SNAP, Small-world Network Analysis and Partitioning: an open-source parallel graph framework for the exploration of large-scale networks

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

Abstract

We present SNAP (Small-world Network Analysis and Partitioning), an open-source graph framework for exploratory study and partitioning of large-scale networks. To illustrate the capability of SNAP, we discuss the design, implementation, and performance of three novel parallel community detection algorithms that optimize modularity, a popular measure for clustering quality in social network analysis. In order to achieve scalable parallel performance, we exploit typical network characteristics of small-world networks, such as the low graph diameter, sparse connectivity, and skewed degree distribution. We conduct an extensive experimental study on real-world graph instances and demonstrate that our parallel schemes, coupled with aggressive algorithm engineering for small-world networks, give significant running time improvements over existing modularity-based clustering heuristics, with little or no loss in clustering quality. For instance, our divisive clustering approach based on approximate edge betweenness centrality is more than two orders of magnitude *faster than a competing greedy approach, for a variety of large graph instances on the Sun Fire T2000 multicore system. SNAP also contains parallel implementations of fundamental graph-theoretic kernels and topological analysis metrics (e.g., breadth-first search, connected components, vertex and edge centrality) that are optimized for small-world networks. The SNAP framework is extensible; the graph kernels are modular, portable across shared memory multicore and symmetric multiprocessor systems, and simplify the design of high-level domain-specific applications.*

1 Introduction

Data-intensive applications have emerged as a prominent computational workload in the petascale computing era. Massive data sets with millions, or even billions, of entities are frequently processed in financial, scientific, security, and several other application areas. Further, the data are dynamically generated in many cases, and may be assimilated from multiple sources. Thus, the modeling and analysis of massive, transient data streams raises new and challenging research problems. There are several analytical methods for the analysis of interaction data. Algorithms in the data stream and related models [34] have been shown to be effective for statistical analysis, and for mining trends in large-scale data sets. Alternately, a graph or a network representation is a convenient and intuitive abstraction for analyzing data. Unique entities are represented as vertices, and the interactions between them are depicted as edges. The vertices and edges can further be typed, classified, or assigned attributes based on relational information. Analyzing topological characteristics of the network, such as the vertex degree distribution, centrality and community structure, provides valuable insight into the structure and function of the interacting data entities. Common queries on these massive data sets can also be naturally encoded as variants of problems related to graph connectivity, flow, or partitioning.

The modeling and analysis of complex interaction data is an active research topic in the social science and statistical physics communities. Real-world systems such as the Internet, socio-economic interactions, and biological networks have been extensively studied from an empirical perspective [3, 35], and this has led to the development of a variety of models to understand their topological properties and evolution. In particular, it has been shown that technological networks, social interaction graphs, and graph abstractions in biology, exhibit common structural features such as a low graph diameter, skewed vertex degree distribution, self-similarity, and dense subgraphs. Analogous to the small-world (short paths) phenomenon, these real-world data sets are broadly referred to and modeled as *small-world* networks [40, 4]. Practical algorithms for applications such as identification of influential entities, communities, and

978-1-4244-1694-3/08/$25.00 ©2008 IEEE

anomalous patterns in social networks (in general, small-world networks) are well-studied [21, 35].

However, in order to effectively utilize a network abstraction for solving massive data stream problems, we need to be able to compactly represent and process large-scale graphs, and also efficiently support fundamental analysis queries on them. On current workstations, it is infeasible to do exact in-core computations on large-scale graphs (by *large-scale*, we refer to graphs where the number of vertices and edges are in the range of 100 million to 10 billion) due to the limited physical memory. In such cases, parallel computing techniques can be applied to obtain exact solutions for memory and compute-intensive graph problems quickly. For instance, recent experimental studies on Breadth-First Search for large-scale sparse graphs show that a parallel in-core implementation [8] is two orders of magnitude faster than an optimized external memory implementation [2]. Parallel graph algorithms is a well-studied research area, and there is extensive literature on work-efficient PRAM algorithms for several classical graph problems [23]. SNAP, the parallel network analysis framework we present in this paper, is a collection of holistic schemes that couple high-performance computing approaches with classical graph algorithms, social network analysis (SNA) techniques, and optimizations for small-world networks. The source code for SNAP is freely available from our website. We discuss exploratory graph analysis using SNAP in Section 3.

A key problem in social network analysis is that of finding communities, dense components, or detecting other latent structure. This is usually formulated as a graph clustering problem, and several indices have been proposed for measuring the quality of clustering (see [25, 14] for a review). Existing approaches based on the Kernighan-Lin algorithm [28], spectral algorithms [25], flow-based algorithms, and hierarchical clustering work well for specific classes of networks (e.g., abstractions from scientific computing, physical topologies), but perform poorly for small-world networks. We will discuss the related problem of partitioning small-world networks in more detail in Section 2.2. Newman and Girvan recently proposed a divisive algorithm based on edge betweenness [37] that has been applied successfully to a variety of real networks. However, it is compute-intensive and takes $O(n^3)$ time for sparse graphs (n denotes the number of vertices). This algorithm optimizes for a novel clustering measure called modularity, which has become very popular for social network analysis. We apply SNAP to the problem of small-world network clustering in this paper and present novel parallel algorithms that optimize modularity. We design three clustering schemes (two hierarchical agglomerative approaches, and one divisive clustering algorithm) that exploit typical topological characteristics of small-world networks. We also conduct an extensive experimental study and demonstrate that our parallel schemes give significant running time improvements over existing modularity-based clustering heuristics. For instance, our novel divisive clustering approach based on approximate edge betweenness centrality is *more than two orders of magnitude* faster than the Newman-Girvan algorithm on the Sun Fire T2000 multi-core system, while maintaining comparable clustering quality. The algorithms are discussed in detail in Section 4.

2 Preliminaries

The interaction data set is expressed using a graph abstraction $G(V, E)$, where V is the set of vertices representing unique interacting entities, and E is the set of edges representing the interactions. The number of vertices and edges are denoted by n and m respectively. The graph can be directed or undirected, depending on the input data. We will 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 $\langle u_i, u_{i+1} \rangle$, $0 \leq i < l$, 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 $\sigma_{st}(v)$.

2.1 Centrality Metrics

One of the fundamental problems in network analysis is to determine the importance or *criticality* of a particular vertex or an edge in a network. While there has been extensive work on quantifying *centrality* and *connectivity* in a network, there is no single accepted definition. The measure of choice is dependent on the application and the network topology. We briefly define some of the centrality metrics we implement in SNAP, and refer the reader to [13, 9] for a detailed discussion on centrality analysis.

The degree of a vertex, or the *degree centrality*, 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* is a global index that measures the closeness, in terms of distance $\left(CC(v) = \frac{1}{\sum_{u \in V} d(v, u)} \right)$, of a vertex to all other vertices in the network. Vertices with a smaller total distance are considered more important. *Betweenness centrality* is a shortest paths enumeration-based global metric, introduced by Freeman in [20]. Let $\delta_{st}(v)$ denote the *pairwise dependency*, or the fraction of shortest paths between s and

t that pass through v: $\dfrac{\sigma_{st}(v)}{\sigma_{st}}$. Then, betweenness centrality of a vertex v is defined as $BC(v) = \sum_{s\neq v\neq t\in V} \delta_{st}(v)$. Betweenness centrality of a vertex measures the control a vertex has over communication in the network, and can be used to identify critical vertices 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. Key applications of centrality analysis include assessing lethality in biological networks [24, 38], study of sexual networks and AIDS [31], identifying key actors in terrorist networks [16], and supply chain management processes.

2.2 Graph Partitioning

Graph partitioning and community detection are related problems, but with an important difference: the most commonly used objective function in partitioning is minimization of edge cut, while trying to *balance the number of vertices* in each partition. The number of partitions is typically an input parameter for a partitioning algorithm. Clustering, on the other hand, optimizes an appropriate application-dependent measure, and the number of clusters needs to be computed. Multi-level algorithms and spectral heuristics have been shown to be very effective for partitioning graph abstractions derived from physical topologies, such as finite-element meshes arising in scientific computing. Software packages implementing these algorithms (e.g., Chaco [22] and Metis [27, 26]) are freely available, computationally efficient, and produce high-quality partitions in most cases. A natural question that arises is whether these partitioning algorithms, or simple variants, can be applied to small-world networks as well.

Table 1 summarizes results from an experiment to test the quality of existing partitioning packages on small-world networks. We consider graph instances from three different families (a road network, a sparse random graph, and a synthetic small-world network), but of the same size: roughly 200,000 vertices and 1 million edges. We report the edge cut for a balanced 32-way partitioning of each of these graphs, using four partitioning techniques (the default multilevel partitioning approaches from Metis, pmetis and kmetis, and two spectral heuristics from Chaco). Observe that the edge cut for the random and power-law graphs is nearly two orders of magnitude higher than the cut for the nearly-Euclidean road network. Clearly, existing partitioning tools fail to partition small-world networks since these networks lack the topological regularity found in scientific meshes and physical networks, where the degree distribution is relatively constant and most connectivity is localized. Also, random and small-world networks have a lower diameter (O($\log n$), or in some cases O(1)) than physical networks (e.g., O($\sqrt{n}$) for Euclidean topologies). Lang [29, 30] provides further empirical evidence that *cut quality varies inversely with cut balance* for social graphs such as the Yahoo! IM network and the DBLP collaboration data set. Further, he shows that the spectral method tends to break off small parts of the graphs. This finding is corroborated by a recent theoretical result from Mihail and Papadimitriou [33]. They prove that for a random graph with a skewed degree distribution, the largest eigenvalues are in correspondence with high-degree vertices, and the corresponding eigenvectors are the characteristic vectors of their neighborhoods. Spectral analysis in this case ignores structural features of the graph in favor of high-degree vertices.

Recent research efforts have focused on adapting multilevel and spectral partitioning techniques to small-world graphs. Abou-Rjeili and Karypis [1] present new coarsening heuristics for multilevel approaches that outperform (give a lower edge cut) Metis and Chaco. As it is difficult to theoretically analyze general small-world networks, researchers have been looking at applying spectral analysis to synthetic graph models. For instance, Dasgupta et al. [18] provide a normalization of the Laplacian that improves the performance of the spectral approach on a planted-partition random graph model. Clustering heuristics based on the above graph partitioning algorithms optimize for conductance, a measure that compares the cut size to cut balance. However, based on empirical and theoretical evidence that it is difficult to obtain balanced partitions in small-world networks, we focus on optimizing *modularity* [37], a popular heuristic from the complex network analysis community.

2.3 Modularity as a clustering measure

Intuitively, modularity is a measure that is based on optimizing *intra-cluster density over inter-cluster sparsity* [14]. Let $C = (C_1, ..., C_k)$ denote a partition of V such that $C_i \neq \phi$ and $C_i \cap C_j = \phi$. We call C a clustering of G and each C_i is defined to be a *cluster*. The cluster $G(C_i)$ is identified with the induced subgraph $G[C_i] := (C_i, E(C_i))$, where $E(C_i) := \{\langle u, v\rangle \in E : u, v \in C_i\}$. Then, $E(C) := \cup_{i=1}^{k} E(C_i)$ is the set of intra-cluster edges and $\tilde{E}(C) := E - E(C)$ is the set of inter-cluster edges. Let $m(C_i)$ denote the number of inter-cluster edges in C_i. Then, the modularity measure $q(C)$ of a clustering C is defined as

$$q(C) = \sum_i \left[\frac{m(C_i)}{m} - \left(\frac{\sum_{v\in C_i} deg(v)}{2m} \right)^2 \right]$$

To maximize the first term, the number of intra-cluster edges should be high, whereas the second term is minimized by splitting the graph into multiple clusters with small total degrees. If a particular clustering gives no more intra-community edges than would be expected by random

Graph Instance	*Metis-kway*	*Metis-recur*	*Chaco-RQI*	*Chaco-LAN*
Physical (road)	1,856	1,703	2,937	3,913
Sparse random	685,211	706,625	717,960	737,747
Small-world	805,903	736,560	–	–

Table 1. Performance results (edge cut) for a 32-way partitioning of three different graph instances, using standard partitioning algorithms from the Chaco and Metis packages. Chaco-RQI and Chaco-LAN fail to complete for the small-world network instance.

chance, we will get $Q = 0$. Values greater than 0 indicate deviation from randomness, and empirical results show that values greater than 0.3 indicate significant community structure. Modularity has found widespread acceptance in the network analysis community, and there have been an array of heuristics, based on spectral analysis, simulated annealing, greedy agglomeration, and extremal optimization [36] proposed to optimize it. Brandes *et al.* [12] recently showed that maximizing modularity is strongly $\mathcal{NP}$-complete, and this has led to renewed interest in designing better algorithms for modularity maximization. We present three new modularity-maximization heuristics in Section 4 and compare them with the current state-of-the-art approaches discussed in [36].

3 The SNAP Infrastructure for Exploratory Network Analysis

SNAP is a modular graph infrastructure for analyzing and partitioning interaction graphs, targeting multicore and manycore platforms. SNAP is implemented in C and uses POSIX threads and OpenMP primitives for parallelization. The source code is freely available online from our web site.

In addition to partitioning and analysis support for interaction graphs, SNAP provides an optimized collection of algorithmic "building blocks" (efficient implementations of key graph-theoretic kernels) to end-users. In prior work, we have designed novel parallel algorithms for several graph problems that run efficiently on shared memory systems. Our implementations of breadth-first graph traversal [8], shortest paths [32, 17], spanning tree [5], MST, connected components [6], and other problems achieve impressive parallel speedup for arbitrary, sparse graph instances. We redesign and integrate several of our recent parallel graph algorithms into SNAP, with additional optimizations for small-world networks. SNAP provides a simple and intuitive interface for network analysis application design, effectively hiding the parallel programming complexity involved in the low-level kernel design from the user. In this section, we highlight some of the algorithmic and data structure choices involved in the design of the SNAP framework, and discuss analysis techniques that are currently supported.

Data Representation

Efficient data structures and graph representations are key to high performance parallel graph algorithms. In order to process massive graphs, it is particularly important that the data structures are space-efficient. The primary graph representation supported in SNAP is a vertex adjacency list representation, implemented using cache-friendly adjacency arrays. This representation is simple and the preferred choice for static graph algorithms. However, for algorithms that require dynamic structural updates to the graph, we need to efficiently support insertion and deletion of edges. We use an auxiliary graph representation that uses dynamic, resizable adjacency arrays. To speed up deletions, adjacencies can be ordered by sorting them by their vertex or edge identifier. Further, several optimizations are possible for small-world graphs. Small-world networks typically have an unbalanced degree distribution – the majority of the vertices are low-degree ones, and there are a few vertices of very high degree. In such cases, we could have a threshold on the degree and represent low-degree vertex adjacencies in a simple, unsorted adjacency representation, but adjacencies of high-degree vertices in a tree structure such as treaps [39]. Treaps are randomized search trees that support fast insertion, deletion, searching, joining and splitting. In addition, there are efficient parallel algorithms for set operations on treaps such as union, intersection and difference. Based on the graph update rate, and the insertion to deletion ratio for an application, we could choose an appropriate graph representation.

Graph Kernels

The SNAP graph kernels are primarily designed to exploit fine-grained thread level parallelism in graph traversal. We apply one of the following two paradigms in the design of parallel kernels: *level-synchronous* graph traversal, where vertices at each level are visited in parallel; or *path-limited searches*, in which multiple searches are concurrently executed and aggregated. The level-synchronous approach is particularly suitable for small-world networks due to their low graph diameter. Support for fine-grained efficient synchronization is critical in both these approaches. We try to aggressively reduce locking and barrier con-

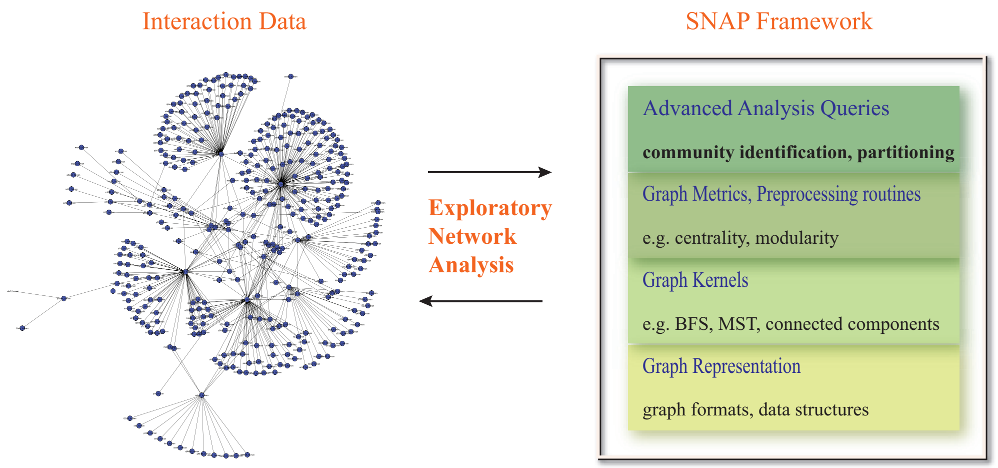

Interaction Data
SNAP Framework
Exploratory Network Analysis
Advanced Analysis Queries
community identification, partitioning
Graph Metrics, Preprocessing routines
e.g. centrality, modularity
Graph Kernels
e.g. BFS, MST, connected components
Graph Representation
graph formats, data structures

Figure 1. The SNAP Framework Overview

structs through algorithmic changes, as well as implementation optimizations. For the BFS kernel, we use a lock-free level-synchronous algorithm that significantly reduces shared memory contention. The minimum spanning tree algorithm uses a lazy synchronization scheme coupled with work-stealing graph traversal to yield a greater granularity of parallelism. While designing fine-grained algorithms for small-world networks, we also consider the unbalanced degree distributions. In a level-synchronized parallel BFS where vertices are statically assigned to processors without considering their degree, it is highly probable that there will be phases with severe work imbalance. To avoid this, we first estimate the processing work to be done from each vertex, and then assign them accordingly to processors. We visit adjacencies of high degree vertices in parallel for better load balancing. With these optimizations, we demonstrate that the performance of our fine-grained BFS and shortest path algorithms [8, 32] is mostly independent of the graph degree distribution.

We utilize these efficient kernel implementations as building blocks for higher level algorithms such as centrality and partitioning. For these algorithms, we also consider performance trade-offs associated with memory utilization and parallelization granularity. In cases where the input graph instance is small enough, we can trade off space with a coarse-grained parallelization strategy, thus reducing synchronization overhead. We utilize this technique in the compute-intensive ($O(mn)$ work) exact betweenness centrality calculation, where the centrality score computation requires n graph traversals. The fine-grained implementation parallelizing each graph traversal requires $O(m + n)$ space, whereas the memory requirements of the coarse-grained approach, in which the n traversals are distributed among p processors, are $O(p(m + n))$. We also incorporate small-world network specific optimizations in the choice of data structures for centrality computations. For instance, the predecessor sets of a vertex in shortest path computations, required in centrality computations, are implemented differently for low-degree and high-degree vertices. The parallel algorithms, coupled with small-world network optimizations, enable SNAP to analyze networks that are three orders of magnitude larger than the ones that can be processed using commercial and research software packages for SNA (e.g., Pajek [11], InFlow, UCINET).

Network Analysis Metrics and Preprocessing Routines for Small-world Networks

The crux of exploratory graph analysis is a systematic computational study of the structure and dynamics of a network, using a discriminating selection of topological metrics. SNAP supports fast computation of simple as well as novel SNA metrics, such as average vertex degree, clustering coefficient, average shortest path length, rich-club coefficient, and assortativity. Most of these metrics have a linear, or sub-linear computational complexity and are straightforward to implement. When used appropriately, they not only provide insight into the network structure, but also help speed up subsequent analysis algorithms. For instance, the average neighbor connectivity metric is a weighted average that gives the average neighbor degree of a degree-k vertex. It is an indicator of whether vertices of a given degree preferentially connect to high- or low-degree vertices. Assortativity coefficient is a related metric proposed by Newman, which is an indicator of community structure in a network. Based on the these metrics, it is easy to identify instances of specific graph classes, such as bipartite graphs, and networks with pronounced community structure. This helps us choose an appropriate community detection algorithm and a clustering measure for which to optimize. Other preprocessing kernels include computation of connected and biconnected components of the graph. If a graph is composed of several large connected components, it can be decomposed and individual components can be analyzed concurrently. In case of protein interaction networks in computational biology, we find that vertices that are articulation points (determined from computing biconnected components), but have a low degree, are unlikely to be essential to the network [10]. All these preprocessing steps combined together potentially offer an order of magnitude speedup or more [9] for key analysis kernels on real-world network instances.

4 Parallel Community Identification Algorithms

The parallel algorithms we present for community identification are based on modularity maximization. Intuitively, modularity captures the idea that a good division of a network into communities is one in which there are *fewer than expected* edges between communities, and not one that just minimizes edge cut. Since the general problem of modularity optimization is $\mathcal{NP}$-complete [14], we explore greedy strategies that maximize modularity. Existing algorithms fall into two broad classes, divisive or agglomerative, based on how the division is done. In the agglomerative method, each vertex initially belongs to a singleton community, and two communities whose amalgamation produces an increase in the modularity score are merged together. The agglomeration can be represented by a tree, referred to as a dendrogram, whose internal nodes correspond to joins. In the following discussion, we present three novel *parallel* algorithms, one divisive and two agglomerative approaches, that are built on top of optimized SNAP analysis kernels.

Approximate betweenness-based divisive algorithm (pBD)

Our first approach is a divisive algorithm in which we initially treat the entire network as one community, and iteratively determine *critical* links in the network that can be cut. By doing this repeatedly, we divide the network into smaller and smaller components, and can also keep track of the clustering quality at each step by computing the modularity score. Algorithm 1 gives the high-level pseudocode for this iterative approach, and our parallelization strategy. We explain each step in more detail below.

There are several possible approaches to select the *critical link* on each iteration. Newman and Girvan [36] suggest picking edges based on their betweenness score, and show that this approach results in significantly higher modularity scores compared to other known greedy heuristics.

Algorithm 1: Approximate betweenness-based divisive algorithm (pBD)

```
Input: $G(V, E)$, length function $l : E \rightarrow \mathbb{R}$
Output: A partition $C = (C_1, ..., C_k)$ ($C_i \neq \phi$ and
        $C_i \cap C_j = \phi$) of $V$ that maximizes modularity; A dendrogram $D$
        representing the clustering steps.
1 Optional step: Run biconnected components, identify
  articulation points and bridges.
2 $numIter \longleftarrow 0$;
3 while $numIter < m$ do
4     Find edge $e_m$ with the highest approximate
      betweeness centrality score in parallel.
5     Mark edge $e_m$ as deleted in the graph $G$.
6     Run connected components on $G$, update
      dendrogram and number of clusters in parallel.
7     Compute modularity of the current partitioning
      in parallel.
8     $numInter \longleftarrow numIter + 1$;
9 Inspect the dendrogram, set $C$ to the clustering with
  the highest modularity score.
```

The problem with this approach is that it is computationally expensive — we need to recompute edge betweenness centrality scores at each step of the algorithm, and there can be O(m) iterations in the worst case. Although it might be tempting to compute betweenness scores only once and then remove edges in that order, Newman and Girvan show that this results in inferior clustering quality for several small-world networks.

We rely on extensive algorithm engineering and parallelization to speed up the Newman-Girvan edge betweenness technique, while trying to maintain the quality of clustering. First, observe that on each iteration, we only need to identify the edge with the highest centrality score. We recently proposed a novel betweenness computation algorithm based on adaptive sampling [7] for estimating the centrality score of a *specific vertex or edge* in a general network. It is adaptive in that the number of samples (graph traversals) varies with the information obtained from each sample; further, we show high-probability bounds on the estimated error. In practice, after extensive experimentation on real-world networks, we show that on an average, we can estimate betweenness scores of high-centrality (the top 1%) entities with less than 20% error, by sampling just 5% of the vertices. We replace the exact centrality computation algorithm with the approximate betweenness approach, and only recompute aproximate betweenness scores of the known high-centrality edges (step 4 of Algorithm 1).

A second effective optimization is to vary the granularity of parallelization as the clustering algorithm proceeds.

In the initial iterations of the algorithm, before the graph is split up into connected components of smaller sizes, we parallelize computation of approximate betweenness centrality. Once the graph is decomposed into a large number of isolated components, we can switch to computing exact centrality. We can then exploit parallelism at a coarser granularity, by computing centrality scores of each component in parallel. This switch in the parallelism granularity is semi-automatic (controller by a user parameter) in our SNAP implementation. In addition, we parallelize the $O(m)$-work kernels such as modularity computation (step 7 of Algorithm 1) and dendrogram updates (step 6 of Algorithm 1). Note that varying the parallelization granularity does not affect the quality of clustering (the modularity score) in any manner.

From empirical evidence, we observe that bridges in the network (determined by computing biconnected components) are likely to have high edge centrality scores. We apply this heuristic as an optional step (step 1 of Algorithm 1) to determine a set of potential high-centrality edges in the graph, and to accelerate approximate betweenness computation.

Modularity-maximizing agglomerative clustering algorithm (pMA)

Algorithm 2: Modularity-maximizing agglomerative clustering algorithm (pMA)

```
Input: $G(V, E)$, length function $l : E \rightarrow \mathbb{R}$
Output: A partition $C = (C_1, ..., C_k)$ ($C_i \neq \phi$ and
        $C_i \cap C_j = \phi$) of $V$ that maximizes modularity.

1  $nC \longleftarrow n$;
2  Max heap $H \longleftarrow \phi$;
3  foreach $v \in V$ do
4      $\Delta Qd[v]$ (dynamic array) $\longleftarrow \phi$;
5      $\Delta Qb[v]$ (multilevel bucket) $\longleftarrow \phi$;
6      Add modularity update value corresponding to
       each neighbor (adjacent community) of $v$ to both
       $\Delta Qb[v]$ and $\Delta Qd[v]$.
7      Add community-pair with the maximum
       modularity update value to $H$.
8  while $nC > 1$ do
9      Select the community pair $(i, j)$ corresponding
       to the largest value in $H$.
10     Update $\Delta Qd$, $\Delta Qb$, $H$ in parallel, and
       increment modularity score.
11     $nC \longleftarrow nC - 1$;
12 Inspect $Q$, set $C$ to the clustering with the highest
   modularity score.
```

A greedy agglomerative approach starts from a state of n singleton communities, and iteratively merges the pair of communities that result in the greatest increase in modularity. Clauset *et al.* [15] give an algorithm that runs in $O(md \log n)$ time, where d is the depth of the resulting dendrogram. The primary data structure is an implicitly-maintained sparse matrix ΔQ, where $\Delta Q(i, j)$ corresponds to a increase in modularity on merging clusters C_i and C_j. We design a new parallel algorithm (pMA, see Algorithm 2) that performs the same greedy optimization as Clauset *et al.*'s approach, but uses data representations supported in SNAP for the modularity update matrix. We store each row of the matrix as a sorted dynamic array (so that elements can be identified or inserted in $O(\log n)$ time), as well as a multi-level bucket (to identify the largest element quickly). We parallelize two steps in every iteration of the greedy approach – the matrix rows representing the two communities C_i and C_j are merged in parallel; secondly, if C_i and C_j are connected to other communities, the corresponding ΔQ updates can be parallelized. This algorithm is significantly faster than the divisive clustering approach, with the trade-off of loss in clustering quality for some graph instances.

Greedy local aggregation algorithm (pLA)

Algorithm 3: Greedy local aggregation algorithm (pLA)

```
Input: $G(V, E)$, length function $l : E \rightarrow \mathbb{R}$
Output: A partition $C = (C_1, ..., C_k)$ ($C_i \neq \phi$ and
        $C_i \cap C_j = \phi$) of $V$ that maximizes modularity.

1  Run biconnected components to identify bridges.
2  Delete bridges, run connected components.
3  foreach connected component $C$ in $G$ do
4      $n_C \longleftarrow$ number of vertices in the component;
5      while $n_C > 1$ do
6          Select a vertex $v$ at random in parallel.
7          Merge vertices/clusters adjacent to $v$ and
           create a new cluster, based on an appropriate
           local clustering metric (e.g., degree,
           clustering coefficient).
8          Accept the new cluster if the overall
           modularity score increases.
9          Update the value of $n_C$, the number of
           remaining vertices in the graph.
```

Note that the above approaches rely on global metrics for community identification, and parallelism can only be exploited at a very fine granularity (at the level of an iteration). We consider relaxing this further and design an agglomerative partitioning heuristic in which multiple execution threads concurrently try to identify communities.

The algorithm proceeds as follows. We first compute biconnected components to determine if the graph has any bridges. If it does, we remove bridge edges and run the connected components kernel. If this splits the graph into multiple isolated components, we run a greedy agglomerative clustering heuristic on each of these components, and finally amalgamate the clusters at the top level. Note that we still optimize for modularity. However, while doing agglomerative clustering, to avoid global synchronization after each iteration, we use a local measure such as degree or clustering coefficient to decide whether an edge needs to be added to a cluster. To initiate clustering, we need to pick a set of seed vertices – this can be done randomly, or obtained from a breadth-first ordering of the vertices. Vertices are greedily added to the clusters, and we exploit parallelism using the *path-limited search* paradigm discussed in the previous section. In practice, this heuristic performs well for networks with a pronounced community structure, and does not rely on any global centrality metrics.

5 Experiments and Performance Evaluation

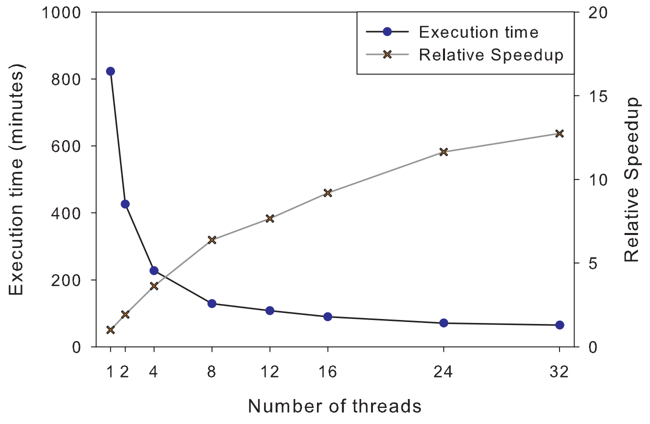
We evaluate the performance of the community identification heuristics on twelve different real-world network instances. In Table 2, we compare values of modularity obtained using our new approaches against the Girvan-Newman (GN) algorithm. We show results for six different networks, all of which have been used in previous studies (please see [36, 19] for sources). We also report the best-known modularity score (higher scores indicate better community structure) for each network, obtained by either an exhaustive search, extremal optimization [19], or a simulated annealing-based technique. It should be noted that the approaches used to obtain the best-known modularity scores are computationally very expensive, and may only be applied to small networks. Table 2 shows that our divisive betweenness-based approach pBD performs extremely well in practice, and the modularity scores are comparable to GN. In fact, for the larger *E-mail* and *Key signing* networks, pBD outperforms GN. pMA and pLA, the faster agglomerative algorithms also perform favorably, with pLA giving a better partitioning for the *Karate* and *Key signing* networks.

The real benefit of our algorithms lies in the fact that they are significantly faster than existing approaches, and facilitate analysis of networks that were considered too large to be tractable. We now report execution time and parallel speedup on a multicore parallel system for several real-world graph instances. Table 3 lists a collection of small-world networks gathered from diverse application domains: a protein-interaction network from computational biology, a citation network, a web crawl, and two social networks. We ignore edge directivity in the community detection algorithms. Our test platform for reporting parallel performance results in this paper is the Sun Fire T2000 server, with the Sun UltraSPARC T1 (Niagara) processor. This system has eight cores running at 1.0 GHz, each of which is four-way multithreaded. The cores share a 3 MB L2 cache, and the system has a main memory of 16 GB. We compile SNAP with the Sun C compiler v5.8 and the default optimization flags.

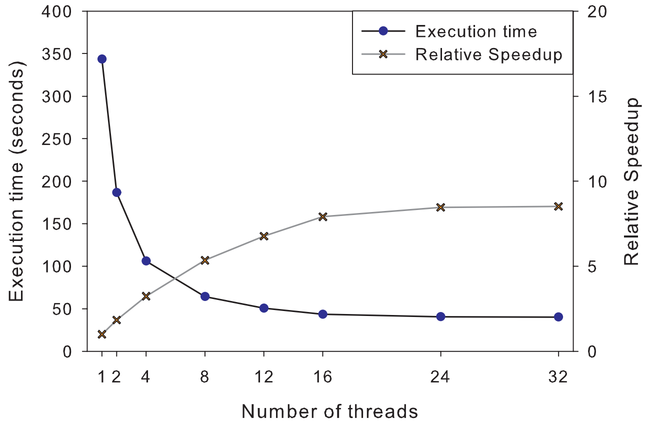
Figure 2 gives the execution time and relative speedup on the Sun Fire T2000, for community identification using our three parallel algorithms. The graph instance analyzed is *RMAT-SF*, a synthetic small-world network of 0.4 million vertices and 1.6 million edges. The computationally-expensive divisive approximate betweenness approach is the slowest among the three (note that the execution time in Figure 3(a) is in the order of minutes), while pMA and pLA are comparable in execution time. On 32 threads, we achieve a parallel speedup of roughly 13, 9 and 12 for pBD, pMA, and pLA respectively. These performance results are along expected lines and follow the speedup trends displayed by the SNAP inner kernels such as graph traversal and approximate betweenness centrality [8].

In Figure 3(a), we compare the performance of pBD to the GN approach for the real-world networks listed in Table 3. pBD is faster than GN because of algorithmic differences (we compute approximate betweenness and incorporate additional small-world network optimizations to speed up the partitioning), and also due to the fact that that it is a parallel approach. Since these speedup factors are multiplicative, the overall performance improvement achieved is quite significant. For instance, for the web-crawl *NDwww*, the single-threaded run of pBD is nearly 26 times faster than an optimized implementation of GN using SNAP. This improvement, coupled with a parallel speedup of 13.2 on the Sun Fire T2000, results in an overall speedup of 343. The performance is consistently high across all the real-world networks. Note that the exact *algorithm engineering* speedup achieved depends on the topology of the network: it is comparatively lower for *PPI* as the network is small. For pMA and pLA, since we do not have a baseline heuristic to compare performance against, we just report the relative speedup on 32 threads for the different graph instances. pLA achieves a slightly higher speedup in most cases, while the running times are comparable.

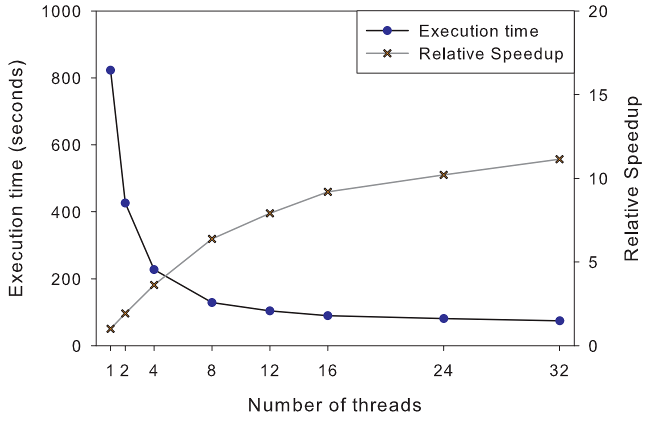
Note that all three parallel algorithms require only O$(m + n)$-space, independent of the number of processors. While we report performance results for graphs with several millions of vertices and edges in this paper, the algorithms are scalable and can process graphs with even billions of entities.



(a) pBD



(b) pMA



(c) pLA

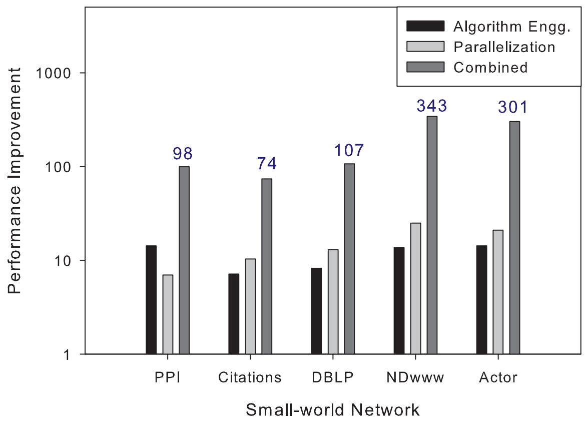
Figure 2. Parallel performance (execution time and relative speedup on the Sun Fire T2000) of our three community detection algorithms, when applied to the `RMAT-SF` graph instance.

Network	n	Modularity Q				
		GN	pBD	pMA	pLA	Best known
`Karate`	34	0.401	0.397	0.381	0.397	0.431 [12]
`Political books`	105	0.509	0.502	0.498	0.487	0.527 [12]
`Jazz musicians`	198	0.405	0.405	0.439	0.398	0.445 [19]
`Metabolic`	453	0.403	0.402	0.402	0.402	0.435 [36]
`E-mail`	1,133	0.532	0.547	0.494	0.487	0.574 [19]
`Key signing`	10,680	0.816	0.846	0.733	0.794	0.855 [36]

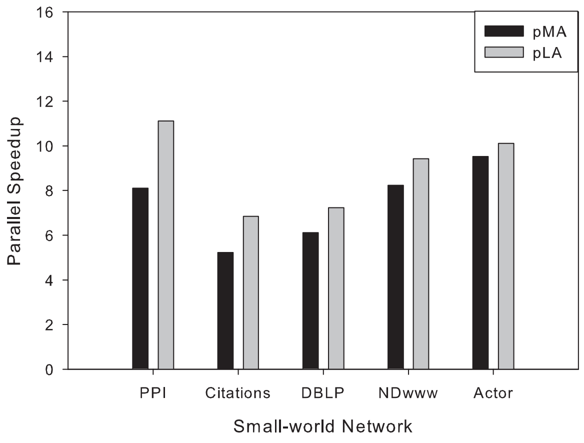
Table 2. A comparison of modularity scores achieved using the algorithms presented in this paper (pBD, pMA, pLA). GN corresponds to the Girvan-Newman edge-betweenness based algorithm. The best known modularity scores are determined either by an exhaustive search, or using non-greedy heuristics.

Label	Network	n	m	Type
`PPI`	human protein interaction network	8,503	32,191	undirected
`Citations`	Citation network from KDD Cup 2003	27,400	352,504	directed
`DBLP`	CS publication coauthorship network	310,138	1,024,262	undirected
`NDwww`	web-crawl (`nd.edu`)	325,729	1,090,107	directed
`Actor`	IMDB movie-actor network	392,400	31,788,592	undirected
`RMAT-SF`	synthetic small-world network	400,000	1,600,000	undirected

Table 3. Small-world networks used in the experimental study



(a) pBD speedup relative to GN



(b) pMA and pLA parallel speedup

Figure 3. Speedup achieved by pBD over the GN algorithm due to algorithm engineering and parallelization (left), and parallel speedup achieved by pMA and pLA approaches (right) for several real-world graph instances. The bar labels indicates the ratio of execution time of GN to the running time of pBD.

6 Conclusions and Future Work

This paper introduces SNAP, a parallel framework for large-scale network analysis. SNAP includes efficient parallel implementations of novel community structure identification algorithms, classical graph-theoretic kernels, topological indices that provide insight into the network structure, and preprocessing kernels for small-world graphs. To illustrate the capability of the SNAP framework, we detail the design, analysis and implemnentation of three novel parallel community identification algorithms. Further, we demonstrate that SNAP parallel approaches are two orders of magnitude faster than competing algorithms – this enables analysis of networks that were previously considered too large to be tractable. As part of ongoing work, we are designing new small-world network analysis kernels and incorporating existing techniques into SNAP. Our current focus is on support for spectral analysis of small-world networks, and efficient parallel implementations of spectral algorithms that optimize modularity. We intend to extend SNAP to support the topological analysis of dynamic networks.

Acknowledgments

This work was supported in part by NSF Grants CAREER CCF-0611589, NSF DBI-0420513, ITR EF/BIO 03-31654, IBM Faculty Fellowship and Microsoft Research grants, NASA grant NP-2005-07-375-HQ, and DARPA Contract NBCH30390004. We thank Nicolas Bitouze (ENS Cachan) for implementing the parallel greedy agglomeration algorithm, and Jon Berry and Bruce Hendrickson (Sandia National Laboratories) for discussions on large-scale graph problems.

References

[1] A. Abou-Rjeili and G. Karypis. Multilevel algorithms for partitioning power-law graphs. In *Proc. Int'l Parallel and Distributed Proc. Symp. (IPDPS 2006)*, Rhodes, Greece, Apr. 2006.

[2] D. Ajwani, U. Meyer, and V. Osipov. Improved external memory BFS implementations. In *Proc. The 9th Workshop on Algorithm Engineering and Experiments (ALENEX)*, New Orleans, LA, Jan. 2007.

[3] R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47–97, 2002.

[4] L. Amaral, A. Scala, M. Barthélémy, and H. Stanley. Classes of small-world networks. *Proc. National Academy of Sciences*, 97(21):11149–11152, 2000.

[5] D. Bader and G. Cong. A fast, parallel spanning tree algorithm for symmetric multiprocessors (SMPs). *Journal of Parallel and Distributed Computing*, 65(9):994–1006, 2005.

[6] 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. IEEE Computer Society.

[7] D. Bader, S. Kintali, K. Madduri, and M. Mihail. Approximating betweenness centrality. In *Proc. 5th Workshop on Algorithms and Models for the Web-Graph (WAW2007)*, Lecture Notes in Computer Science, San Diego, CA, December 2007. Springer-Verlag.

[8] D. Bader and K. Madduri. Designing multithreaded algorithms for breadth-first search and st-connectivity on the Cray MTA-2. In *Proc. 35th Int'l Conf. on Parallel Processing (ICPP)*, Columbus, OH, Aug. 2006. IEEE Computer Society.

[9] D. Bader and K. Madduri. Parallel algorithms for evaluating centrality indices in real-world networks. In *Proc. 35th Int'l Conf. on Parallel Processing (ICPP)*, Columbus, OH, Aug. 2006. IEEE Computer Society.

[10] D. Bader and K. Madduri. A graph-theoretic analysis of the human protein interaction network using multicore parallel algorithms. In *Proc. 6th Workshop on High Performance Computational Biology (HiCOMB 2007)*, Long Beach, CA, March 2007.

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

[12] U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Höfer, Z. Nikoloski, and D. Wagner. On finding graph clusterings with maximum modularity. In *Proc. 33rd Intl. Workshop on Graph-Theoretic Concepts in CS (WG 2007)*, Dornburg, Germany, June 2007.

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

[14] U. Brandes, M. Gaertler, and D. Wagner. Engineering graph clustering: Models and experimental evaluation. *J. Exp. Algorithmics*, 12:1.1, 2007.

[15] A. Clauset, M. Newman, and C. Moore. Finding community structure in very large networks. *Physical Review E*, 70:066111, 2004.

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

[17] J. Crobak, J. Berry, K. Madduri, and D. Bader. Advanced shortest path algorithms on a massively-multithreaded architecture. In *Proc. Workshop on Multithreaded Architectures and Applications*, Long Beach, CA, March 2007.

[18] A. Dasgupta, J. Hopcroft, and F. McSherry. Spectral analysis of random graphs with skewed degree distributions. In *Proc. 45th Ann. IEEE Symp. on Found. of Comp. Sci. (FOCS'04)*, pages 602–610, Washington, DC, USA, 2004. IEEE Computer Society.

[19] J. Duch and A. Arenas. Community detection in complex networks using extremal optimization. *Physical Review E*, 72:027104, 2005.

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

[21] M. Girvan and M. Newman. Community structure in social and biological networks. *Proc. National Academy of Sciences*, 99(12):7821–7826, 2002.

[22] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. In *Proc. Supercomputing '95*, San Diego, CA, Dec. 1995.

[23] J. JáJá. *An Introduction to Parallel Algorithms*. Addison-Wesley Publishing Company, New York, 1992.

[24] H. Jeong, S. Mason, A.-L. Barabási, and Z. Oltvai. Lethality and centrality in protein networks. *Nature*, 411:41–42, 2001.

[25] R. Kannan, S. Vempala, and A. Vetta. On clusterings: Good, bad and spectral. *J. ACM*, 51(3):497–515, 2004.

[26] G. Karypis and V. Kumar. *MeTiS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices*. Dept. of Comp. Sci., Univ. of Minnesota, version 4.0 edition, Sept. 1998.

[27] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998.

[28] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, 1970.

[29] K. Lang. Finding good nearly balanced cuts in power law graphs. Technical report, Yahoo! Research, 2004.

[30] K. Lang. Fixing two weaknesses of the spectral method. In *Proc. Advances in Neurals Information Proc. Systems 18 (NIPS)*, Vancouver, Canada, December 2005.

[31] F. Liljeros, C. Edling, L. Amaral, H. Stanley, and Y. Åberg. The web of human sexual contacts. *Nature*, 411:907–908, 2001.

[32] K. Madduri, D. Bader, J. Berry, and J. Crobak. An experimental study of a parallel shortest path algorithm for solving large-scale graph instances. In *Proc. The 9th Workshop on Algorithm Engineering and Experiments (ALENEX)*, New Orleans, LA, Jan. 2007.

[33] M. Mihail and C. Papadimitriou. On the eigenvalue power law. In J. Rolim and S. Vadhan, editors, *Proc. 6th Intl. Workshop on Randomization and Approximation Techniques (RANDOM)*. Springer-Verlag, September 2002.

[34] S. Muthukrishnan. Data streams: algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2):117–236, August 2005.

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

[36] M. Newman. Modularity and community structure in networks. *Proc. National Academy of Sciences*, 103(23):8577–8582, 2006.

[37] M. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69:026113, 2004.

[38] J. Pinney, G. McConkey, and D. Westhead. Decomposition of biological networks using betweenness centrality. In *Proc. 9th Ann. Int'l Conf. on Research in Comp. Mol. Bio. (RECOMB 2005)*, Cambridge, MA, May 2005. Poster session.

[39] R. Seidel and C. Aragon. Randomized search trees. *Algorithmica*, 16:464–497, 1996.

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