Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Visual Languages and Computing
17 (2006) 225–253
Journal of Visual Languages & Computing
www.elsevier.com/locate/jvlc

Clustering graphs for visualization via node similarities

Xiaodi Huang[a,*], Wei Lai[b]

[a]*Department of Mathematics, Statistics and Computer Science, The University of New England, Armidale, NSW 2350, Australia*
[b]*Faculty of Information and Communication Technologies, Swinburne University of Technology, P.O. Box 218. Hawthorn, VIC 3122, Australia*

Received 13 May 2004; received in revised form 18 July 2005; accepted 19 October 2005

Abstract

Graph visualization is commonly used to visually model relations in many areas. Examples include Web sites, CASE tools, and knowledge representation. When the amount of information in these graphs becomes too large, users, however, cannot perceive all elements at the same time. A clustered graph can greatly reduce visual complexity by temporarily replacing a set of nodes in clusters with abstract nodes. This paper proposes a new approach to clustering graphs. The approach constructs the node similarity matrix of a graph that is derived from a novel metric of node similarity. The linkage pattern of the graph is thus encoded into the similarity matrix, and then one obtains the hierarchical abstraction of densely linked subgraphs by applying the *k*-means algorithm to the matrix. A heuristic method is developed to overcome the inherent drawbacks of the *k*-means algorithm. For clustered graphs we present a multilevel multi-window approach to hierarchically drawing them in different abstract level views with the purpose of improving their readability. The proposed approaches demonstrate good results in our experiments. As application examples, visualization of part of Java class diagrams and Web graphs are provided. We also conducted usability experiments on our algorithm and approach. The results have shown that the hierarchically clustered graph used in our system can improve user performance for certain types of tasks.

Keywords: Graph clustering; Similarity metric; *k*-Means algorithm; Multilevel graph drawing; Graph visualization

*Corresponding author. Tel.: +61 2 6773 2118; fax: +61 2 6773 3312.
E-mail addresses: xhuang@turing.une.edu.au, xiaodih@gmail.com (X. Huang), wlai@it.swin.edu.au (W. Lai).

1045-926X/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jvlc.2005.10.003

1. Introduction

Many graph-drawing algorithms [1,5] have been developed in recent years. Most of these algorithms, however, have difficulties in dealing with large graphs that have hundreds of nodes. Apart from other techniques such as fisheye viewing, hyperbolic geometry [11], and distortion-oriented presentation [10], clustering graphs serves as one efficient method for drawing large graphs. A clustered graph is able to greatly reduce visual complexity by replacing a set of nodes in clusters with abstract nodes. Moreover, a hierarchically clustered graph produces superimposed structures over the original graph through a recursive clustering process.

Generally speaking, the purpose of cluster analysis is to organize data into different groups: data in the same group are highly similar while those from different groups are dissimilar. Clustering is a process of finding these groups based on chosen criteria. According to the criteria, the current clustering approaches are roughly classified into two broad categories: content-based clustering and structure-based clustering. Content-based clustering uses the semantic aspects of data such as category labels, while structure-based clustering takes advantage of structural information about data. In addition, structure-based clustering is domain-independent so that it is suitable for graph visualization.

For clustering a graph, a metric of a node is required to quantify its features. Based on this metric, existing approaches for partitioning graphs [3,9] are loosely divided into the following groups: *connectivity based partitions*, which use standard concepts from Graph Theory such as components, cliques, and k-cores; *distance partitions from selected subsets*, which utilize the neighbourhoods of "central" nodes; *neighbourhood-based partitions*, in which a cluster is a set of units with similar neighbourhoods such as degree partition, regular partition, and colouring; and *other approaches*, such as eigenvector methods. These approaches have already proven quite useful in graph partitioning. However, some of them cannot be applied to graph visualization directly. The reason for this is that they depend on a good initial embedding of a graph, the creation of which is very expensive, particularly for a large graph.

In the field of graph visualization, a node structural metric is widely utilized in many different forms. One simple example is the degree of a node, i.e., the number of edges connected to the node. A metric more specific to trees, called the Strahler metric, is applied to tree graphs in which nodes with the highest Strahler metric values generate a skeleton or backbone, which is then emphasized [6,7]. Using the distance metric, Botafogo et al. [17] constructed a *distance matrix* that has as its entries the distances of every node to every other node, to identify hierarchies in an organization.

A clustered graph is laid out more quickly than the original graph, as clustered nodes and edges significantly reduce graph visual complexity. The storage space of a clustered graph is, relatively speaking, less than that of the original graph. Furthermore, clustering nodes provides a basis for navigation and context clustering, which is accomplished by three possible approaches [4]:

- *Ghosting*: deemphasizing nodes, or relegating nodes to the background.
- *Hiding*: not displaying nodes with a metric under a cutoff value.
- *Grouping*: grouping nodes under a new supernode representation.

This paper presents a new approach to hierarchically clustering a graph. The key idea behind this approach is to use an abstract node so as to express a set of the most similar

nodes in a graph. This is achieved by initially grouping similar nodes from a node similarity matrix that is constructed on the basis of a novel node similarity metric. Each such group potentially represents a set of highly connected nodes in the graph. These groups are then individually replaced with abstract nodes to form a higher abstract level of the graph with a reduced dimension. This clustered graph again produces a new coarse graph and the above procedure recursively iterates until the number of nodes in the graph falls below a threshold.

Our contributions in this paper lie in proposing a new approach to hierarchically clustering a graph, and in presenting a new method for the multilevel multi-windows layout. Our approach relies on a proposed metric for the measure of similarities among the nodes in a graph, and on the well-known k-means algorithm with a new heuristic method.

The remainder of this paper is organized as follows. We present a node vector space model in the following section, and then construct a node similarity matrix in Section 3, followed by several definitions in Section 4. Section 5 presents our algorithms for clustering graphs. Section 6 discusses the experimental results including an example of visualization of part of the Java class diagrams. Section 7 describes the multilevel multi-windows approach using a clustered Web graph as an example. Related work is reviewed in Section 8. The usability experiment of our system using the proposed technique is reported in Section 9, followed by the conclusion.

2. Node vector space model

We suppose a graph denoted by $G(V, E)$ with $|V|$ nodes and $|E|$ edges where V is the set of nodes and E is the set of edges between the nodes. An edge-by-node matrix R (also called an incidence matrix) of G is defined as $R = (r_{ij})_{|E| \times |V|}$, each entry of which is constructed by

$$r_{ij} = \begin{cases} 1 & \text{if node } v_j \text{ is incident with edge } e_i, \\ 0 & \text{otherwise.} \end{cases}$$

As an example, a graph and its matrix R are shown in Fig. 1.

In the field of information retrieval, a vector is used to represent keywords or a document in a collection [13]. Similarly, each node in a graph can be encoded as a vector where each component reflects the appearance of a particular edge in the node. Conversely,

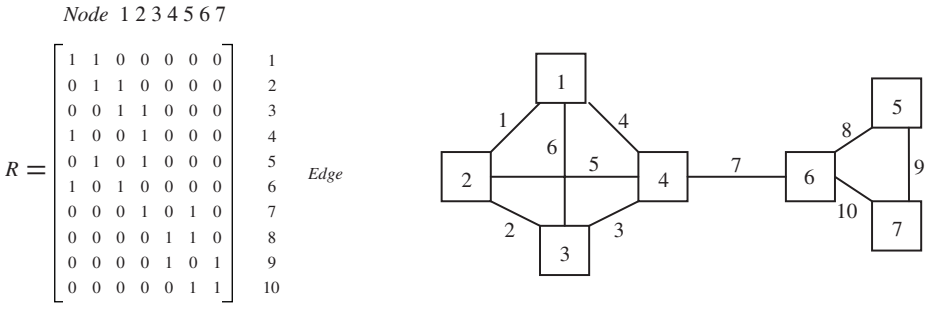


Fig. 1. An example of a graph and its edge-by-node matrix.

each edge is also encoded as an edge vector, and thus the component indicates the appearance of a particular node in that edge.

A vector representing a node is called a *node vector*. The node vectors for all the nodes in a graph are stored as the column space of matrix R. In other words, each column of R represents a node, whilst each row characterizes an edge. As a consequence, a graph is expressed as a matrix. Namely the $|V|$ node vectors may be thought of as forming the $|E| \times |V|$ node matrix, denoting the ith node vector of R as $\boldsymbol{r}_i$. If a graph G is a large graph, two characteristics of R are high dimensionality and sparsity.

3. Node similarity matrix

For the purpose of clustering a graph, a node metric should be defined to quantify an abstract feature, and then the clustering is accomplished by assigning the nodes to a group according to their metric values. Graph edit distance [2,19] and maximum common subgraph [18,19] in pattern recognition and machine vision are used to match graphs by determining the similarity of two graphs. There is, however, little literature on how to quantify similarities among the nodes within a single graph. In this paper, a node structural metric is therefore proposed to measure such similarities, making use of the number of shared edges. The similarity degree between two nodes is partly determined by the number of edges between them. In particular, the greater the number of the edges the two nodes share, the more similar they are. At the same time, the greater the number of edges they do not share, the less similar they are. To this end, a similarity measure function is needed. The measures that occur most in the literature include the dot product, Euclidean distance, and the Jaccard coefficient [3]. From them, Jaccard coefficient is able to measure the degree of overlap, which is defined as

$$sim(\boldsymbol{a}, \boldsymbol{b}) = \frac{\#(a_i = b_i = 1)}{\#(a_i = 1) + \#(b_i = 1) - \#(a_i = b_i = 1)},$$

where $\boldsymbol{a}$ and $\boldsymbol{b}$ are binary vectors. For instance, the numerator in the above equation denotes the number of an attribute i (i.e., edge) occurring in both $\boldsymbol{a}$ and $\boldsymbol{b}$.

From the above definition, the similarity degree of two nodes is calculated by their corresponding node vectors, as stated before. Recall that a node vector is a binary vector in which a component is only 1 or 0 to indicate whether or not a particular edge links to this node. Replacing $\boldsymbol{a}$, $\boldsymbol{b}$ with the node vectors in matrix R, the following equation is derived:

$$sim(\boldsymbol{r}_i, \boldsymbol{r}_j) = \frac{\boldsymbol{r}_i^{\mathrm{T}} \boldsymbol{r}_j}{\boldsymbol{r}_i^{\mathrm{T}} \boldsymbol{r}_i + \boldsymbol{r}_j^{\mathrm{T}} \boldsymbol{r}_j - \boldsymbol{r}_i^{\mathrm{T}} \boldsymbol{r}_j} = \frac{(R^{\mathrm{T}} e_i)(e_j^{\mathrm{T}} R)}{(R^{\mathrm{T}} e_i)(e_i^{\mathrm{T}} R) + (R^{\mathrm{T}} e_j)(e_j^{\mathrm{T}} R) - (R^{\mathrm{T}} e_i)(e_j^{\mathrm{T}} R)}, \tag{1}$$

where i and $j = 1, \ldots, |V|$, and e_i (or e_j) denotes the ith (or jth) canonical vector of dimension e, i.e., $e = (1, 1, \ldots, 1)^{\mathrm{T}}$.

Note that the more similar two nodes, the less links that connect them (in accordance with Eq. (1)). The degree of similarity of two nodes connecting only one edge, for example, will reach the maximum, i.e., 1. In the context of graph clustering, our intention is to group two nodes with many linking edges. This means that we attempt to find those nodes with minimal similarities among them.

It is quite reasonable to utilize the number of shared edges with regard to the measurement of node similarities if we look at the original purposes of what the nodes represent.

The use of the number of shared links is able to find a group of similar nodes in the sense that the connectivity or pattern of linkages between nodes contains a lot of implicit information about common features among the objects which the nodes represent. Namely a group of nodes containing analogous information has the likelihood of many links among them, while dissimilar nodes will have few or no links.

Consider, for example, the case of a Web graph where Web pages are represented by the nodes and hyperlinks among those pages by the edges. It is quite likely that the Web pages with similar topics have more hyperlinks among them than those with different topics. In fact, two well-known algorithms for extracting the relationships among Web pages: PageRank [12] and HITS [8], have been reported in the literature. Both of them employ a bootstrapping approach to determining the quality or "authority" of a Web page using the number and quality of the pages that it links to. Conversely, from a graph perspective, the nodes in a highly connected subgraph are likely to be similar to each other in terms of what these nodes originally represent.

We return now to (1) applying it to the graph shown in Fig. 1. According to (1), for instance, the similarities between nodes v_1 and v_4, and between nodes v_1 and v_2 are: $sim(\boldsymbol{r}_1, \boldsymbol{r}_4) = 0.167$ and $sim(\boldsymbol{r}_1, \boldsymbol{r}_2) = 0.200$, respectively. Note that the similarity score of nodes v_1 and v_4 is less than that of nodes v_1 and v_2. The reason for this is that node v_4 contributes only one shared edge from its total four edges, although nodes v_1 and v_2 have one shared edge as nodes v_1 and v_4 do. Furthermore, we observe that the similarity scores of all pairs of non-neighbour nodes are zero by (1): $sim(\boldsymbol{r}_1, \boldsymbol{r}_6) = 0$, for example. Clearly, this is not accurate in real applications. If node v_1 represents a paper with a reference paper v_4 that in turn refers to a paper v_6, then paper v_1 should be somewhat related to paper v_6. Using a transitive similarity can effectively solve this problem. In the above example, we have $sim(\boldsymbol{r}_1, \boldsymbol{r}_6) = sim(\boldsymbol{r}_1, \boldsymbol{r}_4) * sim(\boldsymbol{r}_4, \boldsymbol{r}_6)$ because an existing path from v_1 to v_6 is (v_1, v_4) and (v_4, v_6).

In general, it is possible for two non-neighbour nodes in a graph to have more than one path between them. We aim to minimize the similarity degree between the two nodes. Obviously, the longer the shortest path between two nodes is, the less likely they will be grouped in the same class. The minimum similarity degree of two nodes can consequently be considered as finding the shortest path with respect to the minimum cost in the graph where the cost of every edge is 1.

For the above reason, finding all-pairs shortest paths in a graph [27] necessitates the prerequisite step. More precisely, the shortest paths between two non-neighbour nodes, defined as a path with the fewest edges, are initially found by the well-known Dijkstra or Floyd's algorithm [27]. The products of sequentially multiplying similarity values of all node pairs in such paths are then calculated. Finally, the minimum value among those products is chosen as the degree of similarity between the two nodes. Formally, it is assumed that one of the shortest paths between nodes v_i and v_j is along a sequence of node pairs $(v_i, v_{k1}), (v_{k1}, v_{k2}), \ldots, (v_{kl}, v_j)$ such that all pairs are edges, and that their corresponding node vectors consist of a set of P: $(\boldsymbol{r}_i, \boldsymbol{r}_{k1}), (\boldsymbol{r}_{k1}, \boldsymbol{r}_{k2}), \ldots, (\boldsymbol{r}_{kl}, \boldsymbol{r}_j)$. The similarity value $s(\boldsymbol{r}_i, \boldsymbol{r}_j)$ minimizes all the values of $sim(\boldsymbol{r}_i, \boldsymbol{r}_j)$ over all the possible shortest path sets, denoted by a union set P', between nodes v_i and v_j. An equation is accordingly arrived at:

$$s(\boldsymbol{r}_i, \boldsymbol{r}_j) = \begin{cases} \min\limits_{P \in P'} \left\{ \prod_{(r_m, r_n) \in P} sim(\boldsymbol{r}_m, \boldsymbol{r}_n) \right\} & \text{if } P' \neq \phi, \\ 0 & \text{if } P' = \phi, \end{cases} \tag{2}$$

where $1 \leqslant i,j \leqslant |V|$ and $i \neq j$. Note that the calculation of similarities between two neighbour nodes can be treated as a special case of that of two non-neighbour nodes in the above equation.

With the combination of (1) and (2), the node similarity matrix of a graph is constructed, where each entry is computed by (2) which tells the extent to which two nodes are similar. The node similarity matrix is thus derived:

$$S = [s(\boldsymbol{r}_i, \boldsymbol{r}_j)]_{|V|\times|V|}.$$

In a case of the graph in Fig. 1, its symmetric similarity matrix is shown in Table 1.

In the following we provide three theorems related to the metric for node similarity.

Theorem 1. *Given a graph $G = (V, E)$, let $p = \langle v_1, v_2, \ldots, v_k\rangle$ be the shortest path from node v_1 to node v_k that minimizes $s(\boldsymbol{r}_1, \boldsymbol{r}_k)$. For any i and j such that $1 \leqslant i \leqslant j \leqslant k$, let $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j\rangle$ be the subpath of p from node v_i to node v_j. Then, p_{ij} is the path that minimizes $s(\boldsymbol{r}_i, \boldsymbol{r}_j)$.*

Proof. If we decompose path p into $v_1 \xrightarrow{p_{1i}} v_i \xrightarrow{p_{ij}} v_j \xrightarrow{p_{jk}} v_k$, then we have $s(\boldsymbol{r}_1, \boldsymbol{r}_k) = s(\boldsymbol{r}_1, \boldsymbol{r}_i) \times s(\boldsymbol{r}_i, \boldsymbol{r}_j) \times s(\boldsymbol{r}_j, \boldsymbol{r}_k)$ according to (2). Now assume that there is a path p'_{ij} from node v_i to node v_j with $s(\boldsymbol{r}'_i, \boldsymbol{r}'_j) < s(\boldsymbol{r}_i, \boldsymbol{r}_j)$. Then $v_1 \xrightarrow{p_{1i}} v_i \xrightarrow{p_{ij}} v_j \xrightarrow{p_{jk}} v_k$ is a path from node v_1 to node v_k whose similarity $s'(\boldsymbol{r}_1, \boldsymbol{r}_k) = s(\boldsymbol{r}_1, \boldsymbol{r}_i) \times s(\boldsymbol{r}'_i, \boldsymbol{r}'_j) \times s(\boldsymbol{r}_j, \boldsymbol{r}_k)$ is less than $s(\boldsymbol{r}_1, \boldsymbol{r}_k)$, which contradicts the definition that $s(\boldsymbol{r}_1, \boldsymbol{r}_k)$ is the minimal similarity degree of node v_1 to node v_k. □

We can use this theorem to reduce the computation complexity of the node similarity matrix.

Theorem 2. *Given any two nodes v_i and $v_j (i \neq j)$ with the shortest path of length p, and $p \geqslant 3$ in a connected graph G that has its diameter denoted as diam(G), the bound of their similarity degree is $1/(2|V| - 3)^{diam(G)} \leqslant s(\boldsymbol{r}_r, \boldsymbol{r}_j) \leqslant 9/(4 \times 3^p)$.*

Proof. Suppose that the shortest path between nodes v_i and v_j is a sequence of node pairs $[(v_i, v_{k1,}), (v_{k1}, v_{k2,}), \ldots, (v_{kl}, v_{j,})]$ such that there are p edges. We first prove the right part of the equation.

Table 1
The node similarity matrix of the graph in Fig. 1

$$S = \begin{bmatrix} 1.000 & & & & & & \\ 0.200 & 1.000 & & & & & \\ 0.200 & 0.200 & 1.000 & & & & \\ 0.167 & 0.167 & 0.167 & 1.000 & & & \\ 0.007 & 0.007 & 0.007 & 0.042 & 1.000 & & \\ 0.028 & 0.028 & 0.028 & 0.167 & 0.253 & 1.000 & \\ 0.007 & 0.007 & 0.007 & 0.042 & 0.333 & 0.250 & 1.000 \end{bmatrix}$$

According to (1) and (2), we have

$$
\begin{aligned}
s(\boldsymbol{r}_i, \boldsymbol{r}_j) &= \prod_{m=i,n=k1}^{kl,j} s(\boldsymbol{r}_m, \boldsymbol{r}_n) = \prod_{m=i,n=k1}^{kl,j} \frac{1}{deg(v_m) + deg(v_n) - 1} \\
&\leqslant \prod_{m=k1,n=k2}^{kl-1,kl} \frac{1}{1+2-1} \times \frac{1}{deg(v_m) + deg(v_n) - 1} \times \frac{1}{2+1-1} \\
&\leqslant \frac{1}{1+2-1} \times \underbrace{\frac{1}{2+2-1} \times \cdots \times \frac{1}{2+2-1}}_{p-2} \times \frac{1}{2+1-1} = \frac{9}{4 \times 3^p},
\end{aligned}
$$

where the degree of node v_m or v_n denoted by $deg(v_m)$ and $deg(v_n)$ is at least 2 except that $deg(v_i)$ or $deg(v_j)$ can be 1. Note that more multiplying items in the above equation will make $s(\boldsymbol{r}_i, \boldsymbol{r}_j)$ smaller.

Now we show the correctness of the left part of the equation. It is easy to verify it if we know that the maximum possible degree of a node is $|V|-1$, and that the diameter of a graph denoted by $diam(G)$ is the longest distance between any two nodes. In fact, there is no restriction on the shortest path for the left part; that is, the similarity degree of any two nodes in a graph has the lower bound. Therefore we have proven that the claim is correct. □

By Theorem 2 we should find those nodes with fewer similarities between them and not far away from each other in order to group highly connected nodes as a subgraph. This corresponds well with intuition.

Eq. (2) is in fact a metric to quantify the degree of similarity between two nodes within a graph, which satisfies fundamental properties: non-negativity, reflexitivity and symmetry.

Theorem 3. *For any two nodes v_i and v_j in a connected graph G, and their corresponding node vectors $\boldsymbol{r}_i$ and $\boldsymbol{r}_j$, respectively, where $i \neq j$, the following properties hold true*:

1. $0 \leqslant s(\boldsymbol{r}_i, \boldsymbol{r}_j) \leqslant 1$,
2. $s(\boldsymbol{r}_i, \boldsymbol{r}_i) = 1$,
3. $s(\boldsymbol{r}_i, \boldsymbol{r}_j) = s(\boldsymbol{r}_j, \boldsymbol{r}_i)$.

By means of the similarity matrix S, we are able to easily obtain the dissimilarity matrix D, which will be used as an input of the k-means algorithm in Section 5:

$$D = [1]_{|V| \times |V|} - S.$$

4. Definitions

In this section we provide a few definitions, beginning with rephrasing the node vector used in the previous sections as Definition 1.

Definition 1. *Node vector:* A vector denoted by $r \in R^{|E|}$ represents a node with respect to the appearances of the edges in a given graph $G = (V, E)$. The component of this vector is "1" if the node is incident with an edge and "0" otherwise.

Definition 2. *Abstract node:* A supernode or metanode representing a subgraph of highly connected nodes in a graph. The node vector of an abstract node is equivalent to the arithmetic mean of all node vectors of the nodes in such a subgraph.

Grouping the "similar" nodes into a number of clusters is achieved minimizing the mean of node similarity values. An abstract node represents the "cluster centre" or median of a cluster.

Suppose l denotes an abstract level of a clustered graph, i.e., $l = 0$, the graph is the original graph without clustering. Also, suppose that el stands for the number of edges in the vector space model, and nl is the total number of nodes in the lth abstract level of the graph. Given the vector space model, the lth level ($l \geqslant 1$) of node vectors is represented by $\boldsymbol{r}_1^l, \boldsymbol{r}_1^l, \ldots, \boldsymbol{r}_{nl}^l$ where each $\boldsymbol{r}_i^l \in R^{el}$. Clearly, the node vectors of the (l+1)th abstract level are directly derived from the node vectors of the lth abstract level. Let $\pi_1^{l+1}, \pi_2^{l+1}, \ldots, \pi_{ml}^{l+1}$ denote a partitioning of the lth abstract level of the node vectors into ml disjoint clusters, i.e.,

$$\bigcup_{i=1}^{ml} \pi_i^{l+1} = \{\boldsymbol{r}_1^l, \boldsymbol{r}_2^l, \ldots, \boldsymbol{r}_{nl}^l\} \quad \text{and} \quad \pi_i^{l+1} \cap \pi_j^{l+1} = \phi \ \text{ if } i \neq j.$$

For each fixed $1 \leqslant i \leqslant ml$, the mean vector or the centroid of the node vectors contained in the cluster π_i^{l+1} is

$$\boldsymbol{m}_i^l = \frac{1}{nl_i} \sum_{\boldsymbol{r} \in \pi_i^{l+1}} \boldsymbol{r},$$

where nl_i is the number of node vectors in cluster i. So we get the (l+1)th abstract level of the norm *abstract node vector* as $\boldsymbol{r}_i^{l+1} = \boldsymbol{m}_i^l / ||\boldsymbol{m}_i^l||$. As a result, the l+1th abstract level of the node vector set is $\cup_{i=1}^{ml} \boldsymbol{r}_i^{l+1}$.

In the field of graph visualization, the focus is mainly on the similarities among a set of nodes. For this reason, an object function is used to evaluate the "coherence" of clustering:

$$Q = \sum_{j=1}^{ml} \sum_{\boldsymbol{r}_i \in \pi_j^{l+1}} ||\boldsymbol{r}_i - \boldsymbol{r}_j^{l+1}||.$$

Intuitively, this objective function measures the combined coherence of all ml clusters for the (l+1)th abstract level of a clustered graph. We attempt to minimize its value to achieve the best clustering results.

Definition 3. *Abstract edge:* An aggregated edge that embodies existing edges between any member nodes of a cluster and other nodes not belonging to this cluster. In other words, internal edges among members in a cluster disappear while external edges, between cluster members and other outside nodes in the graph, are contracted into abstract edges. Suppose that $E_{v_c}^l$ denotes a set of the lth abstract level edges of the abstract node v_c, then we have

$$E_{v_c}^l = \{e_{ij} | v_i \in C^l \wedge v_j \in G - C^l\},$$

where C^l denotes a set of the nodes in a cluster represented by an abstract node v_c.

Definition 4. *Seed nodes*: A set of nodes whose degrees are greater than $\mu + \tau$, denoted by N, where μ is the average degree of a graph G and τ is a user-specified threshold.

Let σ be the *standard deviation* (a statistics term) of node degrees in graph G. The degree of node v_q denoted by $deg(v_q)$ is the number of edges incident with node v_q. The sum of the degrees of nodes over G is $2|E|$, given that every edge has exactly two endpoints. Therefore the average degree of G is $\mu = 2|E|/|V|$.

The threshold τ is usually assigned as a value related to σ such as $0.1\ \sigma$, because the *standard deviation* σ reflects the spread of the degree distribution. In a special case of a fully connected graph where the degree of every node is equal to the average degree μ, there will be no *seed node*. The construction of clusters can then start from any node in the graph.

Definition 5. *k-nearest neighbour*: Given a query node $v_q \in G$ and an integer parameter $k \leq |V|$, the k-nearest neighbour search returns a set $k\_NN(v_q) \subseteq G$ that contains at least k nodes in G, and the following conditions hold:

$$d(v_i, v_q) \leqslant (v_j, v_q), \quad \forall v_i \in k\_NN(v_q), \quad \text{and} \quad \forall v_j \in G - k\_NN(v_q),$$

where $d(v_i,v_q)$ is the shortest distance between nodes v_i and v_q. This means that the distance between a query node v_q and any node in $k\_NN\ (v_q)$ is not greater than that between v_q and any other nodes outside $k\_NN\ (v_q)$. As a consequence, the resulting set $k\_NN\ (v_q)$ is $\{v_i | d(v_i, v_q) \leqslant k \wedge v_i \in G \wedge v_i \neq v_q\}$. Within the graph shown in Fig. 1, for example, we have $1\_NN\ (v_4) = \{v_1, v_2, v_3, v_6\}$, which represents a set of nodes with which node v_4 incidents. Note that node v_q is not included in the set of $k\_NN(v_q)$.

Eades and Feng [14] defined a hierarchically clustered graph as follows:

Definition 6 (*Eades and Feng [14]*). *Hierarchically clustered graph:* A graph G and a rooted tree T whose leaves are exactly the nodes of G, denoted by $C = (G, T)$. T is called the inclusion tree, and represents a recursive inclusion relation called a clustering of G. Internal nodes of T are called abstract nodes or clusters.

It is obvious that a cluster represents a group of its children nodes and children clusters. Ultimately the cluster represents a group of both its children nodes, and recursively the children of its children clusters. For example, the root of T represents a cluster that groups all the nodes of G. As the clustering is an inclusion relation, one of the clusters contains the other if two clusters contain a common node.

Any two clusters cannot contain a common node in our approach. The tree T reflects the abstract levels of a graph and the exclusion relations between parent and children clusters. The edges of a clustered graph in our approach are replaced with abstract edges that include only the edges among nodes that are within different clusters.

Up to this point, we have presented a node similarity matrix and some definitions. It follows that a clustering algorithm should be chosen by which to find clusters from the matrix. The most important and challenging characteristics of the matrix are high dimensionality and sparsity, as mentioned before. For the purpose of efficiency, it is important that the chosen clustering algorithm should exploit the sparsity of the data while producing useful results at the same time. The k-means algorithm satisfies these requirements and hence is our choice of algorithm.

5. Algorithms

From the matrix R and the metric for quantifying node similarities, we obtain the node similarity matrix S of a given graph. It makes possible for clustering the nodes to be based

on such a matrix. Indeed, many clustering algorithms are able to accept a similarity matrix as an input. Here we choose the k-means algorithm on the grounds that it is simple to program and easy to compute on large samples. There are, however, problems with this technique: specification of the number of clusters beforehand, and initial choices of the cluster centroids. In the following sections, an algorithm is first developed to efficiently identify the *seed node* set in a given graph. The number of members and each member node of this set are equivalent to the number of clusters and the initial nodes of the cluster centroids, respectively. The k-means algorithm, together with the *seed node* set as its initial input, is then applied to the node similarity matrix. Finally, the main algorithm is provided for hierachically clustering graphs.

5.1. Determination of the number of clusters and initial nodes

A number of different schemes have been developed for selecting an initial set of *seed nodes* as the centroids of clusters [9]. A commonly used one selects the seeds at random. A clustering solution is then computed using each one of these seeds. The quality of such clusters is evaluated by computing the similarity of each node to the centroid vector of the cluster that it belongs to. The best solution is the one that maximizes the sum of these similarities over the entire set of nodes.

Starting with arbitrary random centroids is, however, a relatively poor solution. An efficient method is presented here to determine the number of clusters and then to choose the initial centroids of the clusters.

The basic idea of this approach is to employ the *seed nodes* in Definition 4. Intuitively, a node with a relatively higher degree, i.e., more connectivities to other nodes, should form a local "community" together with the nodes around it.

The proposed algorithm detects as the s*eed nodes* the nodes whose degrees are greater than the average degree of a graph. Later, these nodes potentially act as the initial members of different clusters. In some cases, two or more *seed nodes*, however, are densely connected and they are not far away in the graph. This suggests they should stay within one cluster. Two or more such *seed nodes* will be combined into one *seed node*, if they share the nodes of their *k-nearest neighbour node* sets, denoted by *k_NN*, and if the number of the shared nodes is no less than half a degree of one of them. Each remaining member of the reduced *seed node* set serves as the initial member of each cluster.

The algorithm for identifying a set of *seed nodes* is given in Fig. 2.

In the above algorithm, a node degree is calculated by summing up the entries of the corresponding column in $\boldsymbol{R}$, namely $deg(v_i) = \boldsymbol{e}_i^{\mathrm{T}} \cdot \boldsymbol{r}_i$. The maximum number of display nodes $N_{\max}$ is determined by the area of a display screen and the average occupied area of a node.

5.2. k-Means algorithm with seed nodes

For the k-means algorithm, the *agglomerative clustering* is employed which starts with the n-way partition and constructs iteratively a k-way partition from the $(k+1)$-way partition. The algorithm consists of a simple re-estimation procedure as follows. First, the nodes in the *seed node* set N are individually assigned to the $|N|$ cluster sets as their initial memberships. The centroid is then computed for each clustering set. These two steps are

Input: graph $G=(V, E)$, *seed set* $N \leftarrow \phi$, threshold k and τ

Output: the *seed node* set N and the number of the clusters nl

Compute the degree of each node in G, $\deg(v_i)$.

Compute the average degree μ and the standard derivation of the degrees σ

$\mu = 2|E|/|V|$

//Find the *seed node* set N

for each $v_q \in V$

if $\deg(v_q) \geq \mu + \tau$ **then**

$N \leftarrow N \cup \{v_q\}$

end if

end for

if $N = \phi$ **then** STOP

for each $v_i \in N$

if $\max_{v_j \in N - \{v_i\}} |(k\_NN(v_i) + \{v_i\}) \cap (k\_NN(v_j) + \{v_j\})| \geq \deg(v_j)/2$ **then**

$N \leftarrow N \setminus \{v_j\}$

end if

end for

if $|N| > N_{max}$ **then**

$nl = N_{max}$

randomly remove the number of $|N|$-N_{max} nodes from N

else

$nl = |N|$

end if

Fig. 2. An algorithm to find *seed nodes*.

alternated until a stopping criterion is met, i.e., when there is no further change in the assignment of the nodes.

The k-means algorithm for clustering nodes is described in Fig. 3.

In the above algorithm, the distance between a node and a cluster centroid equals the sum of the dissimilarity values between this node and all the nodes in this cluster, which are the entries of node dissimilarity matrix D.

5.3. Main algorithm for clustering graphs hierarchically

Fig. 4 describes our algorithm for hierarchically clustering a graph. The main algorithm begins with constructing the edge-by-node matrix R from a given graph G, by expressing each node as the column and each edge as the row. The node dissimilarity matrix D is derived from matrix R using the metric of node similarity. The s*eed nodes* set at each abstract level are then identified by the algorithm in Fig. 2. Next, recursively applying the

Inputs:

$D=[d_{ij}]_{|V|\times|V|}$ (a node dissimilarity matrix to be clustered)

N (a *seed node* set)

V (the node set of a graph G)

Outputs:

$C=\{C_1, C_2, \ldots, C_{|N|}\}$ (cluster centroids)

$m: V \rightarrow \{v_1, v_2, \ldots, v_{|N|}\}$ (cluster membership)

// Set N to C as initial members

$C=N$

for each $v_i \in V \setminus N$

$m(v_i) = \arg\min_{j=1\ldots|N|} \text{distance}(v_i, C_j)$

end

while m has been changed and $\sum_{i=1}^{|N|} |\{v_k \mid m(v_k)=i\}| < |V|$

for each $i \in \{1 \ldots |N|\}$

Recompute C_i as the centroid of $\{v_k \mid m(v_k)=i\}$

end

for each $v_i \in V \setminus N$

$m(v_i) = \arg\min_{j=1\ldots|N|} \text{distance}(v_i, C_j)$

end

end

distance (node v_i, cluster C) {

for each $v_c \in C$

$(v_i, v_c) \rightarrow (v_i, v_j)$

$sim(v_i, v_C) = d_{ij}$

end

$\text{distance} = 1/|C| \sum_{v_C \in C} sim(v_i, v_C)$

}

Fig. 3. The *k*-means algorithm.

k-means algorithm to the dissimilarity matrix produces the clustered graphs. Finally, a sequence of coarse graphs G^l is laid out by the Spring Embedding algorithm [16]. The algorithm is summarized in Fig. 4.

The MMD at the end of the algorithm will be described in Section 7.

Theorem 4. *The time complexity of the algorithm for hierarchically clustering a graph is* $O(|V|^2)$.

Input: $G = (V, E)$ and a threshold

Outputs: Multiple window layouts of clustered subgraphs of G^l

l=0

Do

Construct the edge-by-node incident matrix R of graph G^l

Find the shortest paths between node pairs, if they exist, by using Dijkstra's algorithm

Calculate similarity values of node pairs according to (2)

Construct the node similarity matrix S

$D^l = [1]_{|V|\times|V|} - S$

Find the *seed nodes* set N according to the algorithm in Fig. 2

Apply the K-means algorithm to D^l with the *seed nodes* set N

Replace all nodes in cluster memberships $m(V^l)$ with abstract nodes

Add abstract edges

Store the clustered graph to G^l

$l = l+1$

While $l <$ threshold or $|V|=1$

Input the abstract level l

MMD: Layout C^l

Fig. 4. The main algorithm for hierarchically abstracting a graph.

Proof. The sketch of proof is as follows: The running time for constructing the similarity matrix is $O(|V|^2/2 + |V|/2)$, because this matrix is symmetric. The simplest implementation of Dijkstra's algorithm stores nodes of set V in an ordinary linked list or array. In this case, the running time is $O(|V|^2)$. For sparse graphs; that is, graphs with much less than $|V|^2$ edges, as most cases in graph visualization, Dijkstra's algorithm can be implemented more efficiently by storing the graph in the form of adjacency lists and by using a binary heap or Fibonacci heap as a priority queue. With a binary heap, the algorithm requires $O((|E| + |V|)\log|V|)$ time, and the Fibonacci heap improves this to $O(|E| + |V|\log|V|)$ [27]. It takes $O(|V|^2/2)$ running time to calculate the similarity values of node pairs. Finding the s*eed nodes* set N requires $O(|N|^2)$ computation where $|N|$ is the number of clusters. The time complexity of the k-means algorithm is $O(|V||N|l)$, where l is the number of iterations. The do–while loop is bounded by the value of the threshold, which is the number of the abstract levels. The threshold is 6 in our experiments. Therefore, the worst case of overall running time of the algorithm for hierarchically abstracting a graph is $O(|V|^2)$. □

6. Experiments

All the figures in this section are snapshots extracted from our experimental examples. As we mention in the next section, different abstract views are displayed in various layer

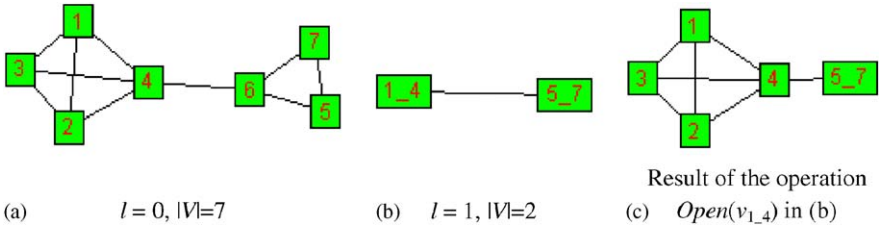


Fig. 5. The layouts of hierarchically clustered graph of Graph 1 in Fig. 1.

windows. Note that some of the following figures have been adjusted due to the limitation of the page size.

A node set $\{v_1, v_2, v_3, v_4, v_6\}$ includes all the nodes in Graph 1 in Fig. 5(a), whose degrees exceed the average degree of the graph. According to the algorithm in Fig. 2, the *seed nodes* set in this instance should be reduced to $\{v_4, v_6\}$ in that nodes v_1, v_2, v_3 and v_4 share at least three nodes of their 1_*NN* with each other. Fig. 5(b) shows the result after applying the *k*-means algorithm with the above s*eed nodes* set as initial cluster members. The $Open(v_{1-4})$ operation in Fig. 5(b) results in the layout in Fig. 5(c), where node v_{1-4} is an abstract node with the first abstract level.

In real applications, the layout sequence of a clustered graph starts from top to bottom in the abstract level tree. The display sequence of Graph 2, for example, is Figs. 6(a)–(c). Figs. 6(a) and (b) show the results of the $ExpandAll(G^2)$ and $ExpandAll(G^1)$ operations defined in the next section, respectively. Alternatively, users are able to interact with the layouts. As an example, the $Open(v_{18\_21})$ operation reveals details of the abstract node v_{18_21} in Fig. 6(b). Note that the labels of the abstract nodes are basically the combination of the labels of their children nodes omitting the numbers in sequence.

The nodes in the graphs in Figs. 7(b) and (c), originally clustered from Graph 3 in Fig. 7(a), vary in colour and size to highlight the relationships between various abstract levels of the layouts.

In our experiments the parameter of the threshold τ is 0 in the algorithm in Fig. 2. The increase of the threshold τ, which is related to standard derivation of node degrees, makes possible the reduction of the number of clusters.

Fig. 8 is a realistic example of hierarchically abstracting graphs, with the original part of Java 1.4 class diagrams in Fig. 8(a) and its clustered graphs in Figs. 8(b) and (c). The underlying purpose of this visualization is to focus mainly on the exploration of the relationships between Java classes, rather than the representation of their inherent hierarchical structures, which may be suitably displayed by a traditional tree drawing. In order to better reveal the relationships between different abstract level views, slightly varied sizes and colours of the nodes are employed to visually encode them. Specifically, the graphs, from the most abstract level view to detailed views, consist basically of the following nodes and edges: the biggest and red, the second biggest and green, the second smallest and blue, and the smallest and white nodes. The readability of the layout will be significantly improved in the presence of only one abstract level view such as the biggest and red nodes. Moreover, users in our prototype are able to interact with the layouts through the operations of MMD presented in the next section (Figs. 13 and 14).

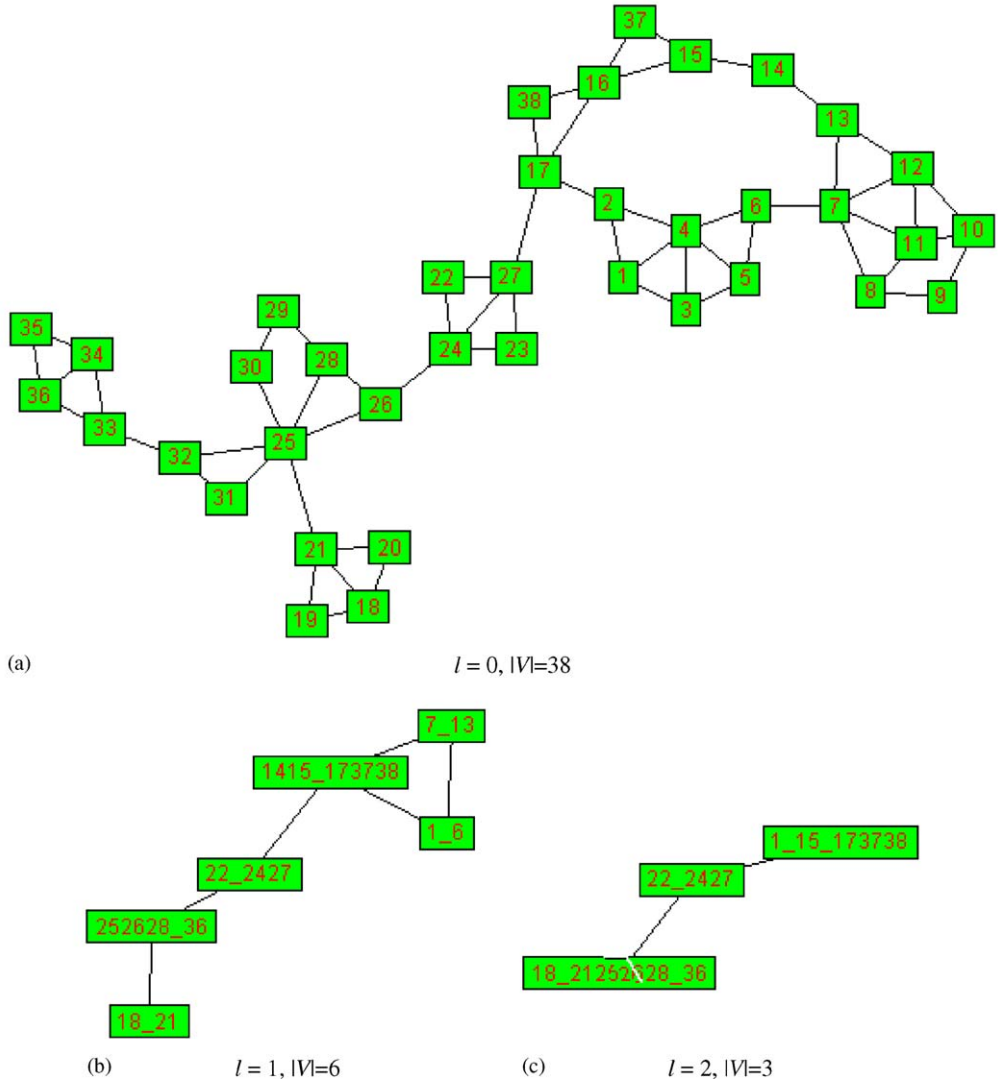
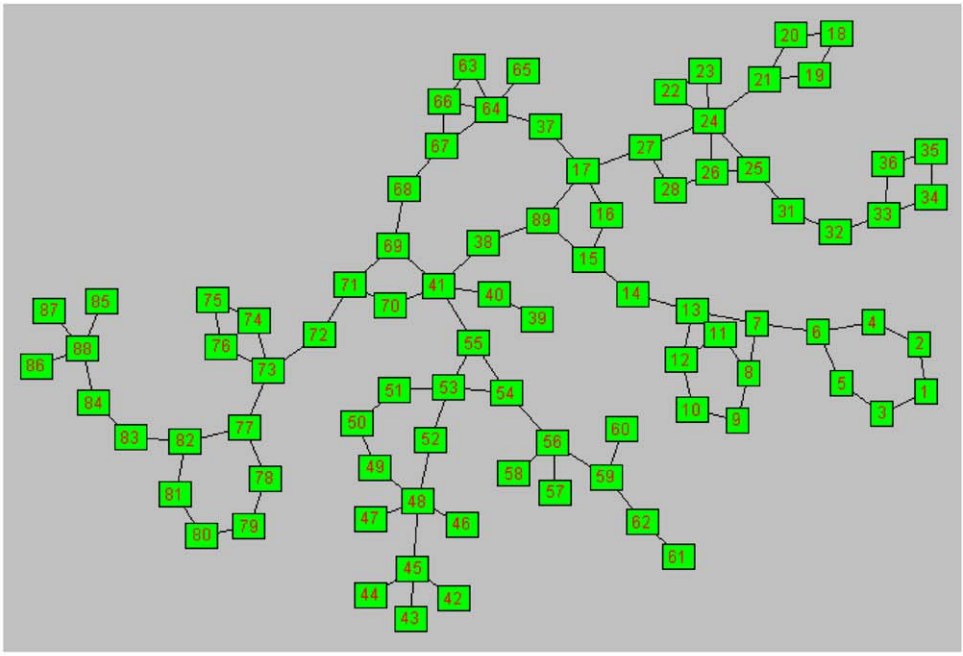


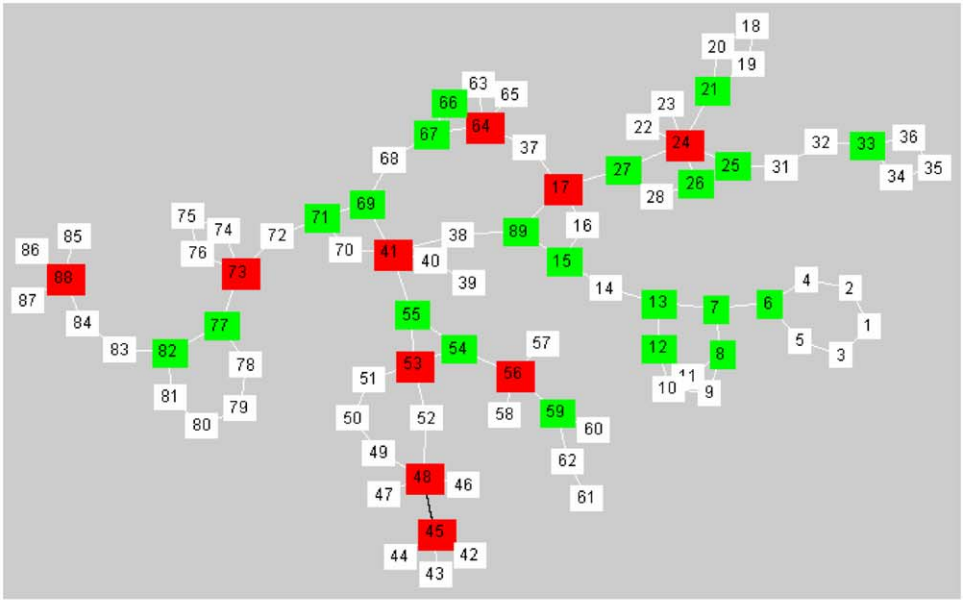
Fig. 6. The layouts of hierarchically clustered graph of Graph 2.

Table 2 lists more results from experiments with our algorithm, which were run on a 1.60 GHz Pentium machine with 512 MB of RAM. Fig. 9 also illustrates these results.

We compare our algorithm to the algorithm by Wu et al. [28]. The approach of Wu et al. found $\{v_1, v_2, v_3, v_4, v_{10}\}$, $\{v_9\}$, and $\{v_4, v_5, v_6, v_7, v_8, v_{10}\}$ in the graph shown in Fig. 10 while our approach identified $\{v_1, v_2, v_3, v_4\}$, $\{v_5, v_6, v_7\}$, and $\{v_8, v_9, v_{10}\}$. As the second example shown in Fig. 11, our approach is able to find three clusters $\{v_1, v_3, v_4, v_5, v_6, v_2\}$, $\{v_8, v_9, v_{15}, v_{16}, v_{17}, v_{10}\}$, and $\{v_7, v_{14}, v_{11}, v_{18}, v_{12}, v_{13}\}$. In contrast, Wu et al's algorithm found $\{v_1, v_2, v_{16}, v_{13}\}$, $\{v_3, v_4, v_5, v_6\}$, and $\{v_8, v_9, v_{10}, v_{15}, v_{17}, v_7, v_{11}, v_{12}, v_{14}, v_{18}\}$. Based on the connectivity of nodes, our algorithm is capable of identifying

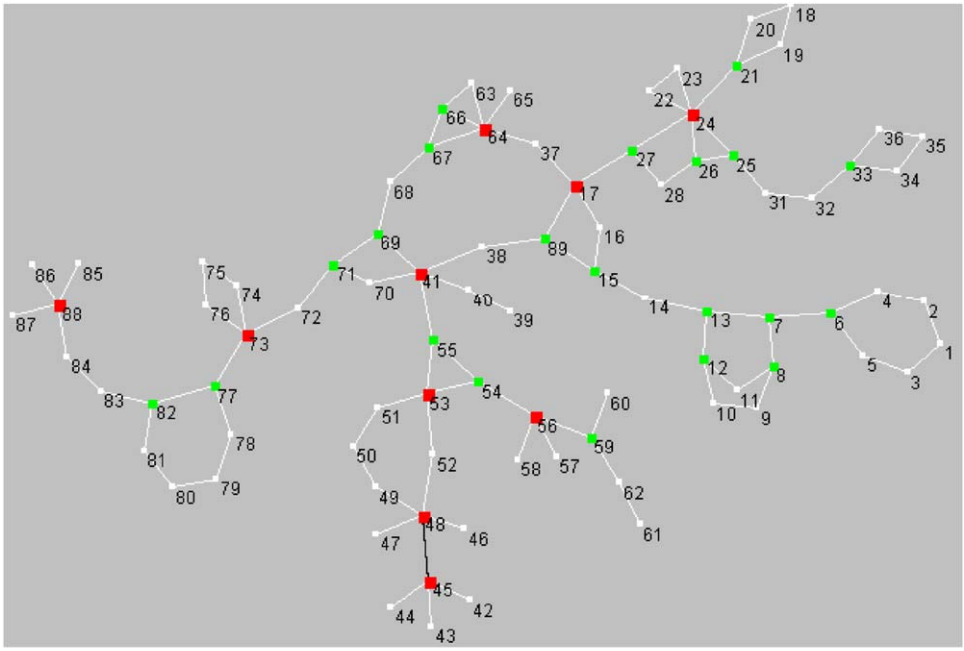


(a) $l = 0$



(b) l = 1(red),2(green) and 3, with inside labels

Fig. 7. The layout of hierarchically clustered graph of Graph 3.



(c) $l = 1$(red),2(green) and 3, without inside labels

Fig. 7. (*Continued*)

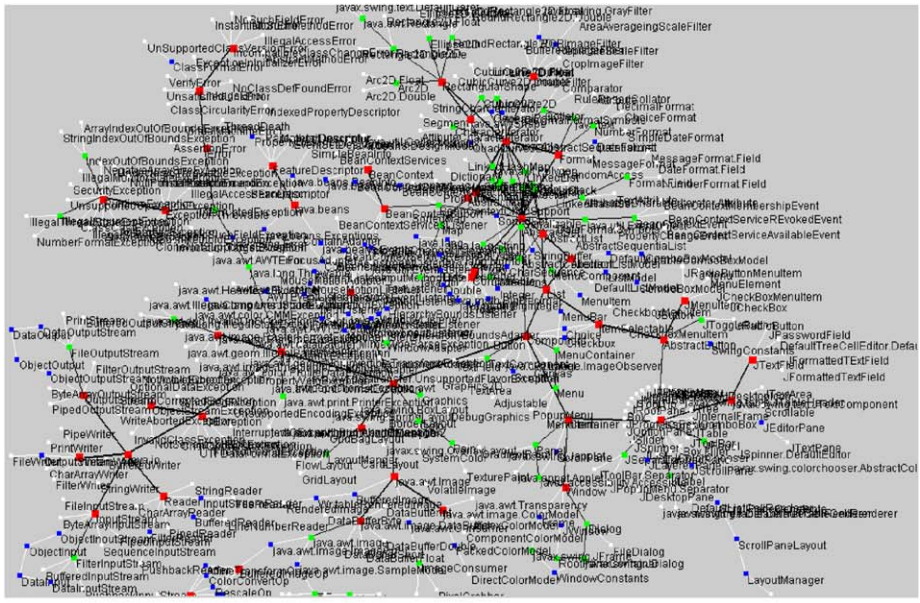
the highly connected clusters in a given graph, and outperforms the algorithm of Wu and Leahy.

7. Multilevel multi-window drawings

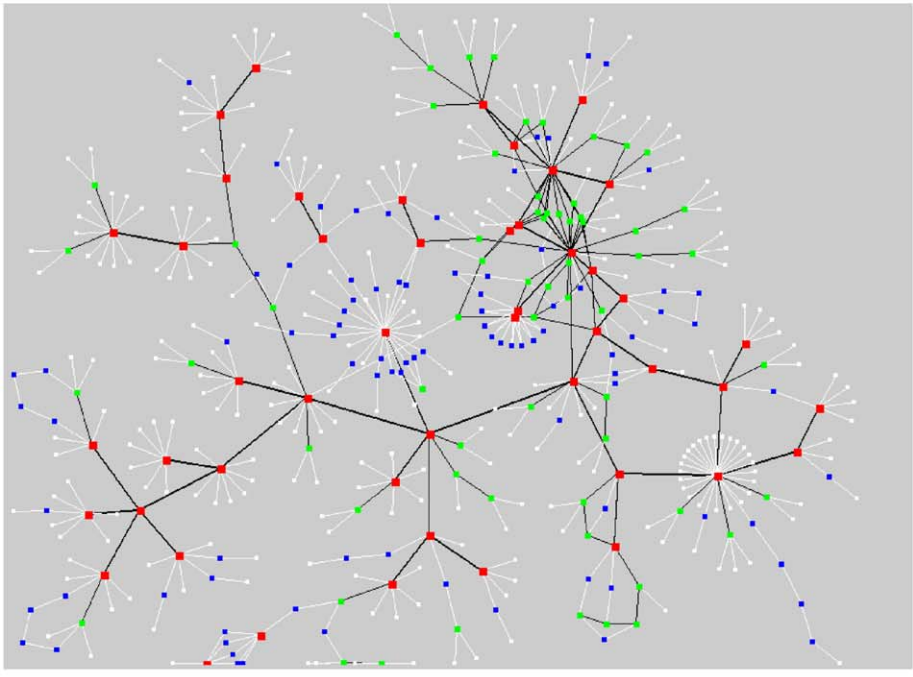
After a clustered graph has been obtained from an original graph using the algorithms described above, a further issue is how to display it in a way that maximizes its readability.

Our approach called MMD [15] makes use of multi-windows to gradually display multilevel abstract views of the clustered graph. The basic characteristics of MMD are as follows:

- A sequence of windows in the x–y plane to display a drawing of each view from the leaf level (level 0) to the root level of a tree T, where the view of abstract level l is drawn on the lth window. The z-level layer of a window corresponds to the abstract level of a clustered graph. The bigger the value of z, the more abstract the clustered graph.
- The windows can be interactively opened or closed by users.
- There are two display modes: one in which users open (or close) an abstract node to navigate the lower (or higher) abstract level view in order to display its cluster member nodes; another in which users are allowed to expand or collapse all abstract nodes and edges in a specified abstract level on the screen.

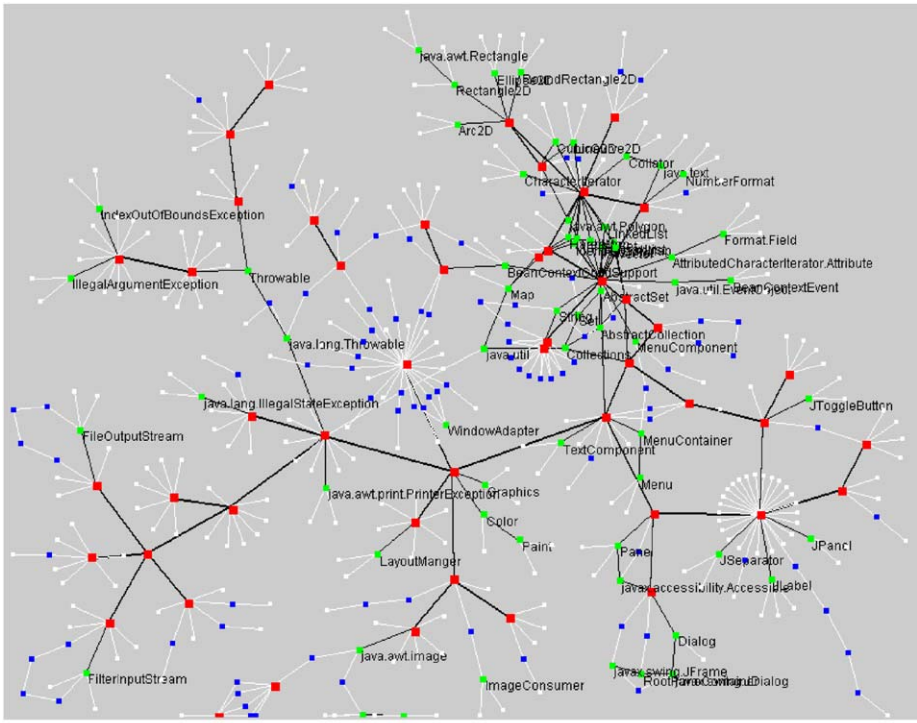


(a) Layout of Java 1.4 class diagrams by the Spring algorithm



(b) Layout of all abstract levels without labels of Java class names (l=0)

Fig. 8. Layout of part of Java class diagrams by MMD.



(c) Layout of the second abstract level with Java class names ($l = 1$)

Fig. 8. (*Continued*)

With the above descriptions, the MMD is defined below.

Definition 7. *MMD*: Given a graph $G = (V, E)$, MMD refers to a sequence of derived graphs, which are recursively generated by using a clustering algorithm:

(1) $G^0 = (V^0, E^0), G^1 = (V^1, E^1), \ldots, G^l = (V^l, E^l)$

$G^{l+1} = C(G^l)$, where l is an abstract level

and another sequence of display windows:

(2) $W^0, W^1, \ldots, W^k$,

where k is an integer.

The relationships between subgraphs and windows are:

(3) $G^l \rightarrow W^k$

$|V^l| \rightarrow sizeOf(W^k)$ and $k \geqslant l$.

In other words, the graph G^{l+1} is derived from the previous abstract level of the graph G^l via clustering. All composition node members of a non-singleton cluster will be replaced with an abstract node in V^l and the edges with an abstract edge in E^l, and then the coarse

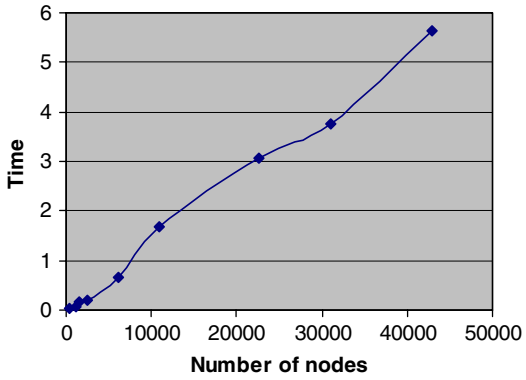


Fig. 9. Number of nodes and running times of the clustering.

Table 2
More experiment results of the algorithm

# Nodes	# Clusters	Time (s)
409	5	0.034
1120	8	0.079
1501	12	0.178
2532	18	0.213
6100	59	0.645
11,005	128	1.674
22,700	240	3.261
31,000	340	3.467
43,000	452	5.621

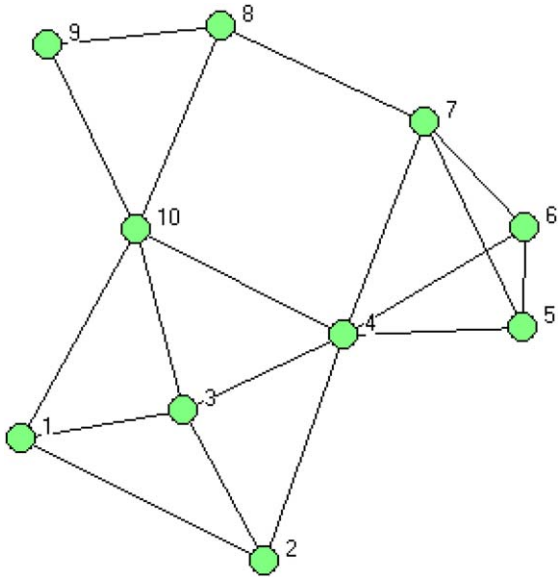


Fig. 10. The first example for comparing our algorithm to Wu and Leahy's.

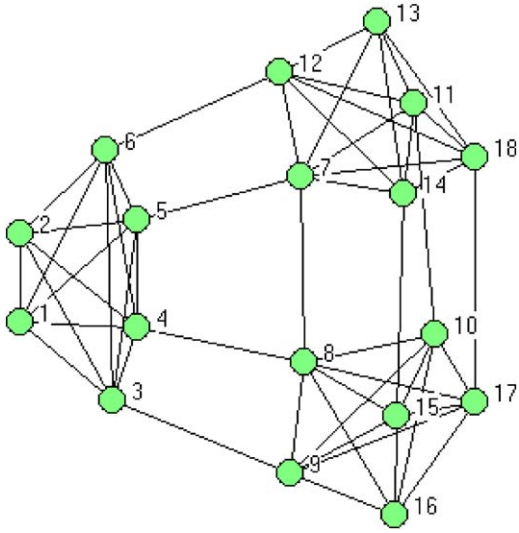


Fig. 11. The second example for comparing our algorithm to Wu and Leahy's.

graph becomes the initial graph in the (l+1)th abstract level drawing. Obviously, we have $|V^{l+1}| \leqslant |V^l|$. The ith abstract level of the graph G^i ($i = 0, \ldots, l$) will be separately drawn in different windows, the sizes of which are dynamically determined by both the number of nodes in the graph and users' interaction.

Definition 8. *Operations of MMD*: The typical operations on MMD are:

- *Open*(v_q): where $v_q \in V^l$: Open an abstract node v_q to display all its cluster member nodes and related abstract edges.
- *Close*(v_q): where $v_q \in V^l$: Contract all current display nodes and edges in a cluster into an abstract node and edge.
- *ExpandAll*(G^l): Layout all the nodes and edges, including abstract nodes and edges, in G^l in a current display window.
- *CloseAll*(G^l): Close all currently open abstract nodes and edges.
- *Shrink*(G^l): Shrink currently open windows in order to free up more space for the focus window.

In the following, part of the Web site of Swinburne University of Technology is used as an example to systematically illustrate the previously proposed algorithms and approaches. Fig. 12 illustrates an overall picture of MMD, where there are two parts to its interface: the left part is the Web graph, and the right one is the content display of a Web page. Except for some abstract nodes, every node in the Web graph is linked to a URL. For example, the node with a label www.swin.edu.au is directly linked to the corresponding Web page, as shown in Fig. 12.

This example was extracted from the Web site of Swinburne University of Technology using the software called WebCrawler. The program takes as input a starting URL address and exploration depth, and then analyses hyperlinks among Web pages. A URL text file

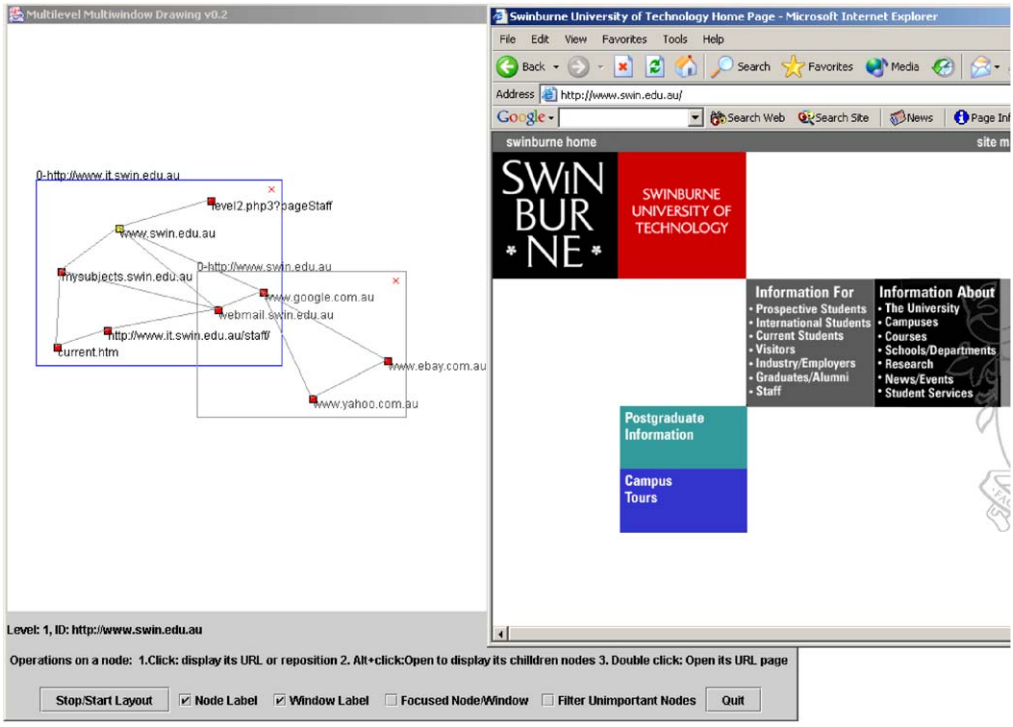


Fig. 12. The interface of the MMD system of clustering.

containing all the extracted URL addresses of two hyperlinked Web pages is finally generated for further processing.

The clustering algorithm is applied to the Web graph generated from the above URL text file, where a node represents a Web page, and an edge indicates a hyperlink between two Web pages. For simplicity, we have chosen to show the highest abstract level of the clustered Web graph in Fig. 13(a). With one expanded abstract node from Fig. 13(a), Fig. 13(b) illustrates its children nodes and corresponding edges in the second abstract level of the Web graph.

Two implemented operations of MMD are illustrated in Figs. 13 and 14, with the *Open* operation shown in Fig. 13 and the *Close* operation in Fig 14. In particular, Fig. 13(b) shows the result of opening the abstract node labelled "*current*" in Fig. 13(a), while Fig. 14(b) is obtained as a result of sequentially closing the two nodes labelled "*program*" and "*current*" in Fig. 14(a).

8. Related work

Although numerous algorithms for cluster analysis have been reported in the literature [29], we briefly review those approaches that are closely related to the structure of a graph, and compare them with our approach.

Matula [20–23] used high connectivity in similarity graphs for cluster analysis, based on a cohesiveness function. This function defines every node and edge of a graph to be the maximum edge-connectivity of any subgraph containing that element. The k-connected subgraphs of the graph are obtained by deleting all elements with cohesiveness less than k

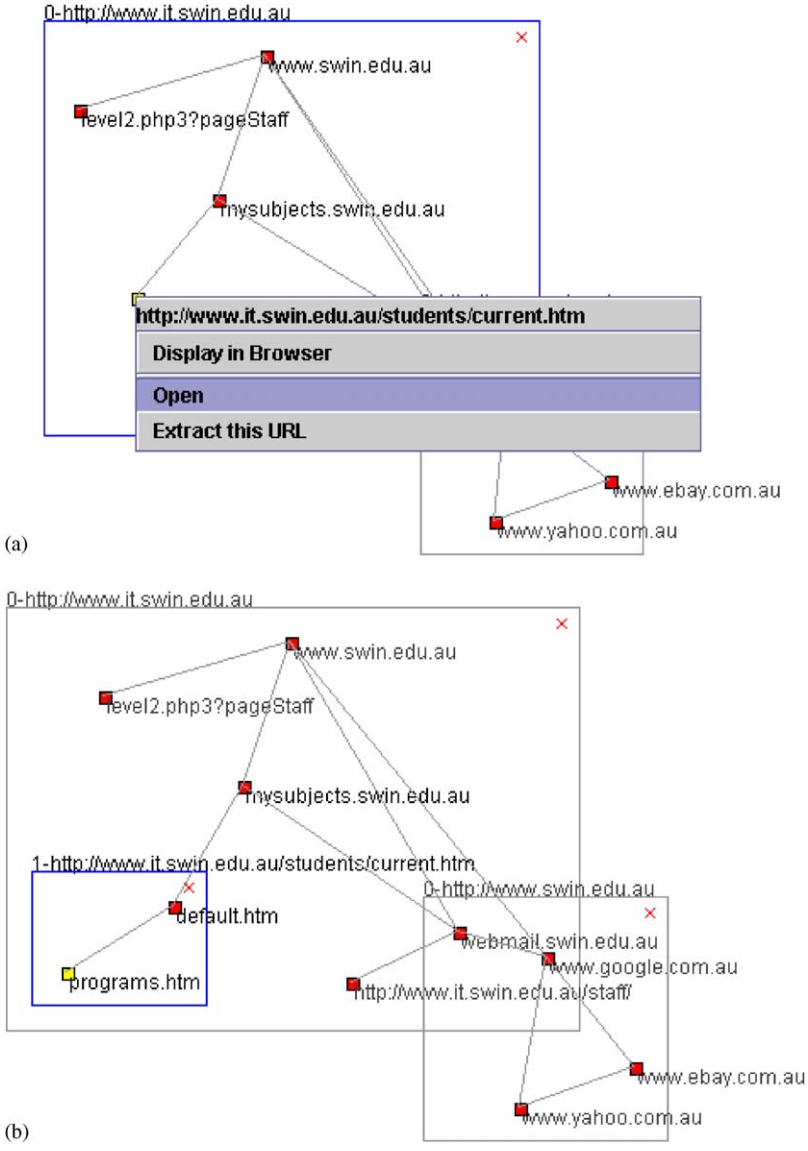


Fig. 13. The *Open* operation of MMD.

in the graph, where k is a constant value. However, it is hard to determine the values of connectivity in real clustering applications with this approach.

There is some recent work related to the clustering of a graph. The HCS algorithms [25] use a similarity graph as the input data, recursively partitioning a current set of elements into two subsets. Highly connected subgraphs are then identified as kernels that are considered as clusters if the number of their edges exceeds half of their corresponding nodes. Unfortunately the result of the clustering is not uniquely determined. Using the same basic scheme as HCS to form kernels, the CLICK algorithm [24] builds on a statistical model. It includes the following processing: singleton adoption, recursive clustering process on the set

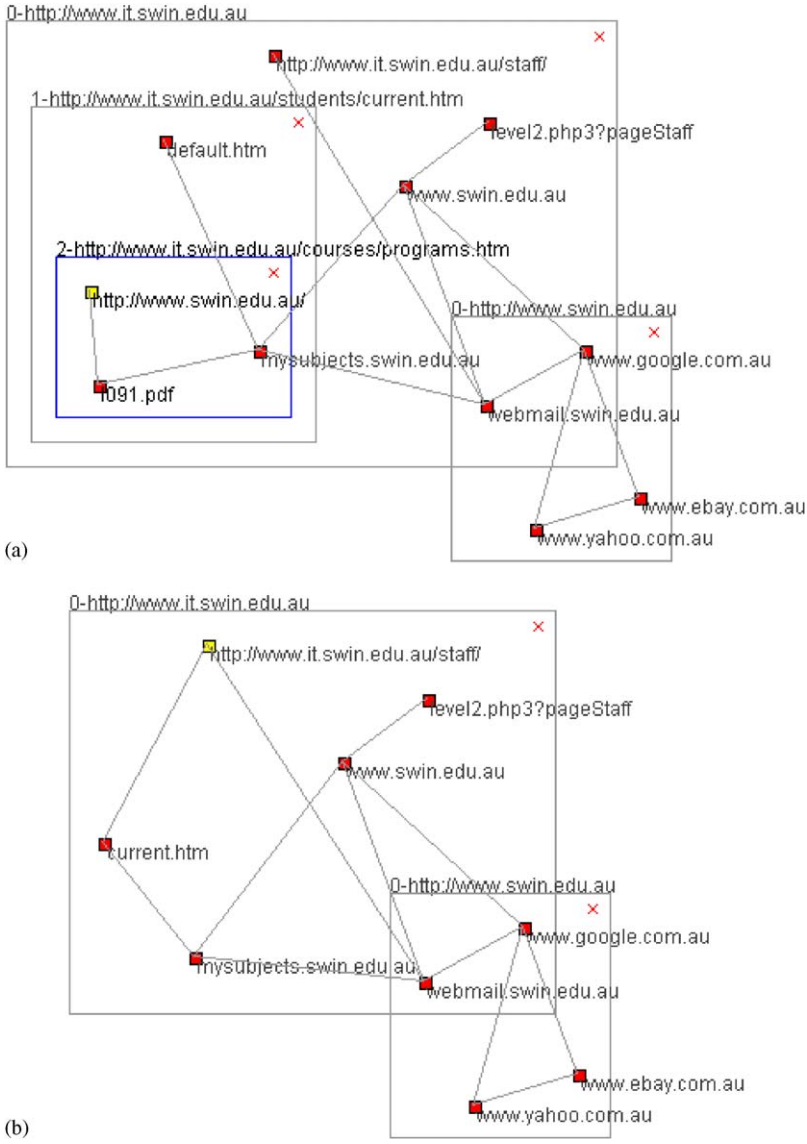


Fig. 14. The *Close* operation of MMD.

of remaining singletons, and an iterative merging step. The CAST [26] starts with a single element and uses a single parameter t. Elements are added or removed from the cluster if their affinities are larger or lower than t, until the process stabilizes.

The features of our algorithm and approach differ from the previous work in the following aspects:

- Our algorithm for graph clustering does not require the initial layout of a graph, and it is suitable for clustering any types of graphs including tree graphs. This is because it is based purely on the structure of a graph.

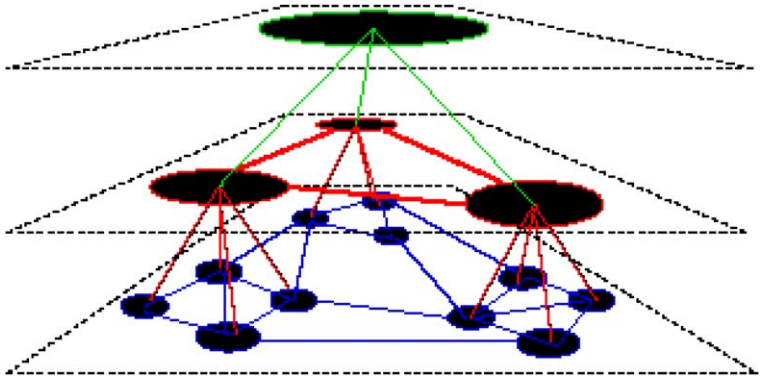


Fig 15. A multilevel drawing (From [14]).

- Without the need to be specified in advance, the number of clusters is automatically detected on the basis of the node degree distribution of a given graph. This differs from the k-means algorithm.
- The number of clusters could be easily adjusted by changing the thresholds.
- More importantly, a metric for the measurement of node similarity within a graph is provided. This may lead to potential application areas such as pattern recognition and Web page clustering.

For the layout of clustered graphs, Eades and Feng [14] proposed a *multilevel drawing* of a clustered graph $C = (G,T)$. This drawing includes:

- A sequence of the x–y planes to represent a drawing of each view from the leaf level (level 0) to the root level of T, where the view of the abstract level l is drawn on the plane $z = l$.
- A three-dimensional drawing of T, with each clustered node of T of the abstract level l drawn as a point on the plane $z = l$.

An example of the multilevel drawing provided by Eades and Feng is shown in Fig. 15.

The obvious advantage of this approach is that the number of all clustered subgraphs can be simultaneously drawn as a whole on one screen. It is, however, impossible to entirely display a large graph using such an approach. Our approach overcomes this drawback by utilizing multilevel and multi-window displays.

9. Usability experiment and discussions

In order to demonstrate the usability of our clustering algorithm and the layout approach, in this section we report an experiment of our MMD system with and without clustered graphs as its user interfaces. In particular, this experiment compares the performance of subjects in answering questions via browsing the Java class documents, accompanied by a graph or a hierarchically clustered graph.

The interfaces used in the experiment are called the *clustering interface* and *non-clustering interface*. Within both interfaces, there is a graph on the left where the nodes represent classes and the edges indicate the relationships between the classes. Both interfaces allow subjects to click on a node to display the corresponding document of a

particular class. Their appearances are similar to the graphs shown in Fig. 12. The maximum abstract level of the graph in the *clustering interface* is 6, meaning that subjects could view the graph at scale 0–5. The initial detailed graph was shown at scale 0, which is exactly the same as that of the *non-clustering interface*. Subjects could only interact with the *non-clustering interface* by clicking on the nodes to display the documents. From the user perspective, the only difference in the two interfaces is the hierarchically clustered graph. The system interfaces are otherwise identical. Timing information and user events are automatically recorded in the system.

We designed six tasks for each interface. The tasks used in the experiment include two types: the "overview" or "relations" between classes; for example, which class is on the top of the AWT event hierarchy? (java.awt.AWTEvent), what is the relationship between the Canvas class and the Graphics class? (Canvas object provides access to a Graphics object via its paint ( ) method), and what is the highest level event class of the event-delegation model? (java.util.EventObject class); a straightforward type of question about classes such as name two subclasses of the TextComponent class (TextField and TextArea) and which containers may have a MenuBar? (Frame).

Twenty-four subjects participated in the experiment, 18 males and 6 females. All were familiar with graphical user interfaces, and none were familiar with our system prior to participating in our experiment. They have basic knowledge of object-oriented programming.

The procedure for the experiment was as follows. Subjects were first introduced to the two interfaces and given a short training task to perform. They then performed the tasks using the *clustering interface*. Half of the subjects also performed the same tasks with the *non-clustering interface*.

After reading the description of the tasks, subjects clicked on a node to see the document of a class. For each task, subjects were asked to proceed to the next one once they had done it for 3 min. After the experiment, subjects filled out a form about which interface they preferred, how they oriented themselves within the hierarchy of clustering, what they liked and disliked about the system, and also other comments.

We analysed the subject's spent times on completing each task. For the completion times of certain tasks such as the "overview" of the relations between classes, the results show that subjects who used the *clustering interface* were faster than subjects who only used the *non-clustering interface*. It is also demonstrated that both interfaces have no difference in the completion times for those tasks that are about the lower levels of the classes.

We analysed the number of correct solutions as a percentage of the whole number of tasks. This was done by simply grading each task as correct or incorrect. Although most subjects did complete the task successfully, 12.5% of them did not. We found no significant difference in the accuracy between the two interfaces. However, tasks regarding the relationships between the super-classes in the Java language were more often answered correctly by using the *clustering interface* than by using the *non-clustering* interface.

We also analysed each subject's number of operations on the clustered graph per task, which provides a quantitative measure of the amount of navigation required to complete the task. With a minimum of 2 operations required, subjects using the *clustering interface* used 3.5 operations on average. It is interesting to note that the most used operation is the *Open*( ) as defined in Definition 8.

One effect beyond those analyses is worth noting. There was little difference in the performance of subjects when locating the lower level of class documents within the

hierarchy using either the *clustering* or *non-clustering* system. This is because the Java documents already provide clear information about the lower level classes independent of the graphs showing the structure of the documents.

In summary, subjects performed certain types of tasks more efficiently using the *clustering interface* than using the *non-clustering interface* technique. In particular, they took less time to complete the "overview" or "relation" tasks, and the amount of navigation was reduced. This corresponded well to the fact that subjects using the *clustering interface* were able to focus directly on this type of task and avoid the need for mentally forming high-level relations among classes. In the questionnaires most subjects also stated that the context provided by the *clustering interface* is a valuable resource for completing certain types of task.

Subjects scored the *clustering interface* significantly higher on the satisfaction questions and commented that the hierarchically clustered graph shows the whole picture of relations and that the high abstract level of the clustered graph was useful for navigation.

Twenty-one subjects preferred to use the *clustering interface*, while three subjects preferred the *non-clustering interface*. The main reasons why subjects prefer the *clustering interface* are listed below:

- The high abstract level of the clustered graph provides information about the relations on the documents. For example, one subject wrote "the clustered graph helps me know the relations among java classes".
- The clustered nodes support navigation. For example, one subject wrote "It is easy for me to switch one class from another".
- The abstract nodes are helpful when examining their children nodes. For example, one subject wrote "I can quickly find the subclasses by opening the abstract node".
- The clustered nodes and edges support comparison of classes. For example, one subject wrote "I followed the edges in the graph to look for the classes".

From their comments and responses to the questionnaires, most subjects greatly preferred the extra context provided by the *clustering interface*. It allowed them to concentrate directly on the task and reduced their cognitive burden of trying to form the structure of the entire class documents.

The three subjects who preferred the *non-clustering interface* provided the following comments:

- The level of abstraction. For example, one subject mentioned that there are too many abstract levels.
- The cluttered layout of graph. Two subjects commented that it is difficult to work on a cluttered graph.
- The labels of nodes. One subject wrote "Sometimes, I was confused by the node labels."

In addition, two others qualified their preference of the *clustering interface* by saying that their choice would depend on the task type and the size of the graph.

We compared the usability of user interfaces with and without a hierarchically clustered graph that was generated by our algorithm and approach. Subjects scored the interface of the hierarchically clustered graph higher on the subjective satisfaction questions, and 87% preferred this interface. We believe that graphs organized with multiple levels are likely

preferable to single-level graphs without clustering in terms of accuracy, task completion time, and satisfaction, particularly for those tasks that involve high-level relations between objects.

10. Conclusions

Graph clustering has applications in many areas such as hypermedia systems, Web communities, and graph drawing. This paper has presented a new approach to clustering a graph. This approach constructs the node similarity matrix of a graph based on a novel metric of node similarity, and then applies the *k*-means algorithm to this matrix in order to obtain a hierarchical abstract of the graph. A heuristic method for determining the number of clusters from a given graph is also proposed to overcome the inherent drawbacks of the *k*-means algorithm. The main advantages of our approach are: (1) no requirement for the initial layout of a graph; (2) the number of clusters is estimated from the node degree distribution of the graph; (3) provision of a metric for measurement of node similarity within a graph. The experiments have demonstrated good results, and an example of visualization of part of Java class diagrams as well as a Web graph is presented. We have also illustrated by examples how to lay out hierarchically clustered graphs in different abstract level views using a proposed method called MMD. This method makes full use of the limited screen space to reduce the visual complexity, thereby improving the readability of large graphs. As well as validating our technique we conducted usability experiments. The results have shown that our approach to producing a hierarchically clustered graph improves the performance of subjects. Our future work will include exploring various applications to which the proposed approach can be applied.

Acknowledgements

We would like to thank the anonymous reviewers for their helpful comments.

References

[1] A. Garg, R. Tamassia, Advances in graph drawing, Lecture Notes in Computer Science 778 (1994) 12–22.
[2] H. Bunke, On a relation between graph edit distance and maximum common subgraph, Pattern Recognition Letter 18 (8) (1997) 689–694.
[3] B. Everitt, Cluster Analysis, third ed., Edward Arnold, London, 1993.
[4] D. Kimelman, B. Leban, T. Roth, D. Zernik, Reduction of Visual Complexity in Dynamic Graphs, in: Proceedings of the Symposium on Graph Drawing GD '93, Springer, Berlin, 1994.
[5] G.D. Battista, P. Eades, R. Tamassia, I.G. Tollis, Graph Drawing: Algorithms for the Visualization of Graphs, Prentice-Hall, Englewood Cliffs, NJ, 1999.
[6] I. Herman, M. Delest, G. Melancon, Tree visualization and navigation clues for information visualization, Computer Graphics Form 17 (2) (1998) 153–165.
[7] I. Herman, M.S. Marshall, D.J. Melançon, M. Duke, Delest, J.-P. Domenger, Skeletal images as visual cues in graphs visualization, Data Visualization '99, in: Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization, Springer, Berlin, 1999, pp. 13–22.
[8] J.M. Kleinberg, Authoritative sources in a hyperlinked environment, Journal of the ACM 46 (5) (1999) 604–632.
[9] J. Hartigan, Clustering Algorithms, Wiley, New York, 1975.
[10] T.A. Keahey, The generalized detail-in-context problem, in: G. Wills, J. Dill (Eds.), Proceedings of the IEEE Symposium on Information Visualization (infoviz '98), Los Alamitos, IEEE Computer Soc. Press, Silver Spring, MD, 1998.

[11] J. Lamping, R. Rao, P. Pirolli, A Focus + Context Technique based on Hyperbolic Geometry for Viewing Large Hierarchies, ACM CHI '95, ACM Press, New York, 1995.
[12] L. Page, S. Brin, R. Motwani, T. Winograd, The pagerank citation ranking: bringing order to the Web, Technical Report, Computer Science Department, Stanford University, 1998.
[13] W.B. Michael, D. Zlatko, R.J. Elizabeth, Matrices, vector spaces, and information retrieval, SIAM Review 41 (2) (1999) 335–362.
[14] P. Eades, Q.-W. Feng, Multilevel Visualization of Clustered Graphs, in: Proceedings of the Symposium on Graph Drawing GD '96, Springer, New York, 1997, pp. 101–112.
[15] X.D. Huang, W. Lai, Multiple level exploration and layout of large Web hypergraph, in: Proceedings of the International Conference on Internet Computing, 2002, pp. 615–621.
[16] P. Eades, A heuristic for graph drawing, Congressus Numerantium 42 (1984) 149–160.
[17] R.A. Botafogo, E. Rivlin, B. Schneiderman, Structural analysis of hypertexts: identifying hierarchies and useful metrics, ACM Transactions on Information System 10 (2) (1992) 142–180.
[18] L.G. Shapiro, R.M. Haralick, Structural descriptions and inexact matching, IEEE Transactions on Pattern Analysis and Machine Intelligence 3 (1981) 504–519.
[19] R. Horaud, T. Skordas, Stereo correspondence through feature grouping and maximal cliques, IEEE Transactions on Pattern Analysis and Machine Intelligence 11 (11) (1989) 1168–1180.
[20] D.W. Matula, The cohesive strength of graphs, in: G. Chartrand, S.F. Kapoor (Eds.), The Many Facets of Graph Theory, Lecture Notes in Mathematics 110 (1969) 215–221.
[21] D.W. Matula, Cluster analysis via graph theoretic techniques, Paper presented at the Proceedings of the Louisiana Conference on Combinatorics, Graph Theory and Computing, Winnipeg, 1970.
[22] D.W. Matula, *k*-Components, clusters and slicings in graphs, SIAM Journal of Applied Mathematics 22 (3) (1972) 459–480.
[23] D.W. Matula, Graph theoretic techniques for cluster analysis algorithms, in: V. Ryzin (Ed.), Classification and Clustering, 1987, pp. 95–129.
[24] R. Sharan, R. Shamir, CLICK: a clustering algorithm for gene expression analysis, in: R.S. Miyano, T. Takagi (Eds.), Currents in Computational Molecular Biology, Universal Academy Press, New York, 2000, pp. 6–7.
[25] E. Hartuv, R. Shamir, A clustering algorithm based on graph connectivity, Information Processing Letters 76 (4–6) (2000) 175–181.
[26] B.D. Amir, R.S.Y. Zohar, Clustering gene expression patterns, Journal of Computational Biology 6 (3–4) (1999) 281–297.
[27] H.C. Thomas, E.L. Charles, L.R. Ronald, S. Clifford, Introduction to Algorithms, 2/e, MIT Press, Cambridge, MA, 2001.
[28] Z. Wu, R. Leahy, An optimal graph theoretic approach to data clustering: theory and its application to image segmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence 15 (11) (1993) 1101–1113.
[29] J. Han, M. Kamber, Data Mining: Concepts and Techniques, Morgan Kaufmann Publishers, Los Altos, CA, 2000.