A Software Tool for the Steady–State Analysis of Google–like Stochastic Matrices*

Tuğrul Dayar
Department of Computer Engineering
Bilkent University
TR-06800 Bilkent, Ankara, Turkey
tugrul@cs.bilkent.edu.tr

Gökçe Nil Noyan
Undersecretariat for Defence Industries
Ziyabey Caddesi, 21 Sokak, No.4
TR-06520 Balgat, Ankara, Turkey
gnnoyan@ssm.gov.tr

ABSTRACT

The problem of computing the steady–state vector of positive stochastic matrices which are convex combinations of sparse, nonnegative matrices possibly having zero rows with appropriately chosen rank–1 matrices is addressed by a software tool. The dynamically changing matrices used by the Google search engine in ranking web pages are among the largest of such matrices. Ranking pages amounts to solving for the steady–state vectors of these matrices. The tool implements the power, quadratically extrapolated power, and iterative methods based on various block partitionings, including those with block triangular form and with triangular blocks obtained using cutsets.

Categories and Subject Descriptors

G.4 [**Mathematical Software**]: Algorithm design and analysis; G.3 [**Probability and Statistics**]: Markov processes; G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra—Sparse, structured, and very large systems (direct and iterative methods)

General Terms

Algorithms, Experimentation, Performance

Keywords

Markov chains, steady–state vector, Google

1. INTRODUCTION

*The work of the first author is supported by the Turkish Academy of Sciences grant TÜBA-GEBİP and that of the second author by The Scientific and Technological Research Council of Turkey.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SMCTOOLS 2009, October 19, Pisa, Italy
Copyright © 2009 ICST 978-963-9799-70-7
DOI 10.4108/ICST.VALUETOOLS2009.7453

We consider positive stochastic matrices of the form

$$S = R + uv, \tag{1}$$

where R of order n is a nonnegative and sparse matrix, possibly reducible with some zero rows, u is a nonnegative column n–vector, and v is a positive row n–vector [11]. Such matrices arise in the page ranking algorithm, PageRank [1], of Google (hence, the term Google–like). The objective therein is to compute the steady–state probability distribution row vector π of S in

$$\pi S = \pi, \pi e = 1, \tag{2}$$

where e is a column vector of ones of suitable length. The first difficulty lies in that although S does not have any zero elements, one must make every effort to work in sparse storage since R is a sparse matrix and uv is an outer product. The second difficulty is related to the reducibility of R, since an arbitrary partitioning of a reducible matrix will not yield irreducible diagonal blocks, and hence, care must be exercised when employing block solvers.

Now, let P be the transition probability matrix associated with the hyperlink structure of the web of pages to be ranked and $\alpha \in (0, 1)$ be the convex combination parameter used to obtain a positive stochastic matrix S so that it is ergodic and therefore can be analyzed for its steady–state [15]. In the PageRank algorithm $R = \alpha P$, $u = e - Re$, and v is the positive personalization probability distribution row vector satisfying $ve = 1$. Note that P may have zero rows corresponding to dangling nodes (that is, web pages without any outgoing hyperlinks). An equivalent formulation and an extensive discussion on PageRank can be found in [9]. The PageRank algorithm computes π in (2) iteratively using the power method. And ranking pages corresponds to sorting the pages (i.e., states of the underlying discrete–time Markov chain, DTMC) according to their steady–state probabilities.

The rate of convergence of the power method depends on the subdominant eigenvalue of S. Convergence is faster when α becomes smaller. However, the smaller α is, the higher the contribution of the second term in (1) and the lesser the hyperlink structure of the web influences page rankings. Slightly different α values can produce very different PageRank vectors, and as α approaches one, sensitivity issues start arising. The founders of Google, Brin and Page, 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 [1]. We remark that, normally $v = e^T/n$; that is, the uniform distribution is used. However, when the web surfer has preferences and is therefore biased, a personalization vector other than the uniform distribution must be used. Hence, ideally the problem in (1) needs to be solved multiple times for different personalization vectors.

In this paper, we present a tool for the steady–state analysis of Google–like matrices in a sequential setting (that is, on a computer with a uniprocessor). The objective is to provide a framework where different iterative solvers can be compared and contrasted. In the next section, we discuss the organization of the code. In section 3, we indicate how the input for the code should be prepared. Section 4 is about running the code and section 5 shows how the output is produced.

2. THE CODE

The gzipped tar file `google_like.tar.z` of about 55 Kbytes that can be obtained from [4] contains the code for the steady–state analysis of Google–like stochastic matrices. However, the same code can also be used to analyze irreducible DTMCs for their steady–state distribution by setting $\alpha = 1$. The directory `google_like` includes the files

```
MATX_r
Makefile
README
iad_methods.c
input
partage.c
partition.c
qpower.c
rec_partition.c
rosen.c
set_data_structures.c
splittings.c
timer.c
utility.c
web_const.h
web_main.c
```

The file `mc13dd.c` from the Harwell Subroutine Library (HSL) [7] and the files `ge.c` (with subroutine `ge_()`), `gth.c` (with subroutine `gth_()`), `pe.c` (with subroutines `pe_()`, `xpe_()`, `smatx_()`) from the MARkov Chain Analyzer (MARCA) [14] 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 [10].

The C code in the file `web_main.c` is able to work with sparse matrices in five formats, namely Harwell-Boeing (HB), sparse row (RB), transposed sparse row (CB), MARCA, and PUA [2]. The file defining the sparse matrices in HB, RB, and MARCA format is `MATX_r`, in CB format is `MATX_c`, and in PUA format is `MATX_pua`.

There are eight solvers available. These are power, power with quadratic extrapolation [8], Jacobi over–relaxation (JOR), successive over–relaxation (SOR), block JOR (BJOR), block SOR (BSOR), iterative aggregation–disaggregation (IAD) [15] with BJOR disaggregation step (IAD_BJOR), and IAD with BSOR disaggregation step (IAD_BSOR). These solvers are implemented respectively in the files `qpower.c`, `splittings.c`, and `iad_methods.c`.

There are five partitionings which can be used with the block solvers. The first three are based on the idea of permuting R to block–triangular form and using cutsets to obtain triangular diagonal blocks. It is known that with Tarjan's algorithm (implemented in `mc13dd.c`), square matrices can be permuted to block triangular form in which the diagonal blocks are irreducible [6]. On the other hand, an irreducible matrix with a zero–free diagonal can be symmetrically permuted and partitioned into four blocks as in

$$\begin{array}{c} \\ n_C \\ n_T \end{array} \begin{array}{c} \begin{array}{cc} n_C & n_T \end{array} \\ \begin{pmatrix} C & Y \\ Z & T \end{pmatrix} \end{array},$$

where C and T are square submatrices of order n_C and n_T, respectively, and T is triangular [3]. Note that T is necessarily nonsingular. It is clear that the 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 (together with the Sherman–Morrison formula since the block becomes positive due to the addition of the second term in (1)), it is useful to obtain a larger triangular block.

Minimizing n_C can be posed as the minimum cutset (or feedback vertex set) problem which is known to be NP–complete for general graphs; therefore, non–optimal solutions need to be considered. Fortunately, a polynomial time algorithm called Cutfind due to Rosen exists [12]. The algorithm runs in linear time and space and finds cutsets of graphs. Although cutsets computed with Cutfind may not be minimum, [3] shows that Cutfind is a fast algorithm for large graphs compared to other approximation algorithms and the size of the cutset computed is generally satisfying. Thus, for a block triangular matrix with irreducible diagonal blocks with zero–free diagonals, 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 alleviates the fill–in problem associated with factorizing diagonal blocks up to a certain extent.

Two other straightforward block partitioning techniques are considered. The *equal* partitioning forms (roughly) equal order blocks and the second partitioning, *other*, uses blocks of order respectively $1, 2, 3, \ldots$ [5, 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 $n - \lfloor\sqrt{n}\rfloor^2$ [5, p. 1693]. We name the former of these partitioning 4 and the latter partitioning 5. We treat dangling nodes within the partitioning algorithms by considering the hyperlinks to them and not in a uniform manner by aggregation as done in the literature. We believe the results obtained by the software tool would improve if we had taken the latter approach.

The parameters of these block partitionings can be expressed as a Cartesian product of four sets. Let set $\mathcal{B} = \{\mathtt{y},\mathtt{n}\}$ denote whether Tarjan's algorithm is used or not, set $\mathcal{C} = \{\mathtt{y},\mathtt{n}\}$ denote whether Rosen's algorithm is used or not, set $\mathcal{R} = \{\mathtt{y},\mathtt{n}\}$ denote whether the number of diagonal blocks is restricted to two or not, and set $\mathcal{O} = \{\mathtt{u},\mathtt{l}\}$ denote whether triangularized diagonal blocks are upper– or

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7453
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7453

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 $\{\mathtt{y}\} \times \mathcal{C} \times \mathcal{R} \times \mathcal{O}$, those using partitioning 3 (in which through a recursive application of the Tarjan and Rosen algorithms, all diagonal blocks are made to be triangular) can utilize $\{\mathtt{y}\} \times \{\mathtt{y}\} \times \{\mathtt{n}\} \times \mathcal{O}$, and those using partitionings 4 and 5 can utilize $\{\mathtt{n}\} \times \{\mathtt{n}\} \times \{\mathtt{n}\} \times \{\mathtt{u}\}$. On the other hand, experiments performed on irreducible matrices using partitionings 1 and 2 can utilize elements of $\{\mathtt{n}\} \times \{\mathtt{y}\} \times \mathcal{R} \times \mathcal{O}$ and those using partitioning 3 can utilize elements of $\{\mathtt{n}\} \times \{\mathtt{y}\} \times \{\mathtt{n}\} \times O$, 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 $\{\mathtt{y}\} \times \{\mathtt{n}\} \times \mathcal{R} \times \mathcal{O}$ are utilized. The five partitionings are implemented in the files `partage.c`, `partition.c`, `rec_partition.c`, and `rosen.c`.

Finally, `timer.c` includes functions to time the numerical experiments, `set_data_structures.c` includes a function to allocate the space required by the different solvers at the outset, read data from input files and set the corresponding data structures, and `utility.c` includes functions that implement sparse matrix–vector product, transposition of sparse matrices, permutation and inverse permutation of vectors, transformation of sparse matrix data structure to a sparse matrix block data structure, forward and backward substituitions.

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

3. PREPARING THE INPUT

The problem parameters α, maximum number of iterations, tolerance on the residual norm, relaxation parameter for those solvers using (B)JOR and (B)SOR, maximum number of inner iterations and inner tolerance for those solvers using BJOR and BSOR are defined in the 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 successive iterates 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 successive iterates for the inner iterations of block iterative solvers, `ERR_PRINT_INNER` also must be set to 1.

The solution vector in power is normalized at multiples of `NORM_COUNT` iterations in the file `web_const.h`. Quadratic extrapolation is performed at multiples of `QUAD_COUNT` in the file `web_const.h`, when power with quadratic extrapolation is used as the solver.

In order to output the contents of some variables during execution for debugging purposes, the constants `VERIFY`, `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 be set to 1. The file `solution_XX` is produced, where `XX` stands for the solver name. For example, when POWER is used as solver `solution_power` is produced. When the constant is set to 0, no such output will be produced.

The constant `PRODDEGREE` in `web_const.h` is used to choose the root node in minimum cutset computation with product outdegree when set to 1 and maximum outdegree when set to 0.

The constant `GE_OR_GTH` in `web_const.h` is used to choose between Gaussian elimination (GE) when set to 0 and the Grassmann–Taksar–Heyman (GTH) method when set to 1 in solving the coupling matrix of the solvers IAD_BJOR and IAD_BSOR.

4. RUNNING THE CODE

The `Makefile` can be used to obtain the executable `web`. The executable can then be run by the command:

```
web Matrix_Type Solver_Type Tarjan_Parameter
    Cutset_Parameter Restriction_Parameter
    Orientation_Parameter Partition_Parameter
```

where

`Matrix_Type` can be:

`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` can be:

1 : Power
2 : Power with quadratic extrapolation
3 : JOR
4 : SOR
5 : BJOR
6 : BSOR
7 : IAD_BJOR
8 : IAD_BSOR

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7453
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7453

`Tarjan_Parameter` can be:

`n` : `mc13d` is not used
`y` : `mc13d` is used

when `Tarjan_Parameter` is empty, it is taken to be `y`;

`Cutset_Parameter` can be:

`n` : cutset is not used
`y` : cutset is used

when `Cutset_Parameter` is empty, it is taken to be `y`;

`Restriction_Parameter` can be:

`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` can be:

`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` can be:

1 : partition 1 is used
2 : partition 2 is used
3 : partition 3 is used (recursive)
4 : partition 4 is used (*equal*)
5 : partition 5 is used (*other*)

Some web matrices can be located at the the University of Florida Sparse Matrix Collection [2]. While others may be obtained from the Stanford Webbase Project [13]. These matrices are reducible and possess zero rows corresponding to dangling nodes. Although this work is geared towards reducible matrices, irreducible matrices as in [5] may also be considered.

5. OBTAINING THE OUTPUT

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 input file and `MATX_r` then:

```
web r 1
```

produces the output

```
Maximum # of iterations                    :       5000
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
```

whereas,

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

produces the output

```
Maximum # of iterations                    :       5000
Precision requested                        :  1.000e-10
Relaxation parameter                       :  1.000e+00
Maximum # of inner iterations              :          5
Inner precision requested                  :  1.000e-10

mc13dd_() is used
Upper-triangular cutset'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 are obtained by adding all output times.

6. REFERENCES

[1] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30 (1998), pp. 107–117.

[2] T. Davis. University of Florida Sparse Matrix Collection, 2009. 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.
`http://www.cise.ufl.edu/research/sparse/matrices`
Last accessed on May 30, 2009.

[3] T. Dayar. *Obtaining triangular diagonal blocks using cutsets*. Technical Report BU-CE-0701, Department of Computer Engineering, Bilkent University, Ankara, Turkey, January 2007.
`http://www.cs.bilkent.edu.tr/tech-reports/2007/BU-CE-0701.pdf` Last accessed on May 30, 2009.

[4] T. Dayar and G. N. Noyan. Software for the steady-state analysis of Google–like matrices, 2009.
`http://www.cs.bilkent.edu.tr/~tugrul/software.html`
Last accessed on May 30, 2009.

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

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7453
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7453

[6] 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.

[7] Harwell Subroutine Library, 2009. `http://www.cse.clrc.ac.uk/nag/hsl` Last accessed on May 30, 2009.

[8] 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.

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

[10] Netlib Repository, 2009. `http://www.netlib.org` Last accessed on May 30, 2009.

[11] G. N. Noyan. *Steady-state analysis of Google–like matrices*. M.S. Thesis, Department of Computer Engineering, Bilkent University, Ankara, Turkey, September 2007.

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

[13] Stanford Webbase Project, 2009. `http://dbpubs.stanford.edu:8091/\~testbed/doc2/WebBase` Last accessed on May 30, 2009.

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

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

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7453
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7453