VNU Journal of Science, Mathematics - Physics 23 (2007) 76-83

Deeper Inside Finite-state Markov chains

Le Trung Kien[1,*] Le Trung Hieu[2], Tran Loc Hung[1], Nguyen Duy Tien[3]

[1]*Department of Mathematics, Hue University, 77 Nguyen Hue, Hue city, Vietnam*
[2]*Mathematics & Mechanics Faculty, Saint-Petersburg State University, Russia*
[3]*Department of Mathematics, Mechanics, Informatics, College of Science, VNU*
334, Nguyen Trai, Hanoi, Vietnam

Received 8 December 2006; received in revised form 2 August 2007

Abstract. The effective application of Markov chains has been paid much attention, and it has raised a lot of theoretical and applied problems. In this paper, we would like to approach one of these problems which is finding the long-run behavior of extremely huge-state Markov chains according to the direction of investigating the structure of *Markov Graph* to reduce complexity of computation. We focus on the way to access to the finite-state Markov chain theory via *Graph theory*. We suggested some basic knowledge about state classification and a small project of modelling the structure and the moving process of the *finite-state Markov chain model*. This project based on the remark that it is impossible to study deeperly the finite-state Markov chain theory if we do not have the clear sense about the structure and the movement of it.

1. Introduction

It is undeniable that the finite-state Markov chain in recent years has lots of important applications in modelling the natural and social phenomena. We may enumerate some branches of science such as weather forecast, system magement, Web information searching, machine learning which the model of finite-state Markov chain is applied for. Markov chain effective application has been paid much attention, and it has raised a lot of theoretical problems as well as applied ones. One of these is that how to find the long-run behavior of Markov chain when the state space is extremely huge. For example, to rank Webs based on the hyperlink structure of Web Graph, PageRank algorithm [1] of information searching engine Google has to identify the stationary distribution of an irreducible aperiodic Markov chain with 6 billion states. In this case, it is obvious that applying the classic methods to identify the stationary distribution is impractical. To solve this problem, some ideas are considered such as measuring approximately the stationary distribution [2-6] or investigating the structure of *Markov Graph* to reduce complexity of computation [7-9].

The problem of measuring approximately the stationary distribution of huge-state Markov chains has been taken into consideration by the scientists through last two decades. Especially, some groups of scientists of Stanford university and other authoritative research centers were interested in identifing the stationary distribution of *Web Markov chain* to evaluate the important of Web. S.Kamvar, T.Haveliwala

* Corresponding author. Tel.: 84-054-822407.
E-mail: hicukien@hotmail.com

et al. [4, 5] suggested using successive intermediate iterates to extrapolate successively better estimates of the true PageRank values. They used the special properties from the second eigenvalue of Google matrix and Power method. J. Kleinberg [6] introduced the notion (ϵ, k)-*detection set* play a role as the evidence for existence of sets which do not have as most k states and have the property: if an adversary destroys this set, after which two subsets of the states, each at least an ϵ fraction of the state space of the Markov chain, that are not accessible from one another. Developing on J. Kleinberg's basic ideas, J. Fakcharoenphol [3] showed that the (ϵ, k)-detection set for state failures can be found with probability at least $1-\delta$ by randomly chossing a subset of nodes of size $O(\frac{1}{\epsilon}k\log k\log\frac{k}{\epsilon}+\frac{1}{\epsilon}\log\frac{1}{\delta})$. F. Chung [2] studied partition property of a Markov chain based on applications of eigenvalues and eigenvectors of its transition probability matrix in combinatorial optimization. The partition property can be used to deal with various problems that often arise in the study of huge-state Markov chains including bounding the rate of convergence and deriving comparison theorems.

In this paper, we would like to access to the problem according to the direction of investigating the structure of Markov Graph to reduce complexity of computation. As we know, the stationary distribution of finite-state Markov chain depends only on the link-structure of its recurrent states and it receives zero value at the transient states. In addition, as a consequence of solving optimally the state classification, we will recognize easilier some new important properties about the graph-structure of this Markov chain. In [8] based on the results in *Random Graph* theory, B. Bollobás proved the correctness of the property: Let n be a positive integer, $0 \leqslant p \leqslant 1$. The *random Markov chain* $\mathcal{M}(n,p)$ is a probability space over the set of Markov chains on the state set $\{1,2,\ldots,n\}$ determined by $\mathbb{P}\{p_{ij}>0\}=p$, with these events mutually independent. Therefore, if n is so large and $p=O(\frac{\ln n}{n})$ then almost sure a Markov chain in $\mathcal{M}(n,p)$ will be irreducible aperiodic. Clearly this is a property of authority; it makes us have a deeper understanding about a fundamental class of finite-state Markov chains, *irreducible aperiodic Markov chain class*. More importantly, it allows us to think about the new way to investigate deeperly the finite-state Markov chain theory basing on the Random Graph Theory.

From this observation, our paper focuses on the way to access to the finite-state Markov chain theory via Graph theory; then model and construct clearerly than basic properties of the finite-state Markov chain theory. Basing on some theoretical results which have been built in Section 2 and Section 3, we have constructed *State Classification* algorithm to classify state of finite-state Markov chain. Our purpose to build this algorithm comes from the idea "All problems will be clearer if we give out the algorithm to solve them". However, our imagination and visual images are completely different from each other. In the reality, no projects have modelled specifically the movement of finite-state Markov chain process; from the theoretically basic algorithms which have just been constructed; in Section 4, we have built a small project with the purpose of modelling specifically our new results. The significance of this project is that we can have a clearer and deeper image about the familiarly theoretical results of Markov chain. More importantly, this project helps us to build a concrete model space *Random Markov chain*, then create a convenient condition for a deeper research in the direction of Random Markov chain. This is also the last section of the paper included our future works and the difficulties we are facing up.

2. The sense in theory graph

In the discrete time domain, a random process $\mathcal{X}=\{X_n\in S|n\geqslant 0\}$ on the state space $S=\{1,2,\ldots,N\}$ is a *Markov chain* if it is a sequence of random variables each taking values in S and

it satisfies the *Markov property*, i.e, its future evolution is independent of the past states and depends only on the present state. Formally, $\mathcal{X}$ is a Markov chain if for all $n \leqslant 1$ and all $j, i_{n-1}, \ldots, i_1, i_0 \in S$,

$$\mathbb{P}\{X_{n+1} = j | X_n = i, X_{n-1} = i_{n-1}, \ldots, X_1 = i_1, X_0 = i_0\} = \mathbb{P}\{X_{n+1} = j | X_n = i\}.$$

If the probabilities governing $\mathcal{X}$ are independent of time, $\mathcal{X}$ is *time-homogeneous*. In this case, we define a matrix $\mathrm{P} = (p_{ij})$ whose element at the i-th row and j-th column,

$$p_{ij} = \mathbb{P}\{X_{n+1} = j | X_n = i\} = \mathbb{P}\{X_1 = j | X_0 = i\}.$$

Matrix P is called 1-*step transition matrix* or simply the *transition matrix* of $\mathcal{X}$.

Consider a digraph $\mathcal{G} = (V, E)$, where the vertex set $V \equiv S = \{1, 2, \ldots, N\}$. The edge space of $\mathcal{G}$ is constructed as follows: an edge from vertex i to vertex j, denote e_{ij}, if and only if in the model of this finite-state Markov chain, the process can visit the state j after one step if now it stays in the state i. In other words, for all i, j: $e_{ij} \in E \Leftrightarrow p_{ij} > 0$. We call the digraph $\mathcal{G}$ the *boolean transition graph* of the Markov chain, and its associated a matrix calling the *boolean transition matrix* of this Markov chain, denote $\mathbf{Q} = Q_{ij}$, is constructed as follows:

$$Q_{ij} = \begin{cases} 1 & \text{if } e_{ij} \in E \\ 0 & \text{otherwise} \end{cases}$$

In the model of digraph $\mathcal{G}$ we give out some related concepts as following: A *path* $\mathcal{P} = i_0 i_1 \ldots i_k$ in a digraph $\mathcal{G} = (V, E)$ and calling $\mathcal{P}$ a path from i_0 to i_k if it is a non-empty sub-digraph of the form:

- Vertex space: $V_{\mathcal{P}} = \{i_0, i_1, \ldots, i_k\} \subset V$, where the i_h are all distinct.
- Edge space: $E_{\mathcal{P}} = \{i_0 i_1, i_1 i_2, \ldots, i_{k-1} i_k\} \subset E$.

The number of edges of the path, k, is called its *length*, the vertex i_0 is called *beginning-vertex* and the vertex i_k is called *ending-vertex*.

In the directed graph $\mathcal{G}$, vertex j is said to be *accessible* from vertex i, denote $i \to j$, if there is a path from vertex i to vertex j. Otherwise, vertex j is said to be *unaccessible* from vertex i and denote $i \nrightarrow j$. Two vertices i and j that are accessible to each other are said to *communicate*, and denote $i \leftrightarrow j$. Vertex i is said *recurrent* if for all vertex j such that $i \to j$ then there will have a path from j to i, $j \to i$. Vertex i is said *transient* if it is not recurrent. Clearly the relation of communication satisfies three properties reflexive, symmetric, and transitive so it is an equivalence. Two vertices that communicate with each other are said to be in the same class; the concept of communication divides the vertex space up into a number of separate classes.

From giving out the concepts: *accessible, communicate, recurrent* and *transient* in the model digraph $\mathcal{G}$, we see the similarity between these concepts and the corresponding concepts in the model of finite-state Markov chain. In other words, if a vertex i is accessible or communicate to a vertex j; or vertex i is recurrent then in the finite-state Markov chain model which is corresponded with, the state i will be accessible or communicate to the state j; or the state i is recurrent state. With this construction, it is obvious that we, basing on its boolean transition graph $\mathcal{G}$, can solve the basic problems of the finite-state Markov chain theory. From the definition, if a vertex is transient then all other vertices that accessible with this vertex will be transient, or if this vertex is recurrent then all other vertices that it accessible with will be recurrent. Thus, when we determine a vertex to be transient or recurrent, the transient and recurrent properties of other vertices that are accessible with these vertices are deduced and of course they are removed from further consideration. Moreover, this identification only depends

on boolean transition graph $\mathcal{G}$ or boolean transition matrix Q. These following concepts and results will specificialize this statement.

We start by defining the *forward* and *backward* sets of a vertex.

Definition 2.1. *The forward set of vertex $i \in V$, denote by $\mathcal{F}(i)$, is the set of vertices that i is accessibles with. That is, $\mathcal{F}(i) = \{j \in V \mid i \rightarrow j\}$. Similarly, the backward set of vertex i, denoted by $\mathcal{B}(i)$, is the set of vertices that are accessible with i. That is, $\mathcal{B}(i) = \{j \in V \mid j \rightarrow i\}$.*

We have the following results:

Proposition 2.1. *A vertex $i \in V$ is recurrent if and only if $\mathcal{F}(i) \subseteq \mathcal{B}(i)$. In other words, i is transient if and only if $\mathcal{F}(i) \nsubseteq \mathcal{B}(i)$.*

Theorem 2.1. [10] *If vertex $i \in V$ is transient, then all vertices in $\mathcal{B}(i)$ are transient. If vertex i is recurrent, on the other hand, all vertices in $\mathcal{F}(i)$ are recurrent. In the latter case, the set $\mathcal{F}(i)$ is a recurrent class, and the set $\mathcal{B}(i) - \mathcal{F}(i)$ (if not empty) contains only transient vertices.*

Proof. Suppose vertex i is transient. By Proposition 2.1, $\mathcal{F}(i) \nsubseteq \mathcal{B}(i)$, i.e., $\exists k \in \mathcal{F}(i)$ such that $k \notin \mathcal{B}(i)$. Now, suppose vertex $j \in \mathcal{B}(i)$, then $k \in \mathcal{F}(j)$. This is because $i \in \mathcal{F}(j)$ so that $\mathcal{F}(i) \subseteq \mathcal{F}(j)$. On the other hand, $\mathcal{B}(j) \subseteq \mathcal{B}(i)$ since $j \in \mathcal{B}(i)$. Therefore, we have vertex $k \in \mathcal{F}(j)$ but $k \notin \mathcal{B}(j)$ since $k \notin \mathcal{B}(i)$, which implies $\mathcal{F}(j) \nsubseteq \mathcal{B}(j)$ so that j is transient by Proposition 2.1.

Now, if vertex i is recurrent, i.e., $\mathcal{F}(i) \subseteq \mathcal{B}(i)$ from Proposition 2.1, then, $\forall j \in \mathcal{F}(i) \Rightarrow i \leftrightarrow j$. So we have $\mathcal{F}(j) \subseteq \mathcal{F}(i)$ and $\mathcal{B}(i) \subseteq \mathcal{B}(j)$. Thus, $\mathcal{F}(j) \subseteq \mathcal{F}(i) \subseteq \mathcal{B}(i) \subseteq \mathcal{B}(j)$, which implies j is recurrent from Proposition 2.1. Finally, if i is recurrent and $\mathcal{B}(i) - \mathcal{F}(i)$ is not empty, let vertex $k \in \mathcal{B}(i) - \mathcal{F}(i)$, we merely need to show that $\mathcal{F}(k) \nsubseteq \mathcal{B}(k)$ so that k is transient. In fact, $k \in \mathcal{B}(i) \Leftrightarrow i \in \mathcal{F}(k)$, and $k \notin \mathcal{F}(i) \Leftrightarrow i \notin \mathcal{B}(k)$, which implies $\mathcal{F}(k) \nsubseteq \mathcal{B}(k)$.

Proposition 2.1 states that we can check if a vertex is recurrent by simply checking if its forward set is contained in its backward set. If it is, then a recurrent class has been found which equals to the forward set so that the vertices of this forward set can be removed from consideration. Moreover, according to Theorem 2.1 if the backward set properly contains the forward set, those vertices in the backward set not belonging to the forward set are all found to be transient. In the case the forward set is not contained in the backward set, we have found a subset of transient vertices equal to $\{i\} \cup \mathcal{B}(i)$.

The important problem in analyzing the long-run behavior of a finite-state Markov chain is determining the recurrent states as exactly as possible. The following results will make Theorem 2.1 clearer and help us to look for the recurrent states easily.

Theorem 2.2. *If vertex $i \in V$ is transient, then all vertices in $\mathcal{B}(i)$ are transient. Moreover, there are some vertices in $\mathcal{F}(i) \backslash \mathcal{B}(i)$ are recurrent; set $\mathcal{F}(i) \backslash \mathcal{B}(i)$ contains a recurrent class.*

Proof. As we know, if $j \in \mathcal{F}(i)$, $i, j \in V$, then $\mathcal{F}(j) \subseteq \mathcal{F}(i)$. So we can prove this theorem with induction method according to the number of vertex of set $\mathcal{F}(i)$.

Let vertex $i \in V$ is transient. Suppose the theorem is right with all transient vertices $u \in V$ such that $|\mathcal{F}(u)| < |\mathcal{F}(i)|$. Consider any vertex $j \in \mathcal{F}(i)$. If vertex j is recurrent, the theorem is right; then $\mathcal{F}(i)$ contains a recurrent class, which is $\mathcal{B}(j)$. Otherwise, if j is transient. $|\mathcal{F}(j)| < |\mathcal{F}(i)|$ so $\mathcal{F}(j)$ contains a recurrent class. The theorem is still right.

We consider a digraph $\mathcal{G}^R$ (correspond with $\mathcal{P}^R$) which has the same vertex space as $\mathcal{G}$ (correspond with $\mathcal{P}$) but in which all edges have been reversed in direction. If we call $n(i)$ and $m(i)$ tuong ung be the number of paths starting and ending at vertex i. From Theorem 2.2 we have an important result as follows:

Theorem 2.3. *The vertex i is recurrent in the digraph $\mathcal{G}$ if*

$$n(i) = \min\{n(u) \mid u \in V\}$$

The vertex j is recurrent in the digraph $\mathcal{G}^R$ if

$$m(j) = \min\{m(u) \mid u \in V\}$$

Proof. Consider a vertex i such that $n(i) = \min\{n(u) \mid u \in V\}$. If vertex i is transient in $\mathcal{G}$, from Theorem 2.2 it exists a vertex i_0 such that

(i) Vertex i_0 is recurrent,

(ii) and existing a path $\mathcal{P} = ii_1 \ldots i_k i_0$, where vertex i_k is transient.

Obviously from all paths starting at vertex i_0, we can make another path starting at vertex i and containing this path by adding path $\mathcal{P}$ forward to this path. In addition, path $\mathcal{P}$ is not a path starting at vertex i_0, so $n(i) > n(i_0)$, contradiction. Therefore, vertex i is recurrent in digraph $\mathcal{G}$.

Basing on the statement that the class property are not affected by reversing all the directed graph's edges, we prove similarly the second idea.

From Theorem 2.3, each recurrent vertex in $\mathcal{G}$ or $\mathcal{G}^R$ is identified the effectiveness via the number of paths starting and ending at it. As we know, in Graph Theory, *Depth-First Search* algorithm (DFS) is known as the most effective algorithm in finding the number of paths starting at one vertex and ending at one vertex. In the following section, we will use the idea of DFS algorithm and Theorem 2.3 to construct an algorithm to classify state of finite-state Markov chain basing on its boolean transition graph.

3. State classification algorithm

In this section, our main purpose is to give *State Classification* algorithm based on the ideas of *Strong Components* algorithm and *DFS* algorithm to classify vertex in a digraph according to transience and recurrence properties. Strong Components algorithm can be found throught any materials mentioning "Design and Analysis of Algorithm & Directed graphs".

From definition of DFS, when we enter a class, every vertex in the class is reachable, so DFS does not terminate until all the vertices in this class have been visited. Thus all the vertices in a class may appear in the same DFS tree of the DFS forest. Unfortunately, in general, many classes may appear in the same DFS tree. Does there always exist a way to order the DFS such that just have only one class appear in any DFS tree? Fortunately, the answer is yes. State Classification algorithm will explain the reason why this answer is yes. In oder to investigate the idea of State Classification algorithm, firstly, we study on the idea of *Depth-First Search* algorithm (DFS).

3.1. Depth-First Search

Assume that we are given a digraph $\mathcal{G} = (V, E)$. To compute effectively all paths starting and ending at a vertex in $\mathcal{G}$ we submit an optimal surf-proposal to surf all paths in $\mathcal{G}$. Concretely, we might use the following strategy. Firstly, we maintain a color for each vertex: `white` means *undiscovered*, `gray` means *discovered* but not finished processing, and `black` means *finished*. Then as the process enter a vertex in V, the color of this vertex will be changed from `white` to `gray` to remind itself that it was already there. Successively travel from vertex to vertex as long as the process comes to a place

it has not already been. When the process returns to the same vertex, try a different edge leaving the vertex (assuming it goes somewhere the process has not already been). When all vertices have been tried in a given vertex, the color of this vertex will be change from gray to black and backtrack. This is the general idea behind *Depth-First Search*. We will associate two numbers with each vertex. There are *time stamps*. When we firstly discover a vertex i store a counter in $d[i]$ and when we finish processing a vertex we store a counter in $f[i]$. The algorithm is showed in Table 1.

Table 1. The code of Depth-First Search Algorithm.

```
Depth-First Search(G) {
 color[.]←white; pred[.]←null;
  time←0;
  for each i in V
    if (color[i] = white)
      Visit(i);
}

Visit(i) {
  color[i]←gray; d[i]←time + 1;
  for each j in Adj(i)
    if (color[j] = white) {
      pred[j]←i;
      Visit(j);
    }
  color[i]←black; f[i]←time + 1;
}
```

3.2. State classification

We have a statement without proof as following: A vertex i of which finish time value, $f[i]$, is maximum will be recurrent in digraph $\mathcal{G}^R$. Moreover, if consider in a new digraph which is created from the digraph $\mathcal{G}^R$ after destroying the vertex i and all its relative edges, then the vertex with maximal finish time value will be recurrent. Otherwise, clearly once the DFS starts within a given class, it must visit every vertex within the class (and possibly some others) before finishing. If we do not start at a recurrent class, then the search may "leak out" into other classes, and put them in the same DFS tree. However, by visiting vertices in reverse topological order of finish times sequence $\{f[i] \mid i \in V\}$, each search cannot "leak out" into other classes, because the DFS always starts within a recurrent class.

Tabel 2. The code of State Classification Algorithm.

```
StClass(G) {
  Run DFS(G), computing finish times f[i] for each vertex i;
  Compute G^R ← Reverse(G);
  Sort the vertices of G^R (by QuickSort) in decreasing order of f[i];
  DFS(G^R) using this order;
  Classes ← DFS tree;
  If there exists an edge connected this class to another class
    This class ← recurrent class;
}
```

This leaves us with the intuition that if we could somehow order the DFS, so that it hits the vertices according to a reverse topological of finish times sequence $\{f[i] \mid i \in V\}$, then we will have

an easy algorithm for finding recurrent classes and transient classes of a directed graph. The code of *State Classification algorithm* to slove the class problem is given out in Table 2.

3. Our project and future work

As we know, the basic knowledge to build the finite-state Markov chain theory is really simple and understandable. However, because of the simple theory, we will find difficulties in realizing some important properties of the finite-state Markov chain theory without basing on experimental images or results in the reality. For example, by means of experiment, we find out an interesting property that "Consider a finite-state Markov chain which its state space is big enough ($|S| > 100$). If its transition matrix is determined by $\mathbb{P}\{p_{ij} > 0\} = 10\%$, $\forall i, j$, with these events mutually independent, then almost sure we can affirm that it is an irreducible aperiodic Markov chain" (See in [8]). Indeed, when the requirement of science is higher and higher, the finite-state Markov chain theory is certainly to be investigated deeperly. We believe that it is impossible to do that if we do not have the clear sense about the structure or the movement of finite-state Markov chain model. From this point, in the time to study on the finite-state Markov chain theory, we have made a small project with the purpose of helping get a good sense to the finite-state Markov chain model. Our project is written with *Visual* $C^\sharp$ language and its code program is very simple and understandable. This project deals with the important problems of finite-state Markov chain theory such as classifying state, finding stationary distribution. More interestingly, this project gives out the specific images about the digraph modelling finite-state Markov chain, and shows clearly the acting process of DFS algorithm and State Classification algorithm.

To model the finite-state Markov chain model by a digraph according to its transition matrix, our porject is written with Visual $C^\sharp$, and is constituted of three *Form* and two *Class*: Form *Finite State Markov Chains*, Form *Table*, Form *Graph*, Class *MouseMove* and Class *PanelArray*. With the purpose of constructing the transition matrix, the Form Table brings out the two dimension array boxes which allow us to input the data of transition matrix. Form Finite State Markov Chains combines with Form Table to form a control system. In the Form Finite State Markov Chains, we have three groups of control button. The group *Construction* helps us to build the digraph to model the finite-state Markov chain; the group *Classification* makes the acting process of DFS and State Classification algorithms; and the group *Distribution* allows us to compute the stationary distribution of finite-state Markov chains. To help the Form Graph modelling the finite-state Markov chain model, the class PanelArray presents a vertex or an edge in digraph as a control panel, and the class MouseMove, with the purpose of moving a control panel, helps us to move a vertex or an edge easily. Thus, the Form Graph creates successfully the active digraph model that we freely move the vertices and the curveness of edges. Moreover, when a digraph is created with its transition matrix, we can change this graph such as omitting some edges or vertices by changing values in its transition matrix. However, the most interest we are self-assured in this project is that the group of buttons Construction can construct the model of random directed graphs. As we know, Random Graphs, an interesting and important branch of science, has some properties we can apply for Markov chain theory. Therefore, we hope that from our project Random Graphs will be easily studied and beneficial from researching finite-state Markov chain theory.

In recent years, lots of scientists all over the world are interested in constructing state classification algorithm and finding the stationary distribution of finite-state Markov chains. Many projects

like [11] are carried out to solve the above problems. However, without the purpose of studying on the finite-state Markov chain theory deeperly, almost projects only focus on the results, not be concerned with the structure and the movement of Markov chain model. From this situation, the establishment of our project has significance in studying the finite-state Markov chain theory deeperly. Besides giving out the result of state classifying and stationary distribution searching, our project has modelled exactly the structure and the acting process of the Markov chain models with 200 states. However, due to the limit of time and our knowledge, our project is poor in both content and formation. It's hope that with the supplementary database knowledge, we will solve all important problems of Markov chains which its state space is large. When the number of state of Markov chain is extremely large, the important properties will be recognized easily. The difficulty we are facing in upgrading this project in order to work with the huge-state Markov chain is how to construct the Form Graph observing and describing the modelled digraph. That's our main future work!

Ackowledgments. This paper is based on the talk given at the Conference on Mathematics, Mechanics, and Informatics, Hanoi, 7/10/2006, on the occasion of 50th Anniversary of Department of Mathematics, Mechanics and Informatics, Vietnam National University, Hanoi. The authors are grateful to the referee for carefully reading the paper and suggestions to improve the presentation.

References

[1] L. Page, S. Brin, R. Motwani, T. Windograd, *The PageRank Citation Ranking: Bring Order to the Web*, Stanford University Technical Report, 1998.
[2] F. Chung, Laplacians and the Cheeger inequality for directed graphs, *Annals of Combinatorics*, to appear.
[3] J. Fakcharoenphol, An Improved VC-Dimension Bound for Finding Network Failures, *Master's thesis*, Berkeley University of California, 2001.
[4] T.H. Haveliwala, S.D. Kamvar, *The second eigenvalue of the Google matrix*, Stanford University Technical Report, 2003.
[5] S. Kamvar, T. Haveliwala, C. Manning, G. Golub, Extrapolation methods for accelerating PageRank computations, *In Proceedings of the Twelfth International World Wide Web Conference*, 2003.
[6] J. Kleinberg, Detecting a Network Failure, *Proc. 41st Annual IEEE Symposium on Foundations of Computer Science*, 2002.
[7] P. Baldi, P. Frasconi, P. Smyth, *Modeling the Internet and the Web*, John Wiley & Sons, Inc. New York, 2003.
[8] B. Bollobás, *Random Graphs*, Cambridge University Press, 2001.
[9] D. Callaway, J. Hopcroft, J. Kleinberg, M. Newman, S. Stragatz, *Are randomly grown graphs really random?* http://arxiv.org/abs/cond-mat/0104546 Vol 2 (2001) 1.
[10] D.M. Mount, *Design and Analysis of Computer Algorithms*, Dept. of Computer Science, University of Maryland, 2003.
[11] H. Tijms, P. Schram, *Orstat: MCQueue*, A software in Dept of Econometrics and Operational Research 2000-2002.