Router: a framework integrating different parallel computation model for large graph mining

ZengFeng Zeng
School of Computer Science
Beijing University of Posts and Telecommunications
Beijing 100876, China
zzfeng1987@gmail.com

Bin Wu
School of Computer Science
Beijing University of Posts and Telecommunications
Beijing 100876, China
zzfeng1987@gmail.com

ABSTRACT

Extracting knowledge from graphs is important for many applications. The increasingly scale of graphs poses challenges to their efficient processing. In this paper, we proposed a framework integrating different parallel computing model to address this task. The framework builds up a well-structured fictitious communications network of the original graph by aggregating the dense subgraph into the Router. The programs are expressed by iterative message passing among the nodes or Routers of the network. A Router of the network presents a dense subgraph and manages the message passing of nodes of the dense subgraph. Nodes of the network can receive messages sent in previous iteration and send messages to other nodes. As nodes of a dense subgraph are aggregated into a Router, much less data moved among different machines for passing messages, which makes our framework very efficient. In addition, the smart message passing scheme of our framework can make a multi-source traversal with one iterative process, improving the parallel efficiency largely. Besides, the divide-and-conquer paradigm is proposed as a complement for message passing approach. Our framework offers a abstract API to hidden the distribution-related details and make it easily to program for large graphs processing. Finally, the programs of our framework are convenient to choose a apt parallel computing model to run the program according the data size and required time.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Theory

Keywords

ACM proceedings, LaTeX, text tagging

1. INTRODUCTION

Large graph mining has become more and more important in various research areas such as for studying web and social networks. The graph dataset we face today are become much larger than before. The modern large search engines crawls more than one trillion links in the internet and the social networking web site contains more than 800 million active users [5]. Besides the large graph on Internet and social networks, the biological networks which represent protein interactions are of the same size [1]. We're rapidly moving to a world where the ability to analyses very-large-scale dynamic graphs (billions of nodes, trillions of edges) is becoming critical.

The above graphs are far too large for a single commodity computer to handle with. The common way to process the large graph is using the parallel computing systems to perform algorithms in which the graph data is distributed across a cluster of commodity machines. The various parallel computing models play an important role in handling these extremely large graphs. Some parallel graph processing systems or libraries based on different computational model have been proposed: Pegasus based on Hadoop's MapReduce, CGMGRAPH /CGMLIB based on MPI, google's Pregel based on the Bulk Synchronous Parallel model [7, 9, 3]. The Pegasus uses a repeated matrix-vector multiplication to express different graph mining operations (PageRank, spectral cluster-ing, diameter estimation, connected components etc), but it is usually not ideal for graph algorithms that often better fit a message passing model. The CGMgraph provides a number of parallel graph algorithms using the Coarse Grained Multi computer(CGM) model based on MPI. However, what the CGMgraph focus is providing implementations of algorithms rather than an infrastructure to be used to implement them. Comparing to CGMgraph, the Parallel BGL provides generic C++ library of graph algorithms and data structures that inherits the flexible interface of the sequential Boost Graph Library to facilitate porting algorithms. Pregel is a distributed programming framework, focused on providing users with a natural API for programming graph algorithms while managing the details of distribution invisibly, including messaging and fault tolerance. It is similar in concept to MapReduce [14], but with a natural graph API and much more efficient support for iterative computations over the graph. The vertex-centric approach

of pregel is flexible enough to express a broad set of algorithms, but it is not convenient and efficient to implement algorithms which need access to small portions of the graph, not just its neighbors, such as such as triangle listing and clique percolation. Besides, none of these systems consider minimizing communication complexity by partitioning the graph data properly and saturating the network becomes a significant barrier to scaling the system up.

Considering the drawbacks of above parallel graph processing systems or libraries, we propose the framework integrating different parallel computing model to address the following questions:

1. **Parallelization with limited effort**: Efficient parallelization of an existing sequential graph algorithm is non-trivial as factors such as communication, data management, and scheduling have to be carefully considered. Implementing graph algorithms on existing frameworks such as PThreads [10] and PFunc [19] for shared-memory systems, MPI [8] for distributed-memory systems and MapReduce built on distributed file systems is time-consuming and requires in-depth knowledge of parallel programming. In our framework, programs are expressed by iterative message passing among the nodes or Routers of the network. The user only need to override the abstract methods of node or Router to finish the algorithm without considering distribution-related details.
2. **Various paradigm for parallel graph algorithm design:** At present, most of parallel graph algorithm express by messages passing among the vertices. However, the message passing approach are not always suitable for different graph algorithm. For some algorithms, the node need to access a small portion topological information of whole graph, not just its adjacent nodes. Other paradigms for parallel graph algorithm design should be considered.
3. **Easy Switch among different Parallel computing model**: Every computing model has both advantages and disadvantages. Our framework is built up on different parallel computing model and each parallel computing model serve as a Runtime of the program. The program can easily switch from one parallel computing model to another computing model according the data size and required time. The user need not to develop different version of the program for different computing model on our framework.
4. **Good data layout to speed up the system**: A good data layout is critical for the parallel graph mining. As the graph is distributed among different machines, a good data layout that minimizes the number of cross partition edges will largely reduce the communications among the different machines. Saturating the network usually becomes a significant barrier to scaling the system up for current parallel graph processing systems. Our framework makes a good balance partition of original graphs by a parallel multi-level partitioning approach. Besides, a good graph partitioning is key preprocessing step to divide-and-conquer graph algorithms.

This paper makes the following contributions:

- The architecture of our framework that integrates different parallel computation model to make the user easily develop a graph algorithm and choose a apt parallel computing model to run the algorithm according the graph size ,required time or other factors. It is without any extra effort for the program to switch among different parallel computation model.
- Novel parallel multi-level graph partitioning algorithm to make a good data layout and speed up our framework largely.
- Smart message passing scheme that expresses the graph algorithm easily and has a high parallel efficiency for multi-source traversal on graph. Besides, divide-and-conquer paradigm for large distributed graph mining sever as a complement for message passing approach.
- Experimental study: We evaluate our framework on different parallel computing models with large graph of different scale and the experiment shows the efficiency and scalability of the framework.

The rest of the paper is structured as follows. Section 2 reviews related works. Section 3 gives some notations and definitions used in this paper. In Section4, we give the detail description of the parallel multi-level graph partitioning. Section 5 presents the parallel multi-level weighted label propagation algorithm and its implementation on MapReduce. In section 6, the stepwise minimizing RatioCut Algorithm is proposed to partition the weighted graph. Section 7 provides a detailed experimental evaluation of out algorithm compared with existing state-of-the-art algorithms and tests the improvement of some graph algorithms by only changing the data layout with our partitioning algorithm. And Finally in Section8, we draw the conclusions and discuss future work.

2. RELATED WORK

As many practical computing problems concern large graphs, such as the Web graph and various social networks, the parallel computing for large-scale graph has attracts many attentions. In this section, we reviews some related parallel computing framework and graph processing systems.

Graph mining on MapReduce: MapReduce is a programming framework for processing huge amounts of unstructured data in a massively parallel way.MapReduce framework relies on the operation of <key, value> pair, both the input and output is a <key, value> pair. Users specify a map function that processes a <key, value> pair to generate a set of intermediate <key, value> pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The Apache Hadoop software library implements MapReduce on its distributed file system HDFS, and provides a high-level language called PIG which is very popular in the industry due to its excellent scalability and ease of use. Many graph has been designed

based on MapReduce, such as PageRank, finding Components and enumerating triangles. U Kang proposed PEGASUS, an open source Graph Mining library implemented on the top of the HADOOP platform [].The main drawback of graph processing based on Hadoop's MapReduce is that the MapReduce framework are not suitable to implement the iterative operation efficiently which is very common in many graph algorithms.

Graph mining on MPI: Message Passing Interface (MPI) is a standardized and portable message-passing system designed by a group of researchers from academia and industry to function on a wide variety of parallel computers. The MPI also has been widely used in graph processing. The Parallel BGL [22, 23] specifies several key generic concepts for defining distributed graphs,provides implementations based on MPI [18].The Parallel BGL sports a modest selection of distributed graph algorithms, including breadth-first and depth-first search, Dijkstraąŕs single-source shortest paths algorithm, connected components, minimum spanning tree, and PageRank. Similar to Parallel BGL, the CGMgraph provides a number of parallel graph algorithms using the Coarse Grained Multi-computer (CGM) model based on MPI.What the CGMgraph focus is providing implementations of algorithms rather than an infrastructure to be used to implement them.

BSP and Pregel: The Bulk Synchronous Parallel (BSP) is a bridging model for designing parallel algorithms. It provides synchronous superstep model of computation and communication. Inspired by BSP, the Pregel computations consist of a sequence of iterations, called supersteps. During a superstep the framework invokes a user-defined function for each vertex, conceptually in parallel. Without assigning vertices to machines to minimize inter-machine communication, performance will suffer due to the message traffic when most vertices continuously send messages to most other vertices.

3. BASIC ARCHITECTURE AND DESCRIPTION

The primary goal of Router is to enable rapid development of parallel graph algorithms that run transparently on different parallel computing model without considering the distribution-related details. To realize this goal, a well-design architecture is proposed to integrating different parallel computing model and offer a unform interface.

The Router is organized into four distinct layers: (1) The user API layer,which provides the programming interface to the users. It primarily consists of abstract classes **Node** that allows users to override to express the graph algorithm in message passing way and **Operator** that offers the interface for designing the divide-and-conquer graph algorithms. (2) The Architecture independent layer, which act as the middle-ware between the user specified programs and the underlying architecture dependent layer. The layer is responsible for constructing the fictitious communications network of the original graph and implementing message passing or operations of Router with user specified parallel computing model.(3) The Architecture dependent layer, which consist of different parallel computing platform that allow Router to run portably on various runtimes. (4) For

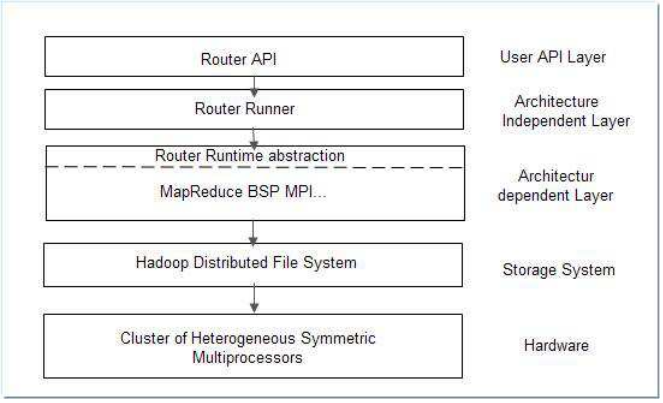

Figure 1: The architecture of Router

the storage layer, the hadoop's distributed file system is used for Storage in our framework.

The mechanism of our framework is the a well-structured fictitious communications network, in which a Router manages a portion of nodes that densely connected. In fact, the Router denotes a dense subgraph of which nodes are stored together in a machine. Hence, the communication of two nodes which are in the some router will not incur realistic data remove among different physical machines. The communication of Router consist of a sequence of iterations, during a iteration the node read the message other nodes send to it in the previous iteration and send the message to other nodes. When the node receiving message located in the same router, the message can directly send to the node, otherwise the message will be forwarded to the router where node locate and the router will transfer the message to the node. During a iteration the framework invokes a user-defined function for each node, conceptually in parallel. The function specifies operation at a single node and a single iteration. The parallel graph algorithms usually can be well expressed by the message passing on the fictitious communications network.

But the message passing approach are not convenient and efficient to express some graph algorithms in which nodes need to obtain the topological information of its vicinity. For example, maximal clique enumeration need to obtain the "two-leap" topological information of a specific node which are not suitable to using message passing approach due to massive information need to pass. As a complement of message passing approach, we define the Operator which consist of **compute** operation and **merge** operation of routers for dive-and-conquer paradigm. The Operator are run in two stage: At first stage, the **compute** operation finish the local computation of a router. At the second stage, **merge** operation will output the final results by combining the intermediate result from Adjacent routers.

4. MODEL OF COMPUTATION

The input of Router framework is a direct graph in which each vertex is uniquely identified by a string vertex identifier. After the direct graph imported by the framework, each vertex is denoted by a node object whose identifier equals vertex's in our fictitious communications network. The di-

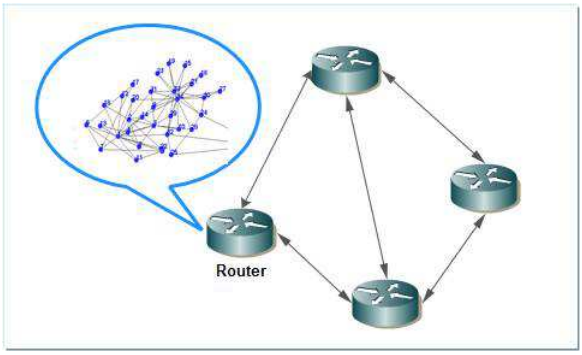

Figure 2: The mechanism of Router

rected edges are associated with their source nodes and each edge consists of a modifiable user defined value and a target node identifier. Our framework includes two models of computation: message passing model which is node-centric and dive-and-conquer model which is router-centric.

4.1 Message passing model

For the existing message passing model, like pregel, a vertex can modify its state to control the execution process of algorithm: when the vertex's state is inactive, the Pregel framework will not execute that vertex in subsequent supersteps unless reactivated by a message and the algorithm as a whole terminates when all vertices are simultaneously inactive and there are no messages in transit. However, this simple vertex state machine is not enough to express some complex graph algorithm efficiently, such as multi-source shortest paths algorithm(MSSP). The pregel can get the single-source shortest paths (sssp) directly: it starts from a specified source vertex s and in each superstep, each vertex first receives, as messages from its neighbors, updated potential minimum distances from the source vertex. If the minimum of these updates is less than the value currently associated with the vertex, then this vertex updates its value and sends out potential updates to its neighbors, consisting of the weight of each outgoing edge added to the newly found minimum distance [9]. The MSSP problem can simply solved by repeatedly using SSSP algorithm. But using SSSP algorithm repeatedly to get multi-source shortest paths will lead to many iterative processes and each iterative process is not fully used as many vertexes are inactive in the iterative process.

Can we solve the MSSP problem in one iterative process? In this paper, we proposed the message state machine compared to the vertex state machine in pregel which can realize the multi-source traversal in one iterative process. In our message passing model, the node is associated with a message table that store the message it receive and the message format is the key for our message passing model to realize the multi-source traverse. The standard message include the following properties: **srcId** record source node of the message, the **state** variable which is active or inactive indicates the state of the message and **content** stores the specified content of the message. In the iterative process, the messages started by some source nodes at the first iteration,

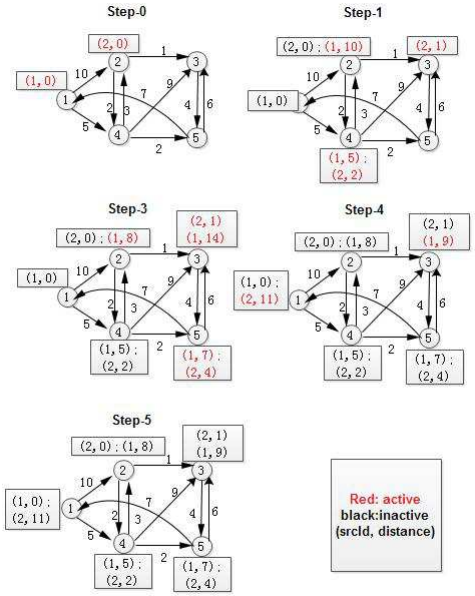

Figure 3: Find multi-source shortest paths in one iterative process

and then the node receives the messages started by different source nodes and stores the message with its message table. The node update the state of messages in the message table and send the message to other nodes. The algorithm as a whole terminates when all vertices have not active messages in message table. In the following, we will illustrate this multi-source message passing model by MSSP problem.

The formulation of this MSSP algorithm will be briefly described in the following.

1. Initialize the specified nodes: the specified node's message table contains a message whose id is the node's id, the state is active and the content is the distance that equals 0;
2. For the message in the message table of each node, if its state is active, add the weight of edge between the vertex and its adjacent vertex to the message's distance, and then send the message to its adjacent vertex, finally set the message inactive.
3. When the router receive a new message, if the message table has a message whose srcId equals the new message's srcId and the new message's distance is less than the old message's distance , update the old message with the new message. If there is no message whose srcId equals the new message's srcId in the mesTable, then put the new message into the message table.
4. Jump 2, until there is no active message in node's message table.

4.2 Dive-and-conquer model

The dive-and-conquer model is router-oriented which support the larger granularity of parallel graph processing compared with the vertex-centric message passing approach with

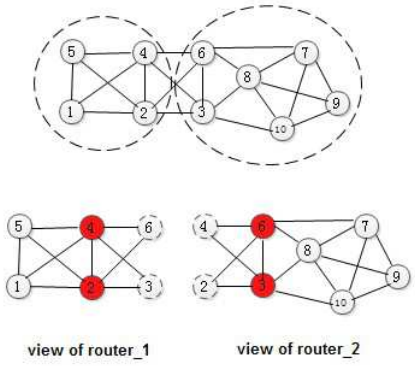

Figure 4: The view of different routers

a small granularity of parallelism . The vertex-centric message passing facilitates the iterative graph algorithm such as PageRank, but it is convenient for non-iterative graph algorithms that need to access local topological information such as maximal clique enumeration. In this section, we propose the dive-and-conquer model to address this problem. In our framework, each router possess a dense subgraph of which the topological information is continuously stored in a machine. Hence, it's very convenient for a router to operate its own subgraph. Our dive dive-and-conquer model consists of two stages: local computation stage and merging stage. In the local computation stage, the router runs the algorithm on local subgraph and send the boarder topological information or other information to its adjacent routers. In the merging stage, the router output the final results by combining the result of local computation stage and the received information. In the following, we will use the a maximal clique enumeration algorithm to illustrate this model.

The Figure 4 presents the different view of two different routers. The red node is the border node of which part of adjacent nodes(dashed node shown in Figure 4) locating in other routers. For the maximal clique enumeration algorithm, each router enumerates maximal cliques without considering dashed nodes in the local computation stage. The router_1 outputs the clique $< 1, 2, 4, 5 >$ and the router_2 output cliques: $< 3, 6, 8 >, < 3, 8, 10 >, < 7, 8, 9, 10 >$. In the merging stage, the router_2 sends its border nodes to router_1 and the router_1 contains all the border nodes and outputs the clique consists of border nodes $< 2, 3, 4, 6 >$.

5. THE JAVA API

In this section, we describe the most important aspects of the Router framework's java API without considering relatively mechanical issues. As our framework offers two computation model for graph processing, there are two suite of API to program.

Writing a program using message passing model involves subclassing the predefined node class (see Fig.5). The user overrides the abstract method **receiveMessage()** and **sendMessage()**which will be executed when the node receives messages that other node send to it at previous iteration and sends messages to other nodes respectively. The messageTable is used to store the message the vertex receives

```
public abstract class Node {
    private String id;
    private List<Edge> edgeList;
    private Hashtable<String,Message> messageTable;
    public abstract void sendMessage (MessageCollector collector);
    public abstract void receiveMessage (Iterator<Message> iterator);
}

public class Router {
    private String id;
    private List<Node> nodeList;
    private Hashtable<String, String> routingTable;
    public void sendMessage (MessageCollector collector);
    public void receiveMessage (Iterator<Message> iterator);
}

public abstract class Operator {
    private Router router;
    private Set interimResults;
    public abstract void compute (MessageCollector collector);
    public abstract void merge (Iterator<Message> iterator);
}
```

Figure 5: The main API of Router framework.

and edgeList stores the directed edges that associated with the node. We present the code of above MSSP algorithm to illustrate how to use API to implement graph algorithm in Figure 6.

When using the dive-and-conquer model to process graph, the user need to implement the **compute()** method and the **merge()** method of Operator. In the following, we will employ the maximal clique enumeration algorithm as a example to illustrate how to use interface of Operator to process graph. In the **compute()** method, maximal cliques are enumerated based on **nodeList** of Router. In the **merge()** method, all the border nodes are gathered and maximal cliques of border node are enumerated. The code of maximal cliques enumeration algorithm are shown in the Figure 7.

6. IMPLEMENTATION

6.1 Build up the fictitious communications network

6.2 Implement message passing

7. EXPERIMENT

8. CONCLUSIONS

This paragraph will end the body of this sample document. Remember that you might still have Acknowledgments or Appendices; brief samples of these follow. There is still the Bibliography to deal with; and we will make a disclaimer about that here: with the exception of the reference to the LaTeX book, the citations in this paper are to articles which have nothing to do with the present subject and are used as examples only.

9. ACKNOWLEDGMENTS

This section is optional; it is a location for you to acknowledge grants, funding, editing assistance and what have you.

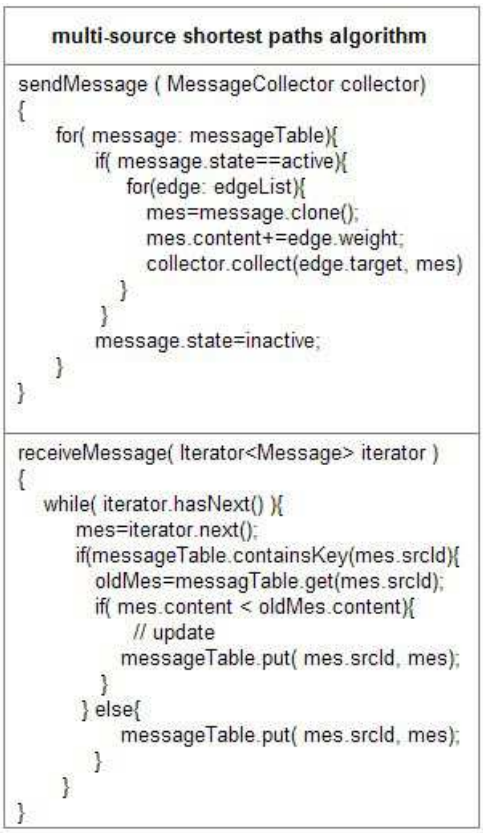

Figure 6: The multi-source shortest paths algorithm implemented in Router Framework.

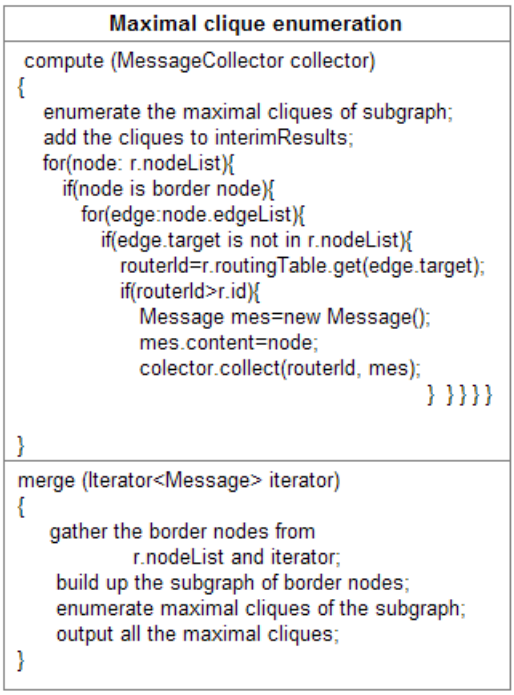

Figure 7: maximal clique enumeration algorithm implemented in Router Framework.

In the present case, for example, the authors would like to thank Gerald Murray of ACM for his help in codifying this *Author's Guide* and the **.cls** and **.tex** files that it describes.

10. REFERENCES

[1] M. Bailly-Bechet, C. Borgs, A. Braunstein, J. T. Chayes, A. Dagkessamanskaia, J.-M. Franfois, and R. Zecchina. Finding undetected protein associations in cell signaling by belief propagation. *CoRR*, pages –1–1, 2011.

[2] S. Barnard. Pmrsb: Parallel multilevel recursive spectral bisection. In *Supercomputing, 1995. Proceedings of the IEEE/ACM SC95 Conference*, page 27, 1995.

[3] A. Chan, F. K. H. A. Dehne, and R. Taylor. Cgmgraph/cgmlib: Implementing and testing cgm graph algorithms on pc clusters and shared memory machines. *IJHPCA*, 19(1):81–97, 2005.

[4] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, pages 107–113, 2008.

[5] Facebook, 2011. http://facebook.com/press/info.php?statistics.

[6] M. Girvan and M. E. Newman. Community structure in social and biological networks. *Proc Natl Acad Sci U S A*, 99:7821–7826, June 2002.

[7] U. Kang, C. Tsourakakis, and C. Faloutsos. Pegasus: A peta-scale graph mining system implementation and observations. In *Data Mining, 2009. ICDM '09. Ninth IEEE International Conference on*, pages 229 –238, dec. 2009.

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

[9] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *SIGMOD Conference'10*, pages 135–146, 2010.

[10] U. N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks, Sept. 2007.