University of Cincinnati

Date: 7/10/2013

I, Chen Lu, hereby submit this original work as part of the requirements for the degree of Master of Science in Computer Science.

It is entitled:
Local K-Core Algorithm in Complex Networks

Student's name: Chen Lu

This work and its defense approved by:

Committee chair: Yizong Cheng, Ph.D.

Committee member: Fred Annexstein, Ph.D.

Committee member: Anca Ralescu, Ph.D.
Local K-Core Algorithm in Complex Networks

A Thesis submitted to the
Graduate School
of the University of Cincinnati
in partial fulfillment of the
requirements for the degree of
MASTER OF SCIENCE
in the Department of
Electrical Engineering & Computing Systems of the College of Engineering

By

CHEN   LU

Bachelor of Industrial Design
Harbin University of Science and Technology, Harbin, CHINA, 2002

Master of Compute Science
Harbin Institute of Technology, Harbin, CHINA, 2006

Thesis Advisor and Committee Chair: Dr. Yizong Cheng
Abstract

Local methods tend to be more effective for community definition and detection than those difficult and large time consuming global ones. In this thesis, we introduce and define the notion of local $k$-core; a simpler local method easily implemented and computed which is based on the knowledge of neighborhood graph of the vertices in the graph. From the definition, we present the localized algorithm for local $k$-core decomposition algorithm. We show its relationship with 3-clique percolation and the $k$-core algorithm. Finally our experiment results reveal the efficiency of the algorithm and the richer community structure it exposed in real networks.
Acknowledgements

Foremost, I would like to thank Dr. Yizong Cheng for continuing support of my study and research and for his patience with the length of time it took me to complete it. Thanks also go to the professors on the thesis committee: Dr. Anca Ralescu and Dr. Fred Annexstein for their valuable and insightful advices and suggestions.

I would also like to thank my labmates: Nan Wang and Qingyan Xie for improving my idea and thesis. Last and certainly no least I would like to thanks my family for their support throughout the process.
Table of Contents

1. Introduction ............................................................................................................... 1

2. Background .................................................................................................................. 4
   2.1 Related Works ......................................................................................................... 4
   2.2 K-Core decomposition ............................................................................................ 6
   2.3 Clique Percolation ............................................................................................... 8

3. Local K-Core algorithm ............................................................................................. 11
   3.1 Local Community Degree ...................................................................................... 11
   3.2 Local K-Core ........................................................................................................ 16
   3.3 Local K-Core algorithm .................................................................................... 19

4. Experiment and result. ............................................................................................... 23
   4.1 Brief of the datasets ........................................................................................... 23

5. Conclusion and Future work ..................................................................................... 36
   5.1 Conclusion ......................................................................................................... 36
   5.2 Future work ....................................................................................................... 36
Table of Figure

Figure 2-1  \( k \)-core decomposition sketch in a small graph. ........................................... 8
Figure 3-1 Illustration of the neighborhood network for real network. ...................... 13
Figure 3-2 \( k \)-core vs local \( k \)-core. ............................................................................ 17
Figure 3-3 A small graph example for finding local 3-core according to algorithm 1. .... 21
Figure 4-1 The number of vertices of \( k \)-core as a function in Network Database .......... 26
Figure 4-2 The number of vertices of \( k \)-core as a function in Network Database. ....... 27
Figure 4-3 The edges number of the \( k \)-core as a function in Network Database. .......... 28
Figure 4-4 The Clustering Coefficient of the \( k \)-core as a function in Network Databases. ........................................................................................................................................... 30
Figure 4-5 The Giant Component sizes and graph size of \( k \)-core as a function in Network Databases. ........................................................................................................................................... 31
Figure 4-6 Local 8-core in Football network. ................................................................ 32
Figure 4-7 Local 3-core in Dolphins network. .......................................................... 33
Figure 4-8 Local 8-core in CA-HepTh network. ...................................................... 35

Table of Table

Table 3-1 Local component degree set, Principal neighborhood component degree and degree of the vertex showed in the Figure 3-1. .......................................................... 15
Table 4-1 Databases for our Experiments. ................................................................. 23
Table 4-2 The comparison with local \( k \)-core and \( k \)-core algorithm in the databases. ................................................................................................................................. 25
1. Introduction

A network is composed by vertices (nodes in some literature), and edges which make a connection between the vertices. In the real world, millions of different Systems are presented as the form of networks (graphs in some mathematical theory literature). Popular examples include social networks[1-3], biological networks[4-7], the Internet[8], the World Wide Web[9,10] and citation networks[11-13]. The analysis of the network structure has been an spotlight in recent research in every aspects of science. The analysis of the network structure is normally to find the heterogeneity of the vertices in a given network and group them together and split the whole graph into a few separate groups. The process of analysis mentioned above is called graph clustering or community detection. Typical network structure analysis is focusing on small graphs and the properties of the vertices or edges within such graph. In recent years, detection of large-scale statistical properties of networks steps out to be a substantial new movement in network research, which turns out to be available for gathering and analyzing the data far larger than before by using more powerful computers. Now it is common to see network with millions or even billions of vertices instead of hundreds when we first started to put an eye on the network in 1970s. In network theory, complex network is the term to call this kind of networks and the graph clustering is a process to find out the community structure. The unambiguous identification of unknown structural groups in networks depends very much on how a community is defined. Normally a group of vertices in the network which is “more densely” connected among themselves than with the rest of the vertices is called a community. A lot of methods are mentioned in the review literature [14,15], which use different their own ways “more densely” to find the grouping vertices
in the network. Since the size of the network is so gigantic, there is not easy way to
analysis the whole graph in a global view. Communities, on the other hand, which seem
like independent entities with their own autonomy, do not have strong ties with the rest of
the network. So the analysis of communities can be considered as independently instead
of as the whole one. Local methods usually focus on sub-graphs for studying, including
small sub-set of the vertices, vertices’ neighborhoods and other local attributes, instead of
browsing the whole network. Currently, most of the network parameters are based on the
global view or vertex’s view. Recently, the research[16-17] on the topic of evolution of
the network shows that such ideas are not absolutely correct. New vertices prefer to be
connected with the vertices which have high degree or “dense” properties, which
reinforces the idea of local methods.

This thesis concentrates on a small area of this problem space. We define new
concept named local community degree which is a good framework for discovering
community structure. Then we extend the idea to a novel notion of local \(k\)-core; a simple
algorithm is also developed to compute the local \(k\)-core which has a connection with the
3-clique percolation and \(k\)-core. The experiments show that the proposed algorithm is
efficient on the real network data presenting the community structure in the \(k\)-core level.

We begin with background information in Section 2, Section2.1 discuss a brief survey
on local methods. Section 2.2 and Section 2.3 introduce the local method called \(k\)-core
algorithm and Clique Percolation method which are good ones to analyze the network
structure.

Section 3.1 develops a new definition called local community degree based on the
local view; Section 3.2 presents the concept of local \(k\)-core and its relationship with the
clique percolation. Section 3.3 covers a new algorithm for local $k$-core decomposition and related attribute of the network.

Section 4 discusses related experiments. Finally the Result and future work are discussed in Section 5.
2. Background

2.1 Related Works

As mentioned above, community structure is not unique which is all based on how researchers defined the density attribute. Clique is the most common definition of community, first developed by social network researchers [18]. Clique describes the condition that all the vertices in the network are connected with each other and each vertex, hence, has the strong similarity between each other’s. In social network analysis, clique can be seen as maximal sub-graph. It is trivial to see that triangle is the clique with size 3 and commonly seen in real network; while bigger Clique is not by its strict definition. Pella et al [19] make an extension; they define a community as a chain of adjacent cliques. His method emphasizes that a vertex possibly belongs to several cliques at the same time. Finding cliques in the networks is an $NP$-Complete problem [20] and the Bron–Kerbosch method [21] is also a time growing exponentially based on the size of the network. Clique is a good definition to analyze the structure of the network; however it’s not so good to find the good communities because it’s strictly definition.

It is better to present a new community structure while gives a little relaxation for the clique’s definition. $N$-clique [18, 22] is among the first concept of clique relaxation; it defines that the distance of each pairwise vertices is at most $n$. Because of small defects of the $n$-clique definition, Mokena [21] has suggested two new clique-like definitions: $n$-clan and $n$-club which both maintain a high density in the network. If the diameter of the group of vertices is not larger than $n$, we called such groups $n$-clan, that is to said, the greatest distances between each pair of vertices in the group are at most $n$. $n$-club, on the
another hand, is a little difference: a maximal subgraph of diameter \( n \). \( N \)-clan and \( n \)-club are all \( n \)-clique, while the reverse is not.

Adjacent of the vertices is another local criterion for community. The basic notion is using the degree of the vertex as a standard: a vertex must connect to a minimum number of other vertices in the subgraph. Here two definitions are presented: \( k \)-plex\([23]\) is a maximal subgraph in which each vertex is connected to at least \( n-k \) other vertices, where \( n \) is the number of vertices in the subgraph; \( k \)-core\([24]\) is a maximal subgraph in which each vertex is connected at least \( k \) other. These two definitions impose the clique-like condition by controlling the number of edges in the subgraph. An efficient algorithm\([25]\) is demonstrated to analysis the whole network by constructing a whole hierarchy of core at different order, which give us a possible view of architecture of the network.

It is necessary to compare the internal versus external cohesion, because being a community, there shall not be a strong cohesion between the community and the rest of the network. Some researcher starts to such requirement for community. Radicchi\([26]\) has developed the concept of the strong community and weak community. The first one is a subgraph in which the internal degree of each vertex is greater than its external degree. While the latter one only stress that internal degree of the subgraph should exceeds its external degree. An alternative definition is presented by Hu et al\([27]\); in Hu’s definition, when any internal degree of vertex in the community exceeds the number of edges which the vertex shares with any other communities, the community is called strong one; When total internal edges of the community only exceed the number of edges connecting the community with any other communities, the community is called weak one. Because of
the slight difference of these definitions, an example like $L_s$-set [28] could be a strong community based on Hu al and a weak ones based on Radicchi’s concept.

Fitness measure is another good method for community identification, which estimates to some extent a subgraph’s cohesion satisfies a giving property. It is almost the same as the quality function, which expresses how good of a graph partition is. Normally, the larger the fitness, the more accurate of the definition; A common used fitness measure is intra-cluster density $\delta_{\text{int}}(C)$, which $C$ is the subgraph for testing. Once the threshold $\xi$ is set, if $\delta_{\text{int}}(C)$ is larger than the threshold, the $C$ can be seemed as a good cluster. When $\xi = 1$, the problem turns into problem mention before: finding the clique in the network [29]. Other measure functions we used are relative density $\rho(C)$ and inter-cluster density $\delta_{\text{ext}}(C)$; the former is defined as the ratio between internal and total degree of the $C$, the latter is as the ration between inter-cluster edges to the maximum number of inter-cluster edges possible [30]. Find subgraphs satisfied such measures are NP-Complete problem. From such definition, Fitness measure is also related with the problem connectivity of sub-graph, which is associated with the cut problem in the graph.

### 2.2 $k$-Core decomposition

The elementary attribute of the network: the degree of the vertex is a popular attribute to analyze the network structure. As the clique-like definition, $k$-core decomposition is put forward to show the complex topologies of all kinds of networks, such as the Internet, the protein interaction network and the WWW [31-33]. Since rich $k$-core architecture is found in real-world networks, it can also be seem as a good method to visualize the network and explain the mutual relationship in them. Jose et al [34] used $k$-core to
analyze the structure and hierarchy attributes beyond the normal degree distribution and found pretty interesting result. The emergence of $k$-core structures in random network was also studied and some criterion was found [35, 36]. Dorogovtsev focused on the emergence of them in complex network and showed that the whole process is a hybrid phase transition [37]. The $k$-core definition is below [34]: If each vertex in the subgraph $G'$ of $G$ has degree in $G'$ not smaller than $k$, we call a $k$-core. $K$ shell $S_k$ is a set of all the vertices in the graph which each vertex belongs to a $k$-core but not a $(k+1)$-core; $K_{max}$ denotes the maximum value $k$ while $S_k$ is not empty.

It is easy to show that by keeping all the vertices of which degree is not smaller than $k$; while removing those not, until the algorithm reaches the fixed point; a $k$-core of Graph $G$ is obtained. From the easy implemented algorithm [32], the position of each vertex totally depends on its shell index and the index of its neighbors.

Figure 2-1 shows a small example of the $k$-core decomposition. From Figure 2-1, the $k$-core decomposition demonstrates the graph layer by layer, show the structure of graph from the outmost shell to the most internal ones. Also in Carmi’s paper[38], A lot of interesting feature of $k$-core were mentioned; for example, the $k$-core of AS map from the DIMES project[39] are $k$-connected[40], which means they always existed $k$ disjointed paths between any two vertices in the $k$-core [38]. So we get the important result that the larger $k$ number in the core, the more robust and routing capacities of them.

However, as the definition show, $k$-core related to the degree of the vertex, which actually is a not perfect definition for communities. $k$-core is a good method to
Figure 2-1  *k*-core decomposition sketch in a small graph. Different types of vertices belong to different *k*-shells, while each closed line correspond to the vertices set falling into a given *k*-core

identify the hierarchy structure of the network. While communities more concern about the “density” of them.

2.3 Clique Percolation

Clique as one of the definition of the community, is popular for the idea that it focus on the local property of the graph instead of the global one for instance, betweenness and modularity, etc. Palla et al [19] extended the idea of clique called Clique Percolation. The basic concept is the high possibility of the internal-edges of the community to form cliques with their high density, and the low possibility of the edges between communities to form cliques. The definition of *k*-clique as a complete graph with *k* vertices is not the
same as the one mentioned in social network [7], which usually called $n$-clique. Two $k$-cliques are adjacent if their intersection is $k-1$-clique that is to say, they share $k-1$ vertices; the $k$-clique chain is the union of adjacent $k$-clique. Then we can call two $k$-cliques are connected if they belong to part of a $k$-clique chain. The definition of $k$-clique community is the largest connected subgraph obtained by the union of all “adjacent” $k$-cliques. The community is like “rolling “ a $k$-clique to all the adjacent $k$-cliques, which by relocating one of its vertices while keeping the other $k-1$ vertices fixed[19]. The trivial fact shows by definition is that the community is actually a connected graph when $k=2$ and disconnected nodes without any edge when $k=1$.

To find $k$-clique communities, the $k$-clique percolation algorithm needs to find maximal cliques, which the running time is known to be exponentially with the size of the network. Palla also points out that because the limited number of cliques and sparse networks are common in the real-world graph, the procedure is actually pretty fast. According to the definition, the algorithm is a good example of local community-finding method; the global network structure can be obtained by browsing all of local $k$-cliques structures. It is trivial to identify the vertices in the network: some might not belong to any community if they are not part of any clique, others can belong to several communities based on location of overlapping two or more communities. Here the parameter $k$ is an important factor which determines nature of the communities. Using different parameter $k$, the nested nature of communities is discovered. $K$-clique percolation focusing on the clique structure may simply overlook other modules which are not with high density and well-connected like trees and so on; on the other hand, it is
pretty inflexible for communities with different $k$ in the graph, because not all the communities structure in the same $k$.

The algorithm gives us a good view of analyzing the structure of overlapping networks. Palla and other researchers extend the algorithm to analysis of weight-graph and directed graph [41] and moreover the evolution of social networks [42]. They developed a good software package $C$Finder implementing the algorithm, which is available for free (www.cfinder.org).
3. Local $k$-core algorithm

The $k$-core Decomposition algorithm focuses on the degree of the vertex, while the community structure is more on “density” side of the graph. The neighborhood of a vertex $v$ is the set of all other vertices directly connected to it. Neighborhood presents the condition of connectivity of nearby vertices; and it is a great attribute for community detection. In the paper, we do not consider the condition that one vertex has the possibility belonging to several different communities, which means not overlapping communities in the paper. Also we focus on the undirected graphs and networks for the thesis.

This section begins with definition of local community degree by extending the neighborhood concept, which is a better description than degree in community condition. Based on the local community degree, local $k$-core concept is put forward. We will demonstrate the relationship between local $k$-core and $k$-clique percolation and the connection between local $k$-core and $k$-core. Finally a local $k$-core algorithm is presented to better describe the community structure of network.

3.1 Local Community Degree

General speaking, a community is a set of vertices with closer relationship under certain measure than rest of vertices in the graph. If a vertex belongs to certain community, the other vertices in the same community should have a high possibility in the vertex’s neighborhood than others not. It also shows local connectivity information of the graph, which is a good property for local algorithm. Some research focus on neighborhood properties of the graph for graph partition. Gleich [43] develops a local community method by using vertex’s neighborhood community conductance. The utility
of improved Jaccard measure of graph scarification for clustering is developed by Satuluri [44]. They all show that the neighborhood is good properties for community. When considering vertex v’s neighborhood, the vertices in the set are normally divided into two classes: some vertices are in the vertex v’s community, others is excluded the community. According to the definition of community, the connections of intra-communities are more densely than the ones in inter-communities; so as vertices in the neighborhood.

First, we recall the Graph concept; A Graph G is consisted of a set of vertices V(G) and a set of edges E(G). For undirected graph, Edges are unordered pair of distinct vertices in V(G). If two vertices are the ends of an edge, they are adjacent and called each other’s neighbor. The degree of a vertex in a graph is total number of edge incidences to the vertex. Here we have definition for neighborhood graph:

**Definition 1**: Let a node \(v_i \in V\), its neighborhood graph is \(N_i = (V_i', E_i')\), which is a subgraph induced by original Graph G. \(V_i'\) is the vertices set for neighborhood of \(v_i\); the order of the neighborhood graph \(N_i\) is \(|V_i'| = d_i\); \(E_i'\) is the edges which vertices in \(V_i'\) connected each other in the original graph.

The above description about two classes in the neighborhood demonstrates that in the neighborhood graph there is some vertices are “closer” together than others. The obvious ways to classify such classes in a vertex’s neighborhood is by checking whether the neighborhood graph \(N_i\) is a connected graph or not. In real-world graph, we find that a lot of neighborhood graph which is in reasonable size is a disconnected graph. We have new definitions below:
**Figure 3-1** Illustration of the neighborhood network for real network 1.a and 1.b from the football database and 1.c and 1.d from the NetScience database.

**Definition 2:** local connected components (LCC) are the connected components in the neighborhood graph. Let $L_j (j = 1, 2, \ldots, n)$ denote a local connected component. $V_{L_j}$ is the vertices set of $L_j$.

**Definition 3:** the order (size) of a local connected component is called local community degree; the local community degree is equal to $|V_{L_j}|$. 
From the definition 2 and 3, it is trivial to know that a vertex may have several local community degrees. It is more like an integer set instead of an integer for normal definition of degree. Here Figure 3-1 shows some small examples. Figure 3-1(a-b) are the neighborhood graph extracted by the Football Network Database [45] and Figure 3-1(c-d) are from NetScience Network Database [45]. It is obvious to see that the vertex Louisiana Monroe University has 3 LCC with local community degree set \{4,3,1\}, Florida state University has 4 LCC with local community degree set \{1,1,1,9\}. Then for Dr. Holmes is 3 LCC with \{2,3,9\} and Dr. TSALYUR is 1 LCC with \{7\}. From Figure 3-1(d), we know that the neighborhood graph sometimes is still connected, which means the whole vertices are all “closer” together and not one is left. Another observation is that the largest LCC seems to be a giant component in the graph. For example in Figure 3-1, the size of giant component show dominating in the neighborhood graph: giant component account for 50% in the vertex Louisiana Monroe University; 75% in Florida State University, then 64.3% in Dr. Holmes and 100% in Dr. TSALYUR. In Nan’s paper [46], he called the largest LCC the principal neighborhood component. As we mention before, community focuses on the cohesive of the vertices in the communities. The vertex has closer relationship one another than no-neighbors; it is probably the same idea that the vertices in principal neighborhood component have higher possibility in the vertex’s community than the other vertices in the neighborhood. It is significant in the social networks, for instance, a student majored in history, some of his friend also majored in history, and others in chemistry. He usually has more communication with their friend in history instead of those in the chemistry class.
The principal neighborhood component is not only an important feature of the neighborhood graph, but a good parameter for community. From the idea of the following the crowd in the graph [47], the vertex tends to connect to the most crowd component. Considering the connectivity as a property in the graph, the principal neighborhood component seems to be a strong cohesive attribute in the neighborhood graph. In a vertex local view, principal neighborhood component is important. While in global view, focusing on principal neighborhood component maybe overlook some other local information. For instance, in Figure 3-1 (a) focusing on the principal neighborhood component will neglect half of the vertices in the graph. The idea of Local community degree collects all the local information by defining an integer set instead of principal component degree individually. In Table 3-1, it shows the local community degree set and degree of the vertices appeared in the Figure 3-1. The Table presents the difference between the local community degree and degree. When the neighborhood graph is a connected component, the degree and the local community degree is equivalent;

<table>
<thead>
<tr>
<th>Vertex</th>
<th>LC degree</th>
<th>degree</th>
<th>PNC degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Louisiana Monroe University</td>
<td>{1, 3, 4}</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>Florida state University</td>
<td>{1,1,1, 9}</td>
<td>12</td>
<td>9</td>
</tr>
<tr>
<td>Dr. Holmes</td>
<td>{2,3,9}</td>
<td>14</td>
<td>9</td>
</tr>
<tr>
<td>Dr. TSALYUR</td>
<td>{7}</td>
<td>8</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 3-1 Local component degree set, Principal neighborhood component degree and degree of the vertex showed in the Figure 3-1

normally, each local community degree in a vertex is smaller than degree. The difference brings the structure information for the graph: if local community degree is the same as degree, It is possible that the vertex is a hotspot in the graph such as a star topology
structure or a member of the community without connecting to any vertices outside the community; if not, local community degree provides the possible size and structure of community [46].

When analyzing the graph in global view, degree of a vertex is more common to used, without any local information. Local community degree demonstrates the number of local edge incidents in partial view condition. This is critical, with the development of the modern network, the size of the graph is enormous, and some global properties like degree is no longer suitable for micro graph structure analysis. Local community degree definition is based on the local view. Such local methods make the graph analysis more practicable and less time-consuming.

3.2 Local $k$-core

The $k$-core is a subgraph $G'$ of $G$ which guarantee that the degree of each vertex in the $G'$ is at least equal or great than $k$. The $k$-core algorithm is based on the connectivity of the graph without considering the community structure. The local $k$-core subgraph proposed in this paper is an expending of the $k$-core subgraph, and we explain and compare them as follow:

**Definition 4**: A local $k$-core is a subgraph $G'$ of $G$ that each vertex of $G$ have local community degrees at least $k$ in the subgraph; that is to say $\min(V_{l_j}) \geq k$ ($j = 1,2,\ldots,n$). Analogously, the local $k$-core number of such a subgraph is $k$. we use $K_L(k)$ for local $k$-core.

**Definition 5**: A vertex $i$ has local $k$-core index $k$ if it belongs to the local $k$-core but not to $(k+1)$ core; it also has local $k$-core number $k$. The maximum local $k$-core index $k$ such that
the local $k$-core number is the maximum is denoted $k_L^{\text{max}}$. We use $K_L(k_L^{\text{max}})$ to denote the local $k$-core subgraph with maximum local $k$-core index.

Local $k$-core actually is more suitable for community structure over $k$-core, which is a relaxing subgraph of clique. We illustrate a local $k$-core example in Figure 3-2. Figure 3-2 is examples for minimum 4-vertex 2-core and local 2-core. Figure 3-2(a) is minimal 4-vertex 2-core. Figure 3-2(b) is minimum 4-vertex local 2-core and we can easily see that the local $k$-core has more edges than $k$-core and closer to a 4-vertex clique. In fact, the 4-vertex 2-core is equal to the 4-vertex local 1-core by definition. From the definition of local $k$-core, we have below theorem:

![Figure 3-2](image)

(a) K-core number=2  
(b) Local K-core number =2

Figure 3-2 $k$-core vs local $k$-core; minimum 2-core are 4 vertices and 4 edges; minimum local 2-core are 4 vertices with 5 edges.

**Theorem1.** When $k=2$, $K_L(2)$ is a set of triangles (3-clique).

**Proof:** when $n=2$, the size of local connected component in vertex’s neighborhood graph is at least 2; So the local connected component and the vertex construct a triangle (3-
clique). Since every vertex in the local connected component also has local community
degree at least 2, they also can form triangles. \( K_L(2) \) is a set of triangles.

**Theorem 2.** When \( k > 2 \), \( K_L(k) \) is a set of 3-clique percolation.

*Proof:* from theorem 1. We know that \( K_L(2) \) is set of triangles (3-clique). Suppose each
local connected component in vertex’s neighborhood graph has \( P_i \) vertices; it will make
up of \( P_i - 1 \) triangles, which share \( P_i - 2 \) edges. From the definition of clique
percolation, those triangles are 3-clique percolation with size \( P_i + 1 \). So altogether \( K_L(k) \)
is a set of 3-clique percolation.

**Theorem 3.** Let \( S(n) \) be the \( n \)-vertex clique, we have \( S(n) = K_L(n - 1) \).

*Proof:* \( S(n) \) is a \( n \)-vertex clique, according to the definition of local \( k \)-core, All the
neighborhood graphs of vertices in the \( S(n) \) are connected components whose the size
are \( n-1 \); so \( n \)-vertex clique is equivalent with local \( n-1 \)-core.

From the theorem, we demonstrate the difference between the \( k \)-core and local \( k \)-core.

The local \( k \)-core is actually based on the triangle (or 3-clique). Let \( P(k) \) (\( k > 3 \)) is the \( k \)-
clique percolation. According to the definition of the clique percolation and theorem 3, it
is easily to know that \( P(k) = K_L(k - 1) \). Since \( P(k + 1) \subseteq P(k) \), \( K_L(k - 1) \) here is
also the set of 3-clique percolation.

Why do we use the triangle (or 3-clique) as a parameter for local \( k \)-core? First, the
triangle is an important argument for clustering coefficient, which is a principle
properties in the graph; secondly, from all kind of definition of community, triangles play
a significant role in the structure. The densest example for community, clique, is actually
constructed by triangles. Some definition of community is directly based on the number of triangles in the graph. However, it is not trivial to describe the structure of the community by the number of triangles itself; other parameters are also added to compliment the definition of community. Finally, when we want to decompose the graph, connectivity is not the only choice; the triangles may also bring some interest result.

3.3 Local k-core algorithm

In section 3.2, we define the concept of local k-core and demonstrate its relationship with k-core and clique percolation. A local k-core is obtained by recursively removing all the local community degree less than k, until all the vertices in the remaining graph have the local community degree at least k. We developed the algorithm 1 for local k-core decomposition. In the algorithm 1, the input is the graph G and k; output is the local k-core subgraph. In step 2, the program is running until satisfying condition in step 10, which means the local k-core reaches a fixed point. Since the local k-core is based on the neighborhood graph, in step 3, we browse the vertices in the graph once at a time; then build a neighborhood network $H$. Each local community degree is computed by finding the local connected components. So in step 6, if the order of local community degree $|L_c|$ is smaller than $p$, all the edges which the vertex $v$ connected to each vertex in local connected components are marked. In step 9, we remove all the edges with mark. Here certain edges may be marked twice or more; we do not need to count the number of marking and the removing is based on the Boolean function of the edges (marked or not). When the graph $G'$ does not change any more, it is guaranteed that no vertices are not satisfied the definition of local k-core; the fixed point is reached; all the vertices in the graph $G'$ belong to local k-core.
Algorithm 1. Detect the Local p-core in the graph

Input Graph G (V,E), p, Output $K_L(p) \subseteq G$, $K_L(p)$ is local p-core subgraph

1) $G' \leftarrow G$;
2) repeat
3) for each $v \in V(G')$
4) $H \leftarrow N_{G'}(v)$
5) for each $L_j \in H$
6) if $|V_{L_j}| < p$
7) then for $u \in V(L_j)$
8) do $\text{mark}(v, u)$
9) $E(G') \leftarrow \{(v, u) \in E(G') | (v, u) \text{is not marked}\}$
10) until $G'$ no longer changes
11) return $K_L(p) \leftarrow G'$

Figure 3-3 is an example of Algorithm 1. Here we try to find the local 3-core subgraph, we begin with browsing the vertex A: from its neighborhood network, 3 local connected components are collected : {B, C, D}, {E}, and {F,G}; the local community degrees of those are 3,1 and 2. Next step, only the component {B, C, D} satisfies the requirement. All the edges which the vertex A connected to {E} and {F, G} are marked. Then vertex B is running the same process and so other vertices. Finally only vertices set {A, B, C, D} are in the local 3-core while those vertices are in the 4-core in k-core algorithm.

The definition of the local k-core gives a clear explanation of the question whether the Graph $G'$ is a local k-core in a fixed point. Each vertex is browsing its neighborhood graph and finds the local connected components which does not meet the local k-core
requirement; the marked edges in the local connected components are removed which affect the connectivity of the vertices in the local connected components themself. Fixed points of the local $k$-core algorithm will have each vertex in the graph $G'$ with at least local community degree $k$. Such result also demonstrates the correctness of the algorithm.

The local $k$-core algorithm focuses on the content of the connectivity which the $k$-core algorithm normally ignores. As we demonstrated before, the local $k$-core algorithm includes the information of the community structure. Although the local $k$-core algorithm is not a community detection algorithm, it shows some clique-like structure in the graph, which may be relative to the community structure and bring interest in analysis. From the definition of the local $k$-core, when the neighborhood graph has only one connected

**Figure 3-3** A small graph example for finding local 3-core according to algorithm 1. $\{A,B,C,D\}$ belongs to local 3-core.
component, the algorithm is the same as the $k$-core one. With the increase of the $k$ in local $k$-core, the local $k$-core becomes denser and more possible to have only one connected component; the local $k$-core algorithm may be degenerated to $k$-core algorithm. It should happened when $k$ number is relative large in the graph.
4. Experiment and Result

In this section, experimental results and analysis are presented. All the experiments are evaluated on a 1.86GHz Core-2 Generation CPU with 8G Ram windows-based system. The datasets can be found on Table 4-1.

<table>
<thead>
<tr>
<th>Network</th>
<th>Vertices</th>
<th>Edges</th>
<th>$d_{avg}$</th>
<th>$d_{max}$</th>
<th>r</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>FangYao</td>
<td>383</td>
<td>3944</td>
<td>20.595</td>
<td>212</td>
<td>-0.1324</td>
<td>0.7467</td>
</tr>
<tr>
<td>Net Science</td>
<td>1589</td>
<td>2742</td>
<td>3.451</td>
<td>34</td>
<td>0.4616</td>
<td>0.6378</td>
</tr>
<tr>
<td>Dolphin</td>
<td>62</td>
<td>159</td>
<td>5.129</td>
<td>12</td>
<td>-0.0436</td>
<td>0.2590</td>
</tr>
<tr>
<td>AS-JULY06</td>
<td>22963</td>
<td>48436</td>
<td>4.2</td>
<td>2390</td>
<td>-0.1984</td>
<td>0.2304</td>
</tr>
<tr>
<td>EMAIL-Enron</td>
<td>36692</td>
<td>183831</td>
<td>10.02</td>
<td>1383</td>
<td>-0.1108</td>
<td>0.4970</td>
</tr>
<tr>
<td>FOOTBALL</td>
<td>115</td>
<td>613</td>
<td>10.66</td>
<td>11</td>
<td>0.1624</td>
<td>0.4032</td>
</tr>
<tr>
<td>CA-ContMa</td>
<td>23133</td>
<td>93497</td>
<td>8.08</td>
<td>280</td>
<td>0.1364</td>
<td>0.6336</td>
</tr>
<tr>
<td>CA-AstroPh</td>
<td>18,772</td>
<td>198,110</td>
<td>21.10</td>
<td>504</td>
<td>0.2053</td>
<td>0.6308</td>
</tr>
<tr>
<td>CA-GrQc</td>
<td>5242</td>
<td>14496</td>
<td>5.53</td>
<td>81</td>
<td>0.6594</td>
<td>0.5302</td>
</tr>
<tr>
<td>CA-HepTh</td>
<td>9877</td>
<td>25998</td>
<td>5.26</td>
<td>65</td>
<td>0.2685</td>
<td>0.4717</td>
</tr>
</tbody>
</table>

Table 4-1 Databases for our Experiments; $d_{avg}$ is the average degree of the network; $d_{max}$ is the maximum degree of the network; r is the clustering assortativity; c is the clustering coefficient.

4.1 Brief of the datasets

The datasets listed in the Table 4-1 can be found on the SNAP website [48] and Newman [45]. For simplify, all the datasets are treated as undirected graph or symmetrized if it is directed. The graph will not allow the self-loop edges. In Table 4-1, we show a lot of different kind of network database:

**Collaboration network**: In these networks, the nodes represent people or object; the edges show communication relationships (like Football, Email-Enron) or collaboration ones (like CA-Hepth, NetScience). Such network usually has a comparative high clustering coefficient.
**Metabolite Network:** these nodes are DNA or metabolite; the edges show they have a positive function or chemical reaction to other metabolites or medicines. Some of the networks are hyper graph (like FangYao).

**Technology Network:** the nodes are routers or server. The edges represent the communication or physical connection between them (like AS-JULY06 and Dolphin).

**A. Comparison with local \(k\)-core and \(k\)-core algorithm**

First, we compare the local \(k\)-core and \(k\)-core algorithm by the number of vertices in \(k\)-core of different level \(k\). Table 4-2 displays the maximum local \(k\)-core index \(k_L^{max}\) and the maximum \(k\)-core index \(k^{max}\) and the number of vertices in both \(k\)-core subgraphs. It is obvious that \(k_L^{max}\) is equal to \(k^{max}\). In fact, the local \(k\)-core algorithm are removing the local connectivity of the graph (local community degree) which still a part of the global connectivity (degree); with the increase of the \(k\), each vertex in the remaining graph will connect each other, which turns out that the local \(k\)-core and \(k\)-core algorithm are equivalence. Another observation is the number of vertices in local \(k\)-core subgraph \(|K_L(k_L^{max})|\) and vertices in \(k\)-core subgraph \(|K(k^{max})|\) are not always identical. For those databases which exist a dominating Giant Component, \(|K_L(k_L^{max})|\) and \(|K(k^{max})|\) are possible to be the same. On the other hand, \(|K(k^{max})|\) and \(|K_L(k_L^{max})|\) tend to be different for those databases, whose degrees are even while local community degrees are not. It is very clear that the local \(k\)-core shows more information for the graph structure.

Figure 4-1 demonstrates comparison of the number of vertices in each \(k\)-core as a function of its index in these two algorithms. For FangYao network, the whole shape is identical. From the Theorem 2, we know that the local \(k\)-core is relative to the triangle,
which is a parameter of the clustering coefficient. FangYao network has a high clustering
coefficient \( c = 0.7467 \) which most of the vertices are well connected. In such condition,
as we mentioned before, Local \( k \)-core and \( k \)-core algorithm have the same

| Network       | \( k_l^{\text{max}} \) | \( k^{\text{max}} \) | \( d_{\text{max}} \) | \( |K_L(k_l^{\text{max}})| \) | \( |K(k^{\text{max}})| \) |
|---------------|----------------|----------------|----------------|----------------|----------------|
| FangYao       | 22             | 22             | 212            | 53             | 53             |
| NetScience    | 19             | 19             | 34             | 20             | 20             |
| Dolphin       | 4              | 4              | 12             | 19             | 36             |
| AS-JULY06     | 25             | 25             | 2390           | 71             | 71             |
| EMAIL-Enron   | 43             | 43             | 1383           | 275            | 275            |
| FOOTBALL      | 8              | 8              | 11             | 63             | 114            |
| CA-Astroph    | 56             | 56             | 504            | 57             | 57             |
| CA-ContMa     | 25             | 25             | 280            | 26             | 26             |
| CA-GrQc       | 43             | 43             | 81             | 44             | 44             |
| CA-HepTh      | 31             | 31             | 65             | 32             | 32             |

Table 4-2 The comparison with local \( k \)-core and \( k \)-core algorithm in the databases.

\( k_l^{\text{max}} \) is the maximum number local \( k \)-core; \( k^{\text{max}} \) is the maximum number of \( k \)-core;

\( |K_L(k_l^{\text{max}})| \) is the number of vertices in the local \( k \)-core subgraph when \( k = k_l^{\text{max}} \);

\( |K(k^{\text{max}})| \) is the number of vertices in the \( k \)-core subgraph when \( k = k^{\text{max}} \).

result. So as NetScience network, its shape is just slight different between these two.

When we look at those four datasets: CA-Astroph, CA-ContMa, CA-GrQc. CA-HepTh:
The Curved shape for local \( k \)-core is all below the \( k \)-core one. That means, from each step,
the number of vertices in local \( k \)-core is smaller than the \( k \)-core one. We get the same
observation from Figure 4-2. It is also pretty trivial to see large distinctions in the Curve
for Football and Dolphins network. Figure 4-1 and 4-2 prove the above statement that the
local \( k \)-cores have less vertices comparing with the \( k \)-core in the same level \( k \). Figure 4-3
shows the comparison of the number of edges in each \( k \)-core as a function of its index in
these two algorithms. The curve shape for FangYao network has highly overlapping ratio.

While on the other three databases, it is clearly that local \( k \)-core
Figure 4-1 The number of vertices in k-core as a function in FangYao, NetScience, CA-AstroPh, CA-CondMat, CA-GrQc and CA-Hepth Database
Figure 4-2 The number of vertices in k-core as a function in Email-Enron, As-July06, Football and Dolphin Database.
Figure 4-3 The number of edges the k-core as a function in FangYao, As-July06, CA-CondMat and Dolphins Database.
algorithm has fewer edges than $k$-core, which also supports the comment we displayed before.

**B. Clustering coefficient comparison of two algorithm**

In *Theorem 1* and 2, we demonstrate the local $k$-core relationship with 3-clique percolation. Figure 4-4 shows the Variation of the clustering coefficient of $k$-core in different $k$ in both algorithms. The local $k$-core algorithm always has a higher clustering coefficient in the same level $k$ than $k$-core algorithm; That is to say, the local $k$-core algorithm keeps the structure information well comparing with $k$-core. Here local $k$-core definitely is a $k$-core; while reverse is not. Another observation is when $k=2$, local $k$-core often has a big jump in the curved. As mentioned before, when $k=2$, All the non-triangle structures are removed; since the clustering coefficient is relative with the triangles’ number in the graph, the clustering coefficient increases a lot by removing the non-triangle structures. Also when $k=56$ in CA-Astroph, $k=6$ in NetScience and $k=9$ in CA-HepTh network, the clustering coefficient all finally become 1, which means the $k$-core subgraph become a set of clique. When the $k$-core subgraph become cliques, it is usually no need to decompose any further, where is kind of balance condition for the graph. It is also obvious that local $k$-core is earlier to reach such balance condition than $k$-core.

**C. Community structure in local $k$-core**

From the above two experiments, they display that the local $k$-core algorithm have more strictly requirement than $k$-core. Let us focus on the $k$-core subgraph of the database. Figure 4-5 demonstrated the variation of the giant component size of those two algorithms in different level $k$. Here the giant component is the largest connected
Figure 4-4 The Clustering Coefficient of the k-core as a function in CA-AstroPh, Email-Enron, NetScience and CA-HepTh Databases.
Figure 4-5 The Giant Component sizes and graph size of k-core as a function in CA-HepTh, As-July06, Football and Dolphins Databases.
Figure 4-6 Local 8-core in Football network, totally 63 vertices; Giant Component 21 vertices. Graph displayed by Java Jung package [49].
Figure 4-7 Local 3-core in Dolphins network, totally 36 vertices, Giant Component 20 vertices.
Graph displayed by Java Jung package [49].
components in the graph. In Figure 4-5, we notice that, for \( k \)-core algorithm, the giant component is almost the same size as the \( k \)-core graph, which means \( k \)-core is not good at showing the community structure of the graph. For local \( k \)-core Algorithm, especially in CA-HepTh, Football and Dolphins network, the giant component sizes are much smaller than \( k \)-core ones, which gives us a possible view for community structure of graph.

Figure 4-6 displays the local \( 8 \)-core of football network which shows 5 connected components; each connected component is clique-like structure. Also it is obvious to see that even in Giant Component (marked with 2) can be easily divided into 2 connected components, if we use an elementary graph clustering algorithm. The Local \( 8 \)-core has 63 vertices and the giant component size is 21; while the \( 8 \)-core has 114 vertices and the giant component size is 114.

For Figure 4-6, it is clearly that the vertices shown in the same connected component should be in the same community. Comparing with the real classified result of the football network, the assumption proves to be true. Vertices in Component 1 belong to the PAC conference; vertices in Component 3 to SEC conference; 4 to ACC one and 5 to B10 one. In the Component 2, the vertices are from B12 and C-USA conferences. The same phenomena are presented in the Figure 4-7 for Dolphins network and Figure 4-8 for CA-HepTh network. They show that the \( k \)-core subgraph always have a larger vertices number and giant component size than the local \( k \)-core at the same level \( k \): 45 vertices and giant component size 45 comparing with 36 vertices and 20 when \( k=2 \) in Dolphins network; 285 vertices and giant component size 172 comparing with 206 vertices and 57 when \( k=8 \) in CA-HepTh network. Here the local \( k \)-core algorithm displays the community structure in \( k \)-core which never shows in normal \( k \)-core algorithm.
Figure 4-8 Local 8-Core in CA-HepTh network, totally 206 vertices, Giant Component 57 vertices. Graph displayed by Java Jung package [49].
5. Conclusion and Future work

5.1 Conclusion

In this thesis, neighborhood graph tends to be a good view for discovering the community structure of the graph; then the concept of local community degree is defined by the size of connected component in the neighborhood graph. Further connection is made between local $k$-core, which guarantees at least $k$ local community degree in the subgraph, a local view of the graph, and Clique percolation, as clique–like community structure with high density. Also the local $k$-core algorithm is possible to regress to $k$-core algorithm if each neighborhood graph has only one connected component, which needs high density of the graph. The local $k$-core is an extending idea of $k$-core with extra limited condition.

Through the experiment, this thesis demonstrates that the local $k$-core algorithm can has fewer edges and vertices than $k$-core algorithm, while keeping the graph with higher density. At the same time, the local $k$-core algorithm shows a good community structure in the process of $k$-core decomposition, which is a better way to view the graph structure.

5.2 Future work

The theories and experiments we presented here is still preliminary and need to be extended by future work. First, the theory and experiments about local $k$-core only concentrate on real network data; for research, such as the random network or the low bound of the appearance the local $k$-core, are not mentioned in the thesis. It is also good idea to construct a random local $k$-core network base on the parameters like $k$ and clustering coefficient. Another new area is, instead of focusing on the local community
degree (the sizes of connected components in the neighborhood graph), the number of the connected components might be an argument worthy further research; it is also relative to the community structure and maybe is a good local method for graph. If a vertex’s neighborhood graph has many connected components, it may mean the vertex is possible a hotspot, instead of a member of community.

In the Theorem 1 and 2, we present that the local $k$-core has a relationship with 3-clique percolation. Is there other new community structure like local $k$-core which may have a connection with 4-clique percolation or even higher one? From the definition of the local $k$-core, it seems like the vertex’s neighborhoods’ neighborhood satisfies the condition of 4-clique percolation and such condition is also relative to the $k$-connected graph. It is worthy to take a look in the future.
Reference


package mlc.datamining.graph;

// Graph.java by Chen Lu May 2013

import java.io.*;
import java.util.*;

public class GraphSum{
    static String[] labels = null; // vertex labels
    int numberOfVertices = 0;
    int[] starts = null; // indices in adj starting lists
    int[] adj = null; // array for all adjacency lists

    int[] components = null; // used by findComponents
    static int depth = 0; // used by stopCriterion

    public GraphSum(int n, int[] s, int[] a){ // constructor
        numberOfVertices = n;
        starts = s;
        adj = a;
    }
}
public static GraphSum readAdjLists(String filename) { // constructor reading adjacency lists

Scanner in = null;
try {
    in = new Scanner(new File(filename));
} catch (FileNotFoundException e) {
    System.err.println(filename + " not found");
    System.exit(1);
}
int n = 0; int m = 0; // first scan determines numbers of vertices and edges
while (in.hasNextLine()) {
    n++;
    String[] terms = in.nextLine().split(" ");
    m += terms.length - 1;
}
in.close();
int[] st = new int[n + 1]; // allocate memory
int[] a = new int[m];
labels = new String[n];
try {
    in = new Scanner(new File(filename));
} catch (FileNotFoundException e) {
    System.err.println(filename + " not found");
    System.exit(1);
}
}
int j = 0;
for (int i = 0; i < n; i++) {
    st[i] = j;
    String[] terms = in.nextLine().split(\" \");
    labels[i] = terms[0]; // first word in each line is vertex label
    int len = terms.length; // others are indices of adjacent vertices
    for (int k = 1; k < len; k++) a[j++] = Integer.parseInt(terms[k]);
}
in.close();
st[n] = j;
return new GraphSum(n, st, a); // using the other constructor
}

GraphSum neighborhood(int host) { // neighborhood of host vertex as a graph
    int[] t = new int[adj.length]; // temporary array for new adj
    int offset = starts[host];
    int degree = starts[host + 1] - offset;
    int[] st = new int[degree + 1];
    int m = 0;
    for (int i = 0; i < degree; i++) {
        st[i] = m;
        int a = adj[offset + i]; // the ith neighbor of host is a
        for (int j = starts[a]; j < starts[a + 1]; j++) {
            int b = adj[j]; // b is the jth neighbor of a
            if (b == host) continue;
        }
    }
    return new GraphSum(n, st, a); // using the other constructor
}
int k = 0; for (; k < degree; k++)
    if (b == adj[offset + k]) break;
if (k < degree) t[m++] = k; // kth neighbor of host is adjacent to b
}
}
st[degree] = m;
int[] a = new int[m]; // this is the actual array for new adj
for (int i = 0; i < m; i++) a[i] = t[i];
return new GraphSum(degree, st, a);
}

void dfs(int vertex, int label){ // depth-first search finding components
    components[vertex] = label;
    int upper = starts[vertex + 1];
    for (int i = starts[vertex]; i < upper; i++){
        int d = adj[i];
        if (components[d] == -1) dfs(d, label);
    }
}

void findComponents(){
    components = new int[numberOfVertices];
    for (int i = 0; i < numberOfVertices; i++) components[i] = -1;
    for (int i = 0; i < numberOfVertices; i++)
        if (components[i] == -1) dfs(i, i); // i will be a component leader
}
int[] followCrowdLocal(int size) { // returns vertices not in the local crowd
    findComponents();
    int[] sizes = new int[numberOfVertices]; // for component sizes
    for (int i = 0; i < numberOfVertices; i++) sizes[i] = 0;
    for (int i = 0; i < numberOfVertices; i++) sizes[components[i]]++;
    int kickSize = 0;
    for (int i = 0; i < numberOfVertices; i++)
        if (sizes[i] < size && sizes[i] > 0) {
            for (int j = 0; j < numberOfVertices; j++)
                if (components[j] == i)
                    kickSize++;
            components[j] = -1;
        }
    int[] kicks = new int[kickSize]; // array for vertices not in the crowd
    int n = 0;
    for (int i = 0; i < numberOfVertices; i++)
        if (components[i] == -1) kicks[n++] = i;
    return kicks;
}

boolean stopCriterion(int degree, int depth) { // stop criterion can be substituted
    return depth > 1;
}

GraphSum removeEdges(boolean[] removes) { // remove edges and return the resulting graph
    for (int i = 0; i < numberOfVertices; i++)
        for (int j = starts[i]; j < starts[i + 1]; j++)
            if (removes[j]) {
                int p = adj[j]; int k = starts[p];
                for (; k < starts[p + 1]; k++)
                    if (i == adj[k]) break;
                if (!removes[k]) removes[j] = false; // removes[k] = true; for unilateral edge removal
            }
    int len = 0; for (int i = 0; i < adj.length; i++)
        if (!removes[i]) len++;
    int[] newAdj = new int[len];
    int[] newSt = new int[numberOfVertices + 1];
    int m = 0;
    for (int i = 0; i < numberOfVertices; i++)
        newSt[i] = m;
    for (int j = starts[i]; j < starts[i + 1]; j++)
        if (!removes[j])
            newAdj[m++] = adj[j];
newSt[numberofVertices] = m;
return new GraphSum(numberofVertices, newSt, newAdj);

int[] unfriendlocal(boolean top, int size) { // recursive unfriending top=true when depth=0
depth++;
boolean[] removes = new boolean[adj.length];
for (int i = 0; i < numberofVertices; i++) {
int base = starts[i];
int degree = starts[i + 1] - base;
for (int j = base; j < starts[i + 1]; j++) removes[j] = false;
if (!stopCriterion(degree, depth)) { // conditional recursive call
if (top) System.err.println(i + "/", numberOfVertices + " " + degree);
int[] unlinks = neighborhood(i).unfriendlocal(false, size);
for (int k: unlinks) removes[base + k] = true; // unilateral removal requests
}
}
GraphSum g = removeEdges(removes); // graph after edge removal
depth--;
if (top) { g.present(); return null; } // all done and display resulting graph
return g.followCrowdlocal(size); // return from a recursive call with removal
}

void present() { // display the graph as components
findComponents();
for (int i = 0; i < numberofVertices; i++) if (components[i] == i) {
for (int j = 0; j < numberofVertices; j++) if (components[j] == i)
System.out.print(labels[j] + " ");
System.out.println();
}