A Customizable Hybrid Approach to Data Clustering

Yu Qian Kang Zhang
Department of Computer Science, The University of Texas at Dallas, Richardson, TX 75083-0688, USA
qianyu_cs@yahoo.com

ABSTRACT

Most current data clustering algorithms in data mining are based on a distance calculation in certain metric space. For Spatial Database Systems (SDBS), the Euclidean distance between two data points is often used to represent the relationship between data points. However, in some spatial settings and many other applications, distance alone is not enough to represent all the attributes of the relation between data points. We need a more powerful model to record more relational information between data objects. This paper adopts a graph model by which a database is regarded as a graph: each vertex of the graph represents a data point, and each edge, weighted or unweighted, is used to record the relation between two data points connected by the edge. Based on the graph model, this paper presents a set of cluster analysis criteria to guide data clustering. The criteria can be used to measure clustering results and help improving the quality of clustering. Further, a customizable algorithm using the criteria is proposed and implemented. This algorithm can produce clusters according to users' specifications. Preliminary experiments show encouraging results.

1. INTRODUCTION

Data clustering has been considered as a primary data mining method for knowledge discovery. There have been many clustering algorithms in the literature. From the perspective of whether graphs are used, we can categorize the clustering methods into graph-based and non-graph-based. Graph-based approaches take extra time on graph construction. It, however, has the ability to represent a large number of relationships between data points [17] while non-graph-based approaches cannot. Recently published graph-based approaches include: CHAMELEON [15], AUTOCLUST [16], Subdue [17] and Random Walk [14]. From the perspective of whether user inputs play a role in the cluster analysis, we can categorize clustering methods into automatic clustering and semi-automatic clustering. Automatic clustering is data-driven and does not accept user inputs. The representative automatic approach is AUTOCLUST [16]. On the other hand, many clustering algorithms use some parameters or thresholds to control the clustering process or results. They include: CHAMELEON [15], BIRCH [12], CLARANS [10], DBSCAN [8], STING [11], and Random Walk [14]. The parameters or thresholds in these algorithms can be regarded as channels between the clustering algorithm and the external environment.

SAC 2003, Melbourne, Florida, USA

An ideal clustering algorithm, however, should be both data-driven and user-centric (user-oriented). Data-driven [18] means that no prior information about the given dataset is needed while user-centric (user-oriented) aims at accepting users' requirements. Although many current clustering algorithms use some parameters to control the clustering process and results, their parameters are not user-oriented but algorithm-oriented. This paper presents a customizable and graph-based approach to data clustering. Compared with current data clustering methods, this approach has two major features: first, it is based on graph structure analysis. Current clustering analyses are mainly based on distance computation [13]. In many applications, distance alone is not enough to represent all the attributes of the relation between data points. By modeling data with a graph, the approach proposed in this paper can control and measure the clustering process. As a result, the derived clusters can meet users' requirements and be easily visualized. Second, this approach can accept users' inputs. Different applications may impose different criteria. Realistically, we consider the user's requirement as the basis of our clustering criteria. In order to suit various application domains, the approach needs to be customizable with criteria settings. This paper first proposes a set of measurement criteria for graph-based data clustering. Based on the set of criteria, this paper proposes a set of user-input parameters aiming at increasing the customizability of the clustering algorithms. No prior information is needed when setting the parameter values in order to maintain the data-driven property. Further, a corresponding clustering algorithm that uses the criteria is proposed and implemented.

The rest of the paper is organized as follows. Section 2 reviews the previous work on data clustering and approximation parameters. Section 3 proposes a set of criteria for measuring the quality of clustering. Our clustering algorithm is described in Section 4. Section 5 reports the experimental results. Section 6 concludes the paper.

2. PREVIOUS WORK ON CLUSTER MEASUREMENT

There have been many works on cluster analysis and measurement. Some of them are based on computational geometry, some for spatial database system applications, while others focus on graph theoretical properties.

2.1 Computational Geometry Analysis

Hambrusch *et al*. [1] defines that the ideal size (the number of vertices) of a cluster is c, $n=cp$, n is the number of total vertices in the given graph, and p is the number of clusters. The size of cluster C_i must satisfy $(1-\alpha/2)c \le |C_i| \le c(1+\alpha)$, $1 \le i \le p$, $0 \le \alpha < 1$.

Agarwal [6] proposes another definition of cluster size based on geometry space: the size of a cluster C_i is the maximum distance between a fixed-point *fp*, called the center of the cluster, and any vertex of C_i. Let S be a set of n nodes in a metric space, a

k-clustering of S is a partition C of S into k subsets $C_1, C_2,..., C_k$. The size of C is the maximum size of a cluster in C.

A functional approach [2] describes the clustering criterion as a function $P: \Phi \rightarrow R^+$. The process of finding the best clustering is to determine the clustering $C^* \in \Phi$, for which $P(C^*) = \min P(C)$ for $C \in \Phi$ where Φ is a set of feasible clustering layouts. (what's the role of R^+?)

2.2 Graph Theoretical Analysis

A *clique* (or a complete graph) is a simple graph in which there is one edge between every pair of vertices. The purpose of node grouping [7] is to abstract small node-disjoint cliques or sub-graphs that are almost-cliques (almost complete graphs). Node-disjoint cliques and almost-cliques are called *groups*, which form sets of high-cohesive nodes.

Harel and Koren[3] provides a heuristic to help deciding which vertices should be drawn closely. The heuristic is based on the observation that a nice layout of a graph should convey visually the relational information that the graph represents, so the vertices that are closely related in the graph (i.e., the graph theoretic distance is small) should be drawn close together.

A very useful measure is applied in FADE [4]: the number of inter-cluster edges versus that of intra-cluster edges. The overall drawing improves if nodes connected with edges are drawn closer than nodes unconnected.

2.3 Spatial Database System (SDBS)

Ester *et al.* [8] proposes DBSCAN (Density Based Spatial Clustering of Applications with Noise) for large spatial databases. It is no longer based on distance but on the number of near neighbors of a point. DBSCAN requires that for every point p in a cluster C there is a point q in C so that p is inside *Eps* of q and *Eps* of q contains at least *MinPts* points. DBSCAN requires a human participant to determine the global parameter *Eps*. The parameter *MinPts* is fixed at 4 in the algorithm to reduce the computational complexity.

CLARANS (Clustering Large Applications based upon RANdomized Search) [10] is motivated by two well-known cluster analysis algorithms: PAM (Partitioning Around Medoids) and CLARA (Clustering LARge Applications). The key concept of PAM is *medoid* which is a representative object for each cluster. The quality of clustering is measured by the average dissimilarity between an object and the *medoid* of its cluster. CLARA draws a sample of the data set and applies PAM on the sample to approximate the *medoid*s of the whole dataset.

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) [12] introduces a threshold T to limit the *diameter* (or *radius*) of the cluster. The larger the T is, the smaller the tree is. Several basic concepts like *radius*, *diameter*, and *centroid* are used in BIRCH to describe the distance properties of a cluster.

CHAMELEON [15] proposes a more objective definition of similarity, which is composed of relative inter-connectivity and relative closeness. It needs an appropriate MINSIZE (1%~5% of the overall number of data points) to perform a satisfactory partition.

The aforementioned cluster analyses address the quality measurement problem individually. None of them systematically measures the entire clustering quality at all the three levels: intra-cluster, inter-cluster, and overall clustering. This has motivated our work. The next section describes a set of criteria proposed for measuring the quality of clustering.

3. CLUSTERING REQUIREMENTS

Our criteria are based on a graph model. Given a graph $G = (V, E)$, $|V| = n$ is the number of vertices, $|E| = m$ is the number of edges. A *clustering* of G partitions the vertices of G into k disjoint sets $C_1, C_2, C_3,... C_k$. Each set C_i ($1 \le i \le k$) is called a *cluster*. $|C_i|$ is the number of vertices of cluster C_i. A *clustered graph* is a graph with a recursive clustering, or partitioning, of the vertex set of G.

Given $u,v \in V$, $(u,v) \in E$, we say that edge (u,v) is *inside* C_i iff both u and v are in C_i. The number of edges *inside* cluster C_i is denoted as E_i. The maximal possible number of edges *inside* C_i is denoted as M_i, $M_i = |C_i|*(|C_i|-1)/2$, when C_i is a clique. For clusters C_i and C_j, *inter(i,j)* is the number of edges between clusters C_i and C_j. Given $u \in C_i$, $v \in C_j$, $(u,v) \in E$, we say (u,v) is an edge between clusters C_i and C_j. The quality of a clustering result can be measured with the following criteria.

1. ***Cohesiveness*** is the minimum value of ratio E_i to M_i, for the ith cluster C_i , denoted as a real number α, $0<\alpha\le 1$.

$$\alpha = Min((E_i / M_i)), \forall i \in \{1,2,....,k\}, k \text{ is the number of clusters} \quad (1)$$

α is used to measure whether every cluster is a cohesive group. The goal is to maximize the value of α for every cluster. When α=1, the cluster is a clique.

2. ***Coupling Bound*** is the biggest number of inter-cluster edges, represented by an integer U_{inter}.

$$U_{inter} = Max(inter(i,j)), \forall i,j \in \{1,2,....,k\}, i \neq j, k \text{ is the number of clusters} \quad (2)$$

U_{inter} represents the clustering quality between every pair of clusters. A lower U_{inter} helps maintaining the natural clustering.

3. ***Coupling Ratio*** is the ratio of the number of inter-cluster edges to the total number of edges in G, denoted as a real number β, $0<\beta<1$.

$$\beta = \left(\sum_{i=1}^{k-1} \sum_{j=i+1}^{k} (inter(i,j))\right) / m \quad (3)$$

β is used to measure an overall cohesiveness for a graph. A smaller β means less inter-cluster edges and more edges are inside clusters.

4. ***Granularity*** is defined as the number of clusters, denoted by an integer k.

$$k = number\ of\ clusters \quad (4)$$

Granularity indirectly measures the flexibility and the applicability of a clustering result. Different applications may require different *granularities*.

The concept of ***Clustering Measurement (CM)*** is defined as a 4-tuple (α, U_{inter}, β, k), where α, U_{inter}, β, k are defined as above. ***CM*** can be used to describe the quality of a clustering result, and it can also be extended to represent the user's requirements, as described below.

The definition of ***Clustering Requirement (CR)*** is extended from ***CM*** and is a 5-tuple (α', U_{inter}', β', K_{min}, K_{max}). α' represents the requirement on *cohesiveness*; U_{inter}' represents the requirement on *coupling bound*; β' describes the requirement on *coupling ratio*

while K_{max} and K_{min} are a pair of integers: the upper bound and the lower bound of the number of clusters.

Given the *CM,* a 4-tuple (α, U_{inter}, β, k), of the final clustering result, *CR* describes the user's requirements in terms of *CM*:

$$\alpha \geq \alpha' \quad (5)$$

$$U_{inter} \leq U_{inter}' \quad (6)$$

$$\beta \leq \beta' \quad (7)$$

$$K_{max} \geq k \geq K_{min} \quad (8)$$

In summary, the *CR* can be regarded as the quality level that a clustering algorithm must maintain, or, can represent the quality of a clustering result quantitatively. A high quality clustering implies a big α, a small β, a low U_{inter} and desirable K_{max} and K_{min}. It is important to note that cohesiveness, coupling, and granularity are 3 orthogonal measurements. For an existing clustering result they can be used together as a *CM* to measure the clustering quality effectively. Or they can be used together as a *CR* to accept the user's requirements. It is possible that a clustering which meets all the four conditions may not exist.

4. A HYBRID APPROACH TO CLUSTERING

Current clustering methods can be divided into two categories:

- Clustering according to some specific requirements: it can be effective for some special cases but cannot be applied to all situations.
- General-purpose clustering: clustering is based on some assumed criteria such as low coupling between clusters. It may be useful for some applications but may be useless for others.

In this section we propose a customizable algorithm that accepts a *CR* as input and outputs a satisfactory *CM*. The whole algorithm consists of 5 steps: we first use *k*-core approach (detailed in Section 4.1) to partition the given graph. After this, we create corresponding matrix for each core, and then apply the Bond Energy Algorithm (BEA, Section 4.2) to each matrix produced. Then we combine all the small matrices into a big matrix. This matrix still represents the same graph but becomes a relatively "clustered" matrix. Finally we manipulate this matrix to meet the user's requirements.

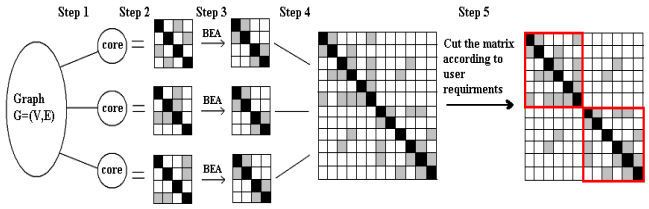

Figure 1 The procedure of the hybrid algorithm

Figure 1 shows five steps of the hybrid algorithm. Step 1 is the *k*-core approach: the big graph is partitioned into cores. Step 2 creates the corresponding matrix for every main core. Step 3 uses the BEA to compute and rearrange the small matrix for each core. Step 4 combines the resulting small matrices into a big matrix. Step 5 accepts the user's specifications and requirements to partition the matrix into clusters.

As the first step is the *k*-core approach, we will discuss it first.

4.1 The *k*-core Approach

Batagelj [2] proposes a fast clustering approach based on core decomposition: i.e. the *k*-core approach. The sub-graphs it produces are not only easy to handle but also contain useful connectivity-based properties for future analysis.

Before briefly introducing the *k*-core algorithm, let us firstly define a *core* (as by Batagelj [2]). Let $G = (V, E)$ be a graph. V is the set of vertices and E is the set of edges. A sub-graph $H = (W, E \mid W)$ induced by the set W is a *k*-core or a core of order k iff $\forall v$ in W: degree $(v) \geq k$ and H is a maximum sub-graph with this property. The core of maximum order is also called the *main core*. The algorithm of finding *k*-core is simple: find the main core of a given graph first, if it is too big, do some block model analysis and divide it further; otherwise, consider the main core as a cluster by removing it from the graph. For the residual graph, the same procedure is repeated until all cores have been removed.

The *k*-core approach's advantage is its $O(m)$ efficiency (m is the number of the edges). However, pure *k*-core approach may not be practically useful since its result totally depends on the structure of the input graph. The *k*-core approach without further clustering provides only one clustering result because the cores of a given graph are fixed. As a result, it rarely meets the user's requirements.

There are two important observations that may help: (1) the small cliques (K_3, K_4, K_5) appear often in the typical structures laid out by graph visualization tools [5]; (2) a clique of degree $k+1$ can possibly exist only in a *k*-core. This implies that if a *k*-clique exists in the given graph, it must be a sub-graph of the $(k+1)$-core of the given graph. It is no doubt that cliques or nearly cliques are the most cohesive parts of the graph. They exist only in cores. This convinces us that the *k*-core approach can be a good start for further partitioning.

Because Step 2 is merely a process of creating the adjacency matrix for a graph we will skip it and explain Step 3.

4.2 The Bond Energy Algorithm

The bond energy algorithm (BEA), proposed by McCormick *et al* [19], is a cluster-analysis method for identifying natural groups and clusters in complex data arrays. It introduces the concept of measure of effectiveness (*ME*), aiming at maximizing the summed bond energy over all row and column permutations of an input array. That is, find

$$max\ (ME) = max\ \{\sum_{i=1}^{M}\sum_{j=1}^{N} a_{i,j}\,[a_{i,j-1} + a_{i,j+1} + a_{i-1,j} + a_{i+1,j}]\}$$

for all $N!M!$ permutations.

Because an adjacency matrix for a graph is symmetric, we need only to find a row or a column permutation that creates the strongest "bond energy" by driving the larger matrix elements together. The elements of the matrix will be grouped as the way they should be. This is achieved by calculating the measure of effectiveness (*ME*) for each permutation by

$$ME = \sum_{i=1}^{N}\sum_{j=1}^{N} a_{i,j}(a_{i,j-1} + a_{i,j+1})$$

The permutation that gives the maximal ME represents a desirable component placement. Although finding such a permutation requires exponential time, a near-optimal algorithm in $O(n^4)$ time produces results that are close to those of exhaustive search, according to Zhang and Gorla [20]. Compared with other data grouping methods, the BEA is accurate and produces good results.

More importantly, BEA can be measured and customized by users' specifications and requirements. The procedure of applying BEA to a symmetric matrix is illustrated in Figure 2.

M	2	0	0	1
2	M	0	0	3
0	0	M	1	0
0	0	1	M	0
1	3	0	0	M

BEA →

M	2	1	0	0
2	M	3	0	0
1	3	M	0	0
0	0	0	M	1
0	0	0	1	M

Figure 2 The original symmetric matrix becomes a "clustered" matrix after applying BEA to it

In Figure 2, *M* is an integer larger than any other element. For example, for a weighted graph, *M* can be any value greater than the maximal weight. After BEA is applied the cohesive elements will be placed together in the matrix.

According to the *k*-core and BEA's special properties, we adapt the BEA to compute each sub-graph produced by the *k*-core algorithm. The main idea of the hybrid algorithm is to divide and conquer as illustrated in Figure 2. We first use the *k*-core approach to partition the given graph. As a result, the size of the produced sub-graphs (cores) is much smaller than the original whole graph, and then we use BEA to group precisely the vertices inside each core. After the vertex grouping finishes for all the cores, we combine all the optimized sub-matrices into a big matrix. This matrix still represents the same graph but becomes a "clustered" matrix. Finally we manipulate this matrix to meet the user's requirements. We designed a method to partition the "clustered" matrix. Because the Step 4 is also straightforward like Step 2, we will skip its explanation but explain Step 5, i.e., our matrix partition method in Section 4.3.

4.3 User-Directed Partitioning

After applying the *k*-core approach, creating matrix, BEA algorithm, and combining all the produced small matrices into a big one, as Steps 1, 2, 3 and 4, the resulting matrix consists of many "clusters". The "clustered" matrix is illustrated in Figure 5 (a). It is time to accept the user's requirements and partition the matrix accordingly. This forms the final step of our approach. We now explain the process of matrix partitioning based on the input parameters from the user.

Given a graph $G = (V, E)$, $n = |V|$, $m = |E|$, for every matrix index i, $1<i<n$, "*cut* at *i*" means partitioning the matrix at the *i*th column and *i*th row. A *cut* is acceptable only if it locates at a border of two clusters. Let us now analyze the relationship between the *cohesiveness* α and the *coupling ratio* β. According to the definitions of α and β, we obtain the following formula:

$$\beta=1-(i*(i-1)*\alpha1+(n-i)(n-i-1)*\alpha2)/(2*m) \quad (9)$$

Our purpose is to minimize β, i.e., to maximize $(i*(i-1)*\alpha1+(n-i)(n-i-1)*\alpha2)/(2*m)$. For cut at *i*, we compute the value of α for each of the two produced clusters. After we get the pair of α ($\alpha1$ and $\alpha2$) for every cut at each matrix index, we can draw a graph like the bottom graph of Figure 3 (b). Then we use the formula (9) to compute the value of β for each pair of α. We can draw a graph like the top graph of Figure 3 (b), from which it is easy to see the smallest *coupling ratio* β. The minimum β means a best cut. After choosing the best cut, we can choose the second best cut for the matrix. Every cut will produce an additional cluster in each round. This procedure is repeated until the number of produced clusters meets the requirement (exceeds the K_{min}). Then we judge if such a clustering is satisfactory, if so, the result is generated as an output; otherwise, choose the next cut *i* to partition the matrix. Repeat the procedure until a satisfactory clustering is found or the number of produced clusters exceeds K_{max}. If a satisfactory clustering is found, the result is generated as an output. If the number of produced clusters exceeds K_{max}, no clustering result meeting the requirement can be found.

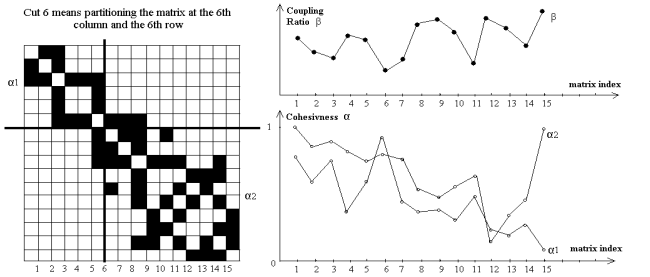

Figure 3 (a) The "clustered" matrix and the best cut
(b) Choosing the best cut from the "clustered" matrix

Figure 3 (a) is the "clustered" matrix, Figure 3 (b) illustrates the pair of values of α for each cut (the bottom graph) and the corresponding value of coupling ratio β for each cut (the top graph). We can see that the best cut is at index 6.

We now analyze the time complexity of each step of the algorithm. Our approach consists of five steps. Given the input graph of *n* vertices and *m* edges, first, according to Batagelj [2], *k*-core step costs $O(m)$ in time for *m* edges. Second, we create the corresponding matrix for each produced core, which costs $O(n^2/num)$, *num* is the maximum number of cores, so the worst time is $O(n^2)$ when $num=1$. Third, after the *k*-core partitioning, the number of elements of the produced sub-graph is much smaller than the original graph. So the time complexity of applying BEA depends on the graph structure. Our experiments show that it is acceptable if the given graph contains many clusters. Then, we combine these optimized small matrices together, which costs $O(n)$ for *n* vertices. Finally we divide the resulting matrix according to the user's specification, which costs $O(n\log n)$.

5. EXPERIMENTS ON USER REQUIREMENTS

Our preliminary experiments are focused on the customizability. We use a graph of 300 vertices, and apply our algorithm with different user requirements. According to the *CR* defined in Section 3, the user's requirements can be simplified into three classes, depending on the following emphases:

1. intra-cluster quality, i.e., high cohesiveness.
2. inter-cluster quality, i.e., less coupling.
3. appropriate granularity.

Without losing generality, three classes have $2^3 = 8$ combinations of requirements. Every requirement can be one or more of these 8 combinations. Table 1 compares the clustering results of these 8 sets of different requirements for the same graph.

Table 1. Experimental comparison of different user requirements

	α	β	U_{inter}	K_{max}	K_{min}	Clusters Exist?
1	0.3	0.2	10	20	2	Yes
2	0.3	0.2	10	10	7	Yes
3	0.3	0.1	5	20	2	Yes
4	0.3	0.1	5	10	7	Yes
5	0.7	0.2	10	20	2	Yes
6	0.7	0.2	10	10	7	No
7	0.7	0.1	5	20	2	Yes
8	0.7	0.1	5	10	7	No

Among the 8 sets of requirements, the ones in rows 5,6,7,8 require the produced clusters to be cohesive vertex groups, which means a high α (α=0.7). The ones in rows 3,4,7,8 say they need low coupling between the clusters, which means a low β (β=0.1) and a low U_{inter} (U_{inter}=5). The ones in rows 2,4,6,8 imply a strict limit on the scope of the number of clusters, which means a close pair of K_{min} and K_{max}. For the ones in rows 1,2,3,4,5, and 7, our algorithm produced satisfactory result. For the ones in rows 6 and 8, our algorithm could not find any result meeting the user's requirement. For the row 6, the number of clusters is strictly limited while a high cohesiveness is required; such kind of result does not exist in the given graph. For the strictest requirement, the row 8, it is not surprising that no result is found.

6. CONCLUSIONS

Like the process of software development, the process of clustering involves design and verification. The basic purpose of clustering is to distinguish two nodes if the cohesiveness between them cannot exceed a pre-specified threshold. We believe that both how to customize a clustering and how to measure it relate to the clustering criteria directly or indirectly. This paper has proposed a set of criteria for measuring data clustering quality and presented a customizable algorithm that finds the most appropriate clusters according to the user-supplied parameters. A flexible mechanism has been proposed for the user to express his/her requirements through input parameters. A graph-based partitioning clustering model is established with this mechanism. This approach not only helps large database clustering but also provides effective graph visualization of the resulting clusters. Furthermore, our algorithm is of high efficiency if the graph contains many clusters. Further study is ongoing to observe if the time complexity of our algorithm is proportional to n, the number of graph vertices, i.e., the data points. Future work also includes the application of the algorithm on Web data mining.

7. REFERENCES:

[1] S. E. Hambrusch, C-M. Liu, and H-S. Lim, Clustering in Trees: Optimizing Cluster Sizes and Number of Subtrees, *Journal of Graph Algorithms and Applications,* Vol. 4, No. 4, pp.1-26 (2000).

[2] V. Batagelj, A. Mrvar, and M. Zaversnik, Partitioning Approaches to Clustering in Graphs, *Proc. GD'1999,* LNCS, pp. 90-97 (2000).

[3] D. Harel and Y. Koren, A Fast Multi-scale Method for Drawing Large Graphs, *Proc. GD'2000*, LNCS, pp. 183-196 (2001).

[4] A. Quigley and P. Eades, FADE: Graph Drawing, Clustering, and Visual Abstraction, *Proc. GD'2000*, LNCS, pp. 197-210 (2001).

[5] J. May-Six, Vistool: A Tool For Visualizing Graphs, PhD Thesis, The University of Texas at Dallas (2000).

[6] P. K. Agarwal and C. M. Procopiuc, Exact and Approximation Algorithms for Clustering, *Proc. 9th ACM-SIAM Symp., Discrete Algorithms (*1998).

[7] J. May-Six and I. G. Tollis, Effective Graph Visualization Via Node Grouping, *Proc. IEEE Symposium on information Visualization 2001*, pp. 51-58 (2001).

[8] M.Ester, H. P. Kriegel, J.Sander, and X.Xu, A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise, *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD-96)*, AAAI Press, pp. 226-231 (1996).

[9] M. Ester, H. P. Kriegel, J. Sander, and X. Xu, Clustering for Mining in Large Spatial Databases. *KI (Artificial Intelligence), Special Issue on Data Mining,* ScienTec Publishing, pp. 18-24 (1998).

[10] R. T. Ng and J. Han, Efficient and Effective Clustering Methods for Spatial Data Mining, *Proc. 20th Int. Conf. on Very Large Data Bases,* Morgan Kaufmann, pp. 144-155 (1994).

[11] W. Wang, J. Yang, and R. Muntz, STING: A Statistical Information Grid Approach to Spatial Data Mining, *Proc. 23rd Int. Conf. on Very Large Data Bases,* Morgan Kaufmann, pp. 186-195 (1997).

[12] T. Zhang, R. Ramakrishnan, and M. Linvy, BIRCH: An Efficient Data Clustering Method for Very Large Databases, *Proc. ACM SIGMOD Int'l Conf. on Management of Data*, ACM Press, pp.103-114 (1996).

[13] M. S. Chen, J. Han and P. S. Yu, Data Mining: An Overview from Database Perspective, *IEEE Transactions on Knowledge and Data Engineering,* IEEE Computer Society Press, Vol. 8, No.6, pp. 866-883 (1996).

[14] D. Harel and Y. Koren, Clustering Spatial Data Using Random Walks, *Proc. 7th Int'l Conf. Knowledge Discovery and Data Mining (KDD-2001)*, ACM Press, New York, pp. 281-286 (2001).

[15] G. Karypis, E. Han, and V. Kumar, CHAMELEON, A Hierarchical Clustering Algorithm Using Dynamic Modeling, *IEEE Computer* pp. 68-75, 32 (1999).

[16] V. Estivill-Castro and I. Lee, AUTOCLUST: Automatic Clustering via Boundary Extraction for Mining Massive Point-Data Sets, *5th Int'l Conf. on Geocomputation*, Geo Computation CD-ROM: GC049, ISBN 0-9533477-2-9 (2000).

[17] I. Jonyer, L. B. Holder and D. J. Cook, Graph-Based Hierarchical Conceptual Clustering, *Proc. of the Thirteenth Annual Florida AI Research Symposium* (2000).

[18] A. K. Jain, M. N. Murty, and P. J. Flynn, Data Clustering: A Review, *ACM Computing Surveys*, Vol.31, No. 3, pp. 264-323 (1999).

[19] W. T. McCormick, P. J. Sweitzer, and T. W. White: Problem decomposition and data reorganization by a clustering technique. *Oper. Res.*, (September-October), pp. 993-1009 (1972).

[20] K. Zhang and N. Gorla, Locality Metrics and Program Physical Structures, *Journal of Systems and Software*, 54 (2000), pp. 159-166 (2000).