StanfordBerkeley* problem when α=0.85.

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		143	8e−11	6.1		20.1	26.2	115
QPOWER		115	9e−11	5.9		18.5	24.4	152
J		144	8e−11	6.2		22.4	28.7	152
GS		78	7e−11	6.2		12.2	**18.4**	152
BJ(n,n,n,u,1e−5,3)	4	124	8e−11	6.8		20.4	27.2	254
BJ(n,n,n,u,1e−5,5)	5	124	8e−11	6.8		23.7	30.5	254
BJ(y,n,n,u,1e−5,5)	1	29	5e−11	7.6		20.1	27.7	254
BJ(y,n,y,l,1e−3,5)	1	29	5e−11	7.5		19.8	27.3	254
BJ(y,y,n,l,1e−3,3)	1	167	9e−11	8.3		51.0	59.3	315
BJ(y,y,y,l,1e−5,3)	1	224	9e−11	8.2		72.3	80.5	315
BJ(y,n,n,u,1e−5,3)	2	126	9e−11	7.5		51.4	58.9	254
BJ(y,y,n,u,1e−5,3)	2	148	8e−11	8.6		37.2	45.8	320
BJ(y,y,n,u)	2	174	8e−11	11.7		38.4	50.0	357
BGS(n,n,n,u,1e−3,5)	4	78	7e−11	6.6		13.2	19.8	254
BGS(n,n,n,u,1e−3,3)	5	78	7e−11	6.9		13.2	20.1	254
BGS(y,n,n,u,1e−5,5)	1	19	6e−12	7.7		12.6	20.3	254
BGS(y,n,y,u,1e−10,3)	1	29	1e−11	7.5		12.6	20.1	254
BGS(y,y,n,u,1e−10,3)	1	62	2e−11	7.1		18.0	25.1	315
BGS(y,y,y,u,1e−10,3)	1	62	5e−11	7.0		19.0	26.0	315
BGS(y,n,n,u,1e−5,3)	2	60	7e−11	7.8		24.4	32.2	254
BGS(y,y,n,l,1e−5,3)	2	59	8e−11	8.1		15.6	23.7	320
BGS(y,y,n,u)	3	64	8e−11	12.6		14.9	27.5	357
IADBJ(n,n,n,u,1e−5,5)	4	114	1e−10	6.7	3.9	30.4	41.0	425
IADBJ(n,n,n,u,1e−5,3)	5	119	9e−11	7.0	3.4	31.7	42.1	425
IADBJ(y,n,n,u,1e−10,5)	1	30	8e−11	8.5	26.0	18.5	53.0	425
IADBJ(y,y,n,u)	3	142	9e−11	11.8	1,485.6	155.3	1,652.6	438
IADBGS(n,n,n,u,1e−10,3)	4	48	8e−11	6.7	3.9	14.6	25.2	425
IADBGS(n,n,n,u,1e−10,3)	5	48	6e−11	7.0	3.4	14.7	25.1	425
IADBGS(y,n,n,l,1e−10,5)	1	17	7e−11	7.5	19.7	10.1	37.3	425
IADBGS(y,y,n,u)	3	63	7e−11	12.8	1,453.1	69.7	1,535.6	438

Table 5.18: Solver statistics for the *StanfordBerkeley* problem when α=0.90.

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		292	9e−11	6.0		40.9	47.0	115
QPOWER		216	1e−10	6.2		34.8	41.0	152
J		294	9e−11	6.2		45.5	51.8	152
GS		143	9e−11	6.2		22.0	**28.2**	152
BJ(n,n,n,u,1e−5,3)	4	246	1e−10	6.6		40.2	46.8	254
BJ(n,n,n,u,1e−5,3)	5	244	9e−11	6.9		40.6	48.0	254
BJ(y,n,n,l,1e−3,5)	1	58	8e−11	7.5		39.6	47.0	254
BJ(y,n,y,l,1e−5,5)	1	58	8e−11	7.5		39.2	46.7	254
BJ(y,y,n,u,1e−5,3)	1	333	9e−11	8.8		91.8	100.6	315
BJ(y,y,y,u,1e−5,3)	1	400	9e−11	8.2		117.1	125.3	315
BJ(y,n,n,u,1e−5,3)	2	257	9e−11	7.5		104.0	111.4	254
BJ(y,y,n,u,1e−3,3)	2	298	9e−11	8.3		74.9	83.2	320
BJ(y,y,n,u)	3	320	9e−11	11.9		70.0	81.8	357
BGS(n,n,n,u,1e−3,3)	4	143	9e−11	6.8		23.6	30.4	254
BGS(n,n,n,u,1e−10,3)	5	125	9e−11	7.5		26.0	33.6	254
BGS(y,n,n,u,1.0e−10,3)	1	49	3e−11	7.5		20.9	28.5	254
BGS(y,n,y,u,1e−5,3)	1	49	3e−11	7.6		20.9	28.5	254
BGS(y,y,n,u,1e−10,3)	1	120	7e−11	7.3		34.4	41.7	315
BGS(y,y,y,u,1e−10,3)	1	122	9e−11	8.3		36.8	45.2	315
BGS(y,n,n,u,1e−5,3)	2	120	9e−11	7.5		48.6	56.1	254
BGS(y,y,n,u,1e−5,3)	2	117	9e−11	8.5		29.7	38.2	320
BGS(y,y,n,u)	3	128	8e−11	11.8		29.4	41.1	357
IADBJ(n,n,n,u,1e−5,5)	4	235	9e−11	6.9	3.9	64.7	75.5	425
IADBJ(n,n,n,u,1e−5,3)	5	246	9e−11	6.6	3.4	67.9	77.9	425
IADBJ(y,n,n,u,1e−10,5)	1	60	9e−11	7.8	26.1	36.4	70.3	425
IADBJ(y,y,n,u)	3	292	9e−11	11.8	1,503.8	324.2	1,839.8	438
IADBGS(n,n,n,u,1e−10,3)	4	96	8e−11	7.3	4.0	28.7	39.9	425
IADBGS(n,n,n,u,1e−10,3)	5	90	8e−11	9.1	3.4	27.9	40.4	425
IADBGS(y,n,n,l,1e−10,5)	1	31	8e−11	7.9	21.7	18.9	48.5	425
IADBGS(y,y,n,u)	3	125	8e−11	11.8	1,482.2	138.5	1,632.5	438

Table 5.19: Solver statistics for the *StanfordBerkeley* problem when α=0.95.

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		490	1e−10	5.8		68.5	74.3	115
QPOWER		360	2e−10	5.8		58.0	63.8	152
J		493	9e−11	6.7		76.8	83.4	152
GS		239	9e−11	6.5		36.6	43.1	152
BJ(n,n,n,u,1e−5,5)	4	406	1e−10	7.3		66.5	73.8	254
BJ(n,n,n,u,1e−5,5)	5	404	1e−10	7.1		67.0	74.1	254
BJ(y,n,n,u,1e−5,5)	1	97	8e−11	7.8		65.9	73.6	254
BJ(y,n,y,l,1e−3,5)	1	97	8e−11	7.7		65.9	73.6	254
BJ(y,y,n,u,1e−3,3)	1	526	1e−10	8.3		145.3	153.5	315
BJ(y,y,y,u,1e−3,3)	1	586	9e−11	8.3		171.7	180.1	315
BJ(y,n,n,u,1e−3,3)	2	432	9e−11	7.4		176.1	183.6	254
BJ(y,y,n,u,1e−3,3)	2	496	1e−10	8.3		125.0	133.2	320
BJ(y,y,n,l)	3	525	1e−10	12.0		115.1	127.1	357
BGS(n,n,n,u,1e−10,3)	4	169	9e−11	7.3		36.0	43.2	254
BGS(n,n,n,u,1e−10,3)	5	173	9e−11	6.9		36.7	43.6	254
BGS(y,n,n,u,1e−10,3)	1	79	4e−11	7.9		33.5	41.4	254
BGS(y,n,y,u,1e−10,3)	1	79	4e−11	7.8		33.3	**41.0**	254
BGS(y,y,n,u,1e−10,3)	1	199	9e−11	7.0		58.3	65.3	315
BGS(y,y,y,u,1e−10,3)	1	203	9e−11	7.0		61.8	68.8	315
BGS(y,n,n,u,1e−10,3)	2	199	9e−11	7.5		79.7	87.2	254
BGS(y,y,n,u,1e−10,3)	2	193	9e−11	8.2		48.6	56.8	320
BGS(y,y,n,u)	3	211	9e−11	11.9		48.0	59.9	357
IADBJ(n,n,n,u,1e−5,5)	4	395	9e−11	7.2	3.9	112.4	123.5	425
IADBJ(n,n,n,u,1e−5,5)	5	391	1e−10	6.8	3.4	115.2	125.3	425
IADBJ(y,n,n,u,1e−10-5)	1	102	1e−10	7.8	26.0	60.8	94.5	425
IADBJ(y,y,n,u)	3	491	9e−11	11.8	1,477.9	533.8	2,023.5	438
IADBGS(n,n,n,u,1e−10,3)	4	158	9e−11	6.8	3.9	48.3	59.0	425
IADBGS(n,n,n,u,1e−10,3)	5	146	9e−11	6.6	3.4	46.0	56.1	425
IADBGS(y,n,n,l,1e−10,5)	1	49	9e−11	7.7	21.9	29.6	59.2	425
IADBGS(y,y,n,u)	3	205	9e−11	12.0	1,428.5	225.0	1,665.6	438

Table 5.20: Solver statistics for the *StanfordBerkeley* problem when α=0.97.

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		1,000*	1e−08	5.7		131.8	137.5	115
QPOWER		1,000*	2e−10	6.0		160.9	166.9	152
J		1,000*	1e−08	6.2		161.2	167.3	152
GS		719	1e−11	6.8		112.1	118.9	152
BJ(n,n,n,u,1e−5,5)	4	1,000*	7e−10	6.6		162.7	169.4	254
BJ(n,n,n,u,1e−5,5)	5	1,000*	5e−10	7.2		166.1	173.3	254
BJ(y,n,n,u,1e−3,5)	1	284	9e−11	7.5		192.8	200.3	254
BJ(y,n,y,l,1e−5,5)	1	285	9e−11	7.8		192.9	200.7	254
BJ(y,y,n,u,1e−3,3)	1	1,000*	2e−08	8.5		276.5	285.0	315
BJ(y,y,y,u,1e−3,3)	1	1,000*	2e−08	8.7		291.9	300.5	315
BJ(y,n,n,u,1e−10,3)	2	1,000*	2e−09	7.4		402.5	409.9	254
BJ(y,y,n,u,1e−10,3)	2	1,000*	1e−08	8.1		250.4	258.5	320
BJ(y,y,n,u)	3	1,000*	2e−08	12.0		205.4	217.3	357
BGS(n,n,n,u,1e−10,3)	4	454	1e−10	6.5		94.6	101.1	254
BGS(n,n,n,u,1e−10,3)	5	411	1e−10	7.5		85.7	**94.2**	254
BGS(y,n,n,u,1e−5,3)	1	219	1e−10	7.7		94.0	101.7	254
BGS(y,n,y,u,1e−3,3)	1	219	1e−10	7.9		93.0	100.9	254
BGS(y,y,n,u,1e−10,3)	1	584	1e−10	7.2		164.9	172.1	315
BGS(y,y,y,u,1e−10,3)	1	591	1e−10	7.1		177.3	184.4	315
BGS(y,n,n,u,1e−10,3)	2	583	1e−10	7.9		233.1	240.9	254
BGS(y,y,n,u,1e−5,3)	2	552	9e−11	8.1		139.8	147.9	320
BGS(y,y,n,l)	3	559	1e−10	12.2		127.9	140.0	357
IADBJ(n,n,n,u,1e−5,5)	4	1,000*	7e−10	6.7	3.9	339.0	349.6	425
IADBJ(n,n,n,u,1e−10,3)	5	1,000*	4e−10	6.6	3.4	385.3	395.3	425
IADBJ(y,n,n,u,1e−10,5)	1	314	1e−10	7.7	26.2	180.1	214.0	425
IADBJ(y,y,n,u)	3	1,000*	1e−08	11.7	1,506.5	1,096.9	2,615.1	438
IADBGS(n,n,n,u,1e−10,3)	4	438	1e−10	7.0	4.0	151.2	162.2	425
IADBGS(n,n,n,u,1e−10,3)	5	402	9e−11	7.0	3.4	145.8	156.2	425
IADBGS(y,n,n,l,1e−10,5)	1	145	9e−11	7.7	21.7	80.7	110.1	425
IADBGS(y,y,n,u)	3	586	1e−10	12.5	1,528.5	645.9	2,186.9	438

Table 5.21: Solver statistics for the *StanfordBerkeley* problem when α=0.99.

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		90	8e−11	14.9		24.0	38.9	256
QPOWER		73	1e−10	14.4		21.2	35.6	302
J		81	8e−11	15.2		23.8	39.0	340
GS		48	5e−11	16.1		14.3	30.3	340
BJ(n,n,n,u,1e−10,3)	4	70	7e−11	16.4		25.4	41.8	542
BJ(n,n,n,u,1e−3,3)	5	82	8e−11	16.9		25.6	42.5	542
BJ(y,n,n,u,1e−10,3)	1	30	4e−11	17.6		25.4	43.0	542
BJ(y,n,y,l,1e−3,3)	1	30	4e−11	17.5		25.2	42.7	542
BJ(y,y,n,u,1e−3,3)	1	95	8e−11	19.4		51.1	70.4	656
BJ(y,y,y,u,1e−3,3)	1	97	8e−11	18.5		52.3	70.8	656
BJ(y,n,n,l,1e−3,3)	2	79	8e−11	17.4		27.2	44.6	542
BJ(y,y,n,l,1e−3,5)	2	85	8e−11	19.0		27.8	46.7	663
BJ(y,y,n,l)	3	89	8e−11	33.2		42.2	75.4	746
BGS(n,n,n,u,1e−5,3)	4	44	5e−11	16.0		14.1	**30.1**	542
BGS(n,n,n,u,1e−5,3)	5	44	6e−11	16.3		14.2	30.5	542
BGS(y,n,n,l,1e−5,5)	1	11	6e−12	17.5		14.8	32.2	542
BGS(y,n,y,l,1e−3,5)	1	11	6e−12	17.7		14.8	32.6	542
BGS(y,y,n,u,1e−3,3)	1	44	5e−11	19.2		24.1	43.2	656
BGS(y,y,y,u,1e−10,3)	1	43	7e−11	19.5		23.3	42.8	656
BGS(y,n,n,l,1e−3,3)	2	48	3e−11	17.4		17.1	34.6	542
BGS(y,y,n,l,1e−3,3)	2	44	6e−11	19.3		15.5	34.7	663
BGS(y,y,n,l)	3	41	7e−11	32.5		20.5	53.0	746
IADBJ(n,n,n,u,1e−10,3)	4	50	7e−11	15.9	390.1	28.9	434.9	903
IADBJ(n,n,n,u,1e−10,5)	5	52	7e−11	17.1	377.0	30.9	425.0	903
IADBJ(y,n,n,u,1e−3,5)	1	83	7e−11	19.2	1,606.8	36.0	1,662.6	903
IADBJ(y,y,n,u)	3	85	8e−11	31.3	6,675.9	1,266.8	7,974.0	921
IADBGS(n,n,n,u,1e−3,3)	4	40	6e−11	16.0	387.7	20.7	424.4	903
IADBGS(n,n,n,u,1e−10,3)	5	30	6e−11	17.0	368.7	18.0	403.6	903
IADBGS(y,n,n,u,1e−10,5)	1	10	4e−11	17.6	1,592.1	13.1	1,622.7	903
IADBGS(y,y,n,u)	3	47	9e−12	31.2	6,112.7	700.2	6,844.2	921

Table 5.22: Solver statistics for the *Eu2005* problem when α=0.85.

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		137	8e−11	14.1		36.4	50.5	256
QPOWER		115	9e−11	14.0		33.5	47.5	302
J		124	8e−11	15.3		35.6	50.9	340
GS		71	4e−11	15.8		20.7	**36.6**	340
BJ(n,n,n,u,1e−10,3)	4	105	8e−11	15.9		37.6	53.5	542
BJ(n,n,n,u,1e−3-3)	5	125	9e−11	18.2		38.5	56.7	542
BJ(y,n,n,u,1e−10,3)	1	46	3e−11	18.2		38.1	56.3	542
BJ(y,n,y,l,1e−3,3)	1	46	4e−11	17.1		38.5	55.6	542
BJ(y,y,n,l,1.0e−10,3)	1	140	9e−11	19.3		75.5	94.9	656
BJ(y,y,y,u,1.0e−10,3)	1	146	9e−11	19.3		76.2	95.4	656
BJ(y,n,n,l,1e−3,3)	2	123	8e−11	18.0		42.8	60.8	542
BJ(y,y,n,l,1e−3,3)	2	127	8e−11	19.2		41.3	60.5	663
BJ(y,y,n,u)	3	137	9e−11	32.5		64.3	96.7	746
BGS(n,n,n,u,1e−10,3)	4	52	7e−11	17.3		19.9	37.2	542
BGS(n,n,n,u,1e−5,5)	5	65	5e−11	18.0		20.8	38.8	542
BGS(y,n,n,u,1e−5,3)	1	27	5e−12	17.3		22.5	39.8	542
BGS(y,n,y,l,1e−3,5)	1	17	2e−12	17.5		22.5	40.0	542
BGS(y,y,n,u,1.0e−5,3)	1	66	7e−11	19.2		35.5	54.8	656
BGS(y,y,y,u,1.0e−5,3)	1	66	7e−11	19.3		35.1	54.4	656
BGS(y,n,n,l,1e−3,3)	2	72	2e−11	17.7		25.9	43.5	542
BGS(y,y,n,l,1e−5,3)	2	65	8e−11	19.1		23.4	42.5	663
BGS(y,y,n,l)	3	63	7e−11	33.3		31.0	64.3	746
IADBJ(n,n,n,u,1e−10,5)	4	74	6e−11	16.0	398.7	44.1	458.8	903
IADBJ(n,n,n,u,1e−10,3)	5	75	8e−11	16.8	376.3	45.0	438.1	903
IADBJ(y,n,n,u,1e−10,5)	1	27	7e−11	17.7	1,737.9	38.4	1,793.9	903
IADBJ(y,y,n,u)	3	127	9e−11	31.1	6,660.6	1,868.9	8,560.6	921
IADBGS(n,n,n,u,1e−10,3)	4	43	5e−11	16.5	390.8	24.7	431.9	903
IADBGS(n,n,n,u,1e−10,5)	5	43	7e−11	16.7	387.9	25.9	430.6	903
IADBGS(y,n,n,l,1e−10,5)	1	14	6e−11	18.6	214.4	18.2	251.3	903
IADBGS(y,y,n,u)	3	70	9e−12	31.6	6,069.9	1,040.9	7,142.4	921

Table 5.23: Solver statistics for the *Eu2005* problem when α=0.90.

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		269	9e−11	15.0		70.1	85.1	256
QPOWER		238	1e−10	14.1		68.4	82.5	302
J		249	9e−11	15.9		71.8	87.7	340
GS		140	2e−11	15.2		40.3	55.4	340
BJ(n,n,n,u,1e−10,3)	4	209	9e−11	15.9		72.3	88.1	542
BJ(n,n,n,u,1e−10,3)	5	212	9e−11	15.9		76.3	92.2	542
BJ(y,n,n,u,1e−10,3)	1	91	5e−11	17.8		73.8	91.6	542
BJ(y,n,y,u,1e−10,5)	1	57	3e−11	17.8		74.3	92.1	542
BJ(y,y,n,u,1e−5,3)	1	290	9e−11	20.3		149.8	170.1	656
BJ(y,y,y,u,1e−10,3)	1	296	9e−11	19.5		153.5	173.1	656
BJ(y,n,n,l,1e−3,5)	2	244	1e−10	18.8		83.5	102.3	542
BJ(y,y,n,l,1e−3,3)	2	252	9e−11	19.3		80.3	99.6	663
BJ(y,y,n,l)	3	280	9e−11	32.3		130.8	163.1	746
BGS(n,n,n,u,1e−10,3)	4	96	9e−11	16.0		35.1	**51.1**	542
BGS(n,n,n,u,1e−10,3)	5	98	8e−11	16.0		35.9	51.9	542
BGS(y,n,n,u,1e−5,5)	1	33	4e−12	17.3		43.3	60.6	542
BGS(y,n,y,u,1e−10,3)	1	52	7e−12	17.9		42.2	60.1	542
BGS(y,y,n,u,1e−5,3)	1	130	8e−11	19.2		68.3	87.4	656
BGS(y,y,y,u,1.0e−3,3)	1	130	8e−11	19.4		68.7	88.1	656
BGS(y,n,n,u,1e−5,3)	2	134	7e−11	18.4		48.6	67.0	542
BGS(y,y,n,l,1e−3,3)	2	132	7e−11	19.3		44.1	63.4	663
BGS(y,y,n,l)	3	127	9e−11	32.4		61.5	93.9	746
IADBJ(n,n,n,u,1e−10,5)	4	135	7e−11	16.2	395.2	78.3	489.6	903
IADBJ(n,n,n,u,1e−10,3)	5	133	8e−11	15.8	381.3	79.9	477.1	903
IADBJ(y,n,n,u,1e−10,5)	1	53	9e−11	18.8	1,754.0	73.2	1,846.1	903
IADBJ(y,y,n,u)	3	259	9e−11	32.1	6,618.5	3,822.0	10,472.6	921
IADBGS(n,n,n,u,1e−10,5)	4	78	7e−11	16.2	396.0	45.4	457.6	903
IADBGS(n,n,n,u,1e−10,3)	5	79	9e−11	17.3	378.9	47.7	443.9	903
IADBGS(y,n,n,l,1e−10,5)	1	23	8e−11	17.3	226.6	33.6	277.4	903
IADBGS(y,y,n,u)	3	137	1e−11	32.2	6,112.3	2,158.6	8,303.1	921

Table 5.24: Solver statistics for the *Eu2005* problem when α=0.95.

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		434	1e−10	14.1		114.1	128.6	256
QPOWER		394	2e−10	14.3		113.0	127.3	302
J		416	1e−10	15.1		120.1	135.2	340
GS		228	2e−11	15.2		66.6	81.8	340
BJ(n,n,n,u,1e−10,3)	4	345	9e−11	17.2		118.7	136.0	542
BJ(n,n,n,u,1e−10,3)	5	350	9e−11	17.4		121.5	138.9	542
BJ(y,n,n,u,1e−3,5)	1	94	3e−11	17.9		121.0	138.9	542
BJ(y,n,y,u,1e−5,3)	1	150	6e−11	17.6		121.3	138.9	542
BJ(y,y,n,u,1e−3,3)	1	485	1e−10	20.0		249.6	269.6	656
BJ(y,y,y,u,1e−3,3)	1	494	1e−10	20.9		255.4	276.3	656
BJ(y,n,n,l,1e−3,5)	2	345	7e−10	17.3		116.0	133.4	542
BJ(y,y,n,u,1e−3,3)	2	412	1e−10	19.4		130.4	149.8	663
BJ(y,y,n,u)	3	470	1e−10	33.3		218.0	251.3	746
BGS(n,n,n,u,1e−10,3)	4	168	9e−11	17.5		59.3	**76.8**	542
BGS(n,n,n,u,1e−10,3)	5	171	9e−11	18.1		61.0	79.1	542
BGS(y,n,n,l,1e−5,5)	1	53	4e−12	17.5		69.3	86.7	542
BGS(y,n,y,u,1e−10,3)	1	84	9e−12	17.9		68.5	86.4	542
BGS(y,y,n,u,1e−10,3)	1	213	9e−11	19.2		111.5	130.7	656
BGS(y,y,y,u,1e−5,3)	1	213	9e−11	20.5		112.0	132.5	656
BGS(y,n,n,l,1e−3,3)	2	230	3e−11	18.3		79.8	98.0	542
BGS(y,y,n,l,1e−3,3)	2	214	8e−11	19.1		69.7	88.8	663
BGS(y,y,n,l)	3	210	9e−11	33.6		100.9	134.5	746
IADBJ(n,n,n,u,1e−10,3)	4	206	7e−11	16.0	394.8	117.3	528.2	903
IADBJ(n,n,n,u,1e−10,5)	5	196	9e−11	17.0	373.2	119.4	509.6	903
IADBJ(y,n,n,u,1e−10,5)	1	91	1e−10	17.9	1,930.0	127.9	2,075.8	903
IADBJ(y,y,n,u)	3	435	1e−10	33.2	6,640.2	6,519.9	13,193.3	921
IADBGS(n,n,n,u,1e−10,3)	4	121	7e−11	16.9	392.8	68.5	478.2	903
IADBGS(n,n,n,u,1e−10,3)	5	126	9e−11	15.8	376.6	76.0	468.4	903
IADBGS(y,n,n,l,1e−10,5)	1	37	8e−11	17.3	226.2	52.1	295.7	903
IADBGS(y,y,n,u)	3	219	1e−10	33.1	7,880.1	3,240.4	11,153.5	921

Table 5.25: Solver statistics for the *Eu2005* problem when α=0.97.

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		1,000*	1e−09	15.0		262.6	277.5	256
QPOWER		1,000*	3e−10	14.6		286.5	301.1	302
J		1,000*	1e−09	17.5		286.3	303.8	340
GS		634	5e−11	15.0		184.1	199.0	340
BJ(n,n,n,u,1e−10,3)	4	1,000*	1e−10	16.2		348.5	364.8	542
BJ(n,n,n,u,1e−10,3)	5	1,000*	1e−10	16.3		346.7	363.0	542
BJ(y,n,n,u,1e−3,3)	1	266	5e−11	22.1		288.1	310.2	542
BJ(y,n,y,l,1e−3,3)	1	266	5e−11	21.3		286.9	308.2	542
BJ(y,y,n,u,1e−10,3)	1	1,000*	2e−11	18.5		534.2	552.7	656
BJ(y,y,y,u,1e−3,3)	1	1,000*	1e−08	19.2		529.8	549.0	656
BJ(y,n,n,l,1e−3,3)	2	1,000*	1e−09	18.3		342.5	360.7	542
BJ(y,y,n,l,1e−3,5)	2	1,000*	6e−10	19.2		318.9	338.1	663
BJ(y,y,n,l)	3	1,000*	6e−09	34.0		469.9	503.9	746
BGS(n,n,n,u,1e−10,3)	4	502	1e−10	15.6		171.0	**186.6**	542
BGS(n,n,n,u,1e−10,3)	5	511	1e−10	17.2		177.9	195.1	542
BGS(y,n,n,l,1e−10,3)	1	437	8e−11	21.7		399.2	420.9	542
BGS(y,n,y,l,1e−10,3)	1	437	8e−11	20.2		405.9	426.0	542
BGS(y,y,n,l,1e−10,3)	1	608	6e−11	21.9		368.8	390.7	656
BGS(y,y,y,l,1e−10,3)	1	607	1e−10	23.1		383.1	406.2	656
BGS(y,n,n,l,1e−3,3)	2	637	5e−11	18.1		219.9	238.0	542
BGS(y,y,n,u,1e−3,3)	2	590	8e−11	19.2		191.6	210.8	663
BGS(y,y,n,l)	3	594	1e−10	32.6		287.0	319.6	746
IADBJ(n,n,n,u,1e−10,5)	4	489	5e−11	15.9	390.6	297.7	704.2	903
IADBJ(n,n,n,u,1e−10,3)	5	477	7e−11	16.6	383.8	331.9	732.2	903
IADBJ(y,n,n,u,1e−10,5)	1	292	1e−10	17.6	1,770.8	357.8	2,146.3	903
IADBJ(y,y,n,u)	3	1,000*	1e−10	33.4	6,584.8	19,308.7	25,927.0	921
IADBGS(n,n,n,u,1e−10,3)	4	284	6e−11	16.8	389.1	166.7	572.5	903
IADBGS(n,n,n,u,1e−10,3)	5	329	1e−10	16.7	383.8	209.4	610.0	903
IADBGS(y,n,n,l,1e−10,5)	1	109	9e−11	17.2	212.7	158.1	388.0	903
IADBGS(y,y,n,u)	3	601	3e−11	33.0	6,069.1	9,268.7	15,370.7	921

Table 5.26: Solver statistics for the *Eu2005* problem when α=0.99.

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		100	4e−10	13.9		27.3	41.2	252
QPOWER		82	6e−11	13.6		25.7	39.3	325
J		100	4e−11	13.8		32.1	45.9	332
GS		57	6e−11	14.1		18.7	**32.7**	332
BJ(n,n,n,u,1e−5,3)	4	89	4e−10	15.1		30.8	45.9	551
BJ(n,n,n,u,1e−5,5)	5	89	4e−10	14.8		31.5	46.3	551
BJ(y,n,n,l,1e−10,3)	1	34	4e−11	16.7		30.9	47.5	551
BJ(y,n,y,l,1e−3,3)	1	34	4e−11	16.7		31.2	47.9	551
BJ(y,y,n,u,1e−5-3)	1	113	8e−11	18.1		75.0	93.1	680
BJ(y,y,y,u,1e−3,3)	1	139	8e−11	18.5		99.5	118.0	680
BJ(y,n,n,l,1e−10,3)	2	63	9e−11	16.5		56.3	72.8	551
BJ(y,y,n,l,1e−3,3)	2	82	9e−11	18.2		47.4	65.6	690
BJ(y,y,n,l)	3	111	7e−11	55.2		46.2	101.4	771
BGS(n,n,n,u,1e−10,3)	4	40	1e−11	15.4		18.1	33.5	551
BGS(n,n,n,u,1e−10,3)	5	40	1e−11	15.2		18.6	33.8	551
BGS(y,n,n,l,1e−3,3)	1	21	2e−11	16.3		18.6	34.8	551
BGS(y,n,y,l,1e−5,3)	1	21	2e−11	16.4		18.5	34.9	551
BGS(y,y,n,l,1e−5,3)	1	43	5e−12	18.4		25.7	44.1	680
BGS(y,y,y,l,1e−10,3)	1	43	5e−12	18.3		26.2	44.6	680
BGS(y,n,n,l,1e−5,3)	2	39	8e−11	16.5		34.9	51.4	551
BGS(y,y,n,u,1e−5,3)	2	51	7e−11	18.0		30.4	48.4	690
BGS(y,y,n,l)	3	44	6e−11	53.9		19.2	73.2	771
IADBJ(n,n,n,u,1e−5,5)	4	75	7e−11	15.2	10.4	39.1	64.6	913
IADBJ(n,n,n,u,1e−10,3)	5	60	9e−11	15.1	9.3	38.7	63.1	913
IADBJ(y,n,n,u,1e−10,5)	1	21	3e−10	18.0	110.4	28.3	156.7	913
IADBJ(y,y,n,u)	3	94	8e−11	53.6	6,692.1	17,419.2	24,164.9	944
IADBGS(n,n,n,u,1e−10,3)	4	38	7e−12	14.7	10.6	23.2	48.5	913
IADBGS(n,n,n,u,1e−10,3)	5	38	3e−11	15.6	9.2	24.0	48.8	913
IADBGS(y,n,n,u,1e−10,5)	1	13	5e−11	16.7	111.5	17.1	145.3	913
IADBGS(y,y,n,u)	3	45	1e−11	54.0	6,959.9	12,644.6	19,658.6	944

Table 5.27: Solver statistics for the *In2004* problem when α=0.85.

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		151	5e−10	13.5		40.4	53.9	252
QPOWER		112	1e−10	13.0		34.2	47.2	325
J		154	3e−10	13.8		49.0	62.8	332
GS		84	7e−11	13.9		27.3	41.2	332
BJ(n,n,n,u,1e−5,5)	4	137	1e−10	15.5		46.6	62.2	551
BJ(n,n,n,u,1e−5,3)	5	137	2e−10	15.3		47.2	62.5	551
BJ(y,n,n,u,1e−5,3)	1	51	7e−11	16.6		44.8	61.4	551
BJ(y,n,y,l,1e−5,3)	1	50	7e−11	16.1		44.9	61.0	551
BJ(y,y,n,u,1e−5,3)	1	165	8e−11	18.5		106.9	125.3	680
BJ(y,y,y,u,1e−5,3)	1	189	9e−11	19.1		132.7	151.7	680
BJ(y,n,n,l,1e−10,3)	2	94	1e−10	17.1		84.0	101.1	551
BJ(y,y,n,l,1e−10,3)	2	121	1e−10	17.9		69.7	87.6	690
BJ(y,y,n,l)	3	162	8e−11	54.8		65.7	120.5	771
BGS(n,n,n,u,1e−10,3)	4	58	2e−11	15.0		25.9	**40.9**	551
BGS(n,n,n,u,1e−10,3)	5	58	2e−11	15.5		26.3	41.8	551
BGS(y,n,n,l,1e−5,5)	1	20	5e−12	16.4		26.5	42.9	551
BGS(y,n,y,u,1e−5,3)	1	29	2e−11	17.6		25.1	42.6	551
BGS(y,y,n,l,1e−3,3)	1	64	6e−12	18.0		37.2	55.2	680
BGS(y,y,y,l,1e−5,3)	1	64	5e−12	17.7		38.8	56.6	680
BGS(y,n,n,l,1e−10,3)	2	57	9e−11	16.3		50.5	66.9	551
BGS(y,y,n,u,1e−10,3)	2	74	9e−11	19.4		43.4	62.8	690
BGS(y,y,n,l)	3	66	7e−11	54.9		28.3	83.2	771
IADBJ(n,n,n,u,1e−10,3)	4	100	1e−10	15.0	10.4	60.8	86.2	913
IADBJ(n,n,n,u,1e−10,3)	5	89	9e−11	14.9	9.3	57.9	82.1	913
IADBJ(y,n,n,u,1e−10,5)	1	32	4e−10	16.8	112.4	42.8	172.0	913
IADBJ(y,y,n,u)	3	144	9e−11	53.8	6,617.8	23,018.0	29,689.6	944
IADBGS(n,n,n,u,1e−10,3)	4	55	7e−12	16.4	10.3	33.3	60.1	913
IADBGS(n,n,n,u,1e−10,3)	5	55	2e−11	15.2	9.2	34.5	58.9	913
IADBGS(y,n,n,u,1e−10,5)	1	18	5e−11	18.0	112.0	23.7	153.7	913
IADBGS(y,y,n,u)	3	65	2e−11	54.3	6,666.6	14,693.8	21,414.7	944

Table 5.28: Solver statistics for the *In2004* problem when α=0.90.

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		313	3e−10	14.3		84.1	98.4	252
QPOWER		203	2e−10	14.1		61.9	75.9	325
J		319	2e−10	14.0		101.2	115.1	332
GS		159	6e−11	14.1		51.4	65.5	332
BJ(n,n,n,u,1e−5,5)	4	269	9e−11	15.8		91.2	107.0	551
BJ(n,n,n,u,1e−5,5)	5	272	9e−11	16.1		93.7	109.8	551
BJ(y,n,n,l,1e−10,5)	1	60	1e−10	16.5		81.6	98.1	551
BJ(y,n,y,l,1e−3,5)	1	60	1e−10	16.3		81.6	97.9	551
BJ(y,y,n,l,1e−5,3)	1	335	9e−11	17.9		196.0	213.9	680
BJ(y,y,y,l,1e−5,3)	1	367	9e−11	18.5		227.0	245.5	680
BJ(y,n,n,l,1e−10,3)	2	180	1e−10	16.6		160.0	176.6	551
BJ(y,y,n,l,1e−10,3)	2	240	1e−10	17.8		138.0	155.7	690
BJ(y,y,n,l)	3	321	9e−11	54.3		130.0	184.3	771
BGS(n,n,n,u,1e−10,3)	4	110	2e−11	15.5		47.8	63.2	551
BGS(n,n,n,u,1e−10,3)	5	110	2e−11	16.0		48.2	64.2	551
BGS(y,n,n,u,1e−3,5)	1	35	1e−11	16.5		46.2	**62.7**	551
BGS(y,n,y,u,1e−10,5)	1	35	1e−11	16.8		46.4	63.1	551
BGS(y,y,n,l,1e−3,3)	1	124	8e−12	19.1		71.1	90.2	680
BGS(y,y,y,l,1e−10,3)	1	123	9e−12	18.1		72.2	90.3	680
BGS(y,n,n,l,1e−5,3)	2	109	1e−10	16.1		96.0	112.0	551
BGS(y,y,n,u,1e−3,3)	2	146	9e−11	18.1		85.0	103.1	690
BGS(y,y,n,l)	3	128	9e−11	55.0		54.2	109.2	771
IADBJ(n,n,n,u,1e−10,3)	4	204	1e−10	15.9	10.4	122.3	148.6	913
IADBJ(n,n,n,u,1e−10,3)	5	169	1e−10	15.1	9.3	109.6	134.0	913
IADBJ(y,n,n,u,1e−10,5)	1	66	2e−10	18.0	107.8	82.1	207.9	913
IADBJ(y,y,n,u)	3	293	9e−11	54.4	6,723.3	41,186.9	47,964.6	944
IADBGS(n,n,n,u,1e−10,3)	4	104	1e−11	16.3	10.5	62.6	89.4	913
IADBGS(n,n,n,u,1e−10,3)	5	104	5e−11	15.0	9.2	64.7	89.0	913
IADBGS(y,n,n,u,1e−10,5)	1	35	5e−11	16.7	111.5	43.9	172.1	913
IADBGS(y,y,n,u)	3	126	3e−11	55.1	6,638.2	21,576.9	28,270.2	944

Table 5.29: Solver statistics for the *In2004* problem when α=0.95.

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		526	1e−10	12.9		140.8	153.7	252
QPOWER		313	2e−10	13.9		95.2	109.1	325
J		530	2e−10	14.6		169.3	183.9	332
GS		253	9e−11	14.8		83.4	98.2	332
BJ(n,n,n,u,1e−5-3)	4	440	1e−10	15.1		146.5	161.6	551
BJ(n,n,n,u,1e−5,5)	5	442	1e−10	15.9		149.9	165.8	551
BJ(y,n,n,l,1e−10,5)	1	102	1e−10	16.5		137.6	154.1	551
BJ(y,n,y,l,1e−10,5)	1	102	1e−10	17.4		136.7	154.1	551
BJ(y,y,n,l,1e−3,3)	1	542	1e−10	17.8		315.4	333.2	680
BJ(y,y,y,l,1e−10,3)	1	559	9e−11	17.9		341.5	359.4	680
BJ(y,n,n,l,1e−5,3)	2	286	1e−10	16.7		256.5	273.2	551
BJ(y,y,n,l,1e−3,3)	2	389	1e−10	18.7		225.2	243.9	690
BJ(y,y,n,u)	3	533	9e−11	54.3		214.2	268.5	771
BGS(n,n,n,u,1e−10,3)	4	173	3e−11	14.9		74.1	**89.0**	551
BGS(n,n,n,u,1e−10,3)	5	173	3e−11	15.7		74.3	90.0	551
BGS(y,n,n,u,1e−5,5)	1	56	2e−11	16.7		74.3	90.9	551
BGS(y,n,y,u,1e−10,5)	1	56	2e−11	17.3		73.6	91.0	551
BGS(y,y,n,l,1e−3,3)	1	198	1e−11	17.9		115.1	133.0	680
BGS(y,y,y,l,1e−3,3)	1	198	1e−11	17.8		116.2	134.0	680
BGS(y,n,n,l,1e−3,3)	2	173	1e−10	17.6		151.4	169.0	551
BGS(y,y,n,u,1e−5,3)	2	235	1e−10	17.9		137.5	155.4	690
BGS(y,y,n,l)	3	207	9e−11	54.7		86.5	141.2	771
IADBJ(n,n,n,u,1e−10,3)	4	340	1e−10	15.1	10.4	206.4	231.8	913
IADBJ(n,n,n,u,1e−10,3)	5	265	1e−10	15.1	9.2	176.1	200.4	913
IADBJ(y,n,n,u,1e−10,5)	1	110	2e−10	17.4	108.3	134.2	259.9	913
IADBJ(y,y,n,u)	3	490	1e−10	54.1	6,775.7	62,911.1	69,740.9	944
IADBGS(n,n,n,u,1e−10,3)	4	163	1e−11	15.1	10.4	97.6	123.0	913
IADBGS(n,n,n,u,1e−10,5)	5	153	7e−11	15.0	9.2	104.6	128.8	913
IADBGS(y,n,n,u,1e−10,5)	1	53	9e−11	17.6	109.6	67.2	194.5	913
IADBGS(y,y,n,u)	3	204	4e−11	55.1	6,681.6	29,755.2	36,491.9	944

Table 5.30: Solver statistics for the *In2004* problem when α=0.97.

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		1,000*	8e−09	13.6		268.3	281.9	252
QPOWER		856	5e−10	13.7		261.6	275.3	325
J		1,000*	1e−09	15.5		324.2	339.7	332
GS		757	1e−11	14.7		241.9	256.6	332
BJ(n,n,n,u,1e−5,3)	4	1,000*	8e−10	15.1		338.2	353.3	551
BJ(n,n,n,u,1e−5,3)	5	1,000*	9e−10	15.8		338.4	354.2	551
BJ(y,n,n,l,1e−3,3)	1	373	4e−09	16.5		325.3	341.8	551
BJ(y,n,y,l,1e−3,3)	1	373	4e−09	16.6		325.0	341.6	551
BJ(y,y,n,l,1e−3,3)	1	1,000*	3e−08	19.3		579.9	599.2	680
BJ(y,y,y,l,1e−10,3)	1	1,000*	3e−08	17.8		613.4	631.2	680
BJ(y,n,n,l,1e−10,3)	2	753	1e−10	16.7		655.0	671.7	551
BJ(y,y,n,l,1e−3,3)	2	1,000*	1e−10	18.0		566.1	584.0	690
BJ(y,y,n,l)	3	1,000*	3e−08	54.7		407.6	462.3	771
BGS(n,n,n,u,1e−10,3)	4	449	1e−10	15.4		187.9	203.3	551
BGS(n,n,n,u,1e−10,3)	5	448	1e−10	15.7		190.3	206.0	551
BGS(y,n,n,u,1e−10,5)	1	153	2e−11	17.8		199.7	217.5	551
BGS(y,n,y,u,1e−3,5)	1	153	2e−11	16.6		201.0	217.6	551
BGS(y,y,n,l,1e−10,3)	1	527	1e−11	18.8		301.9	320.7	680
BGS(y,y,y,l,1e−3,3)	1	529	1e−11	18.9		309.5	328.4	680
BGS(y,n,n,l,1e−3,3)	2	457	1e−10	16.4		393.0	409.4	551
BGS(y,y,n,u,1e−5,3)	2	624	1e−10	19.1		359.0	378.1	690
BGS(y,y,n,l)	3	556	9e−11	55.4		234.9	290.3	771
IADBJ(n,n,n,u,1e−10,3)	4	968	1e−10	14.9	10.5	632.1	657.5	913
IADBJ(n,n,n,u,1e−10,5)	5	598	1e−10	16.4	9.2	494.7	520.3	913
IADBJ(y,n,n,u,1e−10,5)	1	285	2e−10	16.7	112.5	367.3	496.4	913
IADBJ(y,y,n,u)	3			54.0	6,696.4	very long time		944
IADBGS(n,n,n,u,1e−10,5)	4	373	2e−11	15.6	10.3	253.8	279.7	913
IADBGS(n,n,n,u,1e−10,5)	5	153	7e−11	15.1	9.2	103.2	**127.5**	913
IADBGS(y,n,n,u,1e−10,5)	1	150	1e−10	18.6	110.7	185.3	314.5	913
IADBGS(y,y,n,u)	3			53.6	6,695.9	very long time		944

Table 5.31: Solver statistics for the *In2004* problem when α=0.99.

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		96	7e−11	34.9		60.1	95.0	625
QPOWER		79	1e−10	36.1		55.9	92.0	767
J		96	8e−11	38.8		69.5	108.3	826
GS		44	7e−11	37.1		32.7	**69.8**	826
BJ(n,n,n,u,1e−5,3)	4	88	8e−11	41.0		68.4	109.4	1,341
BJ(n,n,n,u,1e−5,3)	5	92	8e−11	39.3		71.6	110.9	1,341
BJ(y,n,n,u,1e−10,5)	1	21	1e−11	42.6		63.3	105.9	1,341
BJ(y,n,y,l,1e−3,3)	1	32	6e−11	42.7		62.0	104.6	1,341
BJ(y,y,n,l,1e−5,3)	1	100	7e−11	45.4		118.4	163.8	1,637
BJ(y,y,y,l,1e−5,3)	1	105	8e−11	45.4		129.5	175.0	1,637
BJ(y,n,n,u,1e−5,3)	2	89	7e−11	36.0		64.3	100.3	1,341
BJ(y,y,n,l,1e−5,3)	2	96	8e−11	39.0		65.0	104.0	1,657
BJ(y,y,n,l)	3	105	7e−11	69.9		115.6	185.5	1,858
BGS(n,n,n,u,1e−3,3)	4	45	6e−11	40.6		35.9	76.5	1,341
BGS(n,n,n,u,1e−3,5)	5	45	6e−11	41.2		36.5	77.7	1,341
BGS(y,n,n,l,1e−10,5)	1	12	7e−12	41.9		36.2	78.1	1,341
BGS(y,n,y,l,1e−3,3)	1	19	1e−11	42.8		37.1	79.9	1,341
BGS(y,y,n,u,1e−10,3)	1	44	9e−11	45.4		53.2	98.6	1,637
BGS(y,y,y,l,1e−3,3)	1	44	7e−11	47.8		55.9	103.7	1,637
BGS(y,n,n,l,1e−5,3)	2	45	7e−11	37.9		35.4	73.3	1,341
BGS(y,y,n,l,1e−5,3)	2	45	6e−11	40.4		32.8	73.2	1,657
BGS(y,y,n,l)	3	42	6e−11	69.4		49.0	118.4	1,858
IADBJ(n,n,n,u,1e−10,3)	4	66	8e−11	38.7	195.8	85.9	320.4	2,218
IADBJ(n,n,n,u,1e−3,3)	5	95	8e−11	40.8	179.1	110.6	330.4	2,218
IADBJ(y,n,n,u,1e−3,3)	1	98	8e−11	44.3	7,094.1	129.3	7,267.7	2,218
IADBGS(n,n,n,u,1e−3,3)	4	43	7e−11	40.0	196.0	48.5	284.6	2,218
IADBGS(n,n,n,u,1e−5,3)	5	40	8e−11	40.3	182.0	47.7	270.1	2,218
IADBGS(y,n,n,l,1e−5,3)	1	41	7e−11	43.5	3,650.3	54.3	3,748.0	2,218

Table 5.32: Solver statistics for the *Webbase* problem when α=0.85.

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		147	8e−11	35.9		91.9	127.8	625
QPOWER		115	9e−11	36.6		80.5	117.1	767
J		148	9e−11	37.1		106.5	143.6	826
GS		65	8e−11	38.9		48.1	**87.0**	826
BJ(n,n,n,u,1e−5,5)	4	132	1e−10	38.7		101.8	140.5	1,341
BJ(n,n,n,u,1e−5,3)	5	142	9e−11	42.5		109.8	152.3	1,341
BJ(y,n,n,l,1e−5,5)	1	30	4e−11	42.2		89.8	131.9	1,341
BJ(y,n,y,l,1e−3,5)	1	30	4e−11	42.3		89.9	132.2	1,341
BJ(y,y,n,u,1e−10,3)	1	153	5e−11	47.4		173.4	220.8	1,637
BJ(y,y,y,u,1e−3,3)	1	157	9e−11	45.3		193.9	239.2	1,637
BJ(y,n,n,l,1e−5,5)	2	134	9e−11	37.8		100.4	138.2	1,341
BJ(y,y,n,l,1e−5,5)	2	147	9e−11	41.2		104.0	145.2	1,657
BJ(y,y,n,l)	3	160	8e−11	66.8		174.8	241.6	1,858
BGS(n,n,n,u,1e−3,3)	4	66	7e−11	38.6		52.0	90.6	1,341
BGS(n,n,n,u,1e−3,5)	5	65	8e−11	39.1		52.1	91.2	1,341
BGS(y,n,n,l,1e−10,3)	1	27	2e−11	41.9		52.0	93.9	1,341
BGS(y,n,y,l,1e−3,5)	1	17	9e−12	42.5		51.4	93.9	1,341
BGS(y,y,n,u,1e−10,3)	1	66	3e−11	45.6		79.2	124.8	1,637
BGS(y,y,y,l,1e−3,3)	1	66	8e−11	44.7		85.0	129.7	1,637
BGS(y,n,n,l,1e−5,5)	2	67	7e−11	37.1		53.0	90.0	1,341
BGS(y,y,n,l,1e−5,5)	2	67	8e−11	39.3		48.3	87.6	1,657
BGS(y,y,n,l)	3	62	7e−11	68.2		71.4	139.6	1,858
IADBJ(n,n,n,u,1e−5,5)	4	116	9e−11	41.8	195.5	131.5	368.8	2,218
IADBJ(n,n,n,u,1e−10,3)	5	103	8e−11	46.8	179.5	136.1	362.5	2,218
IADBJ(y,n,n,u,1e−10,5)	1	31	8e−11	45.0	7,127.6	95.4	7,268.0	2,218
IADBGS(n,n,n,u,1e−3,5)	4	64	8e−11	41.4	195.8	73.2	310.5	2,218
IADBGS(n,n,n,u,1e−5,5)	5	60	8e−11	42.0	173.7	71.3	287.0	2,218
IADBGS(y,n,n,l,1e−3,5)	1	64	7e−11	44.5	3,495.9	82.6	3,623.0	2,218

Table 5.33: Solver statistics for the *Webbase* problem when α=0.90.

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		300	9e−11	36.5		187.2	223.8	625
QPOWER		239	9e−11	34.5		167.1	201.5	767
J		303	9e−11	37.7		216.1	253.8	826
GS		126	8e−11	39.9		92.9	132.8	826
BJ(n,n,n,u,1e−5,5)	4	268	1e−10	38.6		206.5	245.1	1,341
BJ(n,n,n,u,1e−5-5)	5	288	1e−10	41.8		220.7	262.5	1,341
BJ(y,n,n,u,1e−3,5)	1	58	9e−11	45.7		145.2	190.9	1,341
BJ(y,n,y,l,1e−3,5)	1	58	9e−11	43.8		142.7	186.4	1,341
BJ(y,y,n,u,1e−3,3)	1	309	6e−11	49.3		302.0	351.2	1,637
BJ(y,y,y,l,1e−3,3)	1	314	9e−11	48.9		319.1	368.0	1,637
BJ(y,n,n,u,1e−10,3)	2	275	9e−11	45.1		302.9	348.0	1,341
BJ(y,y,n,l,1e−10,3)	2	302	9e−11	48.1		231.3	279.3	1,657
BJ(y,y,n,l)	3	326	9e−11	68.5		355.4	423.9	1,858
BGS(n,n,n,u,1e−3,3)	4	127	8e−11	38.6		98.7	137.3	1,543
BGS(n,n,n,u,1e−3,3)	5	127	8e−11	41.8		99.2	141.0	1,543
BGS(y,n,n,l,1e−3,3)	1	48	2e−11	46.4		80.0	**126.4**	1,341
BGS(y,n,y,l,1e−3,5)	1	30	1e−11	49.5		81.3	130.8	1,341
BGS(y,y,n,u,1e−3,3)	1	127	3e−11	50.9		128.4	179.3	1,637
BGS(y,y,y,l,1e−3,3)	1	129	9e−11	48.3		138.6	186.8	1,637
BGS(y,n,n,u,1e−10,3)	2	128	9e−11	47.0		161.2	208.2	1,341
BGS(y,y,n,u,1e−10,3)	2	126	9e−11	47.4		104.8	152.2	1,657
BGS(y,y,n,l)	3	121	7e−11	68.8		137.6	206.4	1,858
IADBJ(n,n,n,u,1e−10,3)	4	205	9e−11	38.0	203.6	262.6	504.2	2,218
IADBJ(n,n,n,u,1e−10,3)	5	208	9e−11	39.1	173.7	269.7	482.5	2,218
IADBJ(y,n,n,u,1e−10,3)	1	103	9e−11	42.5	7,215.3	226.0	7,483.8	2,218
IADBGS(n,n,n,u,1e−10,5)	4	105	9e−11	38.8	195.8	143.5	378.1	2,218
IADBGS(n,n,n,u,1e−10,3)	5	106	9e−11	39.2	173.7	139.4	352.3	2,218
IADBGS(y,n,n,l,1e−5,5)	1	113	9e−11	42.2	3,273.3	141.3	3,456.8	2,218

Table 5.34: Solver statistics for the *Webbase* problem when α=0.95.

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		503	1e−10	36.1		312.2	348.3	625
QPOWER		397	9e−11	37.4		276.3	313.7	767
J		508	1e−10	39.1		368.1	407.2	826
GS		205	9e−11	36.8		147.2	183.9	826
BJ(n,n,n,u,1e−5,5)	4	444	1e−10	38.4		340.6	379.0	1,341
BJ(n,n,n,u,1e−5,5)	5	484	1e−10	40.9		519.0	374.5	1,341
BJ(y,n,n,l,1e−3,5)	1	98	8e−11	48.4		456.5	504.9	1,341
BJ(y,n,y,u,1e−3,5)	1	98	8e−11	51.0		455.7	506.7	1,341
BJ(y,y,n,u,1e−3,3)	1	518	8e−11	54.8		903.7	958.6	1,637
BJ(y,y,y,l,1e−3,3)	1	526	1e−10	50.3		976.5	1,026.9	1,637
BJ(y,n,n,l,1e−5,3)	2	463	1e−10	49.2		608.1	657.3	1,341
BJ(y,y,n,l,1e−3,3)	2	505	1e−10	49.8		607.7	657.5	1,657
BJ(y,y,n,l)	3	547	1e−10	67.6		596.2	663.8	1,858
BGS(n,n,n,u,1e−3,3)	4	207	9e−11	39.3		160.6	199.9	1,341
BGS(n,n,n,u,1e−3,3)	5	206	9e−11	43.5		159.4	202.9	1,341
BGS(y,n,n,l,1e−3,3)	1	75	3e−11	47.1		223.3	270.4	1,341
BGS(y,n,y,l,1e−3,3)	1	75	3e−11	48.5		128.4	**176.9**	1,341
BGS(y,y,n,u,1e−3,3)	1	205	6e−11	49.5		372.3	421.9	1,637
BGS(y,y,y,l,1e−3,3)	1	210	9e−11	49.5		394.9	444.4	1,637
BGS(y,n,n,u,1e−3,3)	2	209	9e−11	47.9		279.9	327.9	1,341
BGS(y,y,n,u,1e−3,3)	2	198	9e−11	52.4		255.0	307.4	1,657
BGS(y,y,n,l)	3	196	9e−11	68.4		221.7	290.1	1,858
IADBJ(n,n,n,u,1e−10,3)	4	352	9e−11	38.7	195.7	442.9	677.2	2,218
IADBJ(n,n,n,u,1e−10,3)	5	345	1e−10	40.9	173.3	449.7	663.9	2,218
IADBJ(y,n,n,u,1e−5,5)	1	404	1e−10	44.7	7,307.1	516.7	7,868.6	2,218
IADBGS(n,n,n,u,1e−10,3)	4	171	9e−11	38.6	204.7	219.7	463.0	2,218
IADBGS(n,n,n,u,1e−10,3)	5	171	9e−11	41.1	181.1	226.5	448.7	2,218
IADBGS(y,n,n,l,1e−3,3)	1	205	9e−11	42.4	3,616.7	246.9	3,906.0	2,218

Table 5.35: Solver statistics for the *Webbase* problem when α=0.97.

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		1,000*	2e−08	34.7		623.7	658.4	625
QPOWER		1,000*	2e−10	34.2		700.2	734.4	767
J		1,000*	2e−08	39.0		712.8	751.8	826
GS		583	1e−10	39.4		414.5	453.9	826
BJ(n,n,n,u,1e−5,5)	4	1,000*	2e−09	41.3		765.3	806.6	1,341
BJ(n,n,n,u,1e−3,5)	5	1,000*	2e−08	39.2		775.1	814.3	1,341
BJ(y,n,n,l,1e−3,5)	1	279	1e−10	47.3		673.5	720.8	1,341
BJ(y,n,y,l,1e−3,5)	1	279	1e−10	43.9		670.2	714.2	1,341
BJ(y,y,n,u,1e−3,3)	1	1,000*	3e−08	46.5		1,133.3	1,179.8	1,637
BJ(y,y,y,l,1e−3,3)	1	1,000*	3e−08	47.9		1,235.1	1,283.0	1,637
BJ(y,n,n,l,1e−3,5)	2	1,000*	1e−08	43.3		849.4	892.7	1,341
BJ(y,y,n,u,1e−5,3)	2	1,000*	1e−10	49.4		823.2	872.6	1,657
BJ(y,y,n,u)	3	1,000*	2e−08	69.8		1,099.6	1,169.4	1,858
BGS(n,n,n,u,1e−3,3)	4	587	1e−10	38.4		450.6	489.0	1,341
BGS(n,n,n,u,1e−5,5)	5	592	1e−10	42.3		477.0	519.2	1,341
BGS(y,n,n,l,1e−3,5)	1	131	2e−11	43.9		317.9	**361.8**	1,341
BGS(y,n,y,l,1e−3,5)	1	131	2e−11	43.6		322.5	366.1	1,341
BGS(y,y,n,u,1e−3,3)	1	558	3e−11	48.2		553.9	602.1	1,637
BGS(y,y,y,l,1e−3,3)	1	591	1e−10	48.1		622.5	670.6	1,637
BGS(y,n,n,l,1e−5,5)	2	598	1e−10	40.3		426.9	467.2	1,341
BGS(y,y,n,l,1e−5,5)	2	596	1e−10	43.4		390.7	434.11	1,657
BGS(y,y,n,l)	3	563	1e−10	67.8		631.8	699.6	1,858
IADBJ(n,n,n,u,1e−5,3)	4	1,000*	1e−09	41.5	198.3	1,260.5	1,500.2	2,218
IADBJ(n,n,n,u,1e−5,5)	5	1,000*	7e−10	41.1	173.7	1,345.4	1,560.2	2,218
IADBJ(y,n,n,u,1e−10,5)	1	311	1e−10	43.1	7,337.7	913.4	8,294.2	2,218
IADBGS(n,n,n,u,1e−10,3)	4	479	1e−10	38.8	204.1	632.0	875.0	2,218
IADBGS(n,n,n,u,1e−10,3)	5	475	1e−10	41.1	174.0	655.6	870.7	2,218
IADBGS(y,n,n,l,1e−10,3)	1	183	9e−11	45.5	3,424.4	362.7	3,832.6	2,218

Table 5.36: Solver statistics for the *Webbase* problem when α=0.99.

coefficient matrix to a sparse block representation. The Pe time provides the time to compute and store in sparse representation the block column sums of the coefficient matrix at the outset for IADBJ and IADBGS. This piece of data is later used over and over again in computing the aggregated matrix at each iteration of these solvers. Column Solu gives the time spent by the particular solver during iteration. Column Total provides the total time spent by the solver, and is therefore the sum of the values in the previous three columns. The memory requirement of each solver in megabytes is provided in column nine (MB). Note that, this column includes space for nonzeros in the matrix and its transpose. Memory requirement for IAD solvers based on partitionings 1, 4, and 5 is calculated assuming that, there are 1,000,000 nonzero elements in the aggregated matrix and for partitioning 3, it is calculated for 250,000 nonzero elements in the aggregated matrix. Memory requirement of the block column sums of the coefficient matrix in sparse representation is calculated for the number of nonzero elements in the matrix. Finally, the value of α used for the particular problem is given in the caption of the table.

5.3.2 Experiments with Irreducible Matrices

Results for the 6 irreducible test matrices are presented in Tables 5.49–5.54 as in the previous section after data pertaining to the resulting nonzero structures under the assumed block partitionings are provided in Tables 5.38–5.47. For partitionings 3, 4, and 5 nonzero structure is given only for orientation parameter u, since given information on nonzero structure is the same for orientation parameter l. Note that, in the experiments performed for the *Ncd* problem *maxit* is set to 5,000. IAD solvers based on partitioning 1 are not applied to irreducible test

matrices since they yield an aggregated matrix of order 2.

Part	Para	nb	Max_n	Min_n	Ave_n	Max_{nz}	Min_{nz}	Ave_{nz}	nz_{off}
1	n,y,y,l	2	7,583	7,583	8,320.5	19,452	13,799	16,625.5	32,798
1	n,y,y,u	2	7,583	7,583	8,320.5	19,452	13,799	16,625.5	32,798
1	n,y,n,l	2	7,583	7,583	8,320.5	19,452	13,799	16,625.5	32,798
1	n,y,n,u	2	7,583	7,583	8,320.5	19,452	13,799	16,625.5	32,798
2	n,y,n,l	2	7,583	7,583	8,320.5	19,452	13,799	16,625.5	32,798
2	n,y,n,u	2	7,583	7,583	8,320.5	19,452	13,799	16,625.5	32,798
3	y,y,n,u	3	9,058	90	5,547.0	19,452	90	11,019.0	32,992
4	n,n,n,u	129	129	129	129.0	461	257	29,815.0	27,588
5	n,n,n,u	182	181	1	91.4	645	1	19,473.0	30,608

Table 5.37: Nonzero structure of the *2D* problem for all partitionings.

Part	Para	$n_{T_{00}}$	$nz_{T_{00}}$	n_T	nz_{rem}
1	n,y,y,l	9,058	19,452	7,583	13,799
1	n,y,y,u	9,058	19,452	7,583	13,799
1	n,y,n,l	9,058	19,452	7,583	13,799
1	n,y,n,u	9,058	19,452	7,583	13,799
2	n,y,n,l	7,583	13,799	7,583	19,452
2	n,y,n,u	7,583	13,799	7,583	19,452

Table 5.38: Other information on nonzero structure of the *2D* problem with partitionings 1 and 2.

Part	Para	nb	Max_n	Min_n	Ave_n	Max_{nz}	Min_{nz}	Ave_{nz}	nz_{off}
1	n,y,y,l,	2	15,149	15,149	10,150.5	75,144	5,155	40,149.5	60,205
1	n,y,y,u	2	15,149	15,149	10,150.5	75,144	5,155	40,149.5	60,205
1	n,y,n,l	2	15,149	15,149	10,150.5	75,144	5,155	40,149.5	60,205
1	n,y,n,u	2	15,149	15,149	10,150.5	75,144	5,155	40,149.5	60,205
2	n,y,n,l	2	15,149	15,149	10,150.5	75,144	5,155	40,149.5	60,205
2	n,y,n,u	2	15,149	15,149	10,150.5	75,144	5,155	40,149.5	60,205
3	y,y,n,u	5	5,152	99	4,060.2	5,551	99	4,061.0	120,199
4	n,n,n,u	143	142	137	142.0	872	409	513.2	67,119
5	n,n,n,u	201	201	1	101.0	601	1	301.0	80,003

Table 5.39: Nonzero structure of the *Easy* problem for all partitionings.

5.3.3 Comparison of Solvers

When we look at the experimental results for web matrices, we see that in seven matrices GS is producing minimum solution time, since it does not have much overhead associated with preprocessing. GS produces minimum solution time for

Part	Para	$n_{T_{00}}$	$nz_{T_{00}}$	n_T	nz_{rem}
1	n,y,y,l	5,152	5,155	15,149	75,144
1	n,y,y,u	515 2	5,155	15,149	75,144
1	n,y,n,l	5,152	5,155	15,149	75,144
1	n,y,n,u	5,152	5,155	15,149	75,144
2	n,y,n,l	15,149	75,144	15,149	5,155
2	n,y,n,u	15,149	75,144	15,149	5,155

Table 5.40: Other information on nonzero structure of the *Easy* problem with partitionings 1 and 2.

Part	Para	nb	Max_n	Min_n	Ave_n	Max_{nz}	Min_{nz}	Ave_{nz}	nz_{off}
1	n,y,y,l	2	10,245	10,245	10,245.5	10,305	10,246	10,275.5	80,490
1	n,y,y,u	2	10,245	10,245	10,245.5	10,305	10,246	10,275.5	80,490
1	n,y,n,l	2	10,245	10,245	10,245.5	10,305	10,246	10,275.5	80,490
1	n,y,n,u	2	10,245	10,245	10,245.5	10,305	10,246	10,275.5	80,490
2	n,y,n,l	2	10,245	10,245	10,245.5	10,305	10,246	10,275.5	80,490
2	n,y,n,l	2	10,245	10,245	10,245.5	10,305	10,246	10,275.5	80,490
3	y,y,n,u	3	10,246	15	6,830.3	19,452	90	11,019.0	32,992
4	n,n,n,u	144	143	42	142.3	654	179	585.9	16,665
5	n,n,n,u	202	201	1	101.4	922	1	394.0	21,448

Table 5.41: Nonzero structure of the *Telecom* problem for all partitionings.

α values less than or equal to 0.95. It decreases number of iterations to convergence about 30 to 50 percent, decreases solution time about 15 to 70 percent without increasing space requirement in comparison to QPOWER. In six matrices BGS with partitioning 4 produces minimum solution time. This partitioning yields diagonal blocks with almost equal order and a relatively small number of nonzeros in the off-diagonal blocks. BGS with partitionings 4 and 5 have similar behaviour in number of iterations to convergence, solution time and space

Part	Para	$n_{T_{00}}$	$nz_{T_{00}}$	n_T	nz_{rem}
1	n,y,y,l	10,246	10,246	10,245	10,305
1	n,y,y,u	10,246	10,246	10,245	10,305
1	n,y,n,l	10,246	10,246	10,245	10,305
1	n,y,n,u	10,246	10,246	10,245	10,305
2	n,y,n,l	10,245	10,305	10,245	10,246
2	n,y,n,u	10,245	10,305	10,245	10,246

Table 5.42: Other information on nonzero structure of the *Telecom* problem with partitionings 1 and 2.

Part	Para	nb	Max_n	Min_n	Ave_n	Max_{nz}	Min_{nz}	Ave_{nz}	nz_{off}
1	n,y,y,l	2	11,585	11,585	11,713.0	13,983	11,841	12,912.0	130,202
1	n,y,y,u	2	11,585	11,585	11,713.0	13,983	11,841	12,912.0	130,202
1	n,y,n,l	2	11,585	11,585	11,713.0	13,983	11,841	12,912.0	130,202
1	n,y,n,u	2	11,585	11,585	11,713.0	13,983	11,841	12,912.0	130,202
2	n,y,n,l	2	11,585	11,585	11,713.0	13,983	11,841	12,912.0	130,202
2	n,y,n,u	2	11,585	11,585	11,713.0	13,983	11,841	12,912.0	130,202
3	y,y,n,u	5	11,841	11	4,685.2	11,841	11	4,685.2	132,600
4	n,n,n,u	154	153	17	152.1	811	17	181.6	128,056
5	n,n,n,u	216	215	1	108.5	732	1	115.6	131,050

Table 5.43: Nonzero structure of the *Ncd* problem for all partitionings.

Part	Para	$n_{T_{00}}$	$nz_{T_{00}}$	n_T	nz_{rem}
1	n,y,n,l	11,841	11,841	11,585	13,983
1	n,y,y,u	11,841	11,841	11,585	13,983
1	n,y,n,l	11,841	11,841	11,585	13,983
1	n,y,n,u	11,841	11,841	11,585	13,983
2	n,y,n,l	11,585	13,983	11,585	11,841
2	n,y,n,u	11,585	13,983	11,585	11,841

Table 5.44: Other information on nonzero structure of the *Ncd* problem with partitionings 1 and 2.

Part	Para	nb	Max_n	Min_n	Ave_n	Max_{nz}	Min_{nz}	Ave_{nz}	nz_{off}
1	n,y,y,l,	2	16,384	16,384	19,601.5	22,819	16,384	19,601.5	524,288
1	n,y,y,u	2	16,384	16,384	19,601.5	22,819	16,384	19,601.5	524,288
1	n,y,n,l	2	16,384	16,384	19,601.5	22,819	16,384	19,601.5	524,288
1	n,y,n,u	2	16,384	16,384	19,601.5	22,819	16,384	19,601.5	524,288
2	n,y,n,l	2	16,384	16,384	19,601.5	22,819	16,384	19,601.5	524,288
2	n,y,n,u	2	16,384	16,384	19,601.5	22,819	16,384	19,601.5	524,288
3	y,y,n,u	2	22,819	16,384	19,601.5	22,819	16,384	19,601.5	524,288
4	n,n,n,u	198	394	197	198.0	1,069	197	202.4	523,416
5	n,n,n,u	280	279	1	140.0	279	1	140.0	524,288

Table 5.45: Nonzero structure of the *Mutex* problem for all partitionings.

Part	Para	$n_{T_{00}}$	$nz_{T_{00}}$	n_T	nz_{rem}
1	n,y,y,l	22,819	22,819	16,384	16,384
1	n,y,y,u	22,819	22,819	16,384	16,384
1	n,y,n,l	22,819	22,819	16,384	16,384
1	n,y,n,u	22,819	22,819	16,384	16,384
2	n,y,n,l	16,384	16,384	16,384	22,819
2	n,y,n,u	16,384	16,384	16,384	22,819

Table 5.46: Other information on nonzero structure of the *Mutex* problem with partitionings 1 and 2.

Part	Para	nb	Max_n	Min_n	Ave_n	Max_{nz}	Min_{nz}	Ave_{nz}	nz_{off}
1	n,y,y,l	2	50,970	50,970	52,312.5	159,288	147,693	153,490.5	286,134
1	n,y,y,u	2	50,970	50,970	52,312.5	159,288	147,693	153,490.5	286,134
1	n,y,n,l	2	50,970	50,970	52,312.5	159,288	147,693	153,490.5	286,134
1	n,y,n,u	2	50,970	50,970	52,312.5	159,288	147,693	153,490.5	286,134
2	n,y,n,l	2	50,970	50,970	52,312.5	159,288	147,693	153,490.5	286,134
2	n,y,n,u	2	50,970	50,970	52,312.5	159,288	147,693	153,490.5	286,134
3	y,y,n,u	4	53,655	89	26,156.3	159,288	92	71,242.0	308,147
4	n,n,n,u	324	323	296	322.9	1,394	323	333.7	484,994
5	n,n,n,u	457	456	1	228.9	1,416	1	332.4	486,900

Table 5.47: Nonzero structure of the *Qnatm* problem for all partitionings.

Part	Para	$n_{T_{00}}$	$nz_{T_{00}}$	n_T	nz_{rem}
1	n,y,y,l	53,655	159,288	50,970	147,693
1	n,y,y,u	53,655	159,288	50,970	147,693
1	n,y,n,l	53,655	159,288	50,970	147,693
1	n,y,n,u	53,655	159,288	50,970	147,693
2	n,y,n,l	50,970	147,693	50,970	159,288
2	n,y,n,u	50,970	147,693	50,970	159,288

Table 5.48: Other information on nonzero structure of the *Qnatm* problem with partitionings 1 and 2.

requirement. BGS with partitioning 4 decreases number of iterations to convergence about 40 to 60 percent, decreases solution time about 15 to 40 percent with respect to QPOWER. When only Tarjan's algorithm is employed, BGS with partitioning 1 produces minimum solution time in five matrices. This partitioning also yields smaller number of nonzeros in the off-diagonal blocks compared to other partitionings. It decreases number of iterations to convergence about 80 to 95 percent, decreases solution time about 15 to 40 percent with respect to QPOWER. BGS with partitioning 4 increases space requirement over that of QPOWER by about 60 percent. As α increases all of these solvers become more advantageous in terms of number of iterations to convergence and solution time compared to QPOWER.

A surprising phenomenon is that in the *Stanford* problem time per iteration for BGS with partitioning 3 is shorter than GS, although the same number of

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		1,000*	2e−05	0.2		2.1	2.3	2
QPOWER		1,000*	4e−06	0.2		1.9	2.1	3
J		1,000*	4e−06	0.2		2.9	3.1	2
GS		1,000*	9e−12	0.2		3.5	3.7	2
BJ(n,n,n,u,1e−10,5)	4	1,000*	4e−11	0.2		5.2	5.4	3
BJ(n,n,n,u,1e−10,5)	5	1,000*	2e−11	0.2		5.4	5.6	3
BJ(n,y,n,u,1e−5,5)	1	1,000*	8e−07	0.2		7.7	7.9	4
BJ(n,y,y,u,1e−5,5)	1	1,000*	8e−07	0.2		7.7	7.9	4
BJ(n,y,n,u,1e−5,3)	2	1,000*	1e−07	0.2		3.5	3.7	4
BJ(n,y,y,u,1e−5,5)	2	1,000*	1e−07	0.2		3.5	3.7	4
BJ(y,y,n,u)	3	1,000*	3e−07	0.2		3.6	4.0	4
BGS(n,n,n,u,1e−10,5)	4	455	5e−12	0.2		2.2	2.4	3
BGS(n,n,n,u,1e−10,5)	5	380	5e−12	0.2		2.0	2.3	3
BGS(n,y,n,u,1e−5,3)	1	1,000*	8e−11	0.2		6.5	6.7	4
BGS(n,y,y,u,1e−3,3)	1	1,000*	8e−11	0.2		6.5	6.7	4
BGS(n,y,n,u,1e−3,3)	2	1,000*	3e−11	0.2		3.8	4.0	4
BGS(n,y,y,u,1e−5,5)	2	1,000*	3e−11	0.2		3.7	3.9	4
BGS(y,y,n,l)	3	1,000*	6e−11	0.2		3.8	4.1	4
IADBJ(n,n,n,u,1e−3,3)	4			0.2	failed (reducible coupling matrix)			12
IADBJ(n,n,n,u,1e−10,3)	5			0.2	failed (reducible coupling matrix)			12
IADBJ(y,y,n,l)	3	1,000*	3e−07	0.2	0.0	4.8	5.0	7
IADBGS(n,n,n,u,1e−3,3)	4			0.2	failed (reducible coupling matrix)			12
IADBGS(n,n,n,u,1e−10,5)	5	86	5e−12	0.2	0.0	0.3	**0.5**	12
IADBGS(y,y,n,u)	3	1,000*	9e−11	0.2	0.0	5.2	5.4	7

Table 5.49: Solver statistics for the *2D* problem.

floating point operations are performed. Although BGS with partitioning 3 is the winner consistently in the *Stanford* problem, IAD with partitioning 3 provides mostly large solution times. The main reason behind this is the relatively large computation time of the aggregated matrix, which increases dramatically with the current implementation based on routines from the MARCA package [36] as the number of diagonal blocks increases. Also solution time of the aggregated matrix with GTH increases dramatically as the order of the aggregated matrix increases. In none of the problems, POWER and QPOWER are winners. IAD method with partitioning 1 when only Tarjan's algorithm is employed produces minimum number of iterations and solver time (i.e., Solu) for all web matrices

Solver	Part	Iter	Res	Time Prep	Pe	Solu	Total	MB
POWER		829	9e−11	0.4		5.6	6.0	4
QPOWER		533	6e−11	0.4		3.3	3.7	5
J		1,000*	5e−05	0.4		8.1	8.4	4
GS		31	7e−13	0.4		0.1	**0.5**	4
BJ(n,n,n,u,1e−10,5)	4	1,000*	7e−07	0.4		11.2	11.6	5
BJ(n,n,n,u,1e−10,5)	5	1,000*	1e−06	0.5		11.2	11.7	5
BJ(n,y,n,u,1e−5,3)	1	296	5e−12	0.4		2.0	2.4	6
BJ(n,y,y,u,1e−5,3)	1	296	5e−12	0.4		2.0	2.4	6
BJ(n,y,n,u,1e−10,3)	2	279	5e−12	0.4		0.9	1.3	6
BJ(n,y,y,u,1e−10,3)	2	279	5e−12	0.4		0.9	1.3	6
BJ(y,y,n,l)	3	1,000*	5e−05	0.5		8.3	8.8	7
BGS(n,n,n,u,1e−10,3)	4	12	3e−13	0.4		0.1	**0.5**	5
BGS(n,n,n,u,1e−10,3)	5	12	4e−13	0.5		0.1	0.6	5
BGS(n,y,n,l,1e−5,3)	1	124	1e−12	0.4		1.1	1.5	6
BGS(n,y,y,u,1e−10,3)	1	124	1e−12	0.4		1.1	1.5	6
BGS(n,y,n,u,1e−10,5)	2	125	2e−12	0.4		0.4	0.8	6
BGS(n,y,y,u,1e−3,3)	2	125	2e−12	0.4		0.4	0.8	6
BGS(y,y,n,l)	3	162	2e−12	0.5		0.6	1.1	7
IADBJ(n,n,n,u,1e−10,5)	4			0.4	failed (reducible coupling matrix)			16
IADBJ(n,n,n,u,1e−3,5)	5			0.4	failed (reducible coupling matrix)			16
IADBJ(y,y,n,u)	3	314	4e−12	0.5	2.5	1.7	4.7	11
IADBGS(n,n,n,u,1e−3,5)	4			0.4	failed (reducible coupling matrix)			16
IADBGS(n,n,n,u,1e−5,5)	5			0.4	failed (reducible coupling matrix)			16
IADBGS(y,y,n,u)	3	162	1e−12	0.5	2.5	0.8	3.9	11

Table 5.50: Solver statistics for the *Easy* problem.

when inner tolerance of 10^{-10} and number of inner iterations of 5 are used. BGS with partitioning 3 produces smaller number of iterations than GS in all matrices.

Using Rosen's algorithm shortens time per iteration due to the direct solution of triangular block(s) at the expense of increased number of iterations. As the order of the problem increases, preprocessing time for partitioning 3 also increases due to the relatively large number of recursive calls performed. Note that permutation obtained from a partitioning does not change when different personalization vectors are used for a matrix, since the nonzero structure of the matrix does not change. That is, computation of the permutation vector for a

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		1,000*	9e−07	0.3		2.4	2.7	3
QPOWER		1,000*	9e−07	0.3		1.8	2.1	4
J		1,000*	2e−05	0.3		2.1	2.4	3
GS		1,000*	1e−06	0.3		2.0	2.3	3
BJ(n,n,n,u,1e−3,3)	4	1,000*	2e−06	0.3		2.9	3.2	4
BJ(n,n,n,u,1e−10,3)	5	1,000*	2e−07	0.3		4.9	5.2	4
BJ(n,y,n,l,1e−3,3)	1	1,000*	2e−05	0.3		3.6	3.9	5
BJ(n,y,y,u,1e−5,3)	1	1,000*	2e−05	0.3		3.7	4.0	5
BJ(n,y,n,u,1e−10,5)	2	1,000*	2e−05	0.3		2.5	2.8	5
BJ(n,y,y,l,1e−3,3)	2	1,000*	2e−05	0.3		2.5	2.8	5
BJ(y,y,n,u)	3	1,000*	2e−05	0.3		2.6	2.9	6
BGS(n,n,n,u,1e−3,5)	4	1,000*	9e−07	0.3		2.8	3.1	4
BGS(n,n,n,u,1e−5,5)	5	1,000*	1e−06	0.3		2.9	3.2	4
BGS(n,y,n,l,1e−3,3)	1	1,000*	1e−06	0.3		3.7	4.0	5
BGS(n,y,y,u,1e−3,3)	1	1,000*	1e−06	0.3		3.7	4.0	5
BGS(n,y,n,l,1e−10,5)	2	1,000*	2e−06	0.3		2.5	2.8	5
BGS(n,y,y,l,1e−3,3)	2	1,000*	2e−06	0.3		2.5	2.8	5
BGS(y,y,n,u)	3	1,000*	1e−06	0.3		2.5	2.8	6
IADBJ(n,n,n,u,1e−3,3)	4	1,000*	5e−04	0.3	0.0	3.5	3.8	14
IADBJ(n,n,n,u,1e−3,3)	5	31	6e−13	0.3	0.0	0.1	**0.2**	14
IADBJ(y,y,n,l)	3	1,000*	1e−05	0.3	0.0	3.1	3.5	9
IADBGS(n,n,n,u,1e−10,5)	4	69	3e−13	0.3	0.0	0.2	0.3	14
IADBGS(n,n,n,u,1e−3,3)	5	26	3e−13	0.3	0.0	0.1	**0.2**	14
IADBGS(y,y,n,l)	3	1,000*	2e−06	0.3	0.0	3.2	3.5	9

Table 5.51: Solver statistics for the *Telecom* problem.

partitioning only once is enough. When using Rosen's algorithm, partitioning 2 almost always performs better than partitioning 1. Restricting the number of diagonal blocks to 2 does not seem to have a significant effect on solution time. As α increases, number of iterations to convergence increases, which is inline with expectations.

IADBJ with partitioning 3 decreases the number of iterations compared to BJ with partitioning 3. Zhu et al. also reported 5-7 times faster convergence than power method using the IAD method with BJ smoothing [38]. However, the number of iterations performed by BGS with partitioning 3 and that of IADBGS

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		5,000*	2e−07	0.5		14.8	15.3	5
QPOWER		5,000*	3e−06	0.5		12.1	12.6	6
J		5,000*	7e−05	0.5		13.6	14.1	5
GS		5,000*	9e−07	0.5		13.6	14.1	5
BJ(n,n,n,u,1e−5,5)	4	5,000*	2e−05	0.5		20.1	20.6	6
BJ(n,n,n,u,1e−3,3)	5	5,000*	4e−05	0.5		18.8	19.3	6
BJ(n,y,n,u,1e−5,3)	1	5,000*	8e−05	0.5		24.8	25.3	7
BJ(n,y,y,u,1e−5,3)	1	5,000*	8e−05	0.5		24.5	25.1	7
BJ(n,y,n,l,1e−3,3)	2	5,000*	7e−05	0.5		17.1	17.6	7
BJ(n,y,y,l,1e−10,5)	2	5,000*	7e−05	0.5		16.9	17.4	7
BJ(y,y,n,l)	3	5,000*	7e−05	0.5		16.2	16.7	8
BGS(n,n,n,u,1e−5,5)	4	5,000*	2e−07	0.5		18.4	18.9	6
BGS(n,n,n,u,1e−3,3)	5	5,000*	8e−07	0.5		18.2	18.7	6
BGS(n,y,n,u,1e−3,3)	1	5,000*	9e−07	0.5		19.2	19.7	7
BGS(n,y,y,u,1e−10-3)	1	5,000*	9e−07	0.5		22.6	23.1	7
BGS(n,y,n,u,1e−5,3)	2	5,000*	9e−07	0.5		16.6	17.1	7
BGS(n,y,y,l,1e−3,5)	2	5,000*	9e−07	0.5		16.4	16.9	7
BGS(y,y,n,l)	3	5,000*	9e−07	0.5		16.1	16.6	8
IADBJ(n,n,n,u,1e−10,5)	4	1,556	2e−12	0.5	0.0	9.8	10.3	17
IADBJ(n,n,n,u,1e−3,5)	5	5,000*	1e−04	0.5	0.0	41.5	42.0	17
IADBJ(y,y,n,l)	3	5,000*	1e−05	0.5	0.2	20.0	20.7	12
IADBGS(n,n,n,u,1e−10,5)	4	1,332	8e−13	0.5	0.0	8.5	**9.0**	17
IADBGS(n,n,n,u,1e−3,5)	5	5,000*	3e−09	0.5	0.0	41.1	41.6	17
IADBGS(y,y,n,l)	3	5,000*	9e−07	0.5	0.2	21.1	21.8	12

Table 5.52: Solver statistics for the *Ncd* problem.

with partitioning 3 are almost always identical. This is another surprising phenomenon we have not been able to explain. Note that IADBJ may be parallelized to achieve shorter solution time.

Generally, 3 inner iterations and an inner tolerance of 10^{-10} give better solution times for iterative methods based on block partitionings. As the number of nonzeros in the matrix increases and the number of dangling nodes decreases, increasing the inner tolerance yields better results. However, performing as many as 10 inner iterations does not have a positive effect on solution time.

The POWER solver has the minimum space requirement, whereas the IAD

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		745	1e−10	1.5		5.1	6.6	15
QPOWER		155	8e−11	1.7		1.2	2.9	17
J		1,000*	5e−03	1.6		7.5	9.1	15
GS		27	9e−13	1.6		0.2	**1.8**	15
BJ(n,n,n,u,1e−10,5)	4	422	3e−12	1.7		4.0	5.7	17
BJ(n,n,n,u,1e−3,3)	5	1,000*	5e−03	1.7		9.2	10.9	17
BJ(n,y,n,l,1e−10,3)	1	96	2e−12	1.7		1.1	2.9	20
BJ(n,y,y,u,1e−5,3)	1	96	2e−12	1.7		1.1	2.8	20
BJ(n,y,n,l,1e−5,3)	2	96	3e−12	1.7		0.8	2.5	20
BJ(n,y,y,l,1e−5,3)	2	96	3e−12	1.7		0.8	2.5	20
BJ(y,y,n,l)	3	1,000*	5e−03	1.8		8.4	10.2	23
BGS(n,n,n,u,1e−10,3)	4	20	1e−12	1.6		0.2	**1.8**	17
BGS(n,n,n,u,1e−3-3)	5	27	9e−13	2.0		0.4	2.4	17
BGS(n,y,n,u,1e−5,3)	1	23	8e−13	1.7		0.3	2.0	20
BGS(n,y,y,u,1e−5,3)	1	23	8e−13	1.7		0.3	2.0	20
BGS(n,y,n,u,1e−3,5)	2	24	8e−13	1.7		0.2	1.9	20
BGS(n,y,y,l,1e−5,5)	2	24	8e−13	1.7		0.2	1.9	20
BGS(y,y,n,l)	3	24	8e−13	1.8		0.2	2.1	23
IADBJ(n,n,n,u,1e−10,5)	4	307	3e−12	1.6	0.9	38.5	41.0	34
IADBJ(n,n,n,u,1e−3,3)	5	23	8e−13	1.1	0.7	3.4	5.2	34
IADBJ(y,y,n,u)	3	46	2e−12	1.8	14.8	0.5	17.1	30
IADBGS(n,n,n,u,1e−10,3)	4	12	2e−12	1.6	0.9	1.5	4.0	34
IADBGS(n,n,n,u,1e−3,3)	5	14	7e−13	1.4	0.8	2.1	4.3	34
IADBGS(y,y,n,u)	3	23	8e−13	1.8	14.8	0.3	16.9	30

Table 5.53: Solver statistics for the *Mutex* problem.

solvers have the highest space requirement when Rosen's algorithm is employed. BGS with partitioning 4 and partitioning 1 when only Tarjan's algorithms is employed increases space requirement over that of QPOWER by about 60 percent. Space requirement of GS and QPOWER are about the same.

Observations for the irreducible set of matrices revealed that IAD with partitioning 5 is the timewise winner in the *2D* and *Telecom* problems. IAD with partitioning 4 is the winner in the *Ncd* problem. In the *Easy* and *Mutex* problems there is a tie between GS and BGS with partitioning 4. In the *Qnatm* problem, the winner is GS. Since the order of the aggregated matrix is smaller in this set

				Time				
Solver	Part	Iter	Res	Prep	Pe	Solu	Total	MB
POWER		849	9e−11	1.4		9.5	10.9	18
QPOWER		702	9e−11	1.5		9.8	11.3	24
J		1,000*	1e−06	1.4		12.8	14.2	19
GS		256	3e−11	1.4		3.2	**4.7**	19
BJ(n,n,n,u,1e−10,5)	4	607	2e−11	1.4		12.9	14.3	24
BJ(n,n,n,u,1e−3,5)	5	1,000*	1e−06	1.5		18.0	19.5	24
BJ(n,y,n,u,1e−5,3)	1	493	3e−11	1.6		15.9	17.6	30
BJ(n,y,y,u,1e−3,3)	1	493	3e−11	1.6		15.8	17.4	30
BJ(n,y,n,u,1e−3,5)	2	458	3e−11	1.6		8.2	9.8	30
BJ(n,y,y,u,1e−10,5)	2	458	3e−11	1.6		8.1	9.7	30
BJ(y,y,n,l)	3	478	3e−11	1.9		10.5	12.4	33
BGS(n,n,n,u,1e−3,3)	4	249	3e−11	1.4		4.5	6.0	24
BGS(n,n,n,u,1e−3,3)	5	256	3e−11	2.0		5.9	7.8	24
BGS(n,y,n,l,1e−10,3)	1	218	3e−11	1.6		6.3	8.0	30
BGS(n,y,y,u,1e−10,3)	1	218	3e−11	1.6		6.4	8.0	30
BGS(n,y,n,u,1e−5,3)	2	256	3e−11	1.6		4.6	6.2	30
BGS(n,y,y,u,1e−5,3)	2	256	3e−11	1.7		4.5	6.2	30
BGS(y,y,n,l)	3	247	2e−11	1.9		5.5	7.4	33
IADBJ(n,n,n,u,1e−10,5)	4	336	5e−11	1.4	0.4	12.9	14.7	44
IADBJ(n,n,n,u,1e−3,5)	5	276	6e−11	1.5	0.4	13.9	15.9	44
IADBJ(y,y,n,u)	3	476	3e−11	2.0	12.9	13.3	28.2	41
IADBGS(n,n,n,u,1e−10,3)	4	103	2e−11	1.4	0.4	3.8	5.6	44
IADBGS(n,n,n,u,1e−3,5)	5	107	2e−11	1.5	0.4	5.4	7.4	44
IADBGS(y,y,n,l)	3	246	3e−11	1.9	12.8	6.9	21.6	41

Table 5.54: Solver statistics for the *Qnatm* problem.

of problems, this result is inline with the expectations.

Chapter 6

Conclusion

In this thesis, we propose iterative methods based on various block partitionings, including those with triangular diagonal blocks obtained using cutsets, for the computation of the steady-state vector of Google-like stochastic matrices.

The proposed iterative methods together with power and quadratically extrapolated power methods are coded into a software tool. The effect of the proposed block partitioning algorithms are analyzed through a set of numerical experiments in which the solution times and space requirements are reported. Experimental results on benchmark matrices show that in some cases number of iterations to convergence and solution times decrease with respect to power and quadratically extrapolated power methods. Generally, the space requirement for point methods is 1.3 times, for block methods is 2.5 times, and for iterative aggregation-disaggregation methods is 3.5 times the space requirement of the power method. Note that the space required for quadratically extrapolated power method is 1.3 times the space required by the power method.

Among the iterative methods based on block partitionings, generally the best results in the more difficult problems are obtained with the block Gauss-Seidel solver based on Tarjan's ordering algorithm or equally sized blocks. For easier problems, Gauss-Seidel is recommended.

Although the proposed methods achieve fast PageRank computation, there is still room for improvement. Preprocessing time for the partitioning methodology proposed for obtaining triangular diagonal blocks may increase as the order of the matrix increases due to the number of recursive calls. As future work different implementations of this partitioning methodology should be investigated. For iterative aggregation-disaggregation, the routines that implement the computation of the block column sums of the coefficient matrix can be implemented. As an extension, different strategies for handling dangling nodes can be investigated. For example, dangling nodes may be lumped into a single node and the PageRank of non-dangling nodes may be computed separately.

Bibliography

[1] Harwell Subroutine Library, 2007. Available from http://www.cse.clrc.ac.uk/nag/hsl/.

[2] Larbin home page, 2007. Available from http://larbin.sourceforge.net/index-eng.html/.

[3] Webgraph, 2007. Available from http://webgraph.dsi.unimi.it/.

[4] The Stanford Webbase Project, 2007. Available from http://dbpubs.stanford.edu:8091/~testbed/doc2/WebBase.

[5] Netlib Repository, 2007. Available from http://www.netlib.org.

[6] C. Brezinski and M. Redivo-Zaglia, *The PageRank vector: Properties, computation, approximation, and acceleration*, SIAM Journal on Matrix Analysis and Applications, 28 (2006), pp. 551–575.

[7] S. Brin and L. Page, *The anatomy of a large-scale hypertextual web search engine*, Computer Networks and ISDN Systems, 33 (1998), pp. 107–117.

[8] T. Davis. University of Florida Sparse Matrix Collection, 2007. NA Digest, 92, no. 42, October 16, 1994, NA Digest, 96, no. 28, July

23, 1996, and NA Digest, 97, no. 23, June 7, 1997. Available from http://www.cise.ufl.edu/research/sparse/matrices.

[9] T. Dayar. Technical Report BU-CE-0701, Department of Computer Engineering, Bilkent University, Ankara, Turkey, January 2007. Available from http://www.cs.bilkent.edu.tr/tech-reports/2007/BU-CE-0701.pdf.

[10] T. Dayar and G. N. Noyan. Software for steady-state analysis of Google-like stochastic matrices, 2007. Available from http://www.cs.bilkent.edu.tr/~tugrul/software.html.

[11] T. Dayar and W. J. Stewart, *Comparison of partitioning techniques for two-level iterative solvers on large, sparse Markov chains*, SIAM Journal on Scientific Computing, 21 (2000), pp. 1691–1705.

[12] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Linear Systems*, Oxford University Press, New York, NY, 1986.

[13] I. S. Duff and J. K. Reid, *An implementation of Tarjan's algorithm for the block triangularization of a matrix*, ACM Transactions on Mathematical Software, 4 (1978), pp. 137–147.

[14] P. Festa, P. M. Pardalos, and M. G. C. Resende, *Feedback set problems*, in Handbook of Combinatorial Optimization, D. Z. Du and P. M. Parlados, eds., vol. 4, Boston, MA, 1999, Kluwer Academic Publishers, pp. 209–258.

[15] J.-M. Fourneau and F. Quessete, *Graphs and stochastic automata networks*, in Computations with Markov Chains: Proceedings of the 2nd International Workshop on the Numerical Solution of Markov Chains, W. Stewart, ed., Raleigh, NC, 1995, pp. 217–236.

[16] G. H. Golub and C. D. Meyer, *Using the QR factorization and group inversion to compute, differentiate, and estimate the sensitivity of stationary probabilities for Markov chains*, SIAM Journal on Algebraic and Discrete Methods, 7 (1986), pp. 273–281.

[17] I. C. F. Ipsen and S. Kirkland, *Convergence analysis of a PageRank updating algorithm by Langville and Meyer*, SIAM Journal on Matrix Analysis and Applications, 27 (2006), pp. 952–967.

[18] I. C. F. Ipsen and T. M. Selee, *PageRank computation, with special attention to dangling nodes*, to appear in SIAM Journal on Matrix Analysis and Applications, (2007).

[19] S. D. Kamvar, T. H. Haveliwala, C. D. Manning, and G. H. Golub, *Extrapolation methods for accelerating PageRank computations*, in Proceedings of the 12th international conference on World Wide Web, Budapest, Hungary, 2003, pp. 261–270.

[20] R. M. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., New York, NY, 1972, Plenum Press, pp. 85–103.

[21] A. N. Langville and C. D. Meyer, *Deeper inside PageRank*, Internet Mathematics, 1 (2004), pp. 335–380.

[22] A. N. Langville and C. D. Meyer, *The use of the linear algebra by web search engines*, IMAGE Newsletter, 33 (2004), pp. 2–6.

[23] A. N. Langville and C. D. Meyer, *A survey of eigenvalue methods for web information retrival*, SIAM Review, 47 (2005), pp. 135–161.

[24] A. N. Langville and C. D. Meyer, *Updating Markov chains with an eye on Google's PageRank*, SIAM Journal on Matrix Analysis and Applications, 27 (2006), pp. 968–987.

[25] R. Lempel and S. Moran, *The stochastic approach for link-structure analysis (SALSA) and the TKC effect*, Computer Networks, 33 (2000), pp. 387–401.

[26] I. Marek and D. B. Szyld, *Algebraic Schwarz methods for the numerical solution of Markov chains*, Linear Algebra and its Applications, 386 (2004), pp. 67–81.

[27] A. MERİÇ, *Kronecker representation and decompositional analysis of closed queueing networks with phasetype service distributions and arbitrary buffer sizes*, Master's thesis, Bilkent University, (2007).

[28] C. D. Meyer, *Matrix Analysis and Applied Linear Algebra*, SIAM Press, Philadelphia, PA, 2000.

[29] I. Pultarova, *Tarjan's algorithm in computing PageRank*, in In 13th Annual Conference Proceedings of Technical Computing Prague, C. Moler, A. Procházka, and B. Walden, eds., Praha, 2005, VŠCHT.

[30] B. K. Rosen, *Robust linear algorithms for cutsets*, Journal of Algorithms, 3 (1982), pp. 205–217.

[31] S. M. Ross, *Introduction to Probability Models*, Academic Press, Boston, 1989.

[32] Y. Saad, *Iterative Methods for Sparse Linear Systems*, PWS, Boston, 2nd edition, 2003.

[33] P. Semal, *Analysis of Large Markov Models, Bounding Techniques and Applications*, PhD thesis, University of Louvain, Belgium, 1992.

[34] S. Serra-Capizzano, *Jordan canonical form of the Google matrix: A potential contribution to the PageRank computation*, SIAM Journal on Matrix Analysis and Applications, 27 (2005), pp. 305–312.

[35] A. Shamir, *A linear time algorithm for finding minimum cutsets in reducible graphs*, SIAM Journal on Computing, 8 (1979), pp. 645–655.

[36] W. J. Stewart, *MARCA: Markov chain analyzer*, in Numerical Solution of Markov Chains, W. Stewart, ed., New York, NY, 1991, pp. 687–690.

[37] W. J. Stewart, *Introduction to the Numerical Solution of Markov Chains*, Princeton University Press, Princeton, NJ, 1994.

[38] Y. Zhu, S. Ye, and X. Li, *Distributed PageRank computation based on iterative aggregation-disaggregation methods*, in Proceedings of the 14th ACM Conference on Information and Knowledge Management, ACM Press, New York, NY, 2005, pp. 578–585.

Appendix A

Software User Manual

1. The C code in the file `web_main.c` is able to work with sparse matrices in:

   - Harwell-Boeing (HB) format
   - Sparse row (RB) format
   - Sparse column (CB) format
   - MARCA format
   - PUA format

   The matrix is defined through a file located under directory. The file defining sparse matrices in HB, RB, and MARCA format is `MATX_r`, in CB format is `MATX_c`, and in PUA format is `MATX_pua`. Problem parameters alpha, maximum number of iterations, tolerance, maximum number of inner iterations, and inner tolerance are defined in file `input`. Sample files for a problem are given next.

```
MATX_r:

 6 10
 0.5
 0.5
 0.3333333333333333
 0.3333333333333333
 0.3333333333333333
 0.5
 0.5
 0.5
 0.5
 1.0
 2 3 1 2 5 5 6 4 6 4
 1 3 3 6 8 10 11


input:

 0.85
 5000
 1.0e-10
 1.0
 10
 1.0e-10
```

In order to direct the output written to the screen also to the file `output`, the constant FILE_OUTPUT in the file `web_const.h` must be set to 1. When the constant is set to 0, the file `output` will not be produced.

In order to output the norm of the difference between the successive iterates for the block iterative solver every ERR_COUNT iterations, the constants ERR_COUNT and ERR_PRINT in the file `web_const.h` must be set respectively to the value of ERR_COUNT and 1. In order to output the norm of the difference between the successive iterates for the inner iterations of the block iterative solver ERR_PRINT_INNER also must set to 1.

Quadratic extrapolation is performed at multiples of QUAD_COUNT in the file `web_const.h`, when quadratic extrapolation with power method is used as a solver.

In order to output the contents of some variables during execution for debugging purposes, the constants TESTING and TEST_xx, where xx is the routine name, in the file `web_const.h` must be set to 1. When the constants are set to 0, no such output will be produced.

In order to output the solution vector, the constant SOLUTION in `web_const.h` must set to 1. The file `solution_XX` is produced, where XX stands for the solver name. For example, when power method is used as a solver `solution_power` is produced. When the constant is set to 0, no such output will be produced.

Note that array indices in C start from 0 and so do the row and column indices of the matrices in the code.

Execute the `Makefile` by typing make so that the executable is placed in the file `web`.

The files:

`mc13dd.c`

from the Harwell Subroutine Library (HSL) and

`daxpy.c` `ddot.c`

from BLAS have been left out. The C versions of these files are obtained from the Fortran versions by using the Fortran to C translator `f2c` available at Netlib

[5].

2. Make sure that `web` has executable privileges.

The executable can be run by the command:

`web Matrix_Type Solver_Type Tarjan_Parameter Cutset_Parameter Restriction_Parameter Orientation_Parameter Partition_Parameter`

where

Matrix_Type:

`r` : input matrix is to be read from file `MATX_r`

`c` : input matrix is to be read from file `MATX_c`

`p` : input matrix is to be read from file `MATX_pua`

Solver_Type:

`1` : Power method is used

`2` : Power method with quadratic extrapolation is used

`3` : JOR method is used

`4` : SOR method is used

`5` : BJOR method is used

`6` : BSOR method is used

`7` : IAD method with BJOR disaggregation step is used

`8` : IAD method with BSOR disaggregation step is used

Tarjan_Parameter:

`n` : mc13dd is not used

`y` : mc13dd is used

When Tarjan_Parameter is empty, it is taken to be: `y`

Cutset_Parameter:

`n` : cutset is not used

`y` : cutset is used

When Cutset_Parameter is empty, it is taken to be: `y`

Restriction_Parameter:

`n` : no restriction is made to (2×2) block form

`y` : restriction is made to (2×2) block form

When Restriction_Parameter is empty, it is taken to be: `n`

Orientation_Parameter:

`l` : lower-triangular diagonal blocks are produced

`u` : upper-triangular diagonal blocks are produced

When Orientation_Parameter is empty, it is taken to be: `u`

Partition_Parameter:

`1` : partitioning 1 is used

`2` : partitioning 2 is used

`3` : partitioning 3 is used

`4` : partitioning 4 is used

`5` : partitioning 5 is used

If the code is compiled to run under the Linux operating system on a 3 GHz Pentium IV processor with 2 Gigabytes main memory with `MATX_r` and `input` file then:

```
web r 1
```

produces the output:

```
Maximum # of iterations                  :       1000
Precision requested                      :  1.000e-10

Memory requirement                       :          0 MB
Read matrix (n,nz)                       :       6,10
# of zero rows                           :          1
Non-existent diag elements               :          6
Matrix is a DTMC
Set data structures time                 :       0.00 secs

POWER w/ alpha = 0.85 time               :       0.00 secs
# of iterations                          :         39
Residual                                 :  3.409e-11
```

```
web r 6 y y n u 3
```

produces the output:

```
Maximum # of iterations                      :       1000
Precision requested                          :  1.000e-10
Relaxation parameter                         :  1.000e+00
Maximum # of inner iterations                :          3
Inner precision requested                    :  1.000e-05

mc13dd is used
Upper-triangular transversal's complement

Memory requirement                           :          0 MB
Read matrix (n,nz)                           :       6,10
# of zero rows                               :          1
Non-existent diag elements                   :          6
Matrix is a DTMC
New nz                                       :         16
Set data structures time                     :       0.00 secs

partition time                               :       0.00 secs

nb                                           :          2

# of final diagonal blocks                   :          2
# of nzs in first diagonal block             :          6
Max. number of nz's in a diagonal block      :          6
Index of block with max. nz                  :          1
Min. number of nz's in a diagonal block      :          2
Index of block with min. nz                  :          2
Average number of nz's in a diagonal block   :       4.00
# of nzs in remaining diagonal blocks        :          2
# of nzs in off-diagonal part                :          8
svecsymperm, scale, pt2blk time              :       0.00 secs

moderr = 0    ====> Normal stopping criteria
block_iteration time                         :       0.00 secs
vecsymperm, unscale, res computation time    :       0.00 secs
# of iterations                              :         17
Residual                                     :  7.388e-12
```

Note that total solution times for methods obtained by adding all output times.