Towards Compressing Web Graphs

Micah Adler*
University of Massachusetts, Amherst

Michael Mitzenmacher †
Harvard University

Abstract

We consider the problem of compressing graphs of the link structure of the World Wide Web. We provide efficient algorithms for such compression that are motivated by recently proposed random graph models for describing the Web. The algorithms are based on reducing the compression problem to the problem of finding a minimum spanning tree in a directed graph related to the original link graph. The performance of the algorithms on graphs generated by the random graph models suggests that by taking advantage of the link structure of the Web, one may achieve significantly better compression than natural Huffman-based schemes. We also provide hardness results demonstrating limitations on natural extensions of our approach.

1 Introduction

A snapshot of the World Wide Web can be thought of as a graph, with Web pages represented by nodes and hyperlinks represented by directed edges. This representation is used by a wide variety of Web algorithms, including algorithms for ranking pages based on their connectivity [12, 4] and algorithms for finding natural communities of pages on a shared topic [14]. At least one major search engine has designed a tool called the connectivity server for storing the Web graph [3, 5].

Given this interest, a natural question to ask is how well the Web graph and Web-like graphs can be compressed. Such compression would allow for more efficient storage and transfer of Web graphs, and may improve the performance of Web algorithms by allowing computation to be performed in faster levels of computer memory hierarchies. Good compression requires using the structural properties of the Web graph, and hence an important first step is understanding this structure. Previous work gives us important insights. It is clear that the Web graph appears to be significantly different from the likely graphs resulting from traditional random graph models. In particular, there appear to be natural clusters of related pages with similar connections. Hence, in [13, 15], a new random graph model was introduced with these clustering properties. The basis of this model is that pages and links enter the system dynamically, and new pages may link to other pages by finding one or more *reference* pages and copying links from these references.

*Department of Computer Science, University of Massachusetts, Amherst, MA 01003-4610. E-mail: micah@cs.umass.edu.

†Harvard University, Division of Engineering and Applied Sciences, 33 Oxford St., Cambridge, MA 02138. Supported in part by an Alfred P. Sloan Research Fellowship, NSF CAREER grant CCR-9983832, and an equipment grant from Compaq Computer Corporation. E-mail: michaelm@eecs.harvard.edu.

Recent studies of the Web graph suggest that the structure of the Web is actually more complex than this random graph model; see, for example [5]. However, as a first approximation, this model captures important high level behavior, and it may be especially suitable for specific subdomains, such as all the pages within a given university. Hence, in this paper we focus on variations of this graph model, although we also test our prototype on real data [10].

Our primary results are the following:

- We provide a compression algorithm specifically designed for graph structures with many shared links. Under appropriate assumptions, the running time of our algorithm is $O(n \log n)$, where n is the number of nodes in the graph. The algorithm requires finding a directed minimum spanning tree on a graph associated with the original graph.
- We provide hardness results demonstrating that several natural extensions of our algorithm are NP-Hard.
- We demonstrate the effectiveness of our approach on a testbed of random graphs derived from the random graph models that motivate our approach.

Space limitations require us to limit our exposition here. More details can be found in the complete extended abstract [1].

1.1 Previous and Concurrent Work

Since developing this work, we have found that similar ideas have been used by the creators of the connectivity server, a tool for keeping compressed Web graph data [3]. They have found the reference-based approach extremely effective [17], although they also use locality. Our approach, however, is significantly different from theirs. For example, here we focus on optimal algorithms and corresponding theoretical limitations. The connectivity server project has focused on approximate algorithms that seem to be effective in practice, but without the same theoretical guarantees that we provide. We believe both works represent important steps in understanding and improving compression of Web graphs.

The Web graph can be thought of as a sparse bit array. Methods for compressing sparse bit arrays that make use of the probability distributions of the gaps between successive entries are discussed in [20]. Similar approaches are used by the creators of the connectivity server and are useful for compressing local links between pages on the same host [17]. Our reference approach, in contrast, is designed for links between pages on distinct hosts, and thus is better suited to the simple Web models we examine here.

Directed minimum spanning trees have been used previously in other scenarios to provide good compression. Tate [18] uses such trees to obtain a reordering of the bands of a multispectral image that allows for the optimal compression. More recently, a similar idea is alluded to in [6], in the context of compressing tables of data. There, the authors use one column to compress another, and mention that the problem can be reduced to a minimum spanning tree problem, although in their case, edges are undirected.

1.2 Framework

When we discuss compressing Web-like graphs, there are actually a variety of distinct situations we may wish to consider:

1. Compressing the underlying graph for storage or transmission, up to isomorphism. This setting would be useful if we want to store just the graph structure itself.
2. Compressing the underlying graph for storage or transmission, maintaining a given ordering of the nodes. For example, we might order the nodes according to the sorted order of the URLs (so that the URLs can be compressed by delta encoding, as in [3]).
3. Compressing the underlying graph for use in its compressed form. That is, we desire a compressed form of the graph that still allows for efficient computation on the compressed form.

Our primary focus in the paper is the second setting; however, we will suggest connections between the variations as they arise.

1.3 A Web graph model

We reiterate that in this paper, rather than compress actual subgraphs of the Web, our focus is a recently proposed Web graph model that captures certain aspects of Web graphs. The model, taken primarily from [15], uses the following basic outline. The graph evolves over time by associated node and edge creation and deletion processes. The intuition suggested from [15] is the following: "*A new page adds links by picking an existing page, and copying some links from that page to itself.*" For example, a new page v might examine the outedges from a page w and link to a subset of the pages that w links to; we call this *copying outedges*. This intuition is based on the idea that a user decides what pages to link a new page to based on a page or pages that the user already likes.

Given this framework, there are a variety of possible variations, depending on the specifics of the edge creation and deletion process as well as the copy process. We specify the model we use here. We begin with an initial graph of n_0 nodes, with each having d_0 outedges connected to nodes chosen uniformly at random. There is no deletion process, only a node creation process. One new node is created each time step to a total of n nodes. The creation process is determined by probability distributions $\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}, \mathcal{F}$. The distribution $\mathcal{A}$ provides a number a, such that v is given a outedges with each edge pointing to a node chosen uniformly at random from all nodes existing at that time. Similarly, $\mathcal{B}$ provides a number such that v copies outedges from b nodes, again chosen uniformly at random. The distribution $\mathcal{C}$ yields a probability; for each of the b nodes $w_1, \ldots, w_b$ chosen to copy from, a probability is independently chosen from $\mathcal{C}$, and each outedge from w_i is copied independently with the probability determined from $\mathcal{C}$. Distributions $\mathcal{D}, \mathcal{E}$, and $\mathcal{F}$ are analogous to $\mathcal{A}, \mathcal{B}$, and $\mathcal{C}$ respectively, except that they determine the inedges for a new page. We call such graphs *copy graphs*.

2 A baseline Huffman-based scheme

Experiments have demonstrated that the indegrees and outdegrees of Web pages follow a Zipfian distribution [2, 15, 5]. That is, the fraction of pages with indegree j is roughly proportional to $1/j^{\alpha}$ for some fixed constant α, and similarly the fraction of pages with outdegree j is roughly proportional to $1/j^{\beta}$ for some fixed constant β. One of the features of the copy graph model is that it yields graphs with such Zipfian distributions [15].

Given the large variance in degrees, it is natural to consider Huffman-based compression schemes. A simple such scheme goes through the nodes in order and lists the destination of

each outedge directed from that node. Each page is assigned a Huffman codeword based on its indegree. To separate the outedges of each node we utilize a special stop symbol.

Many simple variations are possible. The compression scheme could also be based on the edges directed into each node, whichever is better. In the case where we only need to store an isomorphism of the graph, we might avoid the stop symbol. Instead, we can send an implicit or explicit representation of the outdegree distribution, sort the nodes by outdegree, and list the outedges for each node as before without the stop symbol.

This approach achieves significant compression with little complexity; it could naturally be used in the framework of the connectivity server [3], or in any system that wanted efficient computation on the compressed form of the graph. Thus we shall use it as a baseline for the comparison of algorithms that compress the Web graph. Note, however, that the Huffman-based scheme ignores the natural clustering structure induced in copy graphs.

3 The FIND-REFERENCE algorithm

Our basic algorithm is based on the following insight: since copying links is a basic operation in our graph model, we can attempt to find nodes that share several common outedges, corresponding to cases where one node might have copied the links of another. Once an appropriate neighbor is identified, the difference, or delta, between the outedges of the two nodes can be identified. When node i is compressed in this way using node j, we say that node j is a *reference* for node i.

For example, let us consider a specific scheme (we describe possible improvements to this scheme in Section 3.1). If node j is labeled as a reference of node i, we can include a 0/1 bit vector denoting which outedges of node j are also outedges of node i. Other outedges of i can then be separately identified, say using $\lceil \log n \rceil$ bits in an n node graph. We must also identify i, which is another $\lceil \log n \rceil$ bits. Let $N(i)$ and $N(j)$ represent the set of outedges for node i and node j respectively. The cost of compressing node i using node j as a reference with this scheme is then

$$\text{cost}(i, j) = \text{out-deg}(j) + \lceil \log n \rceil \cdot (|N(i) - N(j)| + 1) .$$

Given a description of a graph in this kind of compressed format, consider how we would determine where a link from node i encoded using node j as a reference actually points. If the corresponding link from node j is encoded using another node k as a reference, then we would need to determine where the corresponding link from node k points. Eventually, we must reach a link that is encoded without using a reference node. In order to satisfy this requirement, we shall not allow any cycles among references. For example, we shall not allow i to be compressed using j as a reference, j to be compressed using k as a reference, and k to be compressed using i as a reference.

An intermediate structure that FIND-REFERENCE uses is the *affinity graph* G_S for the given Web graph G_W. Specifically, the nodes of G_S are the same as the nodes of G_W. We set $w(i, j)$, the weight of the directed edge from node i to node j, to be the cost of compressing node i using node j as a reference. We add to the affinity graph a *root node* r to which every other node has a directed edge and from which there are no directed edges. The weight of the edge from i to r is the cost of compressing i without using any other node as a reference. We assume that node i has a directed edge to node j if and only if $w(i, j) < w(i, r)$.

Given a Web graph, the algorithm FIND-REFERENCE first computes the corresponding affinity graph for the given cost function, and then finds an optimal set of references under the restrictions that (a) each node has at most one reference, and (b) there are no cycles among references. The problem of finding the globally best mapping from nodes to references (or to the dummy node) is equivalent to finding the minimum weight directed spanning tree with root r on the affinity graph. Thus, a high level description of the compression algorithm is as follows:

Algorithm FIND-REFERENCE

- Given a Web graph G_W, compute the corresponding affinity graph G_S.
- Compute a minimum directed spanning tree D rooted at r for the graph G_S.
- Compress the graph G_W, where node i uses node j as a reference if and only if node i points to node j in D.

Theorem 1 *For a Web graph G_W, let n be the number of nodes in G_W, and let $t_{G_W}(i)$ be the indegree of node i. Algorithm* FIND-REFERENCE *can be realized to run in time $O\left(n \log n + \sum_{i=1}^{n} (t_{G_W}(i)^2)\right)$.*

Proof: The affinity graph G_S can be computed from the original graph G_W by using a matrix multiplication. When G_W is a Web graph, we expect it to be sparse, and so we describe the algorithm in terms of a sparse matrix multiplication. Let M represent the adjacency matrix of G_W. It is easy to verify that $(MM^T)_{ij}$ is the number of nodes that both i and j have outedges to. The matrix (MM^T) can be computed in time $O\left(\sum_{i=1}^{n} t_{G_W}(i)^2\right)$, assuming that we compute a list of the non-zero entries of (MM^T). We also compute an array R, where $R[j] =$ out-deg(j). This requires time $O(n+m)$, where $m = \sum_{i=1}^{n} t_{G_W}(i)$ is the number of edges in G_W. Given $(MM^T)_{ij}$ and $R[j]$, cost(i,j) can be computed in constant time.

Note that there will never be an edge from i to j in G_S unless nodes i and j in G_W have an outgoing edge to at least one shared neighbor. Thus, to compute the set of edges in G_S, we only need to compute cost(i,r) for every i, and then for every edge from i to j such that $(MM^T)_{ij} > 0$, compute cost(i,j), and compare it to cost(i,r). The set of edges in G_S is $\{(i,j)$ s.t. cost$(i,j) <$ cost$(i,r)\}$ and the set of edges from every other vertex to r. This also gives us the weight of each edge in G_S. Since there can be at most $\sum_{i=1}^{n} t_{G_W}(i)^2$ nonzero entries in (MM^T), the total time required to compute the graph G_S is $O\left(n + \sum_{i=1}^{n} t_{G_W}(i)^2\right)$.

Computing a minimum directed spanning tree with root r in a directed graph is generally referred to in the literature as a *branching* with root r.[1] For information on branchings, see for example [7, 9, 11, 19]. Minimum spanning trees in directed graphs with x nodes and y edges can be found deterministically in time $O(x \log x + y)$ [9]. A simpler algorithm that runs in time $O(y \log x)$ is suitable for the case of sparse graphs [19, 7], which will generally be the case in our context. Since the total number of edges in G_S is at most $\sum_{i=1}^{n} t_{G_W}(i)^2 + n$, the total time required to compute the minimum directed spanning tree in G_S is $O\left(n \log n + \sum_{i=1}^{n} t_{G_W}(i)^2\right)$.

All that remains is to perform the compression using the computed directed tree to specify a reference for each node. To do this, we compute for each node i with reference node j a linked list of outedges that i and j have in common. This set of lists can be computed in time

[1] Branchings generally refer to the (equivalent) maximum weight problem. They are sometimes also referred to as arborescences.

$O\left(\sum_{i=1}^{n} t_{G_W}(i)^2\right)$. With the list of edges that i and j have in common, the compressed version of node i can be computed in time $O(\text{out-deg}(i))$. Thus, the entire algorithm runs in time $O\left(n \log n + \sum_{i=1}^{n} t_{G_W}(i)^2\right)$. □

Note that the performance of this algorithm is particularly good when G_W is sparse, as we expect of Web graphs. For example, if the distribution of indegrees in G_W is Zipfian with $\alpha > 3$, then $\sum_{i=1}^{n} t_{G_W}(i)^2 = O(n)$.

3.1 Additional improvements and related problems

In practice, after we have found the references via the directed minimum spanning tree, there are various improvements that can be implemented. For example, we may wish to find additional references for greater compression. This can be done by stripping edges from the original graph handled by the first references, re-calculating the cost function accordingly, and re-running the algorithm. This algorithm is not optimal, since in some cases, better compression is possible if we choose the references of the first stage keeping in mind that we have further stages coming. In general, however, finding multiple references in an optimal manner is NP-hard, as we show in Section 4.

Once we have found the best references, we may again use a Huffman encoding to handle the edges not covered by references. Note that by doing this, we invalidate the cost function we used to determine the references, so that the set of references may not be optimal. However, until we choose the references, we cannot determine the cost of edges not covered by references, so it seems difficult to take this into account properly in the cost function.

Other possible improvements include using different compressed representations. We have suggested using a bit vector to denote which links a node has copied from its reference. These bit vectors can be Huffman or run-length encoded. Although there are a variety of possible enhancements that may slightly improve compression, we believe the main concept of using similar pages for compression provides the bulk of the benefit.

Although our algorithm is designed for storing Web graphs, we believe these techniques can also apply when we wish to compute with the compressed form. The potential problem is that finding the inedges or outedges of a node may require going through multiple references in the directed minimum spanning tree, which may take more computation time than desired. To bound the number of references to pass through in our single-reference setting it is sufficient to bound the depth of the directed minimum spanning tree we find on the affinity graph. Unfortunately, finding the optimal directed minimum spanning tree of bounded depth is NP-hard; for example, if we allow depth at most two, then the problem of finding the optimal directed minimum spanning tree is equivalent to the facility location problem. (This connection to the facility location problem was a major point in the work on compressing tables of data mentioned earlier [6].) In practice, we expect that using the FIND-REFERENCE algorithm to initially find a directed tree and then "chopping the tree" to maintain a depth bound (by changing some nodes to be compressed without a reference and thus linking them to the root r) will be suitable.

4 Hardness results

Since we can find the optimal compression given an appropriate cost metric when we allow a single reference node using branching algorithms, a natural question to ask is whether we can

similarly achieve optimality when we allow more than one reference node. We show hardness results related to this question. We focus on the case where up to two nodes can be used as references, but everything described is easily generalized to any number of reference nodes.

For up to two reference nodes, the affinity graph becomes the following kind of structure:

Definition 1 *A* 2-supergraph *is a directed hypergraph where each hyperedge is directed from a single node to two other nodes. These two other nodes can be the same, but must be different from the source node.*

Given a Web graph, we shall consider the corresponding weighted 2-supergraph, where w_{ijk}, the weight of the hyperedge from i to j and k, represents the cost of encoding i using both j and k as references. The weight w_{ijj} represents the cost of encoding node i using only node j as a reference; in the corresponding hyperedge the node i points to j twice. Note that w_{ijk} varies depending on the overlap between the set of edges of the Web graph that nodes i and j have in common and the set of edges that nodes i and k have in common. We call the resulting 2-supergraph an *affinity 2-supergraph.*

Given a Web graph, computing the affinity 2-supergraph for a given link compression scheme can easily be done in polynomial time. Using the affinity 2-supergraph to compute the best compression using up to two reference nodes is equivalent to the following generalization of finding optimal branchings:

Definition 2 *Given a 2-supergraph $\mathcal{G}$ and a designated root node r, a* 2-branching *is a subset S of the hyperedges of $\mathcal{G}$ such that each node except the designated root has exactly one outgoing hyperedge in S, and r has no outgoing hyperedges in S. In addition, the hypergraph formed by the set of hyperedges in S has no directed cycles. The optimum 2-branching is the 2-branching that minimizes the total weight of the edges in S.*

Unfortunately, in general finding the optimal 2-branching is not only NP-Hard, it is hard to approximate. We demonstrate an approximation preserving polynomial time reduction from the problem of finding the optimal directed Steiner tree to the problem of finding the optimal 2-branching. It is known that if $P \neq NP$, then no polynomial time algorithm can find a $\log n$-approximation to the directed Steiner tree problem [8].

Theorem 2 *Any polynomial algorithm that provides a k-approximation for the 2-branching problem also provides a k-approximation for the directed Steiner tree problem.*

Proof: Please see the extended abstract [1].

The inapproximability result demonstrates the hardness of the general problem of finding an optimal 2-branching. This result does not however directly imply that it is even NP-Hard to find the best compression using at most two references, since the graphs that we reduce the directed Steiner tree problem to may not correspond to actual affinity graphs that arise as a result of a Web graph.

We also provide a more direct reduction showing that it is in fact NP-Hard to find the best compression of a Web graph based on using up to two reference nodes. In fact, even if we ignore the additional difficulty imposed by taking into account the asymmetry of the affinity graph, the problem remains NP-Hard. In particular, we demonstrate that the problem of finding the assignment of reference nodes that maximizes the total number of edges in the Web graph that are represented by a corresponding edge in a reference node is NP-Hard.

Theorem 3 *The problem of finding an encoding for a graph G_W, with each node encoded using up to two reference nodes, that maximizes the total number of edges that are encoded using a reference node is NP-Hard.*

Proof: Please see the extended abstract [1].

5 Experiments

We present the results from a preliminary prototype running on artificial random copy graphs and one subset of a snapshot of the Web graph. We emphasize that these experiments are meant as a preliminary proof of concept. In particular, the prototype does not output a compressed file, but rather the compressed size of the file. Moreover, when using Huffman coding, the compressed size does not include the size of any associated Huffman tables. We chose not to include this as the size of the Huffman table depends on whether one compresses it further; this makes the Huffman scheme look better than it would be in practice.

We first describe the graphs we tested. For the random copy graphs, our tests all had 131072 nodes. (Smaller test graphs had similar performance.) Each graph began with 1024 seed nodes with three outedges, where the end of the outedge was chosen uniformly at random from all nodes. When new nodes were added, they were given only outedges. The outedges were determined by copying edges from some number of nodes and by generating edges with endpoints chosen uniformly at random from all present nodes. We show the parameters for the copy graphs tested in Table 1. The field # random copies denotes the number of nodes whose outedges were copied. A range such as $[1, 2]$ denotes that an integer value was chosen uniformly over that range. Each edge was copied with a fixed probability, listed as the copy probability. The field # random edges gives the number of edges that were generated with random destinations; again, a range denotes that an integer value was chosen uniformly over that range. We note that for the large graphs G_3 and G_4, we were forced by memory considerations to limit the affinity graph to allow edges between nodes i and j only if their outedges share at least three destinations. This can only hurt our compression efforts. Further testing suggests that the difference is minimal if two shared destinations can be handled.

We also tested our compression scheme on real Web data from from the TREC-8 (Text REtrieval Conference 8) Web track [10]. Our data set was the WT2g data set which was chosen as a small subset of the Web for testing information by the TREC retrieval conference. This data set is larger than our random sets; hence again to construct the affinity graph on the TREC database we only created edges between pages with at least three shared links.

Table 2 presents the compression results in terms of total bits required divided by edges in the graph. For the random graphs, we have taken the average of ten trials, where a different random graph is produced for each trial. We note that there is little deviation between the runs. For the uncompressed size in bits per edge, we use the underestimate $\log_2$ (#nodes).

As seen in graph G_1, when the amount of copying is low, and thus the average degree is very small, the reference algorithm alone does slightly worse than the Huffman algorithm, although using a Huffman code in conjunction with the reference algorithm leads to better performance. When the amount of copying is larger, as for G_2, G_3, and G_4, our FIND-REFERENCE algorithm greatly outperforms Huffman coding. We expect repeated passes might allow even greater compression. The Huffman algorithm compresses the outedges for each edge, so the code words are based on the indegree. For the FIND-REFERENCE algo-

rithm, we test both the straightforward algorithm as well as one which first determines the references and then uses Huffman coding on the remaining outedges.

Our results are actually best for the TREC database, demonstrating that our approach should be effective on real Web data as well. Our belief is that our good results on the TREC data set arise because links appear to have significant locality. Indeed, the strong effect of locality has been corroborated by the unpublished related work on the connectivity server [17]. They use the fact that most Web links are between pages on the same server to find good references quickly. This lack of acknowledgment of locality in current Web models is problematic, although it is being addressed [16]. Note that our approach can be used to take advantage of locality; for example, our reference-based algorithm could be run on a host-by-host basis, using for example only the outlinks on a given host, to improve speed.

Name	G_1	G_2	G_3	G_4	TREC
Nodes	131072	131072	131072	131072	247428
# Random Copies	1	1	[1,2]	[0,4]	NA
Copy prob.	0.5	0.7	0.5	0.5	NA
# Random Edges	1	1	[1,2]	[1,2]	NA

Table 1: Parameters of the test graphs.

Name	G_1	G_2	G_3	G_4	TREC
Uncompressed	17.00	17.00	17.00	17.00	18.00
Huffman	14.92	14.27	14.48	13.51	15.00
FIND-REFERENCE	15.08	11.47	11.89	10.48	8.85
F.Ref. + Huff.	13.87	10.82	11.11	9.20	8.35

Table 2: Results from the test graphs; bits per edge.

6 Future Work

We have initiated study into how to compress Web graphs using the copy graph model, a random graph family with properties similar to Web graphs. Using this structure, we have designed a compression algorithm based on finding similarity among the links of the pages and tested it on simple copy graphs. We have also shown that various generalizations of this idea lead to NP-Hard problems.

There are several directions remaining to pursue, including refining algorithms for dealing with the locality of pages in the same domain. Although designing such a system can be done via experimentation, developing an appropriate model that allows us to understand the trade-offs would be an interesting problem. Another interesting issue is whether these compression algorithms can be used to test proposed random graph models. A possible technique to test how accurately a proposed random graph model captures the structure of real Web graphs would be to run our compression algorithm (or any other compression algorithm) on both kinds of graphs, and compare the compression obtained.

References

[1] M. Adler and M. Mitzenmacher. Towards compressing Web graphs. U. of Mass. CMPSCI Technical Report 00-39. Available at ftp://ftp.cs.umass.edu/pub/techrept/techreport/2000.

[2] W. Aiello, F. Chung, and L. Lu. A random graph model for massive graphs. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pages 171-180, 2000.

[3] K. Bharat, A. Broder, M. Henzinger, P. Kumar, and S. Venkatasubramanian. The Connectivity Server: fast access to linkage information on the Web. In *Proceedings of the 7th World Wide Web Conference*, 1998. Available at http://www7.scu.edu.au/programme/fullpapers/1938/com1938.htm.

[4] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. In *Proceedings of the 7th World Wide Web Conference*, 1998. Available at http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm.

[5] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the Web: experiments and models. In *Proceedings of the 9th World Wide Web Conference*, 2000. Available at http://www9.org/w9cdrom/index.html.

[6] A. L. Buchsbaum, D. F. Caldwell, K. W. Church, G. S. Fowler, and S. Muthukrishnan. Engineering the compression of massive tables: an experimental approach. In *Proceedings of 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 175-184, 2000.

[7] P. M. Camerini, L. Fratta, and F. Maffioli. A note on finding optimum branchings. *Networks*, 9:309-312, 1979.

[8] M. Charikar, C. Chekuri, T. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li. Approximation algorithms for directed Steiner problems. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 192-200, San Francisco, California, 25-27 January 1998.

[9] H. N. Gabow, Z. Galil, T. Spencer, and R. E. Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica* 6(2):109-122, 1986.

[10] D. Hawking, E. Voorhees, N. Craswell, and P. Bailey. Overview of the TREC-8 Web Track. In *The 8th Text Retrieval Conference*, 2000. Available at `http://trec.nist.gov/pubs/trec8/t8_proceedings.html`.

[11] R. M. Karp. A simple derivation of Edmonds' algorithm for optimum branchings. *Networks*, 1:265-272, 1972.

[12] J. Kleinberg. Authoritative sources in a hyperlinked environment. In *Proceedings of 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 668-677, 1998.

[13] J. Kleinberg, R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. The Web as a graph: Measurements, Models, and Methods. In *Proceedings of the International Conference on Combinatorics and Computing*, 1999.

[14] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling the Web for emerging cybercommunities. In *Proceedings of the 8th World Wide Web Conference*, 1999.

[15] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Extracting large scale knowledge bases from the Web. In *Proceedings of the 25th VLDB Conference*, 1999.

[16] M. Mitzenmacher and J. Wiener. Improved graph models for the Web. Manuscript in preparation.

[17] R. Stata and J. Wiener. Personal communication, on the Compaq Systems Research Center connectivity server.

[18] S. R. Tate. Band Ordering in Lossless Compression of Multispectral Images. *IEEE Transactions on Computers*, Vol. 46, No. 4, 1997, pp. 477-483.

[19] R. E. Tarjan. Finding Optimum Branchings. *Networks*, 7:25-35, 1977.

[20] I. Witten, A. Moffat, T. Bell. **Managing Gigabytes: 2nd Edition**. Morgan Kaufmann, San Francisco, 1999.