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

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 build up the multilevel hierarchical network of the original graph by aggregating the dense subgraph iteratively.The programs are expressed by iterative message passing among the vertices 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. Vertices of the network can receive messages sent in previous iteration and send messages to other vertices. As vertices in 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. The multilevel hierarchical network and the smart message passing scheme make our framework to express a broad set of algorithms flexibly and efficiently. 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 ap-

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

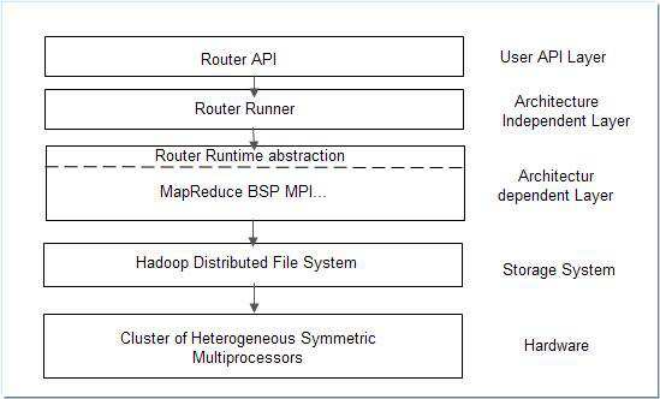

Figure 1: The architecture of Router

The mechanism of our framework is the a well-structed fictitious communications network, in which a Router manage a portion of nodes that densely connected. In fact, the Router denote 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 incurs 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 directed edges are associated with their source nodes and each

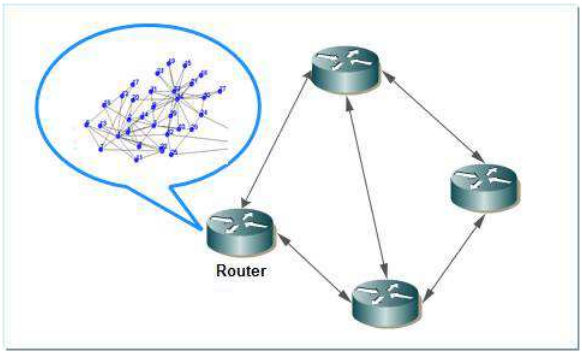

Figure 2: The mechanism of Router

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 []. 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, and then nodes receive the messages come from different

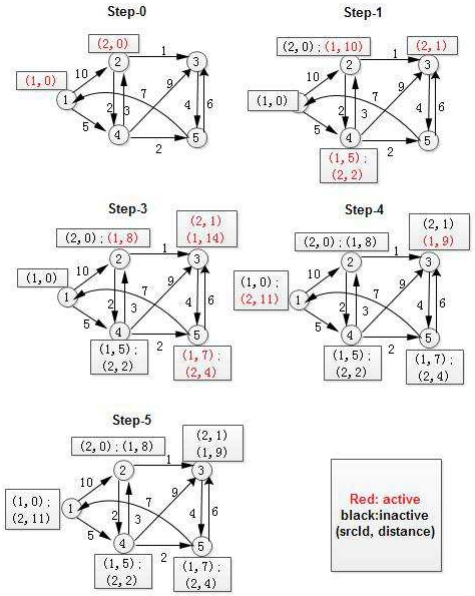

Figure 3: Find multi-source shortest paths in one iterative process

source nodes and store 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: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 a small granularity of parallelism . The vertex-centric message passing facilitates the iterative graph algorithm such as

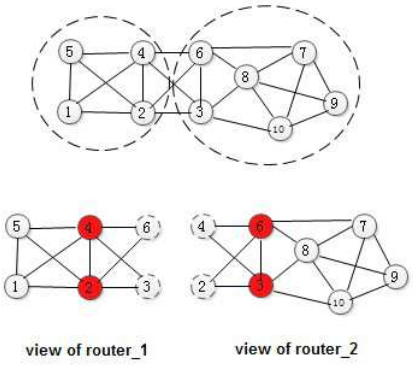

Figure 4: The view of different routers

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.

5. MULTI-LEVEL PARTITIONING ALGORITHM

5.1 Overview of the Multilevel Paradigm

Graphs that we meet in practice are not random. The distribution of edges is not only globally, but also locally inhomogeneous, with high concentrations of edges within special subgraph of vertices, and low concentrations between these subgraphs. This feature of graphs is called community structure or clustering [6]. It may be due to the vertices being geographically close in social networks, or being related to topic or domain on the web. Our graph partition algorithm takes advantage of this locality to make a good partition. The main idea behind our graph partition algorithm is the dense subgraph should not be divided among partitions. So the dense subgraph is treated as an indivisible atom, and the graph partition towards vertexes are transformed into the graph partition towards the dense subgraph. The dense subgraph means that there must be more edges "inside" the subgraph than edges linking vertices of the subgraph with the rest of the graph. The above dense subgraph is known as community.

The basic structure of our graph partitioning algorithm is similar to the multilevel k-way partitioning algorithm. Our graph partitioning algorithm is also a multilevel algorithm and uses the bottom-up method. The algorithm is divided into two phase: aggregate phase and partition phase. In the aggregate phase, we continue to aggregate the original graph to a smaller graph on higher level. In each level, we use a label propagation algorithm to detect dense subgraphs and these dense subgraphs will be a vertex of upper level. Then the label propagation algorithm continues on the upper level graph until the vertex size of upper level graph smaller than the threshold we set. In the partition phase, a balance graph partition performs on the top-level graph which vertex size is much smaller than the original graph. As the top-level graph is a vertex-weighted and edge-weight graph, where the edge-weight reflects the number of edges connecting two different super vertexes and the vertex-weight reflects vertex number in the super vertex, the traditional partition algorithms such as Kernighan-Lin (KL) or spectral method are not suitable to this graph[8, 2]. We propose a novel stepwise partition algorithm using a greedy method to get a small RatioCut step by step. At every step, we extract a connected subgraph with vertex size approximates $\frac{|V|}{k}$ which has minimal edge cuts with the remaining graph, and then remove the subgraph from the original graph. In this way, k subgraphs are iteratively extracted from the graph. A partition example on synthetic graph is shown in Figure 5.

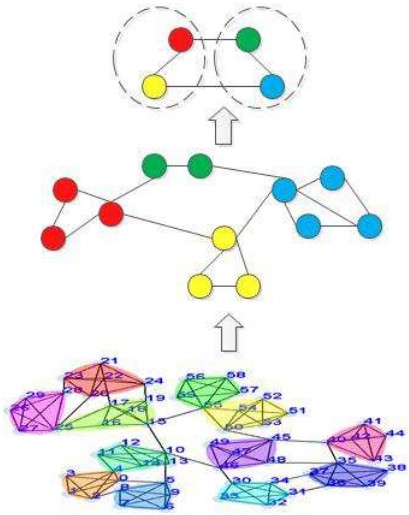

Figure 5: A partition example on synthetic graph

5.2 Weighted label propagation

The reasons we use the label propagation to detect the dense subgraph are: the label need not access other vertex's topological information, what it need is sending the label to its neighbor [10]. For this property, it very suitable to parallel computing environment where access the vertex's topological information is expensive. And the time complexity of label propagation algorithm is linear. However the original label propagation algorithm that performs on unweighted graph is not adapt to the weighted graph on upper level we face in the aggregate phase. In this paper, we propose a weighted label propagation algorithm to handle with the above weighted graph.

Our label propagation algorithm is based on the ideal that which community the vertex v belongs to is determined by

its close neighbors. We assume that each vertex in the graph chooses to join the community that most of its close neighbors belong to. Suppose that a vertex v has neighbors $N(v) = \{u_1, u_2, ..., u_k\}$ and each neighbor owns a label L_u denoting which community they belong to. The neighbor who is more close to the vertex v has more influence on vertex v. Hence we define the u's affinity to v

$$aff(u, v) = \frac{W(u, v)}{\sum_{i \in N(v)} W(u, i)}$$

The degree of membership indicates how much the vertex belongs to a community C is defined as

$$d(v, C) = \frac{\sum_{u \in N(v), L_u = C} aff(u, v)}{\sum_{u \in N(v)} aff(u, v)}$$

The vertex v belongs to the community C which has the maximal degree $d(v, C)$.

The algorithm is preformed iteratively, where at iteration t, each vertex updates its label by computing the degree $d(v, C)$ based on labels of its neighbor vertexes at iteration $t-1$. Hence,$L_v(t) = f(L_{u_1}(t-1), ..., L_{u_i}(t-1))$, where the function returns he label with the maximal degree of membership. As the labels propagate, closely and densely connected subgraphs of vertexes quickly reach a consensus on a unique label. At the end of propagation process, vertexes with the same labels are aggregated as one community.

Ideally, the algorithm should be end when all vertexes' label unchanged after iterations. But in terms of the ambiguous vertexes with the same membership degree of several communities, their label may change after each iteration. So the algorithm should be ended, when most of nodes' labels are stable. Here, we will set the maximal number of iteration of the algorithm in the practice. The algorithm will be intermitted when iteration times equals the maximal number. We can describe our weighted label propagation algorithm in the following steps.

(1) Initial the label of vertex in graph. For a given vertex v, $L_v(0) = v$;

(2) Set $t = 1$;

(3) For each vertex v computes its affinity to its adjacent vertexes $aff(v, u)$, new a Label with weight equals $\frac{aff(v,u)}{|v|}$, where the divisor $|v|$ is used to avoid excessively rapid growth of large-size vertex in upper level,and then send the label to the adjacent vertex.

(4) For each vertex v, $L_v(t) = f(L_{u_1}(t-1), ..., L_{u_i}(t-1))$, where the function f returns the label with the maximal degree of membership.

(5) For most of vertex, the label is unchanged, then stop the algorithm. Else, set t=t+1 and go to (3).

5.3 Implementation on MapReduce

MapReduce is a parallel computing framework which is simple and easy to use [4]. The application, based on the Map-Reduce framework, can run on large-scale commercial clusters, process data in a parallel and fault-tolerant way.

weighted label propagation	
Mapper	input: vertex object v (fields:id, adjacency list and label) key: id of vertex value: vertex object v
	for each vertex adjId in adjacency list compute the aff(v,adjId); label l=v's label l.weight=aff(v,adjId)*decay(lvl) ; output<adjId, l>; output<id,vertex v>;
Reducer	key: id values: list of labels and vertex v
	compute v's membership degree of different labels (community); get the label l that has max degree; if v's label is not equal to l update v's label with l; output<id, vertex v>;
	output: the vertex v

Figure 6: Weighted label propagation on MapReduce

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 overall execution can be simply described as the following streaming:

Input | Mappers | Sort | Reducers | Output

The weighted propagation algorithm is very easy to be implemented in a MapReduce way. The pseudo code in Map and Reduce is presented in Figure 6. In the mapper, the vertex computes the affinity to its neighbors and sends the weighted label to its neighbors. In the reducer, the vertex collects the labels that it receives, then compute the membership degree of different labels and update its label with the label that has the maximal membership degree.

After the label propagation, the vertexes labeled with the same label will be grouped together as super vertex of upper level. The edge weight of these super vertexes must be computed for the further aggregation. A series of MapReduce operations will be designed to finish the construction of upper graph (see Figure 7). The Mapper1 and Redecer1 aggregate the vertexes with the same label to a super vertex. Edges between two different super vertexes are achieved in Mapper2 and Reducer2. Finally, the Mapper3 and Reducer3 transform the above edges to adjacent list format.

construct the upper level graph	
Mapper1	input : vertex object v key:id of vertex; value: vertex v; output<v's label, v>
Reducer1	key: label l; values:list of vertexes New a super vertex sup_v; add the vertexes of list into sup_v; set id of sup_v with label l; set weight of sup_v with sum of vertexes's weight; output<l, sup_v> output: super vertex sup_v

Mapper2	input: super vertex sup_v
	key: id of sup_v; value: sup_v;
	for each vertex v in sup_v
	adj_v :=v's adjacent vertex ;
	if adj_v not in sup_v
	if(v.id>adj_v.id)
	id1:=v.id; id2=adj_v.id;
	else
	id2=v.id; id1=adj_v.id;
	w:=weight between id1 and id2
	output<(id1,id2),(sup_v's id,w)>

Reducer2	key: pair (id1, id2)
	values:(sup_v1.id,w) and (sup_v2.id,w)
	if(sup_v1.id >sup_v2.id)
	output<(sup_v1.id,sup_v2.id),w>;
	else
	output<(sup_v2.id,sup_v1.id),w>;
	output: the edge between two different super vertexes and its weight

Mapper3	input: edge and its weight w key:(id1,id2); value: w; output<id1, (id2, w)> output<id2, (id1,w)>
Reduce3	key: id values:list of pairs sum the weights of some adjacent id; get adjacent list of id; output<id, adjacent list with weights>; the super vertex of upper level graph in adjacent list way

Figure 7: Construct the upper level graph

5.4 Stepwise minimizing RatioCut Algorithm

The top-level graph $G_m(V_m, E_m)$ produced by Multi-level weighted label propagation is a vertex-weighted and edge-weighted graph, where the edge-weight reflects the number of edges connecting two different super vertexes and the vertex-weight reflects vertex number of the super vertex. It is very important to note that the partitioning algorithm must be able to handle the edges and vertex weights. In this section, a partitioning algorithm based on stepwise minimizing RatioCut is proposed to compute a k-way partitioning of graph G_m such that each partition contains roughly $\frac{|V|}{k}$ vertexes of the original graph. Instead of minimizing $RatioCut(V_1, ..., V_k)$ all together, we try to find a vertex set V_i that minimizes $\frac{W(V_i, \overline{V_i})}{|V_i|}$, which is a part of $RatioCut(V_1, ..., V_k)$, as our first partition, and then remove the vertex set V_i from G_m and try to find the next partition in the same way. The partial-RatioCut of vertex set V is defined as

$$PRC(V_i) = \frac{W(V_i, \overline{V_i})}{|V_i|}$$

It is obvious that the problem of minimizing $PRC(V)$ is also NP-hard. Because the problem is a 2-way graph partition problem. And we approximate the $PRC(V_i)$ using a simple greedy algorithm. At each iteration, we greedily select the vertex that minimizing PRC of the currently constructed vertex set. Before the algorithm is proposed formally, we first give a proposition and proof it.

PROPOSITION 1. *There exist weighted graph G_m for which the $PRC(V_i)$ is non-monotonic as the vertex size $|V_i|$ grows.*

PROOF. Consider a current vertex set V_i. Assume that the next vertex selected for inclusion in the set is v. Then, the new set is $V_i \bigcup v$. By the partial-RatioCut definition, the $PRC(V_i \bigcup v)$ is $\frac{W(V_i \bigcup v, \overline{V_i \cup v})}{|V_i|+|v|}$ (note that the vertex of upper level is vertex-weighted super vertex). The $PRC(V_i \cup v)$ will decrease when:

$$\frac{W(V_i \cup v, \overline{V_i \cup v})}{|V_i| + |v|} < \frac{W(V_i, \overline{V_i})}{|V_i|}$$

$$\frac{W(V_i, \overline{V_i}) - k_{v,V_i} + d_v - k_{v,V_i}}{|V| + |v|} < \frac{W(V_i, \overline{V_i})}{|V_i|}$$

Because $W(V_i \cup v, \overline{V_i \cup v}) = W(V_i, \overline{V_i}) - k_{v,V_i} + d_v - k_{v,V_i}$ where $k_{v,V_i} = \sum_{i=v, j \in V_i} W(i,j)$ and the degree of vertex v $d_v = \sum_{i=v} W(i,j)$.

$$W(V_i, \overline{V_i}) - k_{v,V_i} + d_v - k_{v,V_i} < \frac{W(V_i, \overline{V_i})(|V_i| + |v|)}{|V_i|}$$

$$W(V_i, \overline{V_i}) - k_{v,V_i} + d_v - k_{v,V_i} < W(V_i, \overline{V_i}) + \frac{W(V_i, \overline{V_i})|v|}{|V_i|}$$

$$d_v - 2k_{v,V_i} < \frac{W(V_i, \overline{V_i})}{|V_i|}$$

Conversely, The $PRC(V_i \cup v)$ will increase when:

$$d_v - 2k_{v,V_i} > \frac{W(V_i, \overline{V_i})}{|V_i|}$$

□

From the proof of proposition 1, we can know that in the order to minimize the $PRC(V_i)$, in every step the next vertex which minimize $\frac{d_v - 2k_{v,V}}{|v|}$ is selected to add to V_i, as shown in Algorithm. The parameter α is used to avoid the case that

Algorithm 1 Greedy Algorithm for minimizing PRC.

Input:
Graph $G_m = (V_m, E_m)$;
the vertex set size $\frac{|V|}{k}$.
the start vertex v_0.
Output:
constructed vertex set V_i.

$V_i = \emptyset$;
$V_i = V_i \cup v_0$;
while $|V_i| < \frac{V}{k}$ **do**
  $v = \text{argmin}_{v \in V_m - V_i} \frac{d_v - 2k_{v,V}}{|v|}$;
  if $|V_i + |v| < \frac{(1+\alpha)|V|}{k}$ **then**
    $V_i = V_i \cup v$;
  else
    if $|V_i + |v| > \frac{(1-\alpha)|V|}{k}$ **then**
      Break;
    end if
  end if
end while
return V_i;

an exceptional larger vertex add to the vertex set whose size approximate $\frac{|V|}{k}$, making the set size much larger than $\frac{|V|}{k}$.

In the following, we will present the k-way balance partitioning algorithm based on above stepwise minimizing PRC algorithm. The algorithm runs with k steps. In every step, each remaining vertex is selected as a start vertex and gets a vertex subset S by stepwise minimizing PRC algorithm. And get the best subset S^* with minimal PRC from these subsets. Finally, remove the vertexes of S^* from graph and start next step.

Algorithm 2 Stepwise partitioning Algorithm .

Input:
Graph $G_m = (V_m, E_m)$;
the partition number k.
Output:
the vertex set List $setList$.

$setList = \emptyset$;
$bestSet = null$;
for $i = 1$; $i < k$; $i++$ **do**
  for v in V_m **do**
    $S = minimizePRC(v, G_m, \frac{|V|}{k})$;
    if $PRC(S) < PRC(bestSet)$ **then**
      $bestSet = S$;
    end if
  end for
  $setList = setList \cup bestSet$
  $V_m = V_m \setminus bestSet$
end for

6. 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.

7. ACKNOWLEDGMENTS

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

8. 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.