Parallel Algorithms for Evaluating Centrality Indices in Real-world Networks

David A. Bader Kamesh Madduri
College of Computing
Georgia Institute of Technology, Atlanta GA 30332
{bader,kamesh}@cc.gatech.edu

Abstract

This paper discusses fast parallel algorithms for evaluating several centrality indices frequently used in complex network analysis. These algorithms have been optimized to exploit properties typically observed in real-world large scale networks, such as the low average distance, high local density, and heavy-tailed power law degree distributions. We test our implementations on real datasets such as the web graph, protein-interaction networks, movie-actor and citation networks, and report impressive parallel performance for evaluation of the computationally intensive centrality metrics (betweenness and closeness centrality) on high-end shared memory symmetric multiprocessor and multithreaded architectures. To our knowledge, these are the ***first parallel implementations*** *of these widely-used social network analysis metrics. We demonstrate that it is possible to rigorously analyze networks* ***three orders of magnitude*** *larger than instances that can be handled by existing network analysis (SNA) software packages. For instance, we compute the exact betweenness centrality value for each vertex in a large US patent citation network (3 million patents, 16 million citations) in 42 minutes on 16 processors, utilizing 20GB RAM of the IBM p5 570. Current SNA packages on the other hand cannot handle graphs with more than hundred thousand edges.*

1 Introduction

Large-scale network analysis is an exciting field of research with applications in a variety of domains such as social networks (friendship circles, organizational networks), the internet (network topologies, web graphs, peer-to-peer networks), transportation networks, electrical circuits, genealogical research and bio-informatics (protein-interaction networks, food webs). These networks seem to be entirely unrelated and indeed represent quite diverse relations, but experimental studies [5, 23, 11, 36, 35] have shown that they share some common traits such as a low average distance between the vertices, heavy-tailed degree distributions modeled by power laws, and high local densities. Modeling these networks based on experiments and measurements, and the study of interesting phenomena and observations [13, 18, 42, 52], continue to be active areas of research. Several models [26, 37, 40, 51, 15] have been proposed to generate synthetic graph instances with scale-free properties.

Network analysis is currently a very active area of research in the social sciences [44, 50, 47, 25], and seminal contibutions to this field date back to more than sixty years. There are several analytical tools [32, 6, 45] for visualizing social networks, determining empirical quantitative indices, and clustering. In most applications, graph abstractions and algorithms are frequently used to help capture the salient features. Thus, network analysis from a graph theoretic perspective is about extracting interesting information, given a large graph constructed from a real-world dataset.

Network analysis and modeling have received considerable attention in recent times, but algorithms are relatively less studied. Real-world networks are often very large in size, ranging from several hundreds of thousands to billions of vertices and edges. A space-efficient memory representation of such graphs is itself a big challenge, and dedicated algorithms have to be designed exploiting the unique characteristics of these networks. On single processor workstations, it is not possible to do exact in-core computations on large graphs due to the limited physical memory. Current high-end parallel computers have sufficient physical memory to handle large graphs, and a naïve in-core implementation of a graph theory problem is typically two orders of magnitude faster than the best external memory implementation [1]. Algorithm design is further simplified on parallel shared memory systems; due to the globally address memory space, there is no need to partition the graph, and we can avoid the the overhead of message passing. However, attaining good performance is still a challenge, as a large class of graph algorithms are combinatorial in nature, and involve a significant number of non-contiguous, concurrent accesses to global data structures with low degrees of local-

ity.

The **main contributions** of our work are the following: We present *new parallel algorithms* for evaluating network centrality metrics, *optimized for scale-free sparse graphs*. To our knowledge, this is the first work on parallelizing SNA metrics. We have implemented the algorithms on high-end shared memory systems such as the IBM p5 570 and the Cray MTA-2, and report results on several large-scale real datasets, that are three orders of magnitude larger than the graph sizes solved using existing SNA packages. Further, our implementations are designed to handle graph instances on the order of *billions of vertices and edges*.

This paper is organized as follows. Section 2 gives an overview of the various centrality metrics, and the commonly used sequential algorithms to compute them. We present parallel algorithms for evaluating these metrics in Section 3. Section 4 details the performance of these algorithms on parallel shared memory and multithreaded architectures, on a variety of large-scale real-world datasets and synthetic scale-free graphs.

2 Centrality Metrics

One of the fundamental problems in network analysis is to determine the *importance* of a particular vertex or an edge in a network. Quantifying *centrality* and *connectivity* helps us identify portions of the network that may play interesting roles. Researchers have been proposing metrics for centrality for the past 50 years, and there is no single accepted definition. The metric of choice is dependent on the application and the network topology. Almost all metrics are empirical, and can be applied to element-level [10], group-level [21], or network-level [49] analyses. We present a few commonly used indices in this section.

Preliminaries

Consider a graph $G = (V, E)$, where V is the set of vertices representing *actors* or *nodes* in the social network, and E, the set of edges representing the relationships between the actors. The number of vertices and edges are denoted by n and m, respectively. The graphs can be directed or undirected. Let us assume that each edge $e \in E$ has a positive integer weight $w(e)$. For unweighted graphs, we use $w(e) = 1$. A *path* from vertex s to t is defined as a sequence of edges $\langle u_i, u_{i+1} \rangle$, $0 \leq i \leq l$, where $u_0 = s$ and $u_l = t$. The *length* of a path is the sum of the weights of edges. We use $d(s, t)$ to denote the distance between vertices s and t (the minimum length of any path connecting s and t in G). Let us denote the total number of shortest paths between vertices s and t by σ_{st}, and the number passing through vertex v by $\sigma_{st}(v)$.

Degree Centrality

The degree centrality DC of a vertex v is simply the degree $deg(v)$ for undirected graphs. For directed graphs, we can define two variants: in-degree centrality and out-degree centrality. This is a simple local measure, based on the notion of neighborhood. This index is useful in case of static graphs, for situations when we are interested in finding vertices that have the most direct connections to other vertices.

Closeness Centrality

This index measures the closeness, in terms of *distance*, of an actor to all other actors in the network. Vertices with a smaller total distance are considered more important. Several closeness-based metrics [7, 46, 38] have been developed by the SNA community. A commonly used definition is the reciprocal of the total distance from a particular vertex to all other vertices:

$$CC(v) = \frac{1}{\sum_{u \in V} d(v, u)}$$

Unlike degree centrality, this is a global metric. To calculate the closeness centrality of a vertex v, we may apply breadth-first search (BFS, for unweighted graphs) or a single-source shortest paths (SSSP, for weighted graphs) algorithm from v.

Stress Centrality

Stress centrality is a metric based on shortest paths counts, first presented in [48]. It is defined as

$$SC(v) = \sum_{s \neq v \neq t \in V} \sigma_{st}(v)$$

Intuitively, this metric deals with the *work* done by each vertex in a communications network. The number of shortest paths that contain an element v will give an estimate of the amount of stress a vertex v is under, assuming communication will be carried out through shortest paths all the time. This index can be calculated using a variant of the all-pairs shortest-paths algorithm, that calculates and stores all shortest paths between any pair of vertices.

Betweenness Centrality

Betweenness Centrality is another shortest paths enumeration-based metric, introduced by Freeman in [24]. Let $\delta_{st}(v)$ denote the *pairwise dependency*, or the fraction of shortest paths between s and t that pass through v:

$$\delta_{st}(v) = \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Betweenness Centrality of a vertex v is defined as

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

This metric can be thought of as *normalized* stress centrality. Betweenness centrality of a vertex measures the control a vertex has over communication in the network, and can be used to identify key actors in the network. High centrality indices indicate that a vertex can reach other vertices on relatively short paths, or that a vertex lies on a considerable fraction of shortest paths connecting pairs of other vertices. We discuss algorithms to compute this metric in detail in the next section.

This index has been extensively used in recent years for analysis of social as well as other large scale complex networks. Some applications include biological networks [30, 43, 20], study of sexual networks and AIDS [33], identifying key actors in terrorist networks [31, 17], organizational behavior [12], supply chain management [16], and transportation networks [27].

There are a number of commercial and research software packages for SNA (e.g., Pajek [6], InFlow [32], UCINET [2]) which can also be used to determine these centrality metrics. However, they can only be used to study comparatively small networks (in most cases, sparse graphs with less than 40,000 vertices). Our goal is to develop fast, high-performance implementations of these metrics so that we can analyze large-scale real-world graphs of millions to billions of vertices.

2.1 Algorithms for computing Betweenness Centrality

A straightforward way of computing betweenness centrality for each vertex would be as follows:

1. compute the length and number of shortest paths between all pairs (s, t).
2. for each vertex v, calculate every possible pair-dependency $\delta_{st}(v)$ and sum them up.

The complexity is dominated by step 2, which requires $\Theta(n^3)$ time summation and $\Theta(n^2)$ storage of pair-dependencies. Popular SNA tools like UCINET use an adjacency matrix to store and update the pair-dependencies. This yields an $\Theta(n^3)$ algorithm for betweenness by augmenting the Floyd-Warshall algorithm for the all-pairs shortest-paths problem with path counting [8].

Alternately, we can modify Dijkstra's single-source shortest paths algorithm to compute the pair-wise dependencies. Observe that a vertex $v \in V$ is on the shortest path between two vertices $s, t \in V$, *iff* $d(s, t) = d(s, v)+d(v, t)$. Define a set of *predecessors* of a vertex v on shortest paths from s as $pred(s, v)$. Now each time an edge $\langle u, v\rangle$ is scanned for which $d(s, v) = d(s, u) + d(u, v)$, that vertex is added to the predecessor set $pred(s, v)$. Then, the following relation would hold:

$$\sigma_{sv} = \sum_{u \in pred(s,v)} \sigma_{su}$$

Setting the initial condition of $pred(s, v) = s$ for all neighbors v of s, we can proceed to compute the number of shortest paths between s and all other vertices. The computation of $pred(s, v)$ can be easily integrated into Dijkstra's SSSP algorithm for weighted graphs, or BFS Search for unweighted graphs. But even in this case, determining the fraction of shortest paths using v, or the pair-wise dependencies $\delta_{st}(v)$, proves to be the dominant cost. The number of shortest $s-t$ paths using v is given by $\sigma_{st}(v) = \sigma_{sv} \cdot \sigma_{vt}$. Thus computing $BC(v)$ requires $\mathrm{O}(n^2)$ time per vertex v, and $\mathrm{O}(n^3)$ time in all. This algorithm is the most commonly used one for evaluating betweenness centrality.

To exploit the sparse nature of typical real-world graphs, Brandes [8] came up with an algorithm that computes the betweenness centrality score for all vertices in the graph in $\mathrm{O}(mn + n^2 \log n)$ time for weighted graphs, and $\mathrm{O}(mn)$ time for unweighted graphs. The main idea is as follows. We define the *dependency* of a source vertex $s \in V$ on a vertex $v \in V$ as

$$\delta_s(v) = \sum_{t \in V} \delta_{st}(v)$$

The betweenness centrality of a vertex v can be then expressed as $BC(v) = \sum_{s \neq v \in V} \delta_s(v)$. The dependency $\delta_s(v)$ satisfies the following recursive relation:

$$\delta_s(v) = \sum_{w : v \in pred(s,w)} \frac{\sigma_{sv}}{\sigma_{sw}} (1 + \delta_s(w))$$

The algorithm is now stated as follows. First, n SSSP computations are done, one for each $s \in V$. The predecessor sets *pred*(s, v) are maintained during these computations. Next, for every $s \in V$, using the information from the shortest paths tree and predecessor sets along the paths, compute the dependencies $\delta_s(v)$ for all other $v \in V$. To compute the centrality value of a vertex v, we finally compute the sum of all dependency values. The $\mathrm{O}(n^2)$ space requirements can be reduced to $\mathrm{O}(m + n)$ by maintaining a *running centrality score*.

3 Parallel Approaches

In this section, we present our new parallel algorithms and implementation details of the centrality metrics on two classes of shared memory systems: symmetric multiprocessors (SMPs) such as the IBM p5 575, and multithreaded

architectures such as the Cray MTA-2 [19, 14]. The high memory bandwidth and uniform memory access offered by these systems aid the design of high-performance graph theory implementations [3]. We use a cache-friendly adjacency array representation [41] for storing the graph. For algorithm analysis, we use a complexity model similar to the one proposed by Helman and JáJá [28] that has been shown to work well in practice. This model takes into account both computational complexity and memory contention. We measure the overall complexity of an algorithm on SMPs using $T_M(n, m, p)$, the maximum number of non-contiguous accesses made by any processor to memory, and $T_C(n, m, p)$, the upper bound on the local computational complexity. The MTA-2 is a novel architectural design and has no data cache; rather than using a memory hierarchy to hide latency, the MTA-2 processors use hardware multithreading to tolerate the latency. Thus, if there is enough parallelism in the problem, and a sufficient number of threads per processor are kept busy, the system is saturated and the memory access complexity becomes zero. We report only T_C for the MTA-2 algorithms.

Degree Centrality

We store both the in- and out-degree of each vertex in contiguous arrays during construction of the graph from the test datasets. This makes the computation of degree centrality straight-forward, with a constant time look-up on the MTA-2 and the p5 570. As noted earlier, this is a useful metric for determining the graph structure, and for a first pass at identifying important vertices.

Closeness Centrality

Recall the definition of closeness centrality:

$$CC(v) = \frac{1}{\sum_{u \in V} d(v, u)}$$

We need to calculate the distance from v to all other vertices, and then sum over all distances. One possible solution to this problem would be to pre-compute a *distance matrix* using the $\mathrm{O}(n^3)$ Floyd-Warshall All-Pairs Shortest Paths algorithm. Evaluating a specific closeness centrality value then costs $\mathrm{O}(n)$ on a single processor ($\mathrm{O}\left(\frac{n}{p} + \log p\right)$ using p processors) to sum up an entire row of distance values. However, since real-world graphs are typically sparse, we have $m \ll n^2$ and this algorithm would be very inefficient in terms of actual running time and memory utilization. Instead, we can just compute n shortest path trees, one for each vertex $v \in V$, with v as the source vertex for BFS or Dijkstra's algorithm. On p processors, this would yield $T_C = \mathrm{O}\left(\frac{nm+n^2}{p}\right)$ and $T_M = \frac{nm}{p}$ for unweighted graphs. For weighted graphs, using a naïve queue-based representation for the expanded frontier, we can compute all the centrality metrics in $T_C = \mathrm{O}\left(\frac{nm+n^3}{p}\right)$ and $T_M = \frac{2nm}{p}$. The bounds can be further improved with the use of efficient priority queue representations.

Since the evaluation of closeness centrality is computationally intensive, it is valuable to investigate approximate algorithms. Using a random sampling technique, Eppstein and Wang [22] show that the closeness centrality of all vertices in a weighted, undirected graph can be approximated with high probability in $\mathrm{O}\left(\frac{\log n}{\epsilon^2}(n \log n + m)\right)$ time, and an additive error of at most $\epsilon\Delta_G$ (ϵ is a fixed constant, and Δ_G is the diameter of the graph). The algorithm proceeds as follows. Let k be the number of iterations needed to obtain the desired error bound. In iteration i, pick vertex v_i uniformly at random from V and solve the SSSP problem with v_i as the source. The estimated centrality is given by

$$CC_a(v) = \frac{k}{n\Sigma_{i=1}^{k} d(v_i, u)}$$

The error bounds follow from a result by Hoeffding [29] on probability bounds for sums of independent random variables. We parallelize this algorithm as follows. Each processor can run SSSP computations from $\frac{k}{p}$ vertices and store the evaluated distance values. The cost of this step is given by $T_C = \mathrm{O}\left(\frac{k(m+n)}{p}\right)$ and $T_M = \frac{km}{p}$ for unweighted graphs. For real-world graphs, the number of sample vertices k can be set to $\Theta(\frac{\log n}{\epsilon^2})$ to obtain the error bounds given above. The approximate closeness centrality value of each vertex can then be calculated in $\mathrm{O}(k) = \mathrm{O}\left(\frac{\log n}{\epsilon^2}\right)$ time, and the summation for all n vertices would require $T_C = \mathrm{O}\left(\frac{n \log n}{p\epsilon^2}\right)$ and constant T_M.

Stress and Betweenness Centrality

These two metrics require shortest paths enumeration and we design our parallel algorithm based on Brandes' [8] sequential algorithm for sparse graphs. Alg. 1 outlines the general approach for the case of unweighted graphs. On each BFS computation from s, the queue Q stores the current set of vertices to be visited, S contains all the vertices reachable from s, and $P(v)$ is the predecessor set associated with each vertex $v \in V$. The arrays d and σ store the distance from s, and shortest path counts, respectively. The centrality values are computed in steps 22–25, by summing the dependencies $\delta(v)$, $v \in V$. The final scores need to be divided by two if the graph is undirected, as all shortest paths are counted twice.

We observe that parallelism can be exploited at two levels:

Input: $G(V, E)$

Output: Array $BC[1..n]$, where $BC[v]$ gives the centrality metric for vertex v

1 **for** *all* $v \in V$ ***in parallel*** **do**
2 $BC[v] \leftarrow 0$;
for *all* $s \in V$ ***in parallel*** **do**
3 $S \leftarrow$ empty stack;
4 $P[w] \leftarrow$ empty list, $w \in V$;
5 $\sigma[t] \leftarrow 0, t \in V$; $\sigma[s] \leftarrow 1$;
6 $d[t] \leftarrow -1, t \in V$; $d[s] \leftarrow 0$;
7 $Q \rightarrow$ empty queue;
8 enqueue $s \leftarrow Q$;
9 **while** *Q not empty* **do**
10 dequeue $v \leftarrow Q$;
11 push $v \rightarrow S$;
12 **for** *each neighbor* w *of* v ***in parallel*** **do**
13 **if** $d[w] < 0$ **then**
14 enqueue $w \rightarrow Q$;
15 $d[w] \leftarrow d[v] + 1$;
16 **if** $d[w] = d[v] + 1$ **then**
17 $\sigma[w] \leftarrow \sigma[w] + \sigma[v]$;
18 append $v \rightarrow P[w]$;
19 $\delta[v] \leftarrow 0, v \in V$;
20 **while** *S not empty* **do**
21 pop $w \leftarrow S$;
22 **for** $v \in P[w]$ **do**
23 $\delta[v] \leftarrow \delta[v] + \frac{\sigma[v]}{\sigma[w]}(1 + \delta[w])$;
24 **if** $w \neq s$ **then**
25 $BC[w] \leftarrow BC[w] + \delta[w]$;

Algorithm 1: Parallel Betweenness Centrality for unweighted graphs

- The BFS/SSSP computations from each vertex can be done concurrently, provided the centrality running sums are updated atomically.
- The actual BFS/SSSP can be also be parallelized. When visiting the neighbors of a vertex, edge relaxation can be done concurrently.

It is theoretically possible to do all the SSSP computations concurrently (steps 3 to 25 in Alg. 1). However, the memory requirements scale as $\mathrm{O}(p(m+n))$, and we only have a modest number of processors on current SMP systems. For the SMP implementation, we do a coarse-grained partitioning of work and assign each processor a fraction of the vertices from which to initiate SSSP computations. The loop iterations are scheduled dynamically so that work is distributed as evenly as possible. There are no synchronization costs involved, as a processor can compute its own partial sum of the centrality value for each vertex, and all the sums can be merged in the end using a efficient reduction operation. Thus, the stack S, list of predecessors P and the BFS queue Q, are replicated on each processor. Even for a graph of 100 million edges, with a conservative estimate of 10GB memory usage by each processor, we can employ all 16 processors on our target SMP system, the IBM p570. We further optimize our implementation for scale-free graphs. Observe that in Alg. 1, Q is not needed as the BFS frontier vertices are also added to S. To make the implementation cache-friendly, we use dynamic adjacency arrays instead of linked lists for storing elements of the predecessor set S. Note that $|\Sigma_{v \in V} P_s(v)| = \mathrm{O}(m)$, and in most cases the predecessor sets of the few high-degree vertices in the graph are of comparatively greater size. Memory allocation is done as a pre-processing step before the actual BFS computations. Also, the indices computation time and memory requirements can be significantly reduced by decomposing the undirected graph into its biconnected components.

However, for a multithreaded system like the MTA-2, this simple approach will not be efficient. We need to exploit parallelism at a much finer granularity to saturate all the hardware threads. So we parallelize the actual BFS computation, and also have a coarse grained partition at the outer level. We designed a parallel approach for BFS on the Cray MTA-2 in [4], and show that it is possible to attain scalable performance for up to 40 processors on low-diameter, scale-free graphs. The MTA-2 programming environment provides primitives to automatically handle nested parallelism, and the loop iterations are dynamically scheduled.

For the case of weighted graphs, Alg. 1 must be modified to consider edge weights. The relaxation condition changes, and using a Fibonacci heap or pairing heap priority queue representation, it is possible to compute betweenness centrality for all the n vertices in $T_C = \mathrm{O}\left(\frac{mn+n^2 \log n}{p}\right)$. We can easily adapt our SMP implementation to consider weighted graphs also. We intend to work on a parallel multithreaded implementation of SSSP on the MTA-2 for scale-free graphs in the near future.

An approximate algorithm for betweenness centrality is detailed in [9], which is again derived from the Eppstein-Wang algorithm [22]. As in the case of closeness centrality, the sequential running time can be reduced to $\mathrm{O}\left(\frac{\log n}{\epsilon^2}(n+m)\right)$ by setting the value of k appropriately. The parallel algorithms can also be derived similarly, by computing only k dependencies ($\delta[v]$ in Alg. 1) instead of v, and taking a normalized average.

4 Results

This section summarizes the experimental results of our centrality implementations on the Cray MTA-2 and the

Dataset	Source	Network description
ND-actor	[34]	an undirected graph of 392,400 vertices (movie actors) and 31,788,592 edges. An edge corresponds to a link between two actors, if they have acted together in a movie. The dataset includes actor listings from 127,823 movies.
ND-web	[34]	a directed network with 325,729 vertices and 1,497,135 arcs (27,455 loops). Each vertex represents a web page within the Univ. of Notredame *nd.edu* domain, and the arcs represent from $\rightarrow$ to links.
ND-yeast	[34]	undirected network with 2114 vertices and 2277 edges (74 loops). Vertices represent proteins, and the edges interactions between them in the yeast network.
PAJ-patent	[39]	a network of about 3 million U.S. patents granted between January 1963 and December 1999, and 16 million citations made among them between 1975 and 1999
PAJ-cite	[39]	the *Lederberg* citation dataset, produced using HistCite, in PAJEK graph format with 8843 vertices and 41609 edges.

Table 1. Test dataset characteristics

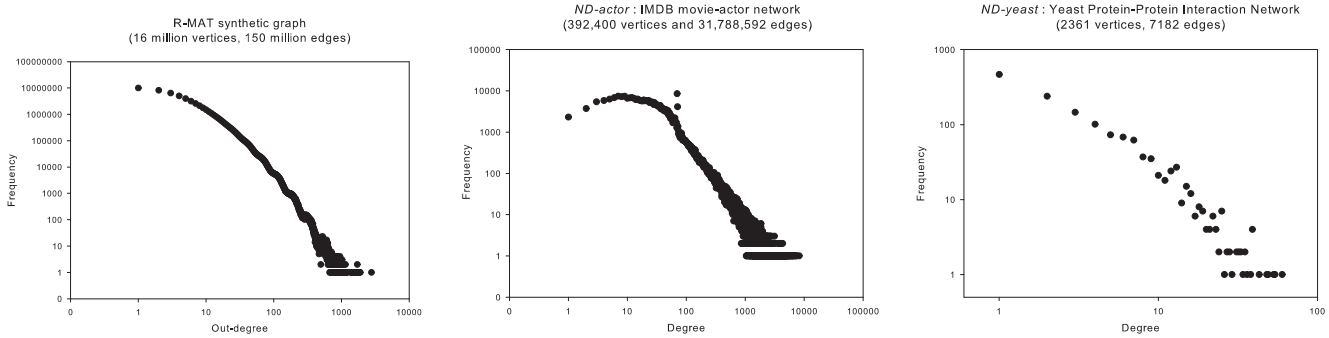

Figure 1. Degree distributions of some network instances

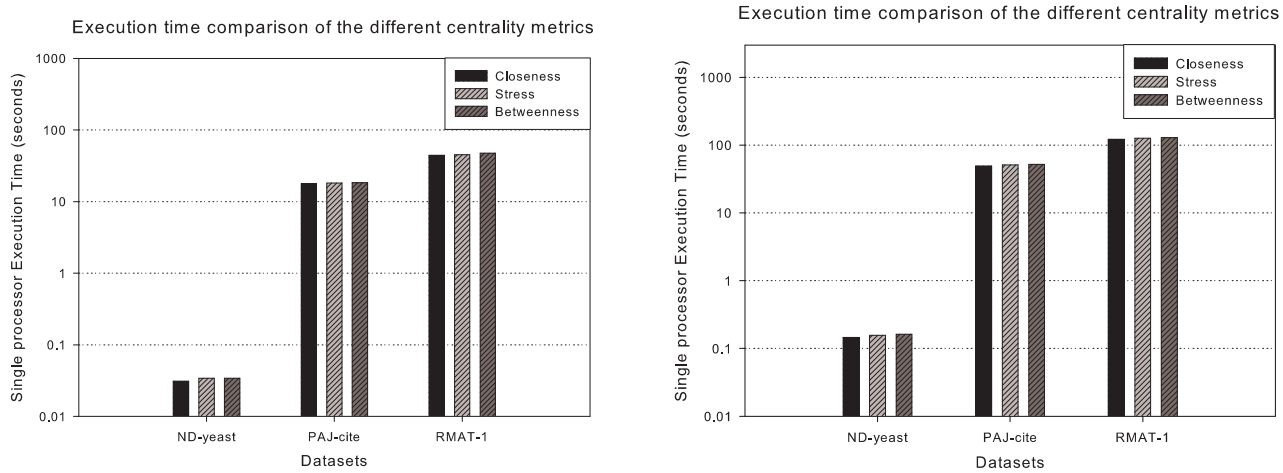

Figure 2. Single processor execution time comparison of the centrality metric implementations on the IBM p5 570 (left) and the Cray MTA-2 (right)

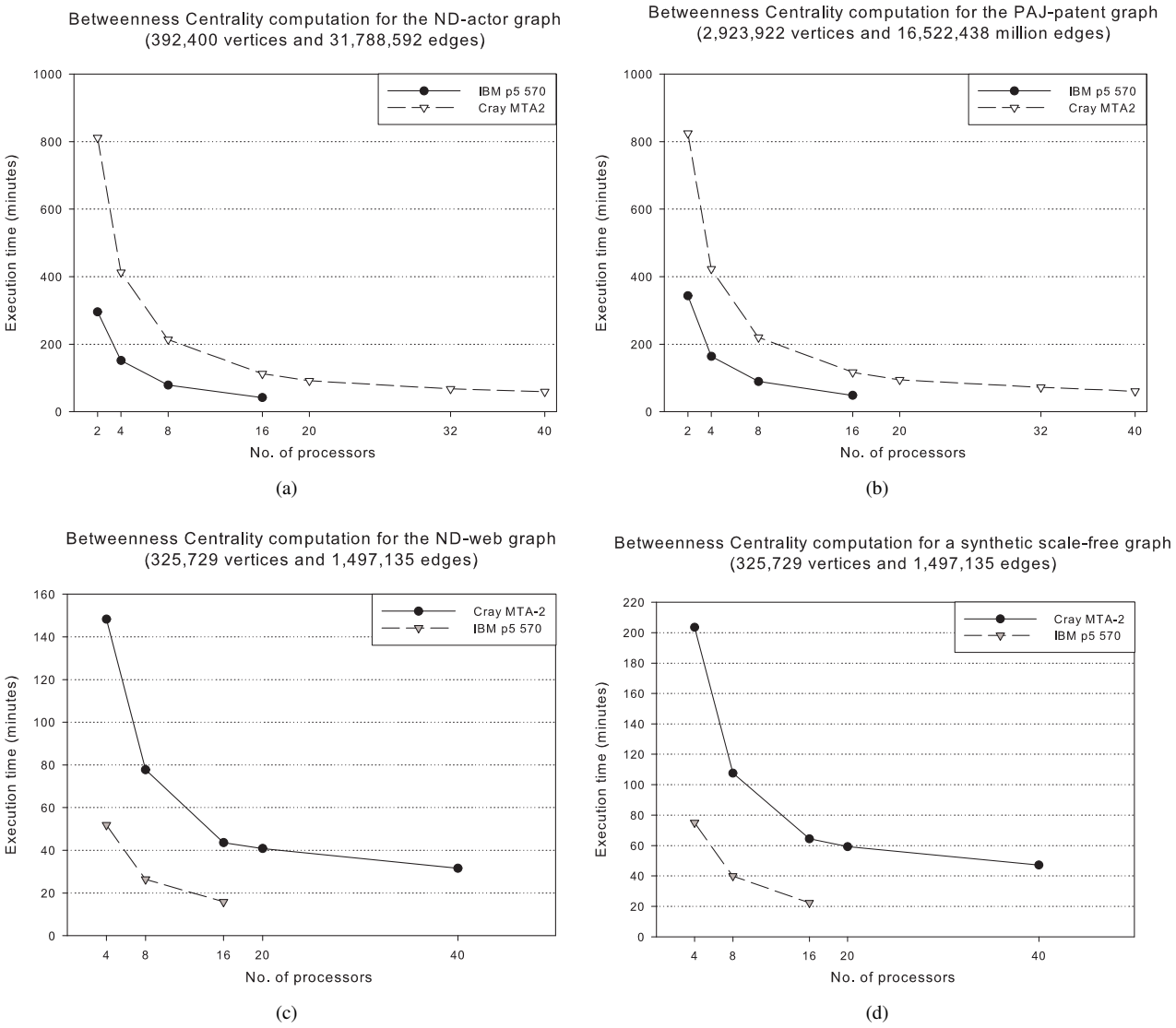

Figure 3. Multiprocessor Performance of Betweenness Centrality for various graph instances on the IBM p5 570 and the Cray MTA-2

IBM p5 570. We report results on a 40-processor MTA-2, with each processor having a clock speed of 220 MHz and 4GB of RAM. Since the address space of the MTA-2 is hashed, from the programmer's viewpoint, the MTA-2 is a flat shared memory machine with 160 GB memory. The IBM p5 570 is a 16-way symmetric multiprocessor with 16 1.9 GHz Power5 cores with simultaneous multithreading (SMT), and 256 GB shared memory.

We test our centrality metric implementations on a variety of real-world graphs, summarized in Table 1. Our implementations have been extended to read input files in both PAJEK and UCINET graph formats. We also use a synthetic graph generator [15] to generate graphs obeying small-world characteristics. The degree distributions of some test graph instances are shown in Fig. 1. We observe that the out-degree distribution contains a heavy tail, which is in agreement with prior experimental studies on scale-free graphs.

Fig. 2 compares the single processor execution time of the three centrality metrics for three graph instances, on the MTA-2 and the p5 570. All three metrics are of the same computational complexity and show nearly similar running times in practice.

Fig. 3 summarizes multiprocessor execution times for computing Betweenness centrality, on the p5 570 and the MTA-2. Fig. 3(a) gives the running times for the ND-actor

graph on the p570 and the MTA-2. As expected, the execution time falls nearly linearly with the number of processors. It is possible to evaluate the centrality metric for the entire ND-actor network in 42 minutes, on 16 processors of the p570. We observe similar performance for the patents citation data. However, note that the execution time is highly dependent on the size of the largest non-trivial connected component in these real graphs. The patents network, for instance, is composed of several disconnected sub-graphs, representing patents and citations in unrelated areas. However, it did not take significantly more time to compute the centrality indices for this graph compared to the ND-actor graph, even though this is a much larger graph instance.

Figs. 3(c) and 3(d) plot the execution time on the MTA-2 and the p5 570 for ND-web, and a synthetic graph instance of the same size generated using the R-MAT algorithm. Again, note that the actual execution time is dependent on the graph structure; for the same problem size, the synthetic graph instance takes much longer than the ND-web graph. Compared to a four processor run, the execution time reduces significantly for forty processors of the MTA-2, but we do not attain performance comparable to our prior graph algorithm implementations [3, 4]. We need to optimize our implementation to attain better system utilization, and the current bottleneck is due to automatic handling of nested parallelism. For better load balancing, we need to further pre-process the graph and semi-automatically assign teams of threads to the independent BFS computations. Our memory management routines for the MTA-2 implementation are not as optimized p570 routines, and this is another reason for drop in performance and scaling.

5 Conclusions

We present parallel algorithms for evaluating several network indices, including betweenness centrality, optimized for Symmetric Multiprocessors and multithreaded architectures. To our knowledge, this is the first effort at parallelizing these widely-used social network analysis tools. Our implementations are designed to handle problem sizes in the order of billions of edges in the network, and this is three orders of magnitude larger than instances that can be handled by current social network analysis tools. We are currently working on improving the betweenness centrality implementation on the MTA-2, and also extend it to efficiently compute scores for weighted graphs. In future, we plan to implement and analyze performance of approximate algorithms for closeness and betweenness centrality detailed the paper, and also apply betweenness centrality values to solve harder problems like graph clustering.

Acknowledgements

This work was supported in part by NSF Grants CAREER CCF-0611589, ACI-00-93039, NSF DBI-0420513, ITR ACI-00-81404, ITR EIA-01-21377, Biocomplexity DEB-01-20709, ITR EF/BIO 03-31654, and DARPA Contract NBCH30390004. We are grateful to Jonathan Berry and Bruce Hendrickson for discussions on large-scale network analysis and betweenness centrality.

References

[1] D. Ajwani, R. Dementiev, and U. Meyer. A computational study of external-memory bfs algorithms. In *Proc. 17th Ann. Symp. Discrete Algorithms (SODA-06)*, pages 601–610. ACM-SIAM, 2006.

[2] Analytic Technologies. UCINET 6 social network analysis software. `http://www.analytictech.com/ucinet.htm`.

[3] D. Bader, G. Cong, and J. Feo. On the architectural requirements for efficient execution of graph algorithms. In *Proc. 34th Int'l Conf. on Parallel Processing (ICPP)*, Oslo, Norway, June 2005.

[4] D. Bader and K. Madduri. Designing multithreaded algorithms for Breadth-First Search and st-connectivity on the Cray MTA-2. Technical report, Georgia Institute of Technology, Feb. 2006.

[5] A.-L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, October 1999.

[6] V. Batagelj and A. Mrvar. Pajek – Program for large network analysis. *Connections*, 21(2):47–57, 1998.

[7] A. Bavelas. Communication patterns in task oriented groups. *J. Acoustical Soc. of America*, 22:271–282, 1950.

[8] U. Brandes. A faster algorithm for betweenness centrality. *J. Mathematical Sociology*, 25(2):163–177, 2001.

[9] U. Brandes and T. Erlebach, editors. *Network Analysis: Methodological Foundations*, volume 3418 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005.

[10] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[11] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. *Comp. Netw.*, 33(1-6):309–320, 2000.

[12] N. Buckley and M. van Alstyne. Does email make white collar workers more productive. Technical report, University of Michigan, 2004.

[13] D. Callaway, M. Newman, S. Strogatz, and D. Watts. Network robustness and fragility: percolation on random graphs. *Physics Review Letters*, 85:5468–5471, 2000.

[14] L. Carter, J. Feo, and A. Snavely. Performance and programming experience on the Tera MTA. In *Proc. SIAM Conf. on Parallel Processing for Scientific Computing*, San Antonio, TX, Mar. 1999.

[15] D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-MAT: A recursive model for graph mining. In *Proc. 4th SIAM Intl. Conf. on Data Mining*, Florida, USA, Apr. 2004.

[16] D. Cisic, B. Kesic, and L. Jakomin. Research of the power in the supply chain. International Trade, Economics Working Paper Archive EconWPA, April 2000.

[17] T. Coffman, S. Greenblatt, and S. Marcus. Graph-based technologies for intelligence analysis. *Communications of the ACM*, 47(3):45–47, 2004.

[18] R. Cohen, K. Erez, D. Ben-Avraham, and S. Havlin. Breakdown of the internet under intentional attack. *Physics Review Letters*, 86:3682–3685, 2001.

[19] Cray Inc. The MTA-2 multithreaded architecture. `http://www.cray.com/products/systems/mta/`.

[20] A. del Sol, H. Fujihashi, and P. O'Meara. Topology of small-world networks of protein-protein complex structures. *Bioinformatics*, 21(8):1311–1315, 2005.

[21] P. Doreian and L. Albert. Partitioning political actor networks: Some quantitative tools for analyzing qualitative networks. *Quantitative Anthropology*, 161:279–291, 1989.

[22] D. Eppstein and J. Wang. Fast approximation of centrality. In *Proc. 12th Ann. Symp. Discrete Algorithms (SODA-01)*, Washington, DC, 2001.

[23] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the Internet topology. In *Proc. ACM SIGCOMM*, pages 251–262, 1999.

[24] L. C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, 1977.

[25] L. C. Freeman. *The development of social network analysis: a study in the sociology of science*. Booksurge Pub., 2004.

[26] J.-L. Guillaume and M. Latapy. Bipartite graphs as models of complex networks. In *Proc. 1st Int'l Workshop on Combinatorial and Algorithmic Aspects of Networking*, 2004.

[27] R. Guimera, S. Mossa, A. Turtschi, and L. Amaral. The worldwide air transportation network: Anomalous centrality, community structure, and cities' global roles. *Proc. Nat. Academy of Sciences USA*, 102(22):7794–7799, 2005.

[28] D. R. Helman and J. JáJá. Prefix computations on symmetric multiprocessors. *Journal of Parallel and Distributed Computing*, 61(2):265–278, 2001.

[29] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of American Statistical Association*, 58:713–721, 1963.

[30] H. Jeong, S. Mason, A.-L. Barabasi, and Z. Oltvai. Lethality and centrality in protein networks. *Nature*, 411:41, 2001.

[31] V. Krebs. Mapping networks of terrorist cells. *Connections*, 24(3):43–52, 2002.

[32] V. Krebs. InFlow 3.1 - Social network mapping software, 2005. `http://www.orgnet.com`.

[33] F. Liljeros, C. R. Edling, L. A. N. Amaral, H. E. Stanley, and Y. Aberg. The web of human sexual contacts. *Nature*, 411:907, 2001.

[34] Notredame CNet resources. `http://www.nd.edu/~networks`.

[35] M. Newman. Scientific collaboration networks: shortest paths, weighted networks and centrality. *Physics Review E*, 64, 2001.

[36] M. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003.

[37] M. Newman, S. Strogatz, and D. Watts. Random graph models of social networks. *Proceedings of the National Academy of Sciences USA*, 99:2566–2572, 2002.

[38] U. J. Nieminen. On the centrality in a directed graph. *Social Science Research*, 2:371–378, 1973.

[39] PAJEK datasets. `http://www.vlado.fmf.uni-lj.si/pub/networks/data/`.

[40] C. Palmer and J. Steffan. Generating network topologies that obey power laws. In *Proc. ACM GLOBECOM*, Nov. 2000.

[41] J. Park, M. Penner, and V. Prasanna. Optimizing graph algorithms for improved cache performance. In *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS 2002)*, Fort Lauderdale, FL, Apr. 2002.

[42] R. Pastor-Satorras and A. Vespignani. Epidemic spreading in scale-free networks. *Physics Review Letters*, 86:3200–3203, 2001.

[43] J. Pinney, G. McConkey, and D. Westhead. Decomposition of biological networks using betweenness centrality. In *Proc. Poster Session of the 9th Ann. Int'l Conf. on Research in Computational Molecular Biology (RECOMB 2004)*, Cambridge, MA, May 2005.

[44] W. Richards. International network for social network analysis, 2005. `http://www.insna.org`.

[45] W. Richards. Social network analysis software links, 2005. `http://www.insna.org/INSNA/soft_inf.html`.

[46] G. Sabidussi. The centrality index of a graph. *Psychometrika*, 31:581–603, 1966.

[47] J. Scott. *Social Network Analysis: A Handbook*. SAGE Publications, 2000.

[48] A. Shimbel. Structural parameters of communication networks. *Mathematical Biophysics*, 15:501–507, 1953.

[49] University of Virginia. Oracle of Bacon. `http://www.oracleofbacon.org`.

[50] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge Univ. Press, 1994.

[51] D. Watts and S. Strogatz. Collective dynamics of small world networks. *Nature*, 393:440–442, 1998.

[52] D. Zanette. Critical behavior of propagation on small-world networks. *Physics Review E*, 64, 2001.