Building Blocks for Graph Based Network Analysis

Vladimir Ufimtsev
Computer Science Department
University of Nebraska at Omaha
Email: vufimtsev@unomaha.edu

Sanjukta Bhowmick
Computer Science Department
University of Nebraska at Omaha
Email: sbhowmick@unomaha.edu

Sivasankaran Rajamanickam
Scalable Algorithms Department
Sandia National Laboratories.
Email: srajama@sandia.gov

***Abstract*—Network analysis using graph abstractions is a powerful tool for studying complex systems. While there are multiple libraries for both graph operations in general and network analysis algorithms in particular, there is no components based standardization of both of these key set of operations. We propose a framework that abstracts the data stuctures, architecture, programming models for the graph algorithms underneath a very simple component based interface. We also build on these graph abstractions to provide a layer of abstraction that are key for network analysis. A reference implementation of the abstractions and its performance is also demonstrated using a new library – ESSENS.**

I. Introduction

Over the last decade, network analysis has emerged as a powerful tool for studying complex systems of interacting entities such as those arising in bioinformatics [1], social sciences [2], software engineering [3] and many other diverse disciplines. Such complex systems are typically modeled using networks (or graphs), where the entities are represented as vertices and their pair-wise interactions as edges. Graph algorithms are used to analyze the structure of these networks, and these structural properties provide insights to the features of the underlying system. For example, high degree vertices often correspond to lethal proteins in a protein-protein interaction network [4] and groups of tightly connected vertices indicate people with similar interests in a social network [5].

The prevalent use of network analysis has extended the use of graph algorithms to a wide range of disciplines beyond scientific computing. Network analysis by itself is a young discipline where new metrics and algorithms are being developed almost on a daily basis. However, there is yet no standard framework for development of network algorithms. The lack of a such a standard framework hampers the efficiency and productivity of network research. Often the algorithms are designed from scratch and the programming language and the data structure use varies widely. This makes the comparison between different network analysis algorithms a difficult process, because it is difficult to ascertain whether the efficiency was due to the algorithm design or implementation of underlying graph algorithms/data structures. In addition, the size of the graphs and the needs of the analysts in network analysis vary widely which result in different approaches to analyzing the graphs, from high peformance computing to map-reduce based distributed computing. The choice also affects the graph algorithm libraries an analyst could use and the hardware available.

A first solution to this is to develop a standard and basic set of building blocks, that would facilitate the development, analysis and comparison of different graph-based network analysis algorithms.

In contrast to the broad field of network analysis, graph algorithms have been developed for decades. The various formats to store graphs (compressed sparse graphs, adjacency matrices etc) have been known and the advantages and disadvantages of using certain data structures with certain algorithms, for example an all-pairs shortest path with adjancey matrices, are well studied. Multiple frameworks ([6], [7], [8], [9]) exist with different requirements for data structures and interfaces optimized for their use-cases. While there is a lot of commonalities between these frameworks, until recently standardization of the frameworks have not been considered [10]. Moreover, due to the varying implementation strategies in the different frameworks, it is not easy to translate or compare the algorithms written in one framework with the algorithms written in another. In most cases, it is also not very easy for the users to modify the available algorithms unless they are extremely familiar with the framework. We posit that a higher level abstract framework, based on the common features, can differentiate between the algorithmic and implementation-based innovations, as well as provide a flexible, easy-to-use platform for algorithm development, comparison and experimentation.

The building blocks for this framework should be abstract, such that they can support multiple underlying implementations that use any techniques (matrix computations or map-reduce etc), but the users of the building block need not be exposed to it. In this paper, we propose such a abstract set of high-level building blocks for that can serve multiple purposes, for graph algorithm developers, network science researchers and vendors as follows;

- *For developers of new algorithms and metrics:* The higher levels of the framework will provide a set of building blocks that will help developers conceptualize and build their algorithms without the need to worry about lower level implementation details, such as what data structure or what parallel paradigm should be used.
- *For developers focusing on performance:* The higher level building block will be able to support and switch between multiple types of implementation, so the effect of different data structures and other implementation

978-1-4799-6233-4/14/$31.00 ©2014 IEEE

strategies can be compared.

- *For vendors or system architects:* The framework can help highlight the set of building blocks most frequently used by the community, and vendors can focus on optimizing those for improved performance.
- *For network analysts/ students:* Researchers who are not primarily trained as computation scientists and students who are new to the field can leverage the higher level blocks to understand the analysis algorithms and make changes according to their problem requirements.

The remainder of this paper is organized as follows; In Section II we provide background on some of the recent work on network analysis software. In Section III we describe the key building blocks. In Section IV, we give a brief outline of an initial software implementation using these blocks and provide some runtime results. We conclude in Section V with a discussion of the key points and future potential of this work.

II. Background

We posit that one can write graph algorithms for both traditional and "big-data" applications on top of a basic set of primitives or building blocks, irrespective of whether the underlying implementation uses linear algebra primitives and matrix operations (like combinatorial BLAS [9]) or map reduce or thread libraries. In order to achieve this the building blocks themselves should be agnostic on the underlying architecture (XMT, Xeon Phi, GPU), underlying model for graph computations (linear algebra primitives, Map-reduce etc) and programming languages (templated C++, Python, R, Fortran).

Over the recent years many network analysis packages have been developed. These include general purpose packages packages such as Cytoscape [11], JUNG [12], Gephi [13] and igraph [14], as well as those focused on specific applications such as EPiSimS [15] (epidemics), Organizational Risk Analyzer [16] (geospatial analysis) and Network WorkBench [17] (scientific collaborations).

As the target networks are extremely large, network analysis software has also been implemented in the parallel domain. The distributed memory based Parallel Boost Graph Library [18], although more a graph algorithm package, includes most of the relevant network analysis methods. Knowledge Discovery Toolkit [19], also distributed memory based, implements the algorithms as linear algebra functions. SNAP [20] and NetworKit [21] focus on shared memory based implementations. Pregel [22] proposes a vertex-based approach to analyzing large graphs. GraphLab [23] is based on the MapReduce framework.

There are fewer projects for dynamic network analysis. GraphStreams [24] is a sequential Java based package that allows quick updates of to networks and can dynamically update the graph connectivity. GraphCT [25], designed for massively multithreaded architectures, contains functions for dynamically updating the network structure, connectivity and clustering coefficients.

All these packages provide efficient network analysis algorithms and are being widely used by their respective communities. However, despite the common functions, the underlying framework and implementation differs from one package to another. This makes it difficult to share, modify or compare algorithms across different competing packages.

III. The Key Building Blocks:

To design the key building blocks we look at the fundamental definition of graphs. Graphs are defined by two sets *(i)* a set of elements (vertices with their properties) and *(ii)* a set of relations between these elements (edges and their properties). Graph algorithms, in general, concern finding other relations based on this initial set. Building on this set-based view, we propose that the following blocks would suffice for all graph algorithms.

- *Graphs:* Set of vertices (with or without properties); Set of edges (with or without properties).
- *Set operations:* Set/Sequence operations on lists of vertices or edges. The set of operations are; Intersection, Difference , Union (or Merge), Subset (identify subsets that follow certain property; equivalent to filtering), Sort (including Priority Queue operations) and Find
- *Traversals:* Traversals are equivalent to finding transitive chains. We start from a set of elements (generally the set is a singleton or the vertex marked as 0 and based on certain relations, continue to find transitive chains until a stopping condition (such as based on number of elements visited, or the length of the chain). Different traversals are distinguished only by which relationship (here we term it *priority*) drives the chain formation.
- *Output:* A set of elements and properties; A set of relations (edges); A scalar value.

Given below is a generalized form of the traversal function

Algorithm 1 General Traversal Function

Input Graph $G(V, E)$; Properties of vertices VP; **Priority** Set of Boolean Conditions PF **Set of Visited Elements** W. Initialized to empty set. **Output** Set of Edges H

```
Initialize Values of VP
Select Set of Start Node(s); S
while Stopping Condition not fulfilled do
    for all v ∈ S, u ∉ W and (v, u) ∈ E do
        if (v, u) fulfills conditions in PF then
            H = H ∪ (v, u)
            Value of VP[u] updated
        end if
    end for
    W = W ∪ S
    Select vertices that can be part of S based on VP.
end while
```

By using such template-based approach we are no longer confined to the traversal functions provided in a software, but by changing the values and criteria associated with the priority,

PF and vertex properties, VP, we can design different traversal based algorithms, including the commonly known ones in graph theory. In Breadth First Search (BFS) the priority is to find a chain of the shortest length. Depth First Search the priority is the longest chain. This argument can be extended to weighted graphs and extracting non-tree subgraphs as well. We consider an edge weight to be the property of a relation. Minimum Spanning Tree is finding transitive chains such that the priority is given to the relation with the least weight. For extracting chordal graphs using the Dearing algorithm [26], the priority is to maintain the chordal properties. Given below is an example of how the Dearing algorithm can be written within the template of the traversal function given in Algorithm 1.

Algorithm 2 Extracting Maximal Chordal Graphs

Input Graph $G(V, E)$; Properties of vertices VP; **Priority** The added edges should maintain chordal graph properties **Set of Visited Elements** W. Initialized to empty set. **Output** Set of Edges H

for all $v \in V$ **do**
  Set $VP(v) = \emptyset$.
end for
Select Set of Start Nodes; Here $S = \{v_0\}$
while All nodes not visited; $|W| \neq |V|$ **do**
  for all $u \notin W$ and $(v_0, u) \in E$ **do**
    if $VP(u) \subseteq VP(v)$ **then** (Criteria for maintaining chordal subgraph)
      $H = H \cup (v, u)$
      Value of $VP(u)$ updated; $VP(u) = VP(u) \cup v_0$
    end if
  end for
  $W = W \cup S$
  Select next vertex v_0; such that $|VP(v_0)| \geq |VP(v)|$ for all $v \notin W$
end while

We can build on these basic functions to design more complicated analysis methods, where graph traversal is part of the algorithm. We show how an elegant strongly connected component algorithm, originally described by Pinar et al. [27], can be implemented using a generalized traversal function and simple set operations. The strongly connected component algorithm is itself given in Algorithm 3. It uses two traversals on each direction of a directed graph and a few set operations. In practice, Algorithm 3 as listed here has implementations in MPI [28], shared memory [29] and map-reduce frameworks [30]. One could implement this in matrix-based primitives as well.

IV. ESSENS: A Reference Implementation

We have implemented some of the network algorithms using these building blocks into a software package ESSENS (Extensible, Scalable Software for Evolving NetworkS). We classified the analysis algorithms into three levels based on

Algorithm 3 Strongly Connected Component Algorithm

1: **procedure** SCC(V)
2: **if** $V = \varnothing$ **then**
3: **return** $\varnothing$
4: **end if**
5: Select a *pivot* $u \in V$
6: $D \leftarrow$ BFS_Traversal($G(V, E(V)), u$)
7: $P \leftarrow$ BFS_Traversal($G(V, E'(V)), u$)
8: $R \leftarrow (V \setminus (P \cup D)$
9: $S \leftarrow (P \cap D)$
10: $S \leftarrow S \cup$ SCC($D \setminus S$)
11: $S \leftarrow S \cup$ SCC($P \setminus S$)
12: $S \leftarrow S \cup$ SCC(R)
13: return S
14: **end procedure**

area of operations. Since most combinatorial methods can be implemented as a combination (consecutive or nested) of these levels, this classification will help us to design and analyze our methods over a generalized framework. The three levels are as follows;

- *Level 1 Vertex Based Computations*. These computations involve only a vertex and its distance-k neighbors, where k is small. These operations are generally the least expensive ones in the analysis process. Examples include computing degree and clustering coefficient.
- *Level 2 Subgraph Based Computations* . These computations involve a specified set of vertices. One example is combining certain groups of vertices in a supernode.
- *Level 3 Graph Based Computations*. These computations involve traversing the entire network. Examples include verifying connectivity, finding articulation points, computing betweenness centrality.

ESSENS, is implemented in STL/C++, is currently sequential and the functions are designed for undirected networks. The framework has a bottom-up design (shown in Figure 1), the lower rows containing functions that are used in the higher rows. We envision that high performing implementations will choose to implement Level 3 operations by themselves without using the underlying Level 1 and Level 2 blocks for performance reasons.

The top level contains abstractions for network-based algorithms, including (i) a Network Transform component transforms the original network, such as in sampling or reordering, (ii) a Computing Metrics component that computes the network and vertex properties and (iii) a Rank and Compare component that ranks these properties and compares networks. The analysis methods can extend across multiple groups. For example, functions for the minimum weight spanning tree requires both Network Transform (spanning tree) and Rank and Compare (edge weights).

The second level contains the graph abstractions that are required by top level algorithms including (i) Traversal of Networks, (ii) Subgraph Operations for combining or dividing two networks or to selecting specified parts of the network and (iii) Matrix Operations, such as those required for spectral methods. The third level includes vertex-level functions for

adding, deleting and selecting vertices and edges and auxiliary algorithms for sorting (selecting edge with minimum weight) and set operations (finding common neighbors). These levels are built on top of an implementation framework, currently called Network Bundle. Users can implement networks as per their favorite data structure (currently the networks are defined as Adjacency List and CSR formats).

Although not currently in place, ESSENS will also provide options to link with other external packages on database storage and parallel programming. ESSENS will also be designed to be extensible and include more capabilities like visualization after the analysis, adding new algorithms in the graph layers and supporting other programming models and the data structures.

There are two important features of ESSENS that contribute to making it truly extensible and flexible and also distinguish it from many other network analysis/ graph algorithm software. The first feature is the separation between the data structure and the algorithm implementation. Because of the modularity between levels, so long as the operations associated with that data structure is defined in Level 1, the algorithms in Level 2 and 3 need not be changed at all. Although packages like Boost offer the options of different data structures, most of the algorithm implementations are heavily entwined with that particular data structure. Therefore, changing a data structure often means rewriting the entire algorithm. ESSENS prevents this duplication of work and thereby also allows users to separately evaluate whether the improvement in performance because of a clever data structure or a superior algorithm design.

The second unique feature of ESSENS is that it does not divide the algorithms into known network analysis categories, such as community detection, centrality, etc, although examples of these methods are provided. Nor does ESSENS distinguish between static and dynamic networks. Our goal is that users can select the particular blocks required and create their necessary algorithms. If the necessary operations are provided then the users have the freedom to experiment with different static and dynamic algorithm schemes. This feature allows users to experiment with different options beyond what is provided in the software. Suppose the user would like to implement a BFS algorithm, but not traverse certain vertices based on specific properties. The template in Algorithm 1 can easily accommodate such a slight change in the algorithm. As far as we know, no other software allows this template-based facility of experimentation. In most cases, users have to go through the code to figure out the appropriate place to change the code.

A. Runtime Comparisons of ESSENS

ESSENS provides users the flexibility to use either the generalized building blocks at the higher level or customize their algorithms using the lower level blocks. The former improves the ease of development and productivity while the later can be used to improve the performance. As an example, we compared these two approaches with graph algorithms

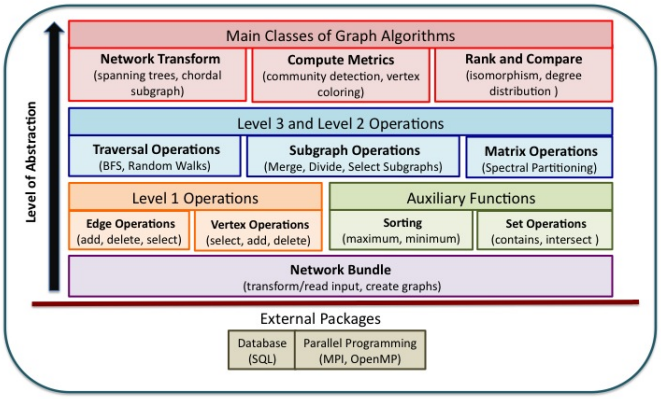

Fig. 1. Blueprint of the ESSENS Framework Showing the Levels of Abstraction

provided in the Boost Graph Library, on a set of benchmarks, listed in Table I, obtained from the Standford Network Database (SNAP) [31]. The runtimes were computed only for the analysis process, and not for graph reading, creation and other auxiliary operations.

We first experimented with executing BFS as the high level traversal algorithm. As can be seen from the top figure in Figure 2, the algorithm in the Boost Graph Library was considerably faster than the one in ESSENS. This is because the implementation is ESSENS is generalized to call any traversal function so long as the appropriate priority is specified. This generalization requires multiple function calls in Levels 2 and 3, which adds to the runtime. However, this also provides the opportunity for the users to mix-and-match traversal operations (for example run BFS for the first half of the vertices and then DFS for the next half), by just changing the appropriate priority function. Other packages do not allow such easy mixing of operations.

Nevertheless, for a user only requiring BFS, the generalized traversal function is expensive. Then the user can invoke lower level functions such as finding the neighbors, and selecting them for the next wave of traversal as used in the traditional BFS computations. We implemented the Brandes algorithm [32] for computing betweenness centrality using the Level 1 operations only. In this case, we do not use any generalized functions and the runtime of ESSENS is considerably lower than that of the algorithm provided by Boost Graph Library. Of course, users familiar with a certain software can improve the runtime by tuning the implementations. However our goal in ESSENS is to differentiate between the high level and low level algorithms, such that new users can easily identify the right blocks to model their algorithms and experienced users can write their algorithms using low level functions to improve performance.

TABLE I
Test Suite of Networks

Name	Vertices	Edges	Description
Caida	16301	65910	Autonomous systems network
WikiVote	58228	214078	Wikipedia Voting Network
Brightkite	58228	428156	Social network
CondMat	23133	186956	Collaboration network
Gnutella	62586	295784	Peer-to-peer file sharing network

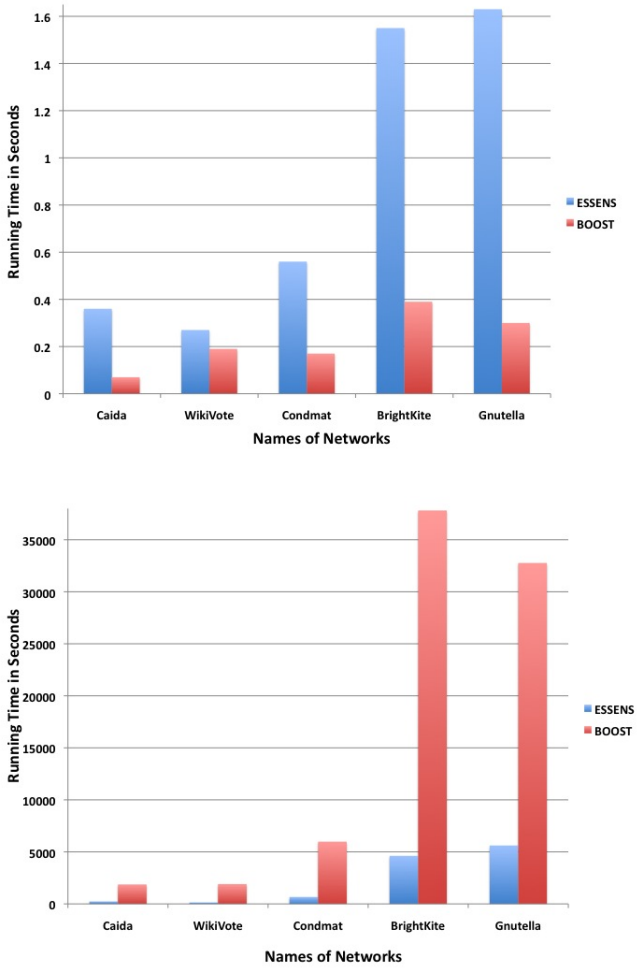

Fig. 2. Comparison of Runtime of ESSENS with Boost Graph Library. The times are given in Seconds. **Top:** Runtimes for BFS. **Bottom:** Runtimes for Betweenness Centrality using the Brandes Method.

V. Conclusion

We presented a general, extensible, component based framework for graph abstractions and extended it for graph-based network analysis. Our framework is independent of data structures, programming models, architectures. This component based interface will allow network science researchers, graph algorithm developers to collaborate efficiently. We also presented a reference implementation of this framework as a software library, ESSENS, and demonstrated a high level algorithm that improves productivity and a low-level high performance implementation of a network analysis algorithm. Our hope is for this paper and framework to simulate discussion between the graph algorithms and network science communities in order arrive at a standard that benefits both the communities.

Acknowledgment

Sandia is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. The University of Nebraska authors are supported by College of IS&T and the Graca Grant for UNO-SPR.

References

[1] K. Voevodski, S. H. Teng, and Y. Xia, "Finding local communities in protein networks," *BMC Bioinformatics*, vol. 10, no. 1, pp. 297+, Sep. 2009. [Online]. Available: http://dx.doi.org/10.1186/1471-2105-10-297

[2] M. A. Porter, P. J. Mucha, M. E. J. Newman, and A. J. Friend, "Community structure in the united states house of representatives," *Physica A: Statistical Mechanics and its Applications*, vol. 386, no. 1, pp. 414–438, Dec. 2007. [Online]. Available: http://www.sciencedirect.com/science/article/B6TVG-4P940N5-1/1/e608cafbf91c360a50ebf6b2a0f006d4

[3] L. Šubelj and M. Bajec, "Software systems through complex networks science: Review, analysis and applications," in *Proceedings of the First International Workshop on Software Mining*, ser. SoftwareMining '12. New York, NY, USA: ACM, 2012, pp. 9–16. [Online]. Available: http://doi.acm.org/10.1145/2384416.2384418

[4] H. Jeong, S. P. Mason, A. L. Barabasi, and Z. N. Oltvai, "Lethality and centrality in protein networks," *Nature*, vol. 411, no. 6833, pp. 41–42, May 2001. [Online]. Available: http://dx.doi.org/10.1038/35075138

[5] K. Wakita and T. Tsurumi, "Finding community structure in mega-scale social networks: [extended abstract]," in *Proceedings of the 16th International Conference on World Wide Web*, ser. WWW '07. New York, NY, USA: ACM, 2007, pp. 1275–1276. [Online]. Available: http://doi.acm.org/10.1145/1242572.1242805

[6] J. G. Siek, L.-Q. Lee, and A. Lumsdaine, *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley, 2002. [Online]. Available: http://www.awprofessional.com/title/0201729148

[7] E. G. Boman, K. D. Devine, V. J. Leung, S. Rajamanickam, L. A. Riesen, M. Deveci, and U. Catalyurek, "Zoltan2: Next-generation combinatorial toolkit." Sandia National Laboratories, Tech. Rep., 2012.

[8] B. W. Barrett, J. W. Berry, R. C. Murphy, and K. B. Wheeler, "Implementing a portable multi-threaded graph library: The mtgl on qthreads," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*. IEEE, 2009, pp. 1–8.

[9] A. Buluç and J. R. Gilbert, "The combinatorial blas: Design, implementation, and applications," *International Journal of High Performance Computing Applications*, vol. 25, no. 4, pp. 496–509, 2011.

[10] T. Mattson, D. Bader, J. Berry, A. Buluc, J. Dongarra, C. Faloutsos, J. Feo, J. Gilbert, J. Gonzalez, B. Hendrickson, J. Kepner, C. Leiserson, A. Lumsdaine, D. Padua, S. Poole, S. Reinhardt, M. Stonebraker, S. Wallach, and A. Yoo, "Standards for graph algorithm primitives," in *High Performance Extreme Computing Conference (HPEC), 2013 IEEE*, Sept 2013, pp. 1–2.

[11] P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker, "Cytoscape: a software environment for integrated models of biomolecular interaction networks," *Genome research*, vol. 13, no. 11, pp. 2498–2504, 2003.

[12] J. O'Madadhain, D. Fisher, S. White, P. Smyth, and Y. biao Boey, "Analysis and visualization of network data using jung."

[13] M. Bastian, S. Heymann, and M. Jacomy, "Gephi: An open source software for exploring and manipulating networks," 2009. [Online]. Available: http://www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154

[14] G. Csardi and T. Nepusz, "The igraph software package for complex network research," *InterJournal*, vol. Complex Systems, p. 1695, 2006. [Online]. Available: http://igraph.org

[15] S. M. Mniszewski, S. Y. Del Valle, P. D. Stroud, J. M. Riese, and S. J. Sydoriak, "Episims simulation of a multi-component strategy for pandemic influenza," in *Proceedings of the 2008 Spring Simulation Multiconference*, ser. SpringSim '08. San Diego, CA, USA: Society for Computer Simulation International, 2008, pp. 556–563. [Online]. Available: http://dl.acm.org/citation.cfm?id=1400549.1400636

[16] Y. Qiuju and C. Qingqing, "A social network analysis platform for organizational risk analysis – ora," in *Intelligent System Design and Engineering Application (ISDEA), 2012 Second International Conference on*, Jan 2012, pp. 760–763.

[17] N. Team, "Network workbench too," *Indiana University, Northeastern University, and University of Michigan*, 2006. [Online]. Available: http://nwb.slis.indiana.edu

[18] D. Gregor and A. Lumsdaine, "The parallel bgl: A generic library for distributed graph computations," in *In Parallel Object-Oriented Scientific Computing (POOSC*, 2005.

[19] "Knowledge discovery toolkit." [Online]. Available: http://kdt.sourceforge.net

[20] D. Bader and K. Madduri, "Snap, small-world network analysis and partitioning: an open-source parallel graph framework for the exploration of large-scale networks," *22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2008.

[21] C. Staudt, A. Sazonovs, and H. Meyerhenke, "Networkit: An interactive tool suite for high-performance network analysis," *CoRR*, vol. abs/1403.3005, 2014.

[22] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '10. New York, NY, USA: ACM, 2010, pp. 135–146. [Online]. Available: http://doi.acm.org/10.1145/1807167.1807184

[23] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed graphlab: A framework for machine learning and data mining in the cloud," *Proc. VLDB Endow.*, vol. 5, no. 8, pp. 716–727, Apr. 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2212351.2212354

[24] Y. Pign, A. Dutot, F. Guinand, and D. Olivier, "Graphstream: A tool for bridging the gap between complex systems and dynamic graphs," *CoRR*, vol. abs/0803.2093, 2008.

[25] D. Ediger, K. Jiang, E. Riedy, and D. Bader, "Graphct: Multithreaded algorithms for massive graph analysis," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 24, no. 11, pp. 2220–2229, Nov 2013.

[26] P. M. Dearing, D. R. Shier, and D. D. Warner, "Maximal chordal subgraphs," *Discrete Appl. Math.*, vol. 20, no. 3, pp. 181–190, Jul. 1988. [Online]. Available: http://dx.doi.org/10.1016/0166-218X(88)90075-3

[27] L. Fleischer, B. Hendrickson, and A. Pinar, "On identifying strongly connected components in parallel," in *Parallel and Distributed Processing*, ser. LNCS. Springer Berlin Heidelberg, 2000, vol. 1800, pp. 505–511.

[28] W. McLendon, III, B. Hendrickson, S. J. Plimpton, and L. Rauchwerger, "Finding strongly connected components in distributed graphs," *Journal of Parallel and Distributed Computing*, vol. 65, p. 2005, 2005.

[29] G. M. Slota, S. Rajamanickam, and K. Madduri, "Bfs and coloring-based parallel algorithms for strongly connected components and related problems," in *2014 IEEE 28th International Parallel & Distributed Processing Symposium (IPDPS), To Appear*, 2014.

[30] S. Salihoglu and J. Widom, "Gps: A graph processing system," in *Scientific and Statistical Database Management*. Stanford InfoLab, July 2013. [Online]. Available: http://ilpubs.stanford.edu:8090/1039/

[31] J. Leskovec, "Stanford Large Network Dataset Collection." [Online]. Available: http://snap.stanford.edu/data/

[32] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of Mathematical Sociology*, vol. 25, pp. 163–177, 2001.