Computers and Mathematics with Applications 63 (2012) 427–436

Contents lists available at SciVerse ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

Optimizing K^2 trees: A case for validating the maturity of network of practices

Quan Shi[a], Yanghua Xiao[b,*], Nik Bessis[c,**], Yiqi Lu[b], Yaoliang Chen[b], Richard Hill[c]

[a] *School of Computer Science and Technology, Nantong University, Nantong, China*
[b] *School of Computer Science, Fudan University, Shanghai, China*
[c] *School of Computing and Mathematics, University of Derby, United Kingdom*

ARTICLE INFO

Keywords:
K^2 tree
Storage optimization
DFS code
Compression algorithm
Network of practices

ABSTRACT

Of late there has been considerable interest in the efficient and effective storage of large-scale network graphs, such as those within the domains of social networks, web and virtual communities. The representation of these data graphs is a complex and challenging task and arises as a result of the inherent structural and dynamic properties of a community network, whereby naturally occurring churn can severely affect the ability to optimize the network structure. Since the organization of the network will change over time, we consider how an established method for storing large data graphs (K^2 tree) can be augmented and then utilized as an indicator of the relative maturity of a community network. Within this context, we present an algorithm and a series of experimental results upon both real and simulated networks, illustrating that the compression effectiveness reduces as the community network structure becomes more dynamic. It is for this reason we highlight a notable opportunity to explore the relevance between the K^2 tree optimization factor with the maturity level of the network community concerned.

1. Introduction

The efficient organization and representation of graph data and the corresponding data processing method is a topical issue for the database research community. In recent years, the exponential rate of growth and pervasiveness of the Internet have resulted in larger scale datasets, particularly, since the advent of social networking applications. For instance, Facebook has in excess of 484 million registered users, with each individual having, on average, 120 'friends' [1].

Clearly, the volume and complexity of these datasets have much to offer the research community, in terms of understanding established and emerging developments in data organization and usage. In contrast, with more traditional repositories such as geographical map stores, these large-scale social networks have the potential to exhibit an increased richness that has not been witnessed before. Existing research can be divided into two categories. In the first category, the methods focus upon graph representation on secondary storage devices. For example, Aggarwal et al. [2] build a connectivity index for massive disk-resident graphs based on a digest graph, which can be directly loaded into memory. The second category considers graph compression as a means of reducing storage costs. Both of these approaches tend to explore the property of an adjacency list or matrix. Amongst them, representative methods include: K^2 tree [3]; compression based upon

* Corresponding author.
** Corresponding address: University of Derby, School of Computing & Maths, Kedleston Road, Derby, DE22 1GB, United Kingdom. Tel.: +44 1332592108; fax: +44 1332597741.
E-mail addresses: sq@ntu.edu.cn (Q. Shi), shawyh@fudan.edu.cn (Y. Xiao), n.bessis@derby.ac.uk (N. Bessis), luyiiqi@gmail.com (Y. Lu), chenyao2@gmail.com (Y. Chen), r.hill@derby.ac.uk (R. Hill).

0898-1221/$ – see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2011.07.060

vertex similarity [4]; ζ-code [5]; use of frequent patterns [6]; compression using graph skeleton by symmetry [7]; social network compression using shingle ordering [8]; finally a back-link schema and a hybrid compression method utilizing dynamic arrays [9]. The next section will offer an introduction to the contextual basis of our work.

2. Contextual basis

2.1. Networks and communities

There has been a considerable amount of research with regard to the concept of Communities of Practice (CoP). This work takes the perspective that a CoP has a social learning theory dimension [10–12] which in effect acknowledges elements of knowledge transformation between connected nodes on the basis of their shared interest.

The emergence of web technologies has concentrated the study of CoP from various technological based developments, including web and virtual communities, online communities and social networks. For brevity we shall use the term 'Network of Practice' (NoP) [13] to refer to the overall set of informal, emergent social networks that facilitate learning and knowledge sharing between individuals conducting practice-related tasks. [12] describes the life-cycle of a CoP as an informal learning and development process encompassing three key stages. We will use the three key stages to annotate user activity and participation within any network of practice (NoP).

During the first stage, an NoP is in its infancy and comprises a scattered set of individuals across a particular network space, that have intent to gain awareness of their space. This is characterized by behavior that promotes and accelerates the process of familiarization between different parties, for instance, by flooding the community network space with message requests for information.

Following on and as a result of stage one interaction, an individual and collective awareness emerges that facilitates an active environment. Individual networked nodes will exhibit more bi-directional communication, with less emphasis upon the broadcasting of messages to the masses. This is where the network space exhibits evidence of user participation and thus, individuals working toward the fulfillment of their goals.

Finally, the individuals demonstrate more fruitful communicative relationships, and commonality between clusters of activity becomes more apparent suggesting the emergence of a particular expertise. The predominant behavior in the network at this stage is that of relationship maintenance; rather than relationship propagation, there is little need to broadcast requests.

Lately, there is an increased interest in understanding what constitutes 'success' in an NoP. Most studies describe success of communities from the perspective of the information system success models. [10] points out that the main concern with this model is that it does not consider the social relationships among members nor the structure of the networked community. For this reason, [10,11] suggest a new approach based on the social network analysis (SNA). Following their work, we are particularly interested in understanding the maturity of an NoP which in effect reflects its relative success.

Specifically, the ability to make an assessment of the maturity of a CoP presents an opportunity for emerging systems to develop more sophisticated behaviors, such as self-awareness. Within this context, we are interested in the development of an optimized K^2 tree method that can be utilized to demonstrate the maturity of an NoP. The remainder of this article describes how we have augmented the simple K^2 tree approach in order that more complex, community network graphs can be compressed effectively for storage, as well as providing an indication as to the relative maturity of that network. First, we introduce the K^2 approach.

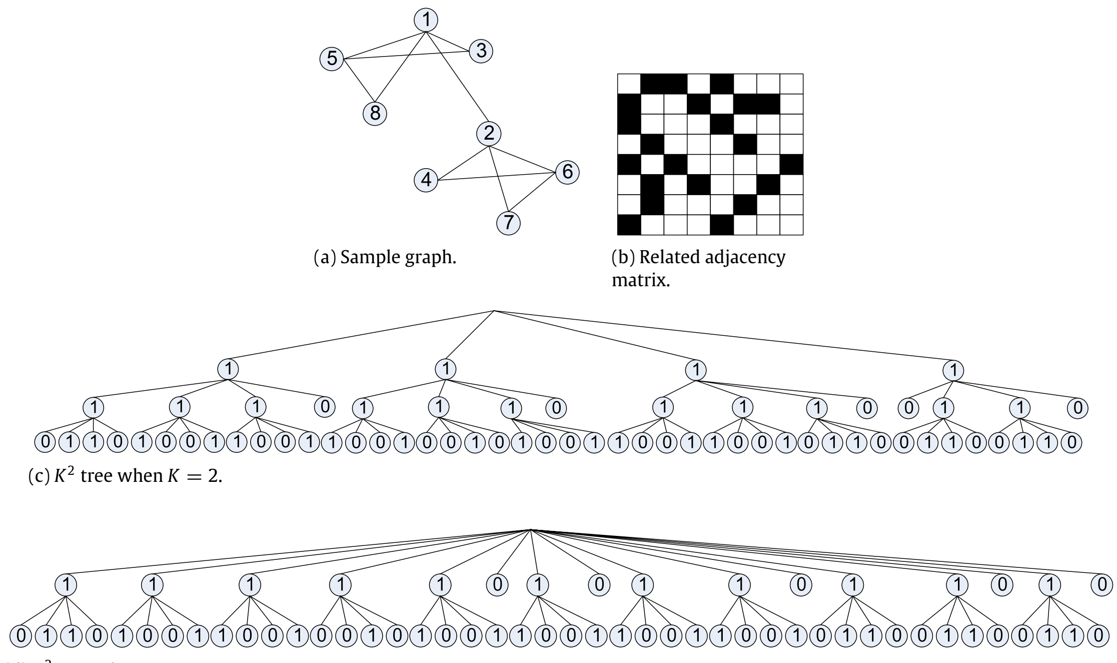
2.2. K^2 tree approach

If a real-world network is represented as a data graph, it will typically demonstrate empty regions. A K^2 tree compresses a large number of these zero value regions in an adjacency matrix, in order that a reduced number of K^2 tree nodes will result, thus demonstrating effective compression. However, a simple K^2 tree is still too abstract for the effective representation of the structural characteristics of real-world networks. In terms of compression efficiency, there is still considerable potential to improve upon a simple K^2 tree. First, many real networks have an evident hierarchy of communities. Nodes inside one community are highly interconnected with other nodes, while links between different communities are relatively sparse.

If it were possible to sort the nodes of a graph, based upon the similarity of their respective indices, then the corresponding adjacency matrix would reflect a concentration of the potential values, in this case either containing values of 'one' or 'zero'. It follows that this concentration of values within the matrix would significantly improve the resulting compression effectiveness.

However, there are two fundamental issues with this approach. First, a simple K^2 tree cannot represent the complex network structure of a community. Second, the established K^2 compression method relies upon a fixed value for K; should the structure of the community alter, and therefore the distribution of values in the adjacency matrix change, the calculated value for K would now be inappropriate resulting in sub-optimal compression.

This presents a challenge that forms the basis of this research, in which we not only explore the optimization of the K^2 method in order to maximize graph compression for storage, but we also take account of the need to represent the maturity of more complex, dynamic community network structures such as an NoP.



(a) Sample graph.

(b) Related adjacency matrix.

(c) K^2 tree when $K = 2$.

(d) K^2 tree when $K = 4$.

Fig. 1. A sample graph and its K^2 trees.

2.2.1. *Basic concepts*

We consider an undirected graph as an exemplar by which we can discuss the challenges of optimization using a K^2 tree. The proposed algorithm can also be extended to solve the difficulties of representation and storage of graphs with multi-edges and self-loops. This section gives some basic concepts and the formal definitions used in this paper, such as the undirected graph and a K^2 tree.

Definition 1. Undirected graph. An undirected graph is a tuple $G = (V, E)$, where V is the set of vertices in graph G and $E \subseteq V \times V$ is the edge set.

For graph G, its corresponding adjacency matrix A is an $N \times N$ matrix where N is the vertex number of graph G and $a_{ij} = 1$ iff $(v_i, v_j) \in E$, otherwise $a_{ij} = 0$. Fig. 1(a) and (b) give an example of the graph and its corresponding adjacency matrix.

Definition 2. K^2 tree. An adjacency matrix can be represented by an unbalanced K^2-dimension tree, called a K^2 tree. For a K^2 tree, in addition to the leaf nodes used to represent the adjacency matrix, the other nodes can be represented by 0 for a leaf node and 1 for an internal node. The lowest level in the tree is the root, followed by the second level (K^2 children of root), each of which is either 0 or 1. In a K^2 tree, every internal node has K^2 children.

Given an adjacency matrix, the construction of a K^2 tree is similar to that of Quad Tree [14]. First, the adjacency matrix is divided equally into parts, that is, divide the original matrix into K^2 $K \times K$ sub-matrixes. These K^2 sub-matrixes correspond to the K^2 children of the root of the K^2 tree. If a sub-matrix has a value of 1, the value of the child node is also 1; otherwise it is assigned a 0. It follows that, a node of value 0 means that the elements in a sub-matrix it represents are all 0, which as a consequence, makes this a leaf node. For those nodes of value 1, we recursively divide the sub-matrix it represents into K^2 parts until all of the values of a sub-matrix are equal to 0, or a sub-matrix which consists of only one element is reached.

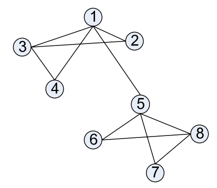
For a K^2 tree, a large K value leads to a tree with relatively small number of layers but a large spread of leaf nodes. Fig. 1(c) and (d) show the K^2 tree when the K value is 2 and 4.

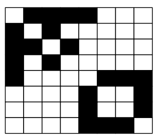
3. Optimization of K^2 tree

In this section, we present various optimization techniques for K^2 tree.

3.1. *DFS code with heuristic rule*

For sparse networks that exhibit a community structure, there must exist some specific node orders which make the elements of value 1 in the corresponding adjacency matrix relatively concentrated in some sub-matrixes, rather than randomly appearing in the adjacency matrix. Therefore, by finding an effective node encoding and reordering schema the number of internal nodes in K^2 tree can be suitably reduced.


(a) Reordered graph.


(b) Corresponding adjacency matrix.

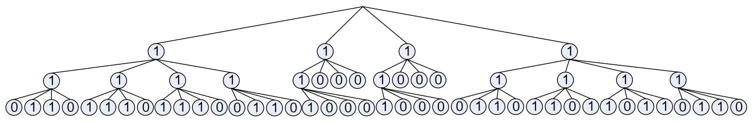

(c) K^2 tree when $K = 2$.

Fig. 2. Reordered graph and its corresponding K^2 tree.

For a graph with N vertices, there are $N!$ possible node orders in total, each corresponding to a unique K^2 tree. It is clear that a different node order leads to different K^2 tree size. As shown in Fig. 2, after reordering the nodes in the graph of Fig. 1(a), the K^2 tree ($K = 2$) obtained has 4 nodes less than the original. Amongst all possible orders of nodes, the one leading to K^2 tree with minimum tree node number is the optimal node order. Finding the optimal node order is intractable. Even when $K = 2$, this optimal node ordering problem can be reduced to a minimum bisection problem, which has been proved to be NP-hard [15]. When K is dynamic, the problem becomes more complex. Therefore, we need to use heuristic rules to find the approximate optimal solution. This article uses depth first search (DFS) with heuristic rules to encode the vertices in a graph, and then takes the DFS order as approximate for the optimal node order.

The optimal node order should be used to encode nodes inside a community continuously. Therefore, an appropriate DFS heuristic rule to guide the process of traversing the nodes within one community is needed. As such, the index of these nodes in one community is continuous which makes them more likely to be in one sub-matrix when the original matrix is being divided. In our experiment, it is shown that this heuristic rule will achieve an improved performance together with the retention of community structural characteristics. We use a simple and effective heuristic rule: *structural similarity*. For a given node pair (u, v), the structural similarity between u and v is $\alpha(u, v)$,

$$\alpha(u, v) = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|}, \tag{1}$$

where $N(u)$ denotes the neighbor set of node u. For every edge $e(u, v)$, we can define its structural similarity $\alpha(u, v)$. Using DFS, the neighbor which has the largest structural similarity is selected as the next node to visit.

We take Fig. 1(a) as an example. In Fig. 1(a), we have two communities where the indices of nodes are randomly assigned. Fig. 2(a) shows the re-encoded graph from node 1 using our heuristic DFS rule. Every node inside a community is indexed continuously. Its corresponding adjacency matrix is given in Fig. 2(b). Edges inside one community tend to appear in the same block, which will assist the compression efficiency of K^2 tree. The graph in Fig. 2(c) has 4 less nodes than that in graph 1(c).

Structural similarity has been widely used in computing the collection similarity, which is also known as the Jaccard coefficient. For graphs represented by an adjacency list, we can first perform a sort on an adjacency list. For an edge (u, v), its Jaccard coefficient can be obtained by using mergesort in $\Theta(d(u) + d(v))$. Based on mergesort, it can be proved that we can obtain the Jaccard coefficient of every edge in the graph in $\Omega(M^2/N)$.

Lemma 1. *For a sparse graph $G(N, M)$, where N is the number of vertices, M is the number of edges. Based on mergesort, the time complexity of calculating the Jaccard coefficient of all edges is $\Omega(M^2/N)$.*

Proof. The cost of sorting an adjacency list is

$$\sum_{u \in V} d(u) \log d(u) \le \log d_{\max} \sum_{u \in V} d(u) \le M \log d_{\max}$$

where $d_{\max}$ is the maximal degree in the graph. Therefore, the total cost of mergesort is

$$\sum_{(u,v) \in E} (d(u) + d(v)) = \sum_{v \in V} d(v)^2 \ge \frac{\left(\sum_{v \in V} d(v)\right)^2}{N} = 4\frac{M^2}{N}.$$

Hence, the time complexity is $\Omega(M^2/N)$.

Large scale, real world networks such as social networks always have a *scale-free network* topography [16]. Its degree distribution obeys a power law form:

$$d(v) = \frac{1}{N^R}(r(v))^R$$

where $r(v)$ is the degree rank of vertex v in the graph, R is a constant and $R < 0$. For real graphs of this kind, it can be proved that the Jaccard coefficient can be calculated in $O(N \log N)$ when $R \le -0.5$. In fact, the R value of most of the scale-free networks is within -2 to -3 [17]. □

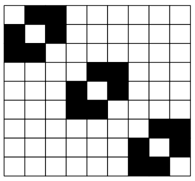


Fig. 3. A 9 × 9 adjacency matrix.

Corollary 1 (*Time Complexity of Calculating the Jaccard Coefficient on Scale-Free Networks*)**.** *For graph with degree distribution of* $d(v) = \frac{1}{N^R}(r(v))^R$ *and* $M = \Theta(N \log N)$*, the time complexity of calculating the Jaccard coefficient of all edges based on mergesort is* $O(N \log N)$.

Proof. The cost of sort on adjacency list is the same as the proof in Lemma 1, which is $O(N \log N)$. The accumulated cost is

$$\sum_{(u,v)\in E} d(u) + d(v) = \sum_{u\in V} d(u)^2 = \sum_{i=1}^{N} i^{2R}/N^{2R}.$$

Therefore, when $R \le -0.5$ the upper bound is $O(N \log N)$.

For large scale networks with more than 1M nodes, the method given above to compute the Jaccard coefficient is still too costly. An alternative is to adopt shingle [18] instead. Shingle has been used widely in similarity computation for large scale networks. It [19] has been proved that for a random permutation π on a given set $U = A \cup B$,

$$\Pr[\pi^{-1}(\min_{a\in A}\{\pi(a)\}) = \pi^{-1}(\min_{b\in B}\{\pi(b)\})] = \frac{|A \cap B|}{|A \cup B|}. \tag{2}$$

That is to say, the probability that the minimum element of A under permutation π equals to the minimum element of B under permutation π is the Jaccard coefficient of set A and B. Based on the theorem above, the $O(N)$ shingle computation method has been proposed [20] to obtain the Jaccard coefficient. In this case, the time complexity of computing the Jaccard coefficient in our heuristic rule is $O(N)$. □

3.2. Self-adaptive K

Another key factor of K^2 tree compression is the selection of K, which is usually fixed for a simple K^2 tree. In many cases, a fixed K may break a sub-matrix which has many values of one into a scatter, resulting in an incremental increase in the number of K^2 tree internal nodes. As shown in Fig. 3, if $K = 3$, we need to divide the matrix according to the community in this graph, otherwise the community will be broken. Therefore, an appropriate value for K is vital.

To address this, we consider K no longer to be a fixed value, which leads to a variable tree structure with every internal node having a different fan-out number. Ideally, an algorithm has to propose the best fan-out number for every internal node, thus presenting a significant computational overhead. We propose that only the sibling nodes that share a common parent node will share also the same value for K. In this section, we will illustrate by means of theoretical analysis and experimental results that the cost of this is relatively low, whilst also achieving good performance.

Algorithm 1 Build K^2 tree(G)

```
Input: Graph $G$
Output: $TG$: Optimal $K^2$ tree;
1: Let M be the adjacent matrix of $G$;
2: TG_set = {};
3: For each $k$ from $k_{\min}$ to $k_{\max}$
4:     TG_set = TG_set ∪ {Construct $K^2$ ($M$, $k$, $0$)};
5: Return the tree with minimal nodes in TG_set as TG;
```

The algorithm for building a K^2 tree with a self-adaptive K value is described in Algorithm 1. We try every possible k value in $[k_{\min}, k_{\max}]$, then call the method **Construct*K*^{2}** to calculate the optimal K^2 tree, before selecting the best value for K. In Algorithm 2, lines 1 and 4 iterate through the current sub-matrix to determine whether it is all zeros or ones, thus identifying a leaf node. Otherwise, go to line 8 to recursively compute the best value for K. Algorithm 2 also attempts to divide the current sub-matrix into every possible value for K, recording the resulting node number (lines 11–13) and storing as *descendant_count*. After all possible values for K have been tried, algorithm 2 takes the value stored in *descendant_count* and stores the minimum value into $m[level]$ (lines 17–19). This results in an optimized K^2 tree (lines 20–21).

Algorithm 2 Construct K^2 (M, K, $level$)

Input: Adjacency matrix M, K (current K value), ***level*** (current level)
Output: *TG*: optimal K^2 tree;

```
1:  IF all the elements in $M$ are 0 Then {
2:  $m[level] = 0$; // $m[level]$ store the number of nodes of the $K^2$ sub-trees corresponding to $M$
3:     Construct a leaf with label 0 as TG; }
4:  IF all the elements in $M$ are 1 Then {
5:     $m[level] = 0$;
6:     Construct a leaf with label 1 as TG; }
7:  Else
8:  min_$TG\_set$ = {};
9:  $m[level] = \infty$;
10: Divide $M$ into $K^2$ SubMatrixes with equal size.
11: For each $k$ from $k_{\min}$ to $k_{\max}$ {
12:    TG_set = {}; //store the $K^2$ tree of submatrix
13:    $descendant\_count = 0$;
14:    For each SubMatrix {
15:       TG_set = TG_set ∪ {Construct $K^2$ (SubMatrix, $k$, $level + 1$)};
16:       $descendant\_count += m[level + 1] + 1$; }
17:    IF $m[level] > descendant\_count$ Then {
18:        min_TG_set = TG_set;
19:       $m[level] = descendant\_count$; }
20:    Create a new tree $TG$ with a root node $R$;
21:    Set R's subtrees as Min_ TG_set;}
22: Return $TG$;
```

The recurrence formula is given below:

$$T(n) = d_k k_i^2 T\left(\frac{n}{k_i^2}\right) + C, \tag{3}$$

where k_i is the current recurrence depth.

Constant C corresponds to lines 6 and 9 in Algorithm 2. By iterating we can determine whether the sub-matrix contains either all zeros or all ones. In formula 3, k_i corresponds to a different recurrence depth, but in practice, k is selected within a small range. In later experiments, we select k in [2,4]. In our analysis, we assume selection of the same k_i for every recurrence depth such as K_{mid}, and then according to the master theorem, the complexity of our algorithm is $O(N^{\alpha})$, where $\alpha = \log_{K^2_{\mathrm{mid}}} d_k k^2_{\mathrm{mid}} = 1 + \frac{1}{2}\log_{K_{\mathrm{mid}}} d_k$. In subsequent experiments, we select a k value in [2,4], which provides good performance in our experiment on a network with 10^5 nodes where the complexity is $O(N^{1.5})$.

4. Experiments

We implemented the proposed algorithm using C++. All the experiments were conducted on Windows 7 Professional with Intel Core Duo 1.60 GHz and 2 GB memory.

4.1. Dataset

The network data used in this experiment is shown in Table 1, where N is the vertex number, M is the edge number and $2M/N$ is the average degree. Gq is the random network generated by Pajek and Lesmis [21]. Football [22], Cond_mat [23] and DBLP [24] are real network data.

4.2. Results

Based on this specification, the following represents our experimental results.

4.2.1. Re-sort of vertices

This experiment is conducted to show the relative compression efficiency of a K^2 tree constructed using the DFS heuristics, compared with that of a naïve K^2 tree.

We compare the number of nodes in a K^2 tree obtained by the original method and DFS, with a heuristic rule on the real network data listed on Table 1, when $K = 2, 3, 4, 5, 6, 7$. The experimental results are given in Figs. 4–8. In Gq, Lesmis,

Table 1
Experimental data.

Network	N	M	$2M/N$
Gq	578	1 068	3.696
Lesmis	77	254	6.597
Foot ball	115	613	10.661
Cond_mat	36 458	171 736	9.421
Dblp	481 433	1 719 320	7.143

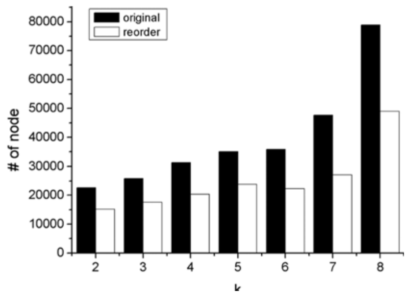


Fig. 4. Results on Gq.

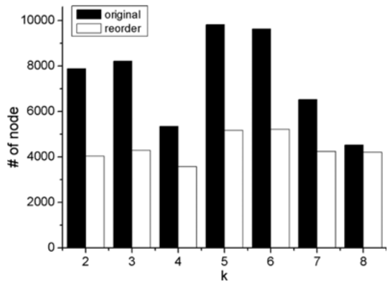


Fig. 5. Results on Football.

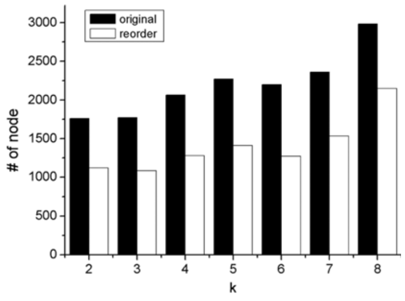


Fig. 6. Results on Lesmis.

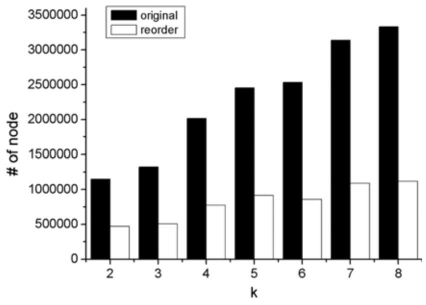


Fig. 7. Results on Cond_mat.

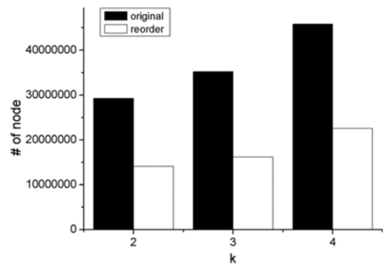


Fig. 8. Results on Dblp.

Table 2
Adaptive K compared with fixed K.

Graph	Original K^2 tree			K^2 tree with vertex ordered by DFS code			
	$K = 2$	$K = 3$	$K = 4$	$K = 2$	$K = 3$	$K = 4$	SA
Lesmis	1760 25.68%	1771 22.52%	2064 21.90%	1120 40.36%	1086 41.62%	1280 35.31%	452
Football	7876 18.68%	8211 17.91%	5340 27.55%	4036 36.45%	4280 34.37%	3568 41.23%	1471
Gq	22 598 29.50%	25 788 25.85%	31 365 21.26%	15 172 43.94%	17 587 37.91%	20 390 32.70%	6667
Cond_mat	1151 592 15.58%	1323 149 13.56%	2014 640 8.91%	472 716 37.96%	508 969 35.26%	771 004 23.28%	179 460

Football, Cond_mat and DBLP networks, the ratio of numbers of nodes of K^2 tree using DFS, with heuristic rule and nodes of the original K^2 tree is 64.22%, 63.40%, 62.14%, 36.73% and 47.8% on average. In those graphs, the number of nodes reduces to 42.07%, 38.0%, 48.76%, 66.5% and 54.0% respectively in the best case. These experiments indicate that the K^2 tree using DFS and a heuristic rule can effectively reduce the number of nodes in a K^2 tree.

Similarly, for different real networks, the optimal K value is also different. Generally, the number of nodes of a K^2 tree will increase with the growth of K. However, there are some counter examples. For the Football network data, when $K = 4$, the compression effectiveness reaches its optimum. It follows, therefore that the optimal K value depends on the structural property of a community network.

4.2.2. *Adaptive K compared with the fixed K*

This experiment is conducted to illustrate the efficiency of an adaptive adjustment strategy for K, in comparison to a naïve K^2 tree.

For a given node order, we compare the number of nodes of the self-adaptive k algorithm with that of the original K^2 tree. In Table 2, we present the experimental results for Lesmis, Football, Gq and Cond_mat when $K = 2, 3, 4$. Based on the DFS encoding with a heuristic rule, the number of nodes in a self-adaptive K^2 tree reduces to 39.10%, 37.35%, 38.18% and 32.17% of the original K^2 tree on average. In the best case, the ratio is 35.31%, 34.37%, 32.70% and 23.28% respectively. So, by self-adaptively altering k, we can minimize the damage to the community structure of the input graph.

From Table 2, we can also observe that by combining the two optimization methods together, we can achieve a compression effectiveness of 21.90%, 17.91%, 21.26% and 8.91% in the best case.

4.2.3. *Selection of K*

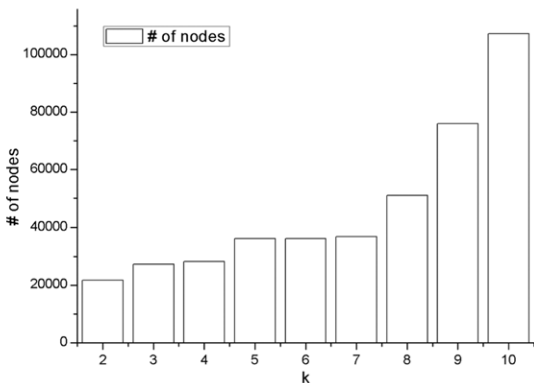
This experiment is conducted to find the best K of an adaptive adjustment strategy when used to build K^2 trees from real networks.

An appropriate k value interval is of great importance to our compress algorithm. We use 2 synthetic networks to determine the best k value interval. The first BA network has 1000 nodes and 1980 edges and the second ER network has 1000 nodes and 5141 edges. For k varies from 2 to 10, the result on those two graphs is given in Fig. 9. The number of nodes in K^2 tree grows with the increment of k value. When k falls in [2,4], K^2 tree has an acceptable result.

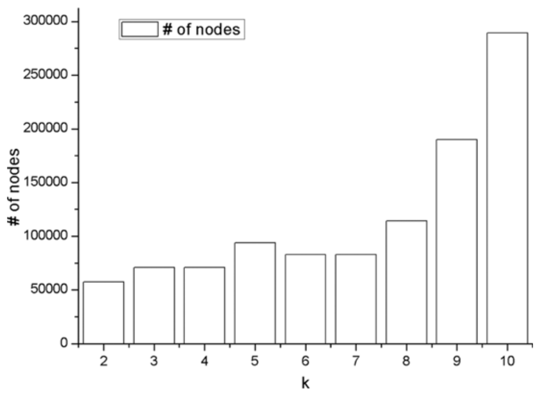
4.2.4. *Influence of community structure*

In this subsection, we will investigate how a community structure of a real network can impact upon on the efficiency of our approach.

In this section, we will discuss the impact of the clarity of a community structure in a network on the method we propose. We divide 1000 nodes into 10 communities. For a node pair (u, v), if they are in the same community, then the edge probability is 0.7, otherwise the probability is 0.001. As a result, we obtain a graph with 1000 nodes and 39 124 edges. It is self-evident from the construction of synthetic networks that the resulting synthetic graphs are of strong community



(a) BA network.



(b) ER network.

Fig. 9. Experimental result under different K.

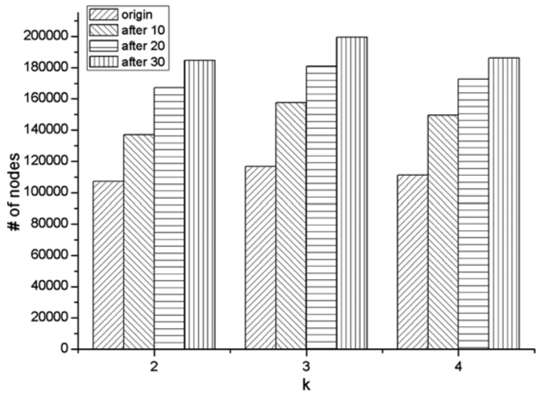


Fig. 10. Experimental result under different community structures.

structure. In order to test the impact mentioned in this section, we do a t time experiment. In each experiment, we randomly disconnect $10t$ edges inside a community and then randomly link $10t$ edges between different communities. In our setting, t varies from 1 to 3. The result on those synthetic networks is given in Fig. 10. It is shown that the compression grows worse with the loss of community structure. These results demonstrate that a mature NoP shall encompass two key characteristics. First, the overall behavior of a mature NoP should be homogeneous which in effect demonstrates that participant nodes interact with other well defined and trusted participant nodes. The results also demonstrate that such a behavior should be expected to occur during stage three of an NoP. This is because participant nodes have developed the skill, knowledge and expertise to process filtered requests. The results show also that the more explicit community structure is reflected in better compression. Second, the structure of the underlying network shall be broadly stable. This is to say that communication links between participant nodes will tend to be altered for the purposes of relationship maintenance, rather than as a general attempt to establish a relationship. Again, the results demonstrate that the more dynamic the structural changes, the worse the compression. It is expected that a newly formed CoP is more likely to exhibit a greater proportion of participant nodes joining and leaving (as per stage one) and thus will not tend to demonstrate stability.

5. Conclusion

A key thrust of this research has been to investigate how complex data graphs, such as those that represent NoP communities, can be appraised in terms of their relative maturity. As described earlier, we consider a network to be mature when two conditions are met; although individual nodes may possess heterogeneous behaviors, the overall behavior of the network would be regarded as homogeneous. Second, the network that represents an NoP will have progressed through three stages of maturity, based on those proposed by [23].

We have also described the simple K^2 tree approach and illustrated how the natural occurrence of sparse regions in real-world networks can be exploited to achieve effective compression. However, since this method does not consider the structural characteristics of a community network, there is still an opportunity to optimize compression further for graphs of NoP. Therefore, we have proposed two optimization methods as follows: encoding nodes using DFS with heuristic rule, and a self-adaptive k building K^2 tree. These two methods take good advantage of community feature in the real network, so as to reduce the number of internal nodes further. Experiments on synthetic and real networks indicate that the K^2 tree has less nodes than simple K^2 tree. To conclude, we have chosen to exploit the observation that an augmented K2 tree compression method can be used to validate that a network has in fact reached stage three maturity.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under grant Nos. 61003001, 61171132, Specialized Research Fund for the Doctoral Program of Higher Education grant No. 20100071120032, and the Natural Science Foundation of Jiangsu Province grant Nos BK2009153 and BK2010280.

References

[1] comScore: http://www.cctime.com/html/2010-4-22/201042291342486.htm.
[2] C. Aggarwal, Y. Xie, P.S. Yu, Gconnect: a connectivity index for massive disk-resident graphs, in: Proc. of VLDB'09, ACM, Lyon, France, 2009, pp. 862–873.
[3] N.R. Brisaboa, S. Ladra, G. Navarro, k^2-trees for compact web graph representation, in: Proc of the 16th Int. Symp. on String Processing and information Retrieval, in: LNCS, vol. 5721, Springer, Berlin, 2009, pp. 18–30.
[4] P. Boldi, S. Vigna, The webgraph framework I: compression techniques, in: Proc of the 13th Int. Conf. World Wide Web2004, ACM, New York, USA, 2004, pp. 595–602.
[5] P. Boldi, S. Vigna, The webgraph framework II: codes for the world-wide web, in: Proc of Data Compression Conference, DCC'04, IEEE Computer Society, Washington, USA, 2004, p. 528.
[6] G. Buehrer, K. Chellapilla, A scalable pattern mining approach to web graph compression with communities, in: Proc of the 1st WSDM'08, ACM, California, USA, 2008, pp. 95–106.
[7] Yanghua Xiao, B.D. MacArthur, Hui Wang, et al., Network quotients: structural skeletons of complex systems, Physical Review E 78 (2008) 046102.
[8] F. Chierichetti, R. Kumar, S. Lattanzi, et al. On compressing social networks, in: Proc of the SIGKDD'09, Paris, France, 2009, pp. 219–228.
[9] Yuming Li, Wanpeng Dong, Yinghong Peng, Study on matrix compressive storage method based on 0–1 property-matrix, Computer Engineering and Applications 39 (2) (2003) 82–84 (in Chinese).
[10] S.L. Toral, M.R. Martınez-Torres, F. Barrero, F. Cortes, An empirical study of the driving forces behind online communities, Internet Research 19 (4) (2009) 378–392.
[11] M.R. Martınez-Torres, S.L. Toral, F. Barrero, F. Cortes, The role of Internet in the development of future software projects, Internet Research 20 (1) (2010) 72–86.
[12] E. Wenger, Communities of Practice: Learning, Meaning, and Identity, Cambridge University Press, Cambridge, 1998.
[13] J.P. Johnson, Collaboration, peer review and open source software, Information Economics and Policy 18 (2006) 477–497.
[14] H. Samet, The quadtree and related hierarchical data structures, ACM Computing Surveys 16 (2) (1984) 187–260.
[15] M.R. Garey, D.S. Johnson, L. Stockmeyer, Some simplified NP-complete graph problems, in: Proc. the 6th Annual ACM Symp. Theoret. Comput. Sci., vol. 1, No. 3, 1974, pp. 237–267, 197.
[16] M. Faloutsos, P. Faloutsos, C. Faloutsos, On power-law relationships of the internet topology, in: Proc. of SIGCOMM, ACM, Cambridge MA, USA, 1999, pp. 251–262.
[17] M.E.J. Newman, The structure and function of complex networks, SIAM Review 45 (2) (2003) 167–256.
[18] A.Z. Broder, S.C. Glassman, M.S. Manasse, et al., Syntactic clustering of the Web, Computer Networks 29 (8–13) (1997) 1157–1166.
[19] A. Broder, M. Charikar, A. Frieze, M. Mitzenmacher, Min-wise independent permutations, Journal of Computer and System Sciences 60 (2000) 630–659.
[20] D. Gibson, R. Kumar, A. Tomkins, Discovering large dense subgraphs in massive graphs, in: Proc. of VLDB'05, ACM, Trondheim, Norway, 2005, pp. 721–732.
[21] D.E. Knuth, The Stanford GraphBase: A Platform for Combinatorial Computing, ACM Press, 1993.
[22] V. Batagelj, A. Mrvar, Pajek datasets, 2006. http://vlado.fmf.uni-lj.si/pub/networks/data/.
[23] M.E.J. Newman, The structure of scientific collaboration networks, Proceedings of the National Academy of Sciences USA 98 (2) (2001) 404–409.
[24] Dblp [2010-05-5]. http://dblp.uni-trier.de/xml/.