A parallel graph partitioning algorithm to speed up the large-scale distributed 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
wubin@bupt.edu.cn

ABSTRACT

For the large-scale distributed graph mining, the graph is distributed over a cluster of nodes, thus performing computations on the distributed graph is expensive when large amount of data have to be moved. A good partitioning of the distributed graph is needed to reduce the communication and scale the system up. Existing graph partitioning algorithms incur high computation and communication cost when applied on large distributed graphs. A efficient and scalable partitioning algorithm is crucial for large-scale distributed graph mining.

In this paper, we propose a novel parallel multi-level stepwise partitioning algorithm. The algorithm first efficiently aggregates the large graph into a small weighted graph without exchange of vertex's topological information, and then makes a balance partitioning on the weighted graph based on a stepwise minimizing RatioCut Algorithm. The experiment results show that our algorithm generally outperforms the existing algorithms and has a high efficiency and scalability for large-scale graph partitioning. Using our partitioning method, we are able to greatly speed up PageRank computation on Spark, a distributed computation system, for large social networks.

Categories and Subject Descriptors

G.2.2 [**Mathematics of Computing**]: Discrete Mathematics—*Graph Theory, Graph Algorithms*; D.1.3 [**Programming Techniques**]: Concurrent Programming—*Distributed programming*

General Terms

Algorithms, Experimentation

Keywords

Graph Partitioning, distributed graph mining, RatioCut, Multi-level

1. INTRODUCTION

1.1 large-scale graph mining

Graph datasets 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 [8]. Besides the large graph on Internet and social networks, the biological networks which represent protein interactions are of the same size [1]. Large graph processing has become more and more important in various research and we're rapidly moving to a world where the ability to analyzes 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 based on different computational model have been proposed: google's Pregel based on the Bulk Synchronous Parallel model, Pegasus based on Hadoop's MapReduce, CGMGRAPH /CGMLIB based on MPI [14, 19, 5] . Unfortunately, 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. Some complex graph algorithms, in which every vertex need access its neighbors frequently, such as triangle listing, clique percolation and Newman fastGN, could not be solve efficiently on these systems, because the graph is randomly distributed among the machines which will lead to a heavy traffic of the system's communication when every vertex try to access its neighbors .

1.2 graph partitioning

Good partition on large graph is critical for the graph mining for the following reasons. First, graph partitioning is key preprocessing step to divide-and-conquer algorithms, where it is often a good idea to break graph into roughly equal-sized dense subgraphs. The graph algorithm can respectively performs on these dense subgraphs and combines the intermediate results in the final phase. Second, a good partition that minimizes the number of cross partition edges can reduce the communications among the different machines at a large scale. As we know that inter-machine communication, even on the same local network, is much more expensive than inter-processor communication. Network latency is measured in microseconds while inter-process communication is measured in nanoseconds. The bad partition of graph will lead to much more data to be moved among the machines when performing graph algorithms on these distributed graph data, which will largely increase the process time and cause network links to become saturated.

The graph partition problem is NP-hard and has been researched many years. A number of high-quality and computationally efficient algorithms have been proposed, even if these solutions are not necessarily optimal, such as Kernighan-Lin algorithm, spectral bisection method and multilevel graph partitioning [16, 3, 15]. However these methods do not scale to large scale graph data, in part because of the running time, and in part because these algorithms require full information about the graph or large portions of the graph, which is impossible in distributed computing environment or will lead to large scale data to move among the machines. Recently, a steaming graph partitioning for large distributed graphs is proposed to read graph data serially from a disk onto a cluster based on simple heuristics [23]. Unfortunately, it is somewhat unrealistic for the distributed system where the graph data is loaded in a parallel way.

1.3 overview of our approach

In this paper, we propose a parallel multi-level graph partitioning algorithm to make a k-way balance partitioning on large graph. The algorithm divide into two phases: aggregate phase and partition phase. In aggregate phase, the algorithm uses a multi-level weighted label propagation to aggregate the large original graph into a small weighted graph. In partition phase, a k-way balance partitioning performs on the weighted graph based on a stepwise minimizing RatioCut method. In our algorithm topological information of vertexes is no need to exchange which makes little data to move among the different machines. Thus it can efficiently partition large-scale graphs on distributed system, where accessing vertex's topological information located on different machines is expensive. The algorithm takes $o(|E|)$ time and well scales with the size of graph and partition number. In addition, compared to the traditional partitioning algorithms, in which partition number k must meet $k = 2^m$ as partitions into more than two clusters are usually attained by iterative bi-sectioning, the partition number of our algorithm can be arbitrary.

1.4 contributions

This paper makes the following contributions:

- The parallel multi-level weighted label propagation algorithm: the algorithm efficiently aggregates the large graph into a small weighted graph without exchange of vertex's topological information. And we implement the algorithm on MapReduce framework.
- The stepwise minimizing RatioCut algorithm: the algorithm minimizes the RatioCut step by step. In every step, a set of vertexes is extracted by minimizing part of RatioCut and remove these vertexes from the small weighted graph. A k-way balance partitioning is obtained by this algorithm.
- The parallel multi-level graph partitioning: the partitioning on large-scale graph is achieved by combining above two algorithms simply.
- Experimental study: We compare our partitioning algorithm to many other partitioning algorithms on various graph dataset. And the results show that our algorithm generally outperforms others. Our algorithm also be evaluated on large-scale graph of different scale and the experiment shows the efficiency and scalability of the algorithm. We finally demonstrate the value of graph partitioning in graph mining by using our algorithm to partition graph for PageRank computation using the Spark cluster system.

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 Section 4, we first give the detail description of the parallel multi-level graph partitioning, and then present the parallel multi-level weighted label propagation algorithm and its implementation on MapReduce, finally, the stepwise minimizing RatioCut Algorithm is proposed to partition the weighted graph. Section 5 provides a detailed experimental evaluation of out algorithm compared with existing state-of-the-art algorithms and tests the efficiency improvement of the PageRank algorithm by only changing the data layout with our partitioning algorithm. And finally in Section 6, we draw the conclusions and discuss future work.

2. RELATED WORK

Graph partitioning is a fundamental issue in many research areas, such as parallel computing, circuit layout and the design of many serial algorithms, including techniques to solve partial differential equations and sparse linear systems of equations. The graph partitioning problem is NP-hard. The Kernighan-Lin algorithm (Kernighan and Lin,1970) is one of the earliest heuristics methods which motivated by the problem of partitioning electronic circuits onto boards [16]. The algorithm starts with an initial partition of the graph in two clusters of the predefined size. The initial partition can be obtained by random partition or suggested by some information on the graph structure. At each iteration the algorithm swaps subsets consisting of equal numbers of vertices between the two sets to reduce the number of edges joining the two sets. The algorithm terminates when it is no longer possible to reduce the number of edges by swapping subsets.The Kernighan-Lin algorithm's time complexity is $O(n^2logn)$ (n being the number of vertices). The Kernighan-Lin algorithm subsequently improved in terms of running time by Fiduccia and Mattheyses [9].

The spectral bisection method is another popular method, which is based on the spectrum of the graph 's Laplacian matrix [3]. A good approximation of the minimum cut size of partition can be attained by choosing the index vector s parallel to the second lowest eigenvector. The first k eigenvectors of the Laplacian can be computed by using the Lanczos method, which is quiet fast if the eigenvalues λ_{k+1} and λ_k are well separated.

Other popular methods for graph partitioning include level-structure partitioning, the geometric algorithm, and multilevel algorithms. A level-structure was provided in SPARSPAK [4], a library of routines for solving sparse systems of equations by direct methods. The algorithm first finds a pseudo-peripheral vertex v in the graph. A breadth-first search from v is used to partition the vertices into levels: the vertex v

belongs to the zeroth level, and all neighbors of vertices in the ith level belong to the $(i+1)th$ level, for $i = 0, 1, ..$ the algorithm in SPARSPAK chooses the vertices in the median level as the vertex separator. The geometric partitioning algorithm is designed by Miller, Teng, Thurston, and Vavasis [20]. This algorithm partitions a graph by using a circle rather than a straight-line to cut the mesh. The basic structure of a multilevel partitioning algorithm is very simple. The graph $G = (V, E)$ is first coarsened down to a small number of vertices, a k-way partitioning of this much smaller graph is computed, and then this partitioning is projected back toward the original graph (finer graph) by successively refining the partitioning at each intermediate level. This three-stage process is coarsening, initial partitioning, and refinement [15].

A parallel graph partition algorithm was proposed in JOSTLE. However, to do it in parallel JOSTLE must first distribute the graph sensibly amongst the processors and to distribute the graph sensibly it must first find a reasonable partition. Then The JOSTLE optimizes the initial crude distribution of graph. Another parallel multilevel graph partitioning is implemented on shared-memory multicore architectures which is not suitable for the distributed computing environment for the frequent topological information exchange among vertexes [24].

3. PRELIMINARIES

In this section,we briefly describe some notations and definitions used in this paper.

3.1 Graph notation

$G = (V, E)$ is an graph where the V is a set of vertex and $E \subseteq V \times V$ is a set of edges.The adjacent matrix of graph is the matrix $W = (\omega_{i,j})_{i,j=1,2,..n}$. $\omega_{i,j} > 0$ indicates that the vertex v_i and v_j are connect by an edge and the weight is $\omega_{i,j}$. As G is unweighted, if the vertex v_i is adjacent to v_j, the $\omega_{i,j} = 1$, otherwise $\omega_{i,j} = 0$. $N(S)$ is the neighbors of vertex set S if $N(S) = \omega \in V \backslash S : \exists v \in S s.t. (v, \omega) \in E$

3.2 Graph partitioning

For any subset of vertices $V_i \subseteq V$,its complement $V \setminus V_i$ is denoted by $\overline{V_i}$. For two For two not necessarily disjoint sets we define

$$W(A, B) = \sum_{i \in A, j \in B} \omega_{ij}$$

The cut set induced by V_i is $C_i = (u, v) \in V_i, \overline{V_i}$, so the edge cut of cut set C_i is $W(V_i, \overline{V_i})$. The subsets of $P = V_1, V_2, .., V_k$ are k-way partitioning of graph G iff (1) $\cup_i V_i = V$ and (2) $\forall i, j : i \neq j \rightarrow V_i \bigcap V_j = \emptyset$. The most directed way to construct a partition of graph is solving the mincut problem which chooses a partition $P = V_1, V_2, .., V_k$ that minimizes

$$cut(V_1, V_2, ..., V_k) = \sum_{i=0}^{k} W(V_i, \overline{V_i})$$

However the solution of mincut which simply separates one individual vertex from the rest of the graph in many cases does not lead to satisfactory partition. The one of most common objective functions to make "balance partition" of graph is RatioCut [11]. The definition is:

$$RatioCut(V_1, V_2, ..., V_k) = \frac{\sum_{i=1}^{k} W(V_i, \overline{V_i})}{|V_i|}$$

The balancing conditions makes the previously mincut problem which is simple to solve become NP hard [6].

4. MULTI-LEVEL STEPWISE PARTITIONING ALGORITHM

4.1 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 [10]. 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 that 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 super 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[16, 3]. 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 1.

4.2 Weighted label propagation

The main reason we use the label propagation to detect the dense subgraph is: the label need not access other vertex's topological information, what it need is sending the

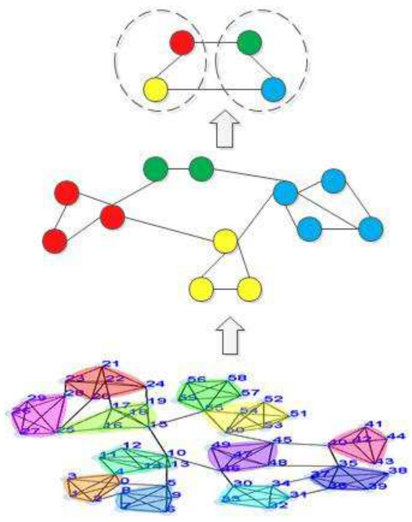
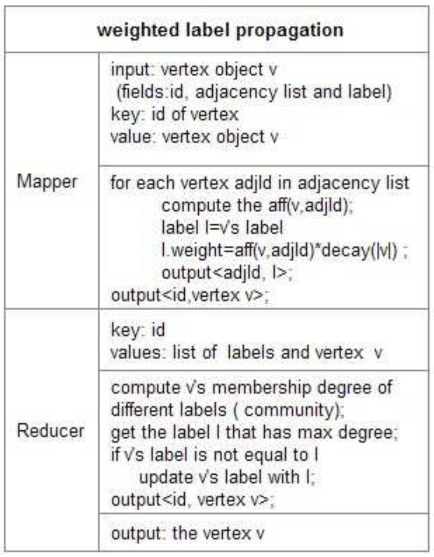


Figure 1: A partition example on synthetic graph.



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(|v|) ; 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 2: Weighted label propagation on MapReduce.

label to its neighbor [21]. For this property, it very suitable to parallel computing environment where access the vertex's topological information is expensive. Besides, 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 vertexes in graph. For a given vertex v, $L_v(0) = v$;

(2) Set $t = 1$;

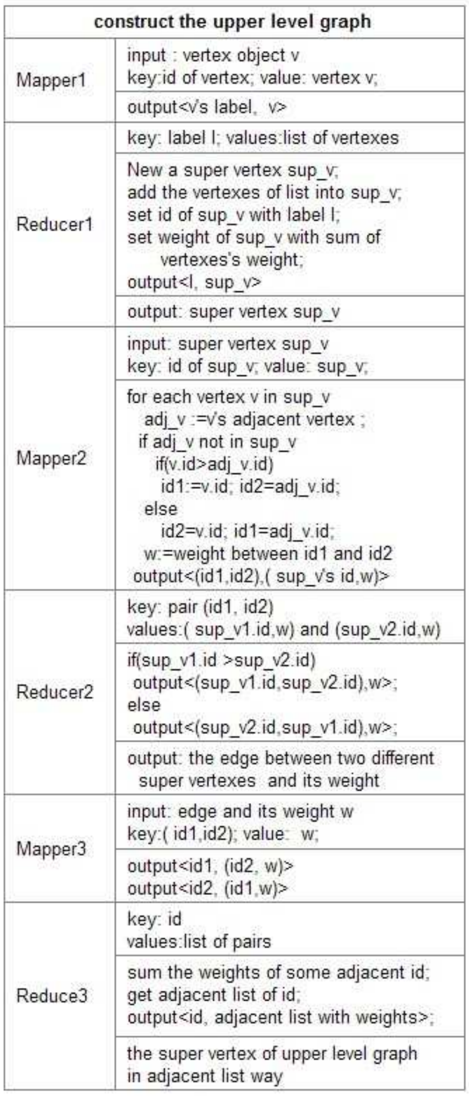
(3) For each vertex v computes its affinity to its adjacent vertexes $aff(v, u)$, new a Label with weight equals $\frac{aff(v,u)}{|v|}$, where the divisor $|v|$ is used to avoid excessively rapid growth of large-size vertex in upper level, and then send the label to the adjacent vertex.

(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).

4.3 Implementation on MapReduce

MapReduce is a parallel computing framework which is simple and easy to use [7]. The application, based on the Map-Reduce framework, can run on large-scale commercial clusters, process data in a parallel and fault-tolerant 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 overall execution can be simply described as the following streaming:



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 3: Construct the upper level graph.

Input | Mappers | Sort | Reducers | Output

The weighted label propagation algorithm is very easy to be implemented in a MapReduce way. The pseudo code in Map and Reduce is presented in Figure 2. 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 computes the membership degree of different labels and updates 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 3). 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.

4.4 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 $d_v = \sum_{i=v} W(i, j)$.

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

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

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

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

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

□

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 1. The parameter α is used to avoid

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 $V_i$.
 1: $V_i = \emptyset$;
 2: $V_i = V_i \cup v_0$;
 3: while $|V_i| < \frac{V}{k}$ do
 4:   $v = \text{argmin}_{v \in V_m - V_i} \frac{d_v - 2k_{v,V}}{|v|}$;
 5:   if $|V_i + |v| < \frac{(1+\alpha)|V|}{k}$ then
 6:     $V_i = V_i \cup v$;
 7:   else
 8:     if $|V_i + |v| > \frac{(1-\alpha)|V|}{k}$ then
 9:       Break;
10:     end if
11:   end if
12: end while
13: return $V_i$;
```

the case that 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 $setList$.
 1: $setList = \emptyset$;
 2: $bestSet = null$;
 3: for $i = 1$; $i < k$; $i++$ do
 4:   for $v$ in $V_m$ do
 5:     $S = minimizePRC(v, G_m, \frac{|V|}{k})$;
 6:     if $PRC(S) < PRC(bestSet)$ then
 7:       $bestSet = S$;
 8:     end if
 9:   end for
10:   $setList = setList \cup bestSet$
11:   $V_m = V_m \setminus bestSet$
12: end for
```

5. EXPERIMENTAL EVALUATION

In this section, we begin to evaluate the performance of our partitioning algorithm compared with other existing partitioning algorithms on various graphs. And the scalability of our partitioning algorithm is also be evaluated on different scale large graph. Finally, we evaluate our partitioning algorithm in a real cluster application.

5.1 Hardware Description

The cluster environment used to test our algorithm composed of one master node and 32 computing nodes (Intel Xeon 3.20GHz $\times$ 2, 2GB RAM, Linux RH4 OS) with 2T-B total storage capacity, and deployed a Hadoop platform(a wildly accepted open source implementation of MapReduce) and a Spark computing framework [27]. As a contrast, the running environment of stand-alone graph partitioning algorithm is: Intel Core2 Duo2.66GHz processor, with 2GB memory, using WindowsXP operating system.

5.2 Dataset Description

We evaluated the performance of our multi-level stepwise graph partition algorithm(**MSP**) on different graphs obtained from various sources. There are a total of 21 different networks: 9 systhetic graph datasets and 12 real-world graph datasets. The synthetic graph datasets are produced by popular generative models, preferential attachment (BA) [2], Watts-Strogatz [26] and a power-law graph generator with clustering [13]. The synthetic datasets based on BA, WS, and PL were created with the NetworkX python package [22]. For graphs BA1, BA2, and BA3, generated by barabasi-albert model, we choose parameters ($n = 10000, m = 3$), ($n = 100000, m = 5$) and ($n = 500000, m = 5$) respectively. And graphs PL1, PL2 and PL3, are generated by powerlaw-cluster model with the same parameters as barabasi-albert model. For graphs WS1, WS2, and WS3, generated by Watts-Strogatz model, we choose parameters ($n = 10000, m = 10, p = 0.1$), ($n = 100000, m = 10, p = 0.1$) and ($n = 500000, m = 10, p = 0.1$) respectively. The real-world graph datasets are mainly come from SNAP[17] and Graph Partitioning Archive [25]. The SNAP graphs used are: email-EuAll,email-Enron, amazon0312, amazon0302, web-NotreDame, web-Stanford, ca-CondMat , ca-help, LiveJournal and Twitter. The Partitioning Archive graphs used are: elt4, and cs4. The characteristics of these graphs are shown in Table 1.

5.3 Experimental Results

5.3.1 *Performance of partitioning algorithm*

We compared our partition algorithm with two existing partitioning algorithms: multi-level graph partitioning algorithm and spectral method. The multi-level graph partitioning algorithm(**MP**) is currently considered to be the state-of-the-art and are used extensively. The multi-level graph partitioning algorithm and the spectral partitioning algorithm are provided by Chaco [12]. Note that the bisections produced by spectral were further refined by using a KL refinement algorithm. Due to space constraints and serial execution of Chaco, the comparison between our partition algorithm and algorithms provided by Chaco are made on small graph datasets. However, the relative performance of our algorithm performed parallelly on a cluster of commodity machines remains the same for larger graph datasets. We ran each experiment on 4 partitions and fixed the imbalance

Table 1: Networks Basic Structural Properties.

Name	$\|V\|$	$\|E\|$	$\|C\|$	$Type$
BA1	10,000	29,992	0.0057	Synth.
BA2	100,000	499,975	0.0011	Synth.
BA3	500,000	2,499,975	0.0010	Synth.
WS1	10,000	55,011	0.5583	Synth.
WS2	100,000	549,735	0.5591	Synth.
WS3	500,000	2,750,988	0.5582	Synth.
PL1	10,000	29,990	0.0605	Synth.
PL2	100,000	499,965	0.0331	Synth.
PL3	500,000	2,499,961	0.0319	Synth.
email-EuAll	265,009	364,481	0.3093	Social
email-Enron	36,692	183831	0.4970	Social
amazon0312	400727	2349869	0.4113	Product
amazon0302	262,111	899,792	0.4240	Product
web-NotreDame	62,586	1,090,108	0.4540	Web
web-Stanford	281,903	1,992,636	0.6109	Web
ca-CondMat	23,133	93,439	0.6334	Citation
ca-help	9,875	25,973	0.4715	Citation
elt4	15,606	504,230	0.4076	FEM
cs4	14,010	16,373	0.0236	FEM
LiveJournal	$4.6 * 10^6$	$77.4 * 10^6$	0.330	Social
Twitter	$41.7 * 10^6$	$1.468 * 10^9$	0.1060	Social

such that no partition held more than 1% more vertices than its share by setting the parameter $\alpha = 0.01$.The fraction of edges cut obtained by the different partition algorithm across the different datasets are shown in Figure 4.

From the figure 4, we can see that all the partition algorithm obtain a larger fraction of edges cut on the graphs: BA1,BA2,BA3,PL1,PL2 and PL3 than other graphs. These graphs have a low average clustering coefficient and more than 35% of edges being cut. And our partition algorithm has little larger fraction of edges cut than the **MP** and spectral method. However in graphs with a high average clustering coefficient, such as WS1, email-EuAll and web-Stanford, our partition algorithm obtain a smaller fraction of edges cut in most cases. It can be explained by the following reasons: a high average clustering coefficient means the nodes in a graph tend to cluster together in which the partition algorithms are more likely to make a good partition. And the weighted label propagation of our algorithm has a good performance in aggregating the dense subgraph on these graphs, which leads to better partition.

In addition, we make a random partition over all the graphs, of which the average fraction of edges cut is 74.9%, much higher than above partitioning algorithms.

5.3.2 *Scalability of partitioning algorithm*

The graph datasets used in above performance comparison are tiny when compared with some of the graphs used in practice. While the above results are promising, it is important to understand whether our partitioning algorithm well scales with the size of the graph. As synthetic graphs produced by same generative model with similar parameter settings have similar graph statistics, we used the synthetic datasets in experiment in order to control for the variance in different graphs. We will present only the results for the Watts-Strogatz graphs, but all other graphs have similar results.We created 10 Watts-Strogatz graphs with a scale of 1 to 10 million vertices. For the Watts-Strogatz model, the probability of rewiring each edge is 0.1 and each node is connected to 10 nearest neighbors in ring topology. We will consider the following questions in the experiment: (1) whether our algorithm's partition performance on large graph is stable with increasing graph size. (2) how the partitioning quality scales with the number of partitions. (3) how the partition on MapReduce framework scales with graph size and computing nodes.

Firstly, the partition performance of 10 Watts-Strogatz graphs is presented on the Figure 5. The Figure shows that the fraction of edges cut well scale with the size of the graph holds - it is approximately 7% for each graph.

The next question is how the partitioning quality scales with the number of partitions. We only present the partition result of one graph in Figure 6, the 5 million vertex Watts-Strogatz graph, but all graphs have similar characteristics. As the figure show, the fraction of edges cut must necessarily increase as we increase the number of partitions. Our partitioning algorithm performs better than other partitioning algorithms when the partition number is small and obtains a approximate performance when the partition number is large.

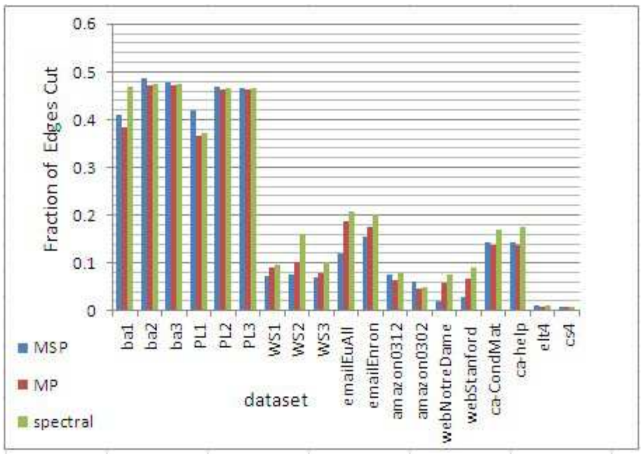


Figure 4: The fraction of edges cut of the different partition algorithm.

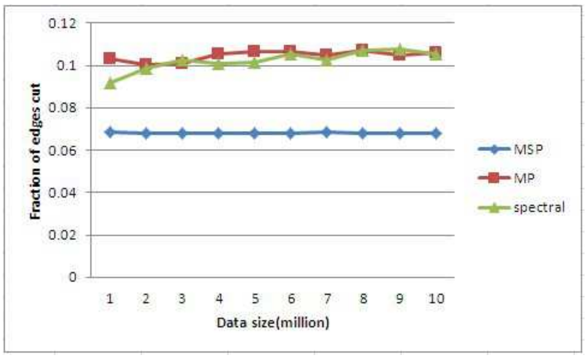


Figure 5: The fraction of edges cut of 10 Watts-Strogatz graphs using different algorithms.

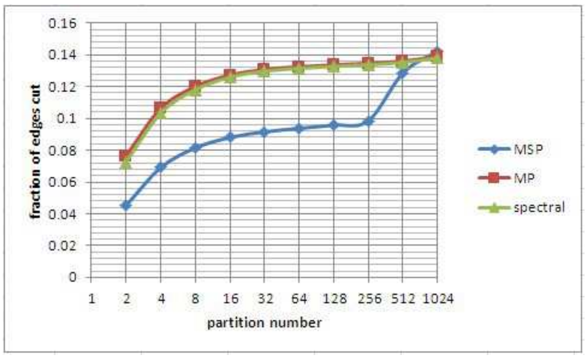


Figure 6: The fraction of edges cut of the Watts-Strogatz graph with different partition number.

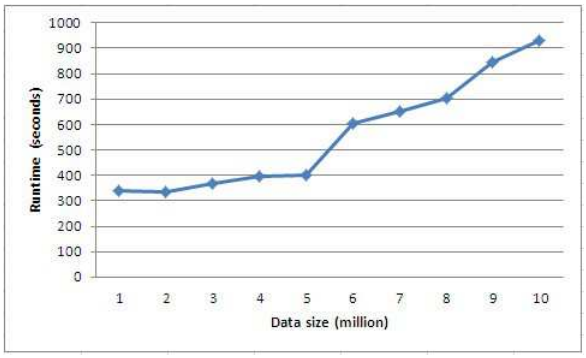


Figure 7: Runtime of 10 Watts-Strogatz graphs with 32 reducers.

To show how our partitioning algorithm scales with graph size in MapReduce, Figure 7 presents runtimes of our partitioning algorithm for above 10 Watts-Strogatz graphs on the MapReduce framework, using a fixed reducer number 32. As an indication of how our algorithm scales with computing nodes, Figure 8 shows runtimes for a Watts-Strogatz graph with 5 million vertices when the number of reducer varies from 2 to 64.

5.3.3 Results on a real system

In this section, we show whether a better graph partitioning will improve the performance of graph mining in real computation systems. To evaluate our partitioning algorithm in a real cluster application, we employed an implementation of PageRank in Spark, a cluster computing framework developed by UC Berkeley [27]. Spark can create distributed datasets from any file stored in the Hadoop distributed file system (HDFS), so the Spark easily obtains the partition results of our partitioning algorithm using Hadoop's MapReduce framwork. Besides, Spark keeps the working set of the application in memory across iterations, so that the algorithm is primarily limited by the cost of communication between nodes. Other distributed graph processing frameworks, such as Pregel [19] and GraphLab [18], also keep data in memory and have similar performance characteristics.

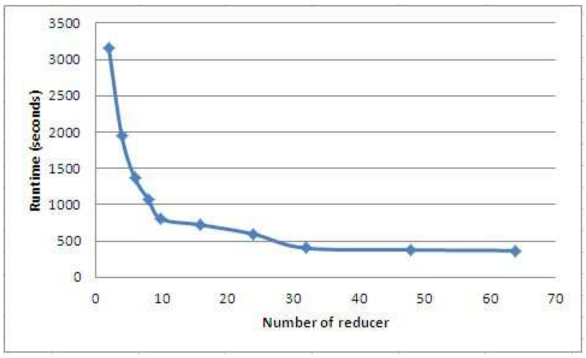


Figure 8: Runtime of algorithm with different reducer number.

Table 2: The averaged runtime of a PageRank iteration with different partitioning approach.

Time (seconds)	Hashing	MSP
LiveJournal	59.1	26.3
Twitter	503.5	294.6

We used two datasets previously mentioned in our experiment, one is LiveJournal with 4.6 million vertexes and 77.4 million edges, and the other is the Twitter graph with 41.7 million vertexes and 1.468 billion edges. This two datasets were partitioned into 100 pieces with imbalance of at most 2% by our partitioning algorithm and Hashing approach, a popular method currently used by many real systems [19]. In most cases, Hashing equivalents to a random partitioning. For LiveJournal, our partitioning algorithm reduced the number of edges cut to 27,401,144 edges compared with 70,224,763 for the Hashing partitioning. For twitter, our partitioning algorithm cut 0.581 billion edges, while the Hashing partitioning cut 1.263 billion. We ran 10 iterations of PageRank of above two datasets, and repeated this experiment 5 times. The average runtime of a iteration of PageRank on two datassets with two different partitioning approach is shown on the Table 2. The result shows that using our graph partitioning algorithm as a preprocessing step for the large-scale distributed graph mining can yield a large improvement in the running time.

6. CONCLUSIONS

In this paper, we proposed a multi-level stepwise partition algorithm that drastically reduces the number of edges cut in distributed graph data. The algorithm first uses a weighted label propagation aggregate the large original graph to a small upper level graph iteratively. And then the k-way balance partition is obtained by a novel method based on stepwise minimizing RatioCut. We conduct various experiments to evaluate our graph partitioning algorithm on Hadoop's MapReduce with different large datasets and make a performance comparison with the existing partitioning algorithm. The results show our partitioning algorithm has a excellent performance and scalability and can largely improve the efficiency of graph mining on real distributed computing system. For future work, we plan to further investigate the interplay between the partition performance and various graph structural properties and their effect on partition.

7. ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China (Grant No.60905025, 90924029, 61074128).

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] A. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509, 1999.

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

[4] J. C.Chu, J.A.George and E.G.Ng. User'sguide for sparspak-a: Waterloo sparse linear equations package. Technical report, Computer Science, University of Waterlo o,Ontario., 1984.

[5] 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.

[6] W. D. and W. F. Between mincut and graph bisection. In *In Proceedings of the 18th International Symposium on Mathematical Foundations of Computer science (MFCS)*, pages 744 – 750, 1993.

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

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

[9] C. Fiduccia and R. Mattheyses. A linear-time heuristic for improving network partitions. In *Design Automation, 1982. 19th Conference on*, pages 175 – 181, june 1982.

[10] 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.

[11] L. W. Hagen and A. B. Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE Trans. on CAD of Integrated Circuits and Systems*, pages 1074–1085, 1992.

[12] B. Hendrickson and R. Leland. The chaco user's guide, version 2.0. (SAND95-2344), 1994.

[13] P. Holme and B. J. Kim. Growing scale-free networks with tunable clustering. *Physical Review E*, 65(2):026107, 2002.

[14] 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.

[15] G. Karypis and V. Kumar. Parallel multilevel k-way partitioning scheme for irregular graphs. In *Supercomputing, 1996. Proceedings of the 1996 ACM/IEEE Conference on*, page 35. TeX Users Group, March 1996.

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

[17] J. Leskovec. Snap, 2011. http://snap.stanford.edu/snap.

[18] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein. Graphlab: A new framework for parallel machine learning. In P. Grĺźnwald and P. Spirtes, editors, *UAI*, pages 340–349. AUAI Press, 2010.

[19] 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.

[20] G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis. Automatic mesh partitioning. In A. George, J. Gilbert, and J. Liu, editors, *Graphs Theory and Sparse Matrix Computation*, The IMA Volumes in Mathematics and its Application, pages 57–84. Springer-Verlag, 1993. Vol 56.

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

[22] A. H. D. Schult and P. Swart. Networkx. http://networkx.lanl.gov/.

[23] I. Stanton and G. Kliot. Streaming graph partitioning for large distributed graphs, 2011. http://research.microsoft.com/apps/pubs/default.aspx?id=1559

[24] X. Sui, D. Nguyen, M. Burtscher, and K. Pingali. Parallel graph partitioning on multicore architectures. In *LCPC'10*, pages 246–260, 2010.

[25] C. Walshaw. Graph partitioning archive. http://staffweb.cms.gre.ac.uk/ wc06/partition.

[26] D. J. Watts and S. H. Strogatz. Collective dynamics of small-world networks. *Nature*, 393(6684):440–442, June 1998.

[27] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, HotCloud'10, page 10, Berkeley, CA, USA, 2010.