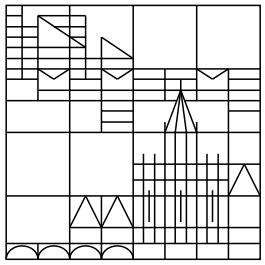

Universität Konstanz

Faster Evaluation of Shortest-Path Based Centrality Indices

Ulrik Brandes

Konstanzer Schriften in Mathematik und Informatik
Nr. 120, Mai 2000

ISSN 1430–3558

Universität Konstanz
Fach D 188, 78457 Konstanz, Germany
Email: preprints@informatik.uni–konstanz.de
WWW: http://www.informatik.uni–konstanz.de/Schriften/

Faster Evaluation of Shortest-Path Based Centrality Indices*

Ulrik Brandes

Preprint 120/2000
University of Konstanz, Department of Computer and Information Science,
Box D 188, 78457 Konstanz, Germany
Ulrik.Brandes@uni-konstanz.de

Abstract. Centrality indices are an important tool in network analysis, and many of them are derived from the set of all shortest paths of the underlying graph. The so-called betweenness centrality index is essential for the analysis of social networks, but most costly to compute. Currently, the fastest known algorithms require $\Theta(n^3)$ time and $\Theta(n^2)$ space, where n is the number of vertices.
Motivated by the fast-growing need to compute centrality indices on large, yet very sparse, networks, new algorithms for betweenness are introduced in this paper. They require $\mathcal{O}(n+m)$ space and run in $\mathcal{O}(n(m+n))$ or $\mathcal{O}(n(m+n\log n))$ time on unweighted or weighted graphs, respectively, where m is the number of edges. Since these algorithms simply augment single-source shortest-paths computations, all standard centrality indices based on shortest paths can now be computed uniformly in one framework. Experimental evidence is provided that this substantially increases the range of networks for which centrality analysis is feasible.

1 Introduction

Social network analysis is a subdiscipline of the social sciences, using graph-theoretic concepts to understand and explain social phenomena. A social network consists of a set of actors, who may be arbitrary entities like persons or organizations, and one or more types of relations between them. For a comprehensive overview of methods and applications see [22, 18].

An essential tool for the analysis of social networks are centrality indices defined on the vertices of the graph [6, 17, 11]. They are designed to value the position of vertices in a graph, and interpreted as the importance of an actor in its surrounding social structure. Many centrality indices are based on shortest paths, measuring, e.g., the average distance from other vertices, or the ratio of shortest paths a vertex lies on. Many network-analytic studies rely at least in part on an evaluation of these indices.

* Part of this research was done while with the Department of Computer Science at Brown University. I gratefully acknowledge financial support from the German Academic Exchange Service (DAAD, Hochschulsonderprogramm III).

With the increasing practicality of electronic data collection and, of course, the advent of the Web, there is a likewise increasing demand for the computation of centrality indices on networks with thousands of actors. However, an $\Theta(n^3)$ bottleneck in existing implementations, mainly due to the particularly popular betweenness centrality index [11], makes centrality analyses of networks with more than a few hundred actors prohibitive. As a remedy, network analysts are now suggesting other indices, for instance based only on linkages between neighboring vertices [8], to at least obtain rough approximations of betweenness centrality.

In this paper, we show that betweenness can be computed exactly even for much larger networks by introducing more efficient algorithms. They are based on a new accumulation technique that integrates well with traversal algorithms solving the single-source shortest-paths problem, and thus exploiting the sparsity of typical instances. The range of graphs for which betweenness centrality can be computed is thereby extended significantly. Moreover, it turns out that all standard centrality indices based on shortest paths can be evaluated simultaneously, further reducing both the time and space requirements of comparative analyses.

The centrality indices relevant here are defined in Sect. 2. In Sect. 3, we review methods computing all shortest paths between all pairs of vertices in a graph. A recursion formula for accumulating betweenness centrality is derived in Sect. 4, and its practical implications are validated by experiments discussed in Sect. 5.

2 Centrality Indices Based on Shortest Paths

Social and other networks are conveniently described as a graph $G = (V, E)$, where the set V of vertices represents actors, and the set E of edges represents ties between actors. We use n and m to denote the number of vertices and edges, respectively. For simplicity, we assume that all graphs are undirected and connected, though they may have loops or multiple edges.

Let ω be a *weight function* on the edges. We assume that $\omega(e) > 0$, $e \in E$, for *weighted* graphs, and define $\omega(e) = 1$, $e \in E$, for *unweighted* graphs. Weights are used to measure, e.g., the strength of a tie.

Define a *path* from $s \in V$ to $t \in V$ as an alternating sequence of vertices and edges, beginning with s and ending with t, such that each edge connects its preceding with its succeeding vertex. The *length* of a path is the sum of the weights of its edges. We use $d_G(s,t)$ to denote the *distance* between vertices s and t, i.e. the minimum length of any path connecting s and t in G. By definition, $d_G(s,s) = 0$ for every $s \in V$, and $d_G(s,t) = d_G(t,s)$ for $s,t \in V$. We assume familiarity with standard algorithms for shortest-paths problems (see, e.g., [7]).

Several measures capture variations on the notion of a vertex's importance in a graph. Let $\sigma_{st} = \sigma_{ts}$ denote the number of shortest paths from $s \in V$ to $t \in V$, where $\sigma_{ss} = 1$ by convention. Let $\sigma_{st}(v)$ denote the number of shortest paths from s to t that some $v \in V$ lies on. The following are standard measures

of centrality:

$$C_C(v) = \frac{1}{\sum_{t \in V} d_G(v,t)} \qquad (\textit{closeness centrality } [17])$$

$$C_G(v) = \frac{1}{\max_{t \in V} d_G(v,t)} \qquad (\textit{graph centrality } [13])$$

$$C_S(v) = \sum_{s \neq v \neq t \in V} \sigma_{st}(v) \qquad (\textit{stress centrality } [19])$$

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}} \qquad (\textit{betweenness centrality } [3, 10])$$

High centrality scores thus indicate that a vertex can reach others on relatively short paths, or that a vertex lies on considerable fractions of shortest paths connecting others. For interpretability, i.e. to control for the size of the network, the above indices are usually normalized to lie between zero and one. Though their definitions extend naturally to directed or disconnected graphs, normalization then becomes a problem with some of the above measures. The inhomogeneity of a centrality index is used to define the *centralization* of a graph with respect to that index [11]. A theoretical foundation for centrality measures not based on shortest paths is given in [12]. See [22] for further details and note that we tacitly generalized some of these definitions of centrality to weighted graphs.

The computationally rather involved betweenness centrality index is arguably the one most frequently analyzed in social network analysis. However, the sheer size of many instances occurring in practice makes the evaluation of betweenness centrality prohibitive. In the following, we therefore focus on computing betweenness. As it turns out, the resulting algorithm can trivially be augmented to compute the other measures as well, at virtually no extra cost. First, recall the following crucial observation.

Lemma 1 (Bellman criterion). *A vertex $v \in V$ lies on a shortest path between vertices $s, t \in V$, if and only if $d_G(s,t) = d_G(s,v) + d_G(v,t)$.*

Given pairwise distances and shortest paths counts, the *pair-dependency* $\delta_{st}(v) = \frac{\sigma_{st}(v)}{\sigma_{st}}$ of a pair $s, t \in V$ on an intermediary $v \in V$, i.e. the ratio of shortest paths between s and t that v lies on, is given by

$$\sigma_{st}(v) = \begin{cases} 0 & \text{if } d_G(s,t) < d_G(s,v) + d_G(v,t) \\ \sigma_{sv} \cdot \sigma_{vt} & \text{otherwise} \end{cases}.$$

To obtain the betweenness centrality index of a vertex v, we simply have to sum the pair-dependencies of all pairs on that vertex,

$$C_B(v) = \sum_{s \neq v \neq t \in V} \delta_{st}(v).$$

Therefore, betweenness centrality is traditionally determined in two steps:

1. compute the length and number of shortest paths between all pairs
2. sum all pair-dependencies

3 Counting the Number of Shortest Paths

In this section, we observe that the complexity of determining betweenness centrality is, in fact, dominated by the second step, i.e. the $\Theta(n^3)$ time summation and $\Theta(n^2)$ storage of pair-dependencies. This situation is remedied in the next section.

The two implementations most widely used to compute betweenness are UCINet [2] and SNAPS.[1] Probably because of a reference to [14] in [11], the latter appear to make use of the following lemma. Recall that the *adjacency matrix* of a graph is the $n \times n$-matrix $A = (a_{uv})_{u,v \in V}$ with $a_{uv} = 1$ if $\{u, v\} \in E$, and $a_{uv} = 0$ otherwise.

Lemma 2 (Algebraic path counting). *Let $A^k = (a_{uv}^{(k)})_{u,v \in V}$ be the k-th power of the adjacency matrix of an unweighted graph. Then $a_{uv}^{(k)}$ equals the number of paths from u to v of length exactly k.*

Since $\mathcal{O}(n^2)$ pair-dependencies need to be summed for each vertex, the overall running time of these implementations is dominated by the time spend on matrix multiplications.

Clearly, algebraic path counting computes more information than needed. Instead of the number of paths of length shorter than the diameter of the network (the maximum distance of any pair of vertices), we are only interested in the number of shortest paths between each pair of vertices.

Some superfluous work is avoided in a suitably defined instance, called *geodetic semiring* [5], of the closed semiring generalization for shortest paths problems [1]. It yields an $\Theta(n^3)$ algorithm for betweenness by augmenting the Floyd/Warshall algorithm for the all-pairs shortest-paths problem with path counting.

To exploit the sparsity of typical instances, we will count shortest paths using traversal algorithms. Both breadth-first search (BFS) for unweighted and Dijkstra's algorithm for weighted graphs start with a specified source $s \in V$ and, at each step, add a closest vertex the set of already discovered vertices in order to find shortest paths from the source to all other vertices. In that process, they naturally discover all shortest paths from the source. Define the set of *predecessors* of a vertex v on shortest paths from s as

$$P_s(v) = \{u \in V \,:\, \{u, v\} \in E,\, d_G(s, v) = d_G(s, u) + \omega(u, v)\}\,.$$

Lemma 3 (Combinatorial shortest-path counting). *For $s \neq v \in V$*

$$\sigma_{sv} = \sum_{u \in P_s(v)} \sigma_{su}\,.$$

Proof. Since all edge weights are positive, the last edge of any shortest path from s to v is an edge $\{u, v\} \in E$ such that $d_G(s, u) < d_G(s, v)$. Clearly, the number of shortest paths from s to v ending with this edge equals the number of shortest paths from s to u. The equality now follows from Lemma 1.

[1] A collection of Gauss-routines [4] by Noah Friedkin of University of California, Santa Barbara.

Both Dijkstra's algorithm and BFS are thus easily augmented to count the number of shortest paths according to this lemma. BFS takes time $\mathcal{O}(m+n)$, and Dijkstra's algorithm runs in time $\mathcal{O}(m+n\log n)$, if the priority queue is implemented with a Fibonacci heap [9].

Corollary 1. *Given a source $s \in V$, both the length and number of all shortest paths to other vertices can be determined in time $\mathcal{O}(m+n\log n)$ for weighted, and in time $\mathcal{O}(m+n)$ for unweighted graphs.*

Consequently, σ_{st}, $s,t \in V$, can be computed in time $\mathcal{O}(n(m+n))$ for unweighted and in time $\mathcal{O}(n(m+n\log n))$ for weighted graphs.

4 Accumulation of Pair-Dependencies

Corollary 1 implies that running time is dominated by the $\Theta(n^3)$ time it takes to sum pair-dependencies. Apparently, this is the approach implemented in UCINet [2]. Clearly, the $\Theta(n^2)$ space bound stems from the need to store the distance matrix and quantities σ_{st}, $s,t \in V$. Both complexities can be reduced substantially by first observing that partial sums of pair-dependencies obey a recursive relation. To see this, define the *dependency* of a vertex $s \in V$ on $v \in V$ as

$$\delta_{s\bullet}(v) = \sum_{t \in V} \delta_{st}(v).$$

In the following special case, the relation between the dependencies of a single vertex is particularly easy to recognize.

Lemma 4. *If there is exactly one shortest path from $s \in V$ to each $t \in V$, the dependency of s on any $v \in V$ obeys*

$$\delta_{s\bullet}(v) = \sum_{w\,:\,v \in P_s(w)} (1 + \delta_{s\bullet}(w))\,.$$

Proof. The assumption implies that the vertices and edges of all shortest paths form a tree. Therefore, v lies on either all or none of the paths between s and some $t \in V$, i.e. $\delta_{st}(v)$ equals either 1 or 0. Moreover, v lies on all shortest paths to those vertices for which it is a predecessor, and on all shortest paths that these lie on (see also Figure 1).

In the general case, a very similar relation holds.

Theorem 1. *The dependency of $s \in V$ on any $v \in V$ obeys*

$$\delta_{s\bullet}(v) = \sum_{w\,:\,v \in P_s(w)} \frac{\sigma_{sv}}{\sigma_{sw}} \cdot (1 + \delta_{s\bullet}(w))\,.$$

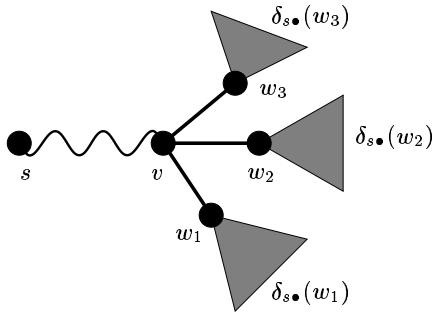

Fig. 1. With the assumption of Lemma 4, a vertex lies on all shortest paths to its successors in the tree of shortest paths from the source

Proof. Recall that $\delta_{st}(v) > 0$ only for those $t \in V \setminus \{s\}$ for which v lies on at least one shortest path from s to t, and notice that on any such path there is exactly one edge $\{v, w\}$ with $v \in P_s(w)$. This slightly more complicated situation is illustrated in Figure 2.

We extend pair-dependency to include an edge $e \in E$ by defining $\delta_{st}(v, e) = \frac{\sigma_{st}(v,e)}{\sigma_{st}}$ where $\sigma_{st}(v, e)$ is the number of shortest paths from s to t that contain both v and e. Then,

$$\delta_{s\bullet}(v) = \sum_{t \in V} \delta_{st}(v) = \sum_{t \in V} \sum_{w\,:\,v \in P_s(w)} \delta_{st}(v, \{v, w\}) = \sum_{w\,:\,v \in P_s(w)} \sum_{t \in V} \delta_{st}(v, \{v, w\}).$$

Let w be any vertex with $v \in P_s(w)$. Of the σ_{sw} shortest paths from s to w, σ_{sv} many first go from s to v and then use $\{v, w\}$. Consequently, $\frac{\sigma_{sv}}{\sigma_{sw}} \cdot \sigma_{st}(w)$ shortest paths from s to some $t \neq w$ contain v and $\{v, w\}$. It follows that the pair-dependency of s and t on v and $\{v, w\}$ is

$$\delta_{st}(v, \{v, w\}) = \begin{cases} \frac{\sigma_{sv}}{\sigma_{sw}} & \text{if } t = w \\ \frac{\sigma_{sv}}{\sigma_{sw}} \cdot \frac{\sigma_{st}(w)}{\sigma_{st}} & \text{if } t \neq w \end{cases}$$

Inserting this into the above yields

$$\begin{aligned} \sum_{w\,:\,v \in P_s(w)} \sum_{t \in V} \delta_{st}(v, \{v, w\}) &= \sum_{w\,:\,v \in P_s(w)} \left(\frac{\sigma_{sv}}{\sigma_{sw}} + \sum_{t \in V \setminus \{w\}} \frac{\sigma_{sv}}{\sigma_{sw}} \cdot \frac{\sigma_{st}(w)}{\sigma_{st}} \right) \\ &= \sum_{w\,:\,v \in P_s(w)} \frac{\sigma_{sv}}{\sigma_{sw}} \cdot (1 + \delta_{s\bullet}(w)) \,. \end{aligned}$$

Corollary 2. *Given the directed acyclic graph of shortest paths from $s \in V$ in G, the dependencies of s on all other vertex can be computed in $\mathcal{O}(m)$ time and $\mathcal{O}(n + m)$ space.*

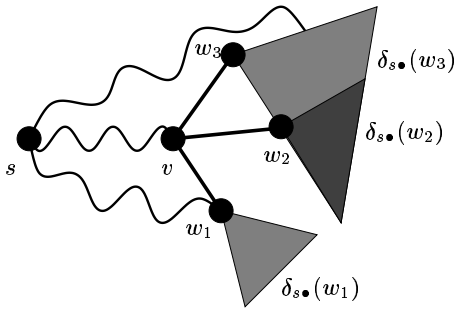

Fig. 2. In the general case of Theorem 1, fractions of the dependencies on successors are propagated up along the edges of the directed acyclic graph of shortest paths from the source

Proof. Traverse the vertices in non-increasing order of their distance from s and accumulate dependencies by applying Theorem 1. We need to store a dependency per vertex, and lists of predecessors. There is at most one element per edge in any of these lists.

With this result, we can determine the betweenness centrality index by solving one single-source shortest-paths problem for each vertex. At the end of each iteration, the dependencies of the source on each other vertex are added to the centrality score of that vertex. For unweighted graphs, this is described more precisely in Algorithm 1. The modifications necessary for weighted graphs are straightforward.

Theorem 2. *Betweenness centrality can be computed in $\mathcal{O}(n(m+n\log n))$ time and $\mathcal{O}(n+m)$ space for weighted graphs. For unweighted graphs, running time reduces to $\mathcal{O}(n(m+n))$.*

The other shortest-path based centrality measures defined in Sect. 2 are easily computed during the execution of single-source shortest-paths traversals. The same holds for a recently introduced index called *radiality* [20]

$$C_R(v) = \frac{\sum_{t \in V} (D(G) + 1 - d_G(v,t))}{(n-1) \cdot D(G)},$$

where $D(G) = \max_{s,t \in V} d_G(s,t)$, and for other potential measures as well. This is a significant practical advantage, reducing the combined time and space spent on computing different measures that are to be compared.

Finally, we note that centrality computations for many shortest-path based indices can be sped up heuristically by first decomposing an undirected graph into its biconnected components.

5 Practical Implications

In this section, we evaluate the practical relevance of the asymptotic complexity improvement achieved by accumulating dependencies. Weighted and un-

Algorithm 1: Betweenness centrality in unweighted graphs

```
C_B[v] ← 0, v ∈ V;
for s ∈ V do
    S ← empty stack;
    P[w] ← empty list, w ∈ V;
    σ[t] ← 0, t ∈ V;   σ[s] ← 1;
    d[t] ← −1, t ∈ V;   d[s] ← 0;
    Q ← empty queue;
    enqueue s → Q;
    while Q not empty do
        dequeue v ← Q;
        push v → S;
        foreach neighbor w of v do
            if d[w] < 0 then
                enqueue w → Q;
                d[w] ← d[v] + 1;
            end
            if d[w] = d[v] + 1 then
                σ[w] ← σ[w] + σ[v];
                append v → P[w];
            end
        end
    end
    δ[v] ← 0, v ∈ V;
    while S not empty do
        pop w ← S;
        for v ∈ P[w] do δ[v] ← δ[v] + (σ[v]/σ[w]) · (1 + δ[w]);
        if w ≠ s then C_B[w] ← C_B[w] + δ[w];
    end
end
```

weighted versions of our algorithm have been implemented for directed and undirected graphs using the Library of Efficient Data Structures and Algorithms (LEDA) [16]. Performance is compared with an implementation that uses the same code to determine the length and number of shortest paths between all pairs of vertices, but explicitly determines all pair-dependencies. Note that this is at most faster than implementations currently in use. The experiment was performed on a Sun Ultra 10 SparcStation with 440 MHz clock speed and 256 MBytes main memory.

Figure 3 shows running times for betweenness centrality on 180 random undirected unweighted graphs with 100 to 2000 vertices. For each number of vertices, there are nine graphs with 10% to 90% density (defined as the number of edges divided by $\frac{n(n-1)}{2}$). First notice that running times of the standard algorithm vary only slightly, indicating that most of the time is spent on summing pair-dependencies, rather than counting paths. Additional experiments confirmed that the overhead to determine the number of shortest paths in weighted graphs is negligible. As expected, accumulation according to Theorem 1 yields a significant speedup. This is true even for dense graphs, because the $\mathcal{O}(m)$ bound of Corollary 2 is, at least in general, overly pessimistic for the number of edges on shortest paths from some source.

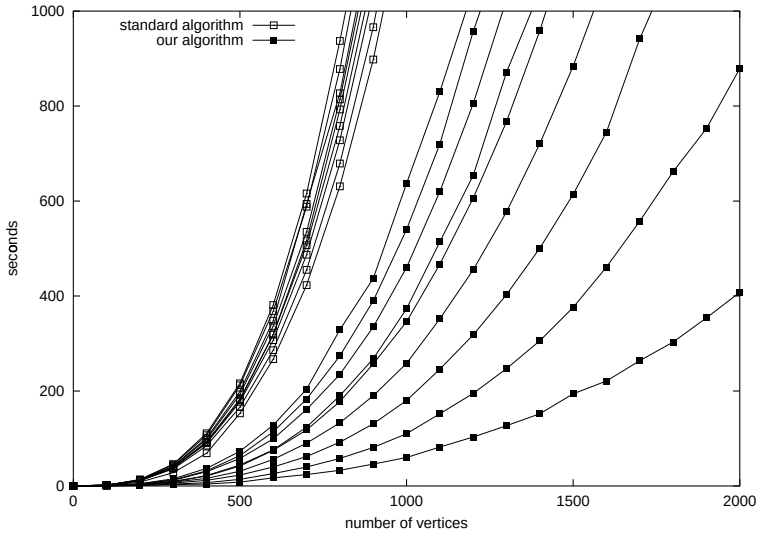

Fig. 3. Seconds needed to the compute betweenness centrality index for random undirected, unweighted graphs with 100 to 2000 vertices and densities ranging from 10% to 90%

Since large instances arising in social network analysis are typically rather sparse, with density well below 10%, the experiment clearly indicates that be-

tweenness centrality can be computed for graphs of significantly larger size by accumulating dependencies instead of summing pair-dependencies.

The speed-up was also validated in practice, by analysis of an instance of 4,259 intravenous drug users with 61,693 directed weighted ties, originating from 197,216 unique contacts.[2] Not only because of running time, but also because of the memory required to store the distance and shortest-paths count matrices, betweenness centrality could not be evaluated for this network to date. The largest subnetwork previously analyzed had 494 actors with 1,774 ties (taking 25 minutes on a 200 MHz Pentium Pro PC). Our implementation determined the betweenness centrality index of the whole network in 448 seconds, using less than 8 MBytes of memory.

Note that the average sum of in- and outdegrees in this network is less than 29, corresponding to 0.3% density. (Clearly, density tends to zero when the average degree is fixed.) Recent experiments estimate an even lower average outdegree of 7.2 for Web pages [15]. Figure 4 gives running times for betweenness index calculations on random graphs with a fixed average vertex degree of 20. These results imply that our implementation can compute, e.g., the betweenness centrality index for an extract of more than 10,000 Web pages in less than an hour on standard equipment.

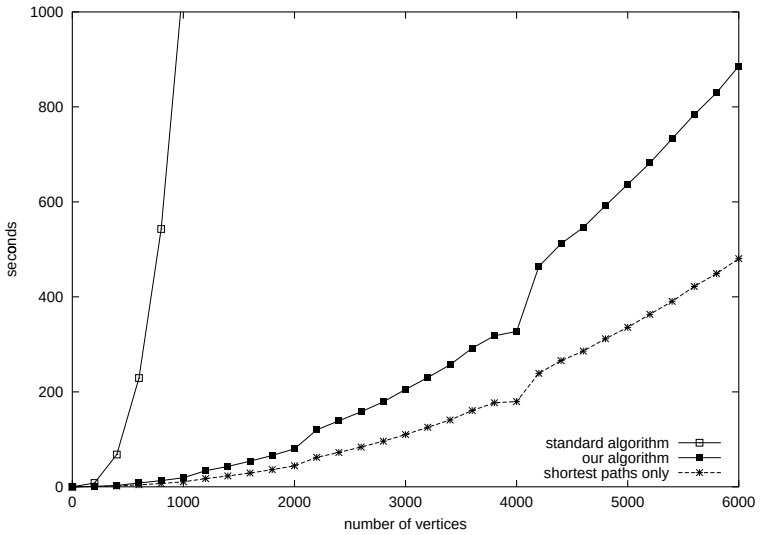

Fig. 4. Seconds needed to the compute betweenness centrality index for random undirected, unweighted graphs with constant average degree 20. The funny jumps are attributed to LEDA's caching behavior

[2] Courtesy of Robert Foreman and Thomas Valente of the Epidemiology Data House at Johns Hopkins University. See [21] for background on the data.

References

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms.* Addison–Wesley, 1974.
2. Analytic Technologies. *UCINet V.* Network analysis software.
3. J. M. Anthonisse. The rush in a directed graph. Technical Report BN 9/71, Stichting Mathematisch Centrum, Amsterdam, Oct. 1971.
4. Aptech Systems, Inc. *The GAUSS Mathematical and Statistical System.* Matrix programming language.
5. V. Batagelj. Semirings for social network analysis. *Journal of Mathematical Sociology*, 19(1):53–68, 1994.
6. A. Bavelas. A mathematical model for group structure. *Human Organizations*, 7:16–30, 1948.
7. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.
8. M. G. Everett, S. P. Borgatti, and D. Krackhardt. Ego-network betweenness. Paper presented at the *19th International Conference on Social Network Analysis (Sunbelt XIX)*, Charleston, South Carolina, 1999.
9. M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the Association for Computing Machinery*, 34(3):596–615, 1987.
10. L. C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40:35–41, 1977.
11. L. C. Freeman. Centrality in social networks: Conceptual clarification. *Social Networks*, 1:215–239, 1979.
12. N. E. Friedkin. Theoretical foundations for centrality measures. *American Journal of Sociology*, 96(6):1478–1504, May 1991.
13. P. Hage and F. Harary. Eccentricity and centrality in networks. *Social Networks*, 17:57–63, 1995.
14. F. Harary, R. Z. Norman, and D. Cartwright. *Structural Models: An Introduction to the Theory of Directed Graphs.* John Wiley & Sons, 1965.
15. J. M. Kleinberg, R. Kumar, P. Raghavan, S. Rajagopalan, and A. S. Tomkins. The Web as a graph: Measurements, models, and methods. In T. Asano, H. Imai, D. T. Lee, S. Nakano, and T. Tokuyama, editors, *Proceedings of the 5th International Conference on Computing and Combinatorics (COCOON '99)*, volume 1627 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 1999.
16. K. Mehlhorn and S. Näher. *The LEDA Platform of Combinatorial and Geometric Computing.* Cambridge University Press, 1999. Project home page at `http://www.mpi-sb.mpg.de/LEDA/`.
17. G. Sabidussi. The centrality index of a graph. *Psychometrika*, 31:581–603, 1966.
18. J. Scott. *Social Network Analysis: A Handbook.* Sage Publications, 1991.
19. A. Shimbel. Structural parameters of communication networks. *Bulletin of Mathematical Biophysics*, 15:501–507, 1953.
20. T. W. Valente and R. K. Foreman. Integration and radiality: Measuring the extent of an individual's connectedness and reachability in a network. *Social Networks*, 20(1):89–105, 1998.
21. T. W. Valente, R. K. Foreman, B. Junge, and D. Vlahov. Satellite exchange in the Baltimore needle exchange program. *Public Health Reports*, 113(S1):90–96, 1998.
22. S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications.* Cambridge University Press, 1994.