Betweenness-based algorithm for a partition scale-free graph*

Zhang Bai-Da(张百达)[a)†], Wu Jun-Jie(吴俊杰)[a)], Tang Yu-Hua(唐玉华)[b)], and Zhou Jing(周 静)[a)]

[a)] *National Laboratory for Parallel and Distributed Processing, School of Computers, National University of Defense Technology, Changsha* 410073, *China*

[b)] *Department of Computer Science and Technology, School of Computers, National University of Defense Technology, Changsha* 410073, *China*

(Received 30 March 2011; revised manuscript received 17 August 2011)

Many real-world networks are found to be scale-free. However, graph partition technology, as a technology capable of parallel computing, performs poorly when scale-free graphs are provided. The reason for this is that traditional partitioning algorithms are designed for random networks and regular networks, rather than for scale-free networks. Multilevel graph-partitioning algorithms are currently considered to be the state of the art and are used extensively. In this paper, we analyse the reasons why traditional multilevel graph-partitioning algorithms perform poorly and present a new multilevel graph-partitioning paradigm, top down partitioning, which derives its name from the comparison with the traditional bottom–up partitioning. A new multilevel partitioning algorithm, named betweenness-based partitioning algorithm, is also presented as an implementation of top–down partitioning paradigm. An experimental evaluation of seven different real-world scale-free networks shows that the betweenness-based partitioning algorithm significantly outperforms the existing state-of-the-art approaches.

Keywords: graph partitioning, betweenness-based partitioning algorithm, scale free network

PACS: 89.75.Hc, 87.23.Ge, 89.20.Hh, 89.75.–k **DOI:** 10.1088/1674-1056/20/11/118903

1. Introduction

Recently, the computerization of data acquisition in all fields has led to the emergence of large databases on the topology of various real networks.[1,2] For example, there are more than 10^{10} vertices in human neural networks[3] and over 500000 vertices in actor cooperation networks. The problems addressed using these networks are so huge that they can be resolved only by parallel computing technology. Graph-partitioning technology, as a technology capable of parallel processing,[4] is used to decompose such computations. However, the graph-partitioning problem is NP-complete (nondeterministic polynomial time-complete),[5] that is, there is no polynomial time algorithm until now. Recently, a number of researchers have investigated a class of algorithms based on multilevel graph partitioning that has moderate computational complexity. Moreover, several toolboxes have been implemented, for example, Metis,[6] Chaco,[7] Scotch[8] and JOSTLE.[9]

Unfortunately, many real-world networks are found to be scale-free,[10] and the results achieved by traditional partitioning algorithms cannot satisfy users' needs. There are several reasons for this. First, the matching is based on local information and a globally optimal result may be missed; second, the matching process is inefficient when some high-degree vertices appear;[11] and third, a great deal of memory space is used to store graphs for all coarsening levels. The details can be found in Section 3.

Another important feature of real networks is the community structure, which is the organization of vertices in clusters, with many edges joining vertices of the same cluster and comparatively few edges joining vertices of different clusters.[12] Vertices in the same community often share common properties or play similar roles within the graph. Community detection is the technology extracting community structure from networks, i.e., dividing the network into sev-

*Project supported by the National Science Foundation for Distinguished Young Scholars of China (Grant Nos. 61003082 and 60903059), the National Natural Science Foundation of China (Grant No. 60873014) and the Foundation for Innovative Research Groups of the National Natural Science Foundation of China (Grant No. 60921062).

†Corresponding author. E-mail: zhangbaida@gmail.com

© 2011 Chinese Physical Society and IOP Publishing Ltd

eral communities.[13,14] They are very similar to partitioning algorithms. A solution to the community detecting problem is, however, not particularly helpful for partitioning networks in general. Because if our goal is merely to break a given network into communities, we probably do not know how many such communities exist and the communities are not required to have roughly the same size. Furthermore, the number of intercommunity edges need not be strictly minimized either, since more such edges are admissible between large communities than between small ones.[2] Whereas partitioning algorithm is used to partition a graph into p part, it balances the number of vertices assigned to each partition and minimizes the number of edges that straddle partition boundaries, where p is the number of processors, the vertices represent computations and the edges indicate data dependencies.[11]

In this paper, a new paradigm of the multilevel graph-partitioning algorithm is presented to partition scale-free graphs. In addition, betweenness-based partitioning algorithm, which is based on a top–down paradigm and community detection technology,[2,12] is presented. Our research primarily focuses on how to obtain a result with fewer edge cuts and how to save memory space used by the multilevel paradigm. We test the betweenness-based partitioning algorithm on seven different real-world networks and compare their performances with those of traditional multilevel partitioning algorithms. The results show that the proposed algorithm significantly outperforms the existing approaches. To summarize, the contributions of this paper are as follows: 1) a classification of the coarsening algorithms is presented; 2) a classification of the multilevel graph-partitioning algorithm based on 1) is presented; 3) a new betweenness-based partitioning algorithm is presented and evaluated.

The rest of this paper is organized as follows. In Section 2, we provide background material and some key concepts that are used throughout the paper. Some illustrative examples are provided in Section 3. Section 4 is devoted to a classification of multilevel graph partitioning. A new multilevel graph-partitioning algorithm is presented in Section 5 and an analysis of its complexity is provided in Section 6. A performance comparison based on seven scale-free networks is presented in Section 7. In Section 8, we discuss the related work. Finally, we draw conclusions and discuss the future work in Section 9.

2. Background

2.1. Scale-free network and disassortativity

The empirical results of many studies demonstrate that many large networks are scale-free,[2,15] i.e., their degree distribution follows power laws in their tails: $p(k) \sim k^{-\alpha}$, where α is a parameter of the distribution.[1,16] Therefore, highly connected vertices are statistically significant in scale-free networks. Moreover, in Ref. [17], it was affirmed that most technological and biological networks are disassortative, i.e., a preference for high-degree vertices attached to low-degree vertices exists.

2.2. Betweenness and community detection

Edge betweenness is the number of shortest paths between all vertex pairs that run along the edges.[12] It is closely related to cascading and traffic dynamics[18] and optimal spreading dynamics.[19] And it is a very important metric in many real applications.[20]

In the network glossary, communities are subsets of vertices within which connections are dense, while between which connections are sparse.[12,21] Community detection is the technology used to find vertex groups like this. Hierarchical clustering has been frequently used to discover communities, based on various metrics of similarity or strength of connection among vertices. And betweenness is one of those metrics. If two communities are joined by only a few intercommunity edges, then all paths through the network from vertices in one community to vertices in the other must pass along one of those few edges. Given a graph, one can count the number of shortest paths that go along each edge in the graph. This number for intercommunity edges will be the largest one. Therefore, betweenness provides a method of identifying intercommunity edges.[20,22] Figure 1 shows a graph with edge-betweenness labeled beside the edge.

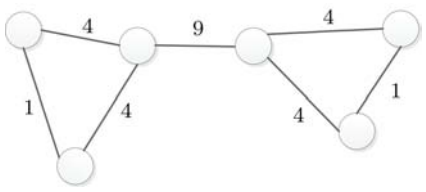


Fig. 1. Betweenness computing.

2.3. Paradigm of multilevel partitioning

Graph partitioning is an NP-complete problem;[5] therefore, it is impossible to find the optimal solution in polynomial time. A number of high-quality and computationally efficient algorithms have been presented.[23] Among them, multilevel graph-partitioning algorithms are considered to achieve good performance.[24−26] There are three stages in traditional multilevel graph-partitioning algorithms.[25,27,28] First, a sequence of smaller and smaller graphs is created from the original graph, a process called coarsening. Second, the smallest graph in the sequence is partitioned carefully. Third, the partitioning is propagated back through the sequence of graphs, with an occasional local refinement; this is called uncoarsening and refining.[6,26,29−31] Maximal matches can be computed in different ways.[26] The method used to compute the matching greatly affects both the quality of the bisection and the time required for the uncoarsening phase. Here, two such matching schemes are described. The first scheme, named random matching (RM), computes the maximal matching with a randomized algorithm. The RM scheme works as follows. The vertices of the graph are visited in a random order. If a vertex u has not yet been matched, then an unmatched adjacent vertex v is randomly selected and the edge (u, v) is included in the matching. The second scheme, called heavy-edge matching (HEM), computes a matching such that the weight of the edges in the match is high. The HEM matching is computed with a randomized algorithm similar to the one used for RM and vertices are visited in a random order. However, instead of randomly matching a vertex with one of its adjacent unmatched vertices, HEM matches it with the unmatched vertex that is connected with the heaviest edge. As a result, the HEM scheme reduces the sum of the weights of the edges in the coarser graph by a larger amount than RM does.

3. Motivations

3.1. Local optimum

Because only local information is used, traditional multilevel-partitioning algorithms may hide some edges that should be in the edge-cut set of the optimal partitioning. For example, Fig. 2(a) shows the original graph in which each vertex is labeled by a single letter and the number beside each edge is the edge weight. When HEM matching is used, edge (c, d) is selected, and vertices c and d will be collapsed together, as Fig. 2(b) shows. The result is that edge (c, d) is impossible to be selected as an inter-partition edge in the initial partitioning. The solution runs into a local optimum. However, if global information is used, then vertices a, b and c are collapsed together and d, e and f are collapsed together. As Fig. 2(c) shows, edge (c, d) will be easily selected to be the cut. Therefore, the best coarsening process should utilize global information and preserve only the information that is critical for graph partitioning.

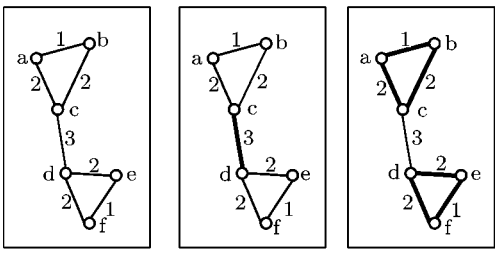


Fig. 2. Problem of matching: (a) original graph, (b) HEM matching, (c) desired result.

3.2. High memory cost

In traditional multilevel matching algorithms, some functions are used to evaluate the degree of vertices belonging to the same partition. However, when scale-free graphs are provided, those functions perform poorly. This is because only a few vertices can collapse during each coarsening process when there are a few high-degree vertices with many neighbouring low-degree vertices. For example, in Fig. 3, the degrees of all vertices are comparable and the number of vertices halves every time the coarsening process is executed. In Fig. 4, however, there are two vertices, $a1$ and $b1$, with higher degree than others. In the coarsening process, if $a1$ and $b1$ are selected, all other vertices lose the chance for contraction and this results in the fact that the speed for reducing the scale of the graph is slowed down. Because all of those graphs are used in the uncoarsening process, they need to be stored temporarily under a traditional paradigm. This will use several times more memory space than needed. As shown in Fig. 5, the x-coordinate is the coarsening level and the y-coordinate is the total memory space needed by Metis Toolbox, which reads the number of times the original graph is stored in the memory space. It is amazing that 9 times and 7 times more memory space is needed by the GOOGLE graph and ACTOR graph, respectively (these two graphs will be

explained in Section 7). Therefore, if the serial graphs do not need to be stored, the memory space required will be dramatically reduced.

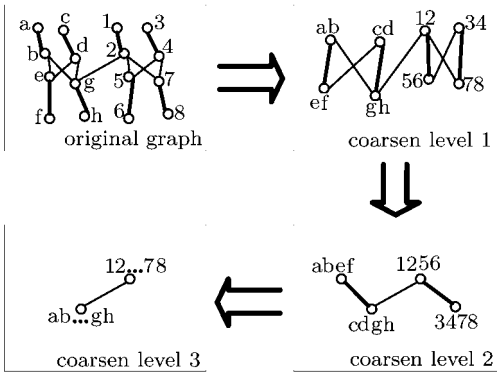


Fig. 3. Match performs well.

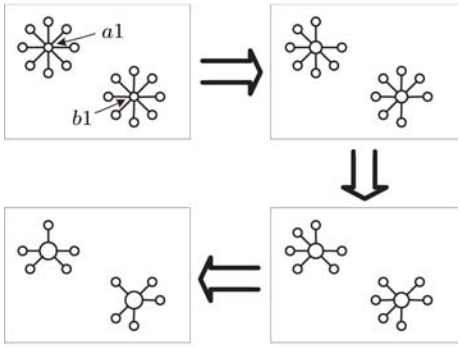


Fig. 4. Match performs badly.

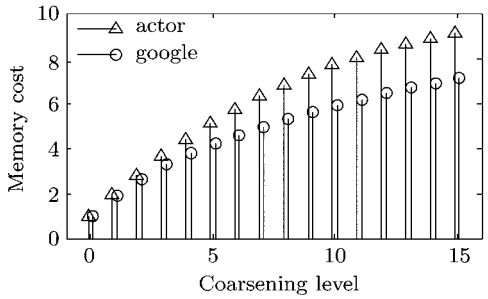


Fig. 5. Memory cost of the GOOGLE graph and ACTOR graph.

4. Classification of multilevel partitioning algorithms

The definition of contraction is provided first. Supposing that $G = (V, E)$ is a graph, E_1 and E_2 are subsets of E, $E_2 = E - E_1$ and a contraction of G from E_1 is graph, denoted by $G_C = (V_C, E_C)$, where each vertex in V_C corresponds to a component of graph G connected by edges in E_1. If v_{C_1} and v_{C_2} are the vertices of graph G_C, they are obtained from the components of C_1 and C_2 of graph G respectively. Then $e_C = (v_{C_1}, v_{C_2}) \in E_C \Leftrightarrow \exists v_1 \in C_1$, $\exists v_2 \in C_2$ and $e = (v_1, v_2)$ is a edge of graph G. If G_β is a contraction of G_α and G_γ is a contraction of G_β, then it is easy to obtain that G_γ is a contraction of G_β. In multilevel partitioning algorithms, the input of the coarsening process is a graph G_0 and the output is a series of graphs $G_1, G_2, \ldots, G_n$, in which G_{i+1} is a contraction of G_i from E_i, for $i = 0, 1, \ldots, n-1$. This problem can be described from another point of view by considering the output of the coarsening process to be a series of edge sets, $E_0, E_1, \ldots, E_{n-1}$, in which E_i is a subset of E_{i+1} and E_{n-1} is a subset of E. Hierarchical clustering techniques aim to identify groups of vertices with high similarity and can be classified into two categories: 1) agglomerative algorithms, in which clusters are iteratively merged if their similarity is sufficiently high; 2) divisive algorithms, in which clusters are iteratively split by removing edges connecting vertices with low similarity.

4.1. Classification of coarsening process

Based on the sequence of the edge set E_i, the coarsening process can be classified as follows.

4.1.1. Bottom–up coarsening

If E_i is achieved earlier than E_{i+1}, the coarsening process will be called bottom–up coarsening. Because in this method the original graph is regarded as a separate vertices set and a combination of methodologies is used, as shown in Fig. 6. Traditional matching algorithms, i.e., HEM[32] and RM,[32] belong to this kind.

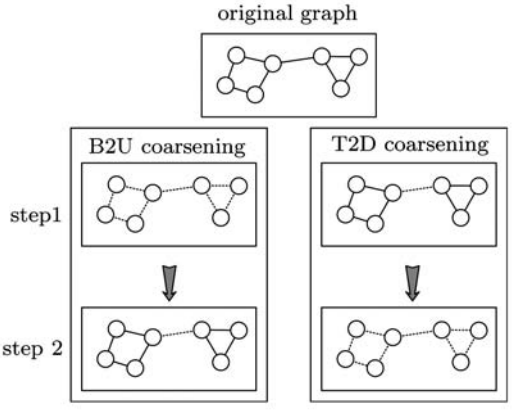


Fig. 6. Two types of coarsening.

4.1.2. Top–down coarsening

If E_i is obtained later than E_{i+1}, the coarsening process will be called top–down coarsening. Unlike the bottom–up coarsening, this is a decreasing method; thus, the original graph is regarded as a single large vertex and more vertices are gained by dividing this large one as shown in Fig. 6. The betweenness-based partitioning algorithm presented in the next section belongs to this kind.

4.2. Classification of partitioning algorithm

Based on the classification of the coarsening process as described above, multilevel partitioning algorithms can be classified as bottom–up multilevel partitioning and top–down multilevel partitioning algorithms.

4.2.1. Bottom–up multilevel partitioning

In the bottom–up multilevel partitioning paradigm, the partitioning algorithm is executed as follows: coarsening, initial partitioning, uncoarsening and refining. As shown in Fig. 7(a), arrows indicate the sequence of the program executed and each solid rectangle represents a vertex in each subgraph. A sequence of successive approximations of the original graph is obtained. Each one of these approximations represents a smaller problem than the original problem. This process continues until a level of approximation is reached when the graph contains only a few dozen vertices. Because the size of this graph is quite small, even simple algorithms lead to reasonably good solutions. The paradigm of the Metis Toolbox[6] belongs to this kind.

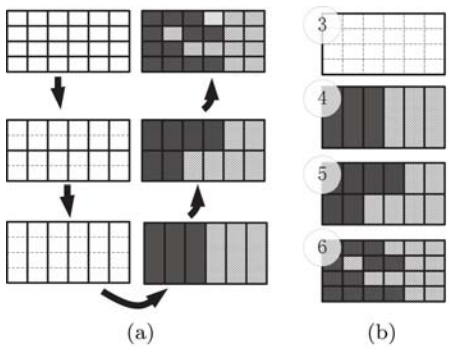


Fig. 7. Two types of partitioning: (a) B2U partitioning, (b) T2D partitioning.

4.2.2. Top–down multilevel partitioning

In the top–down multilevel partitioning paradigm, coarsening processes do not exist and a graph is initially treated as a single vertex. After initial division, a few dozen components are obtained, and an initial partition is computed by taking a component as a vertex; division and refining processes will be executed alternately. The total process is shown in Fig. 7(b). Compared with in the case of bottom–up multilevel partitioning, there is no need to store a series of contraction graphs. Therefore, the memory space can be reduced significantly. The betweenness-based partitioning algorithm presented in the next section belongs to this kind.

5. Betweenness-based partitioning algorithm

To overcome the limitations presented in Section 3, a method that is good at finding a set of closely connected vertices is needed. Community detection is this very method. However, some features are required. 1) Closely connected vertices can be identified in different levels because multilevel partitioning requires several graphs that describe different details of the original graph. 2) The process of community detection is divided and not agglomerative for the sake of generating a graph only when it is needed. The community detection algorithm by Newman and Girvan[20] satisfies these requirements. A new betweenness-based algorithm for partitioning scale-free graph is presented in Algorithm 1 as follows.

Algorithm 1 betweenness-based partitioning algorithm description

```
Data: Graph G, int part
Result: vector partition
N = num of vertices of G;
size = N/part/4;
G1 = contraction (G, size);
partition = initpartition (G1) ;
While size > 1 do
        size = size/2;
        G2 = contraction(G,size);
        partition = projection(G1, G2,
                    partition);
        partition = refine(G1, G2, partition);
end
```

The contraction algorithm of Algorithm 1 is shown in Algorithm 2.

Algorithm 2 Contraction

Data: Graph G, int maxsize

Result: Graph G1

While maximum component size of G less than maxsize **do**
cmpnt = getmaxcomponent of G;
compute betweenness of all edges in cmpnt;
While cmpnt is connective **do**
delete the edge of cmpnt with max betweenness;
end
end
G1 = contraction graph of G from deleted edges set

First, a graph is given and k-way partition is required. Then a contraction $G1$ is generated and $G1$ is small enough to be provided as a better partition. There are three things considered in betweenness-based partitioning algorithm.

(i) The ability to hierarchically identify sets of well-connected vertices. A simple way to identify communities in a graph is to detect the edges that connect vertices of different communities and remove them so that the clusters gradually become disconnected from each other. This is the philosophy of divisive algorithms. The key is to find a feature that can order all edges. In the betweenness-based partitioning algorithm, the edge betweenness is used to order edges. Edge betweenness is the number of shortest paths between all vertex pairs that run along the edge. It is intuitive that inter-cluster edges each have a large value of edge betweenness because the shortest paths connecting vertices of different clusters will be likely to pass through those edges.

(ii) It is forbidden to generate a vertex whose weight is greater than those of other vertices in a contraction graph. This is avoided by selecting the maximum component of graph G in the contraction process. Once a component is selected, it will be divided into two smaller components. This mechanism restricts the generation of very large vertices, which dramatically reduces the performance of the refining process.

(iii) The number of vertices in successive contraction graphs should be balanced. If many vertices are collapsed in a single level, the refining process may not work well because the KL or FM algorithm can only slightly improve the given partitioning result. If few vertices are collapsed together in a single level, too many level contractions are needed and the run time will increase dramatically. This goal is achieved by the parameter scale, which controls the size of the maximum vertices of successive contraction graphs.

An example of betweenness-based partitioning is provided in Fig. 8. The original graph contains 8 vertices and 2-way partition is required. In the first level, according to the betweenness of all edges, edges e and g are removed, and the graph is divided into two communities: com_1 and com_2. The initial partition is shown, com_1 in $\mathrm{partition}_1$ and com_2 in $\mathrm{partition}_2$. Next, com_1, as the largest community, is selected for further division and the communities com_{1-1} and com_{1-2} are produced. Projecting process in Algorithm-2 places com_{1-1} and com_{1-2} into $\mathrm{partition}_1$. Then the refining process is executed and com_{1-1} is placed into $\mathrm{partition}_2$. Dividing, projecting and refining are executed repeatedly until all edges are removed.

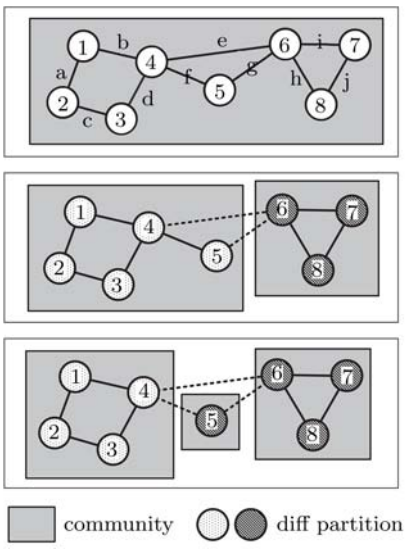


Fig. 8. An example of betweenness-based partitioning.

6. Analysis of betweenness-based partitioning

6.1. Time complexity

Suppose that $G = (V, E)$ is a graph, V is a set of vertices which contains N vertices and E is a set of edges which contains M edges. It has been proved that the time complexity of computing the betweenness of

all edges is $O(N^3)$ and obtaining all of the N^2 shortest paths needs $O(N^3)$ times computing. The time complexity of computing the connectivity of a graph is $O(N^2)$. The amount of time spent on the initial partitioning process is considered to be constant because the process is executed on a small graph whose size is proportional to the number of partitions compared with the original graph. An partition on original graph is obtained by initial-partitioning process. The main loop containing the contraction, the projection and the refinement processes is executed at most $\mathrm{lb}N$ times. Because every time contraction is executed, the largest component is reduced by β times. Supposing that the original size of the graph is N, the size of all components are reduced to 1 at most $\mathrm{lb}N$ times of execution of the main loop. The time complexity of the projecting process is $O(N)$, where N is the size of the output graph. The time complexity of the refining process is at most $O(N^2)$. The main parts of contraction are betweenness computing and connectivity confirming. Every time an edge is deleted, the connectivity is computed once more. Betweenness is computed once when the contraction process is called. It can be concluded that the total time complexity of betweenness-based partitioning algorithm is $O(\mathrm{lb}N(N^3+N^2+N)+N^2E) = O(N^3\mathrm{lb}N+EN^2)$. This is an upper limit and more accurate results are not provided here due to space constraints.

6.2. Space complexity

The space complexity of the traditional multilevel-partitioning algorithm is related to the level of coarsening. If the number of edge is reduced by a ratio α, the total memory cost can be expressed as

$$\begin{aligned} & K(E+E/\alpha+E/\alpha^2+\cdots) \\ &= KE/(1-1/\alpha), 1<\alpha<=2, \end{aligned} \tag{1}$$

where K represents the memory cost of an edge.

In regular networks, α nears 2 and the total memory space is about $2KE$. While in scale-free networks, α is close to 1, and the memory cost will drastically increase. This is what happens in scale-free networks. Because no additional coarsening graph is stored, the memory cost of betweenness-based partitioning algorithm is $O(E)$, no matter what kind of graph is given. This is a very good attribute, especially when a graph with a great many vertices is provided.

As shown in the analysis above, the betweenness-based partitioning algorithm possesses polynomial time complexity and is easy to implement.

7. Experimental results

To evaluate the performance of betweenness-based partitioning algorithm, seven benchmarks are selected, and detailed information is listed in Table 1. To facilitate the comparison, Reference [11] is referenced for the selection of the benchmarks. In Fig. 9, the memory cost is shown.

Table 1. Information of seven benchmarks.

Data set	Vertex No.	Edge No.	Graph source
Citations	27400	352504	www.cs.cornell.edu/projects/kddcup/datasets.html
DBLP	310138	1024262	www.informatik.uni-trier.de/ ley/db/index.html
Actor	498925	1460791	Internet Movie Database
+ Google	198782	295063	www.google.com/programming-contest/
PPI	59191	160737	dip.doe-mbi.ucla.edu
Scan	228263	320149	www.isi.edu/div7/scan/mercator/maps.html
Lucent	112969	181639	www.isi.edu/div7/scan/mercator/maps.html

Table 2. Edge cut of 7 scale-free networks.

Algorithm	CITATIONS	DBLP	ACTOR	GOOGLE	PPI	SCAN	LUCENT
BBP	14414	47967	38762	6359	8563	4887	2358
FCDC	15868	52405	42404	6867	8926	4560	2387
Metis	14925	61242	113962	14758	12853	11575	4323
Spectral	25765	72450	78342	7144	7469	4950	3874

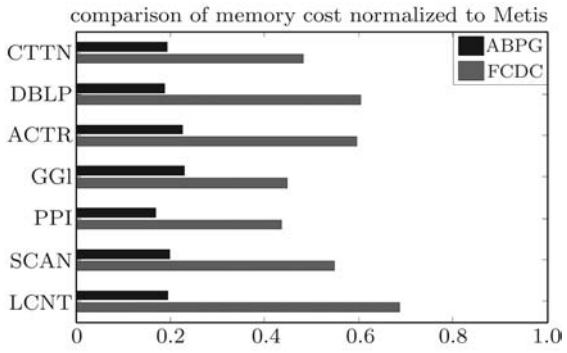


Fig. 9. Memory compare cost.

The Metis Toolbox and the FCDC algorithm[11] are selected for comparison. For convenience, the memory cost of the Metis Toolbox is defined as the baseline and the value shown in Fig. 9 is relative to the cost of the Metis Toolbox. Compared with that in the Metis Toolbox, in the betweenness-based partitioning algorithm only 20.3% of the memory space is used on average. In Table 2, the edge cuts of four different algorithms are listed, i.e., betweenness-based partitioning (BBP for short), FCDC,[11] Metis,[6] and Spectral.[31] The best results are underlined. Table 2 shows that betweenness-based partitioning algorithm outperforms the other 3 algorithms with respect to 6 benchmarks out of 7.

8. Related work

Graph partitioning is an NP-complete problem,[5] and the optimal solution cannot be determined when the size of the problem is excessively large. However, many algorithms have been developed which can perform partitioning reasonably well.[23] A multilevel partitioning algorithm has recently emerged as a highly effective method for computing a k-way partitioning of a graph.[11,31] Unlike the betweenness-based partitioning algorithm, they are bottom–up partitioning algorithms. Community detection was recently used for partitioning in the Commpar algorithm,[33] but it was only a single-level partition. Duch and Arenas[13] presented a top–down community detection algorithm and Zou *et al.*[14] presented a community detection algorithm with consideration of non-topological information. Unlike the betweenness-based partitioning algorithm dividing graphs according to the partition number given by users, they divided graphs according to the intrinsic feature of the graph, i.e. community structure.

9. Conclusion

This paper focuses on improving the performance of multilevel partitioning algorithms, particularly when scale-free graphs are provided. Top–down partitioning is presented as a new partitioning paradigm. A new betweenness-based algorithm under this paradigm is also presented; it is specifically designed to partition scale-free graphs.

There are two central features that distinguish the betweenness-based partitioning algorithm from preceding ones. First, the betweenness-based partitioning algorithm is divisive rather than agglomerative. Divisive multilevel algorithms have not been studied in the past. Second, a community detection algorithm is used to identify closely connected vertices. Betweenness-based partitioning algorithm finds edges with the highest betweenness, where betweenness is a metric that favours edges that lie between partitions and disfavours edges that lie inside partitions:

Through analysis and experimentation, the following conclusions can be made:

1) Top–down multi-level partitioning, which avoids the storage of the coarsening graph, can greatly reduce the memory cost of traditional algorithm. Therefore, more complex problems can be solved;

2) Because it uses global information, the betweenness-based partitioning algorithm can find better solutions.

In the future, a parallel simulation of scale-free networks will be performed to verify the advantages of the betweenness-based partitioning algorithm.

References

[1] Albert R and Barabási A 2002 *Rev. Modern Phys.* **74** 47
[2] Newman M 2003 *SIAM Rev.* **45** 167
[3] Goodman D and Brette R 2008 *Frontiers in Neuroinformatics* **2** 134
[4] Hendrickson B and Kolda T 2000 *Parallel Computing* **26** 1519
[5] Garey M and Johnson L 1976 *Theoretical Computer Science* **1** 237
[6] Karypis G and Kumar V 1995 *The University of Minnesota* **2** 11
[7] Hendrickson B and Leland R 1994 *The Chaco User's Guide–Version 2.0* (Technical report) (Sandia National Laboratories)
[8] Pellegrini F and Roman J 1996 *Proceedings of HPCN'96*, Brussels, April 1996 p. 493
[9] Walshaw C and Cross M 2007 *Mesh Partitioning Techniques and Domain Decomposition Techniques* **21** 27
[10] Barabási A and Albert R 1999 *Science* **286** 509
[11] Abou-Rjeili A and Karypis G 2006 *Parallel and Distributed Processing Symposium* Newyork, April 2006 10

[12] Fortunato S 2010 *Phys. Reports* **486** 75
[13] Duch J and Arenas A 2005 *Phys. Rev.* E **72** 027104
[14] Zou S R, Peng Y J, Liu A F, Xu X L and He D R 2011 *Chin. Phys.* B **20** 018902
[15] Zhang L M, Deng X H, Yu J P and Wu X S 2011 *Chin. Phys.* B **20** 048902
[16] Tian L, Di Z R and Yao H 2011 *Acta Phys. Sin.* **60** 02120
[17] Newman M 2002 *Phys. Rev. Lett.* **89** 208701
[18] Yang R, Wang W X, Lai Y C and Chen G R 2009 *Phys. Rev.* E **79** 026112
[19] Yang R, Zhou T, Xie Y B, Lai Y C and Wang B H 2008 *Phys. Rev.* E **78** 066109
[20] Newman M and Girvan M 2004 *Phys. Rev.* E **69** 26113
[21] Shen Y 2011 *Chin. Phys.* B **20** 040511
[22] Cui A X, Fu Y S, Shang M S, Chen D B and Zhou T 2011 *Acta Phys. Sin.* **60** 803 (in Chinese)
[23] Schloegel K, Karypis G and Kumar V 2003 *Sourcebook of Parallel Computing* (Morgan Kaufmann Publishers Inc.) p. 491
[24] Karypis G and Kumar V 1995 *Proceedings of the International Conference on Parallel Processing* New York, p. 135
[25] Barnard S and Simon H 1994 *Concurrency: Practice and Experience* **6** 101
[26] Karypis G and Kumar V 1999 *SIAM Journal on Scientific Computing* **20** 359
[27] Hendrickson B and Leland R 1995 *SIAM Journal on Scientific Computing* **16** 452
[28] Hendrickson B and Leland R (US Patent) 5587922[1996]
[29] Kernighan B and Lin S 1970 *Bell System Technical Journal* **49** 291
[30] Mattheyses C and Fiduccia C 1982 *Proceedings of the 19th Design Automation Conference* New York, p. 175
[31] Hendrickson B and Leland R 1995 *Proceedings of the 1995 ACM/IEEE Conference on Supercomputing* p. 28
[32] Karypis G and Kumar V 1995 *Proceedings of the 1995 ACM/IEEE Conference on Supercomputing* p. 29
[33] Hou B N and Yao Y P 2010 *2010 IEEE/ACM 14th International Symposium on Distributed Simulation and Real Time Applications* p. 7