

Outlier Detection and Evaluation by Network Flow

Ying Liu

Alan P. Sprague

Department of Computer and Information Sciences

The University of Alabama at Birmingham

Birmingham, AL 35294-1170

{liuyi, sprague}@cis.uab.edu

ABSTRACT

Detecting outliers is an important topic in data mining. Sometimes the outliers are more interesting than the rest of the data. Outlier identification has lots of applications, such as intrusion detection, and unusual usage of credit cards or telecommunication services. In this paper, we propose a novel method for outlier identification which is based on network flow. We use the well known Maximum Flow Minimum Cut theorem from graph theory to find the outliers and strong outlier groups. This outlier detection occurs in a novel setting: to repair poor quality clusters generated by a clustering algorithm.

Keywords

Data Mining, Outlier Detection, Network Flow, Graph Theory, Maximum Flow Minimum Cut

1. INTRODUCTION

Outlier identification has lots of applications, such as intrusion detection, and unusual usage of credit cards or telecommunication services. Outliers are those data objects that are grossly different from or inconsistent with the rest of the data [5]. Clustering algorithms group similar objects together, but many fail to separate out outliers very well. Other clustering algorithms, such as ROCK [9], do detect outliers. Such an algorithm is called a clustering-based method of outlier detection. Other methods for outlier detection include distribution-based, which is good at low dimensions [7]. The distance-based outliers are those objects who do not have “enough” neighbors [5]. The density-based method of [6] uses a Local Outlier Factor (LOF) to find outliers. These methods require that the user set some parameters, which are difficult to understand for the users before they find the outliers. And they can not effectively find the local outlier groups.

In this paper, we propose a novel method for outlier identification which is based on network flow. The algorithm can detect outliers and outlier groups. Especially, it can evaluate outliers at multiple levels of granularity by Maximum Flow. The paper is organized as follows. Section 2 describes network flow, and the intuition underlying this work. Section 3 describes our method, and Section 4 reports the results of experiments.

2. NETWORK FLOW AND OUTLIERS

In this section we explain the network flow intuition underlying this work. We precede that by giving some definition regarding network flow.

2.1 Network Flow

A finite digraph $G = (V, E)$ is given, with no self-loops and no parallel edges. (An undirected graph may be considered directed, by replacing each edge by a pair of oppositely directed arcs.) Two vertices s and t are specified; s is called the source and t , the sink. Each edge $e \in E$ is assigned a non-negative number $c(e)$ called the capacity of e . The flow function $f(e)$ for each edge satisfies:

- (1) For every edge $e \in E$, $0 \leq f(e) \leq c(e)$,
- (2) For every vertex except the source and sink, the flow incoming to a vertex v is equal to the flow outgoing from v .

The maximum flow problem is to find an f for which the total flow is maximum [2]. The total flow is defined as the net flow out from a source, or equivalently the net flow into a sink. The total flow F can be measured at the sink, or it can be measured at any cut separating the source from the sink.

Let X be a subset of the vertex set V and \overline{X} be the complement of X . If $s \in X$ and $t \in \overline{X}$ then (X, \overline{X}) is called a *cut* separating s and t . The *capacity* of cut (X, \overline{X}) is the sum of capacities of edges from X to \overline{X} . The Maximum Flow Minimum Cut theorem states that the maximum amount of flow from s to t equals the minimum of the capacities of cuts separating s and t .

2.2 Intuition

We are interested in the following situation: a clustering algorithm has produced a set C of data points (“vertices”) which it has labeled as a cluster. We wish to determine if C is a cluster, or instead contains points that are only weakly related to the rest. (Those points are the outliers we are interested in.) We propose to use network flow to perform this analysis.

Consider the set C of points as a network: for each pair s, s' of points in C , consider ss' to be an edge, and to have capacity equal to $1/(1+l(ss'))$ where $l(ss')$ is the length of edge ss' . Thus if s is near to s' , ss' has high capacity, while if s is far from s' , ss' has low capacity.

Our intuition is based on the following ideas. Suppose that s is an outlier. Let t be the point in C that is farthest from s . Suppose further that s is far from all other points in C . Then each edge ss' has small capacity, so $(\{s\}, C - \{s\})$ is a cut of small capacity. Then the network flow algorithm will tell us that the maximum flow from s to t is small, and quite likely the minimum cut will equal $(\{s\}, C - \{s\})$. (Alternately, perhaps t is also an outlier and the minimum cut is singling out t .)[8]

Suppose instead that s and another couple of points s' and s'' form a group of outlier points, and t be the point in C that is farthest from s . s, s' and s'' are very similar with each other, so the capacities are very high or even full when they are identical. When the total flow from s to t is less than the capacities between ss' and ss'' , the minimum cut is a cut $(\{s, s', s''\}, C - \{s, s', s''\})$; when the maximum flow is more than the capacities between ss' or ss'' , this flow will saturate the edge ss' or ss'' . I.e., s will use up capacities around it before saturating the edges that connect s, s' and s'' with other vertices. s' and s'' will be at the sink side, the minimum cut is still a cut $(\{s\}, C - \{s\})$. It can't find the outlier group s, s' and s'' together. For the later case, we use a simple way to solve this in 3.1.

3. METHOD OVERVIEW

3.1 Set Up the Graph

The setting for our work is the following. A data file is given. A graph G , whose edges are weighted, is defined on the set of records of the file. Clustering is performed on the vertices of G (i.e. on the records of the file). Lastly, outlier detection is performed. Our interest centers on the last of these steps. However, we set the stage by first giving some information on earlier steps.

The vertices of graph G are the records of the data file. Defining weights of edges depends on the type of data in the file. For instance, with market basket data, each record is a set (of “items”). For such data, the similarity between two records is measured by the Jaccard coefficient. The Jaccard coefficient is defined as the ratio of the number of shared attributes to the total attributes possessed by the two records. Since the Jaccard coefficient measures the similarity

between two records (where the value is 1 if they are identical and 0 if there is no intersection between them), it is reasonable to define the length of an edge as the reciprocal of the Jaccard coefficient, and the capacity of an edge is the reciprocal of the length. For numerical data, compute the k nearest neighbors of each vertex, and make sure all data points in this cluster are connected. So, a graph with a higher k has more edges. In 4.2, we will discuss the relationship between k and the performance.

We use Ford and Fulkerson algorithm to do the network flow. This algorithm may fail, if the capacities are allowed to be irrational numbers [2]. So, we scale the capacity to 0 to 1000 by multiply 1000 of the similarity. The precision of the similarity depends on you. In Fig. 1, suppose the total flow from source S to its farthest vertex t is 1001. t is an outlier, the cutting is around the sink. So, the sink side is t itself and the source side is the rest of the data. Now we create another vertex t' which is identical with sink t . The full capacity of an edge is 1000. So, the capacity between t and t' is 1000. Suppose t connects m edges with other vertices. After adding t' , those vertices connecting to t also connect to t' , including the edge tt' , so there are more $2*(m+1)$ edges(edges are in two directions.). Because the flow from S to t is 1001, the flow from S to t' is 1001 as well. The $t \rightarrow t'$ is a new path from S to t , so, the flow from S to t' (1001) will saturate the edge $t \rightarrow t'$ (1000), the cut will be the same as before: the sink side of the cut contains t alone. The total flow increase 1000, and now it is 2001. We can't find the outlier group t and t' together by the Minimum Cut.

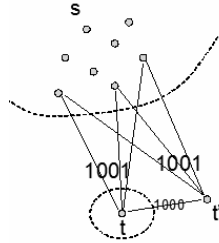


Fig. 1 Each edge are two edges actually, in both directions. From Source to sink side, edges are saturated, from sink to source side, the flow is 0. In the source side, vertices are connected somehow.

When the total flow from source to sink is larger than the maximum capacity or a high capacity between a pair of similar or near vertices, the Minimum Cut can't find the outlier group together. The cuts are usually at the source or sink itself. The results are uninteresting. To solve this problem, the total maximum flow must be lower than the capacities of the edges around the sink or source, such as lower than the full capacity 1000 of the previous example. We hope the flow from source to sink could saturate the edges with lower capacities first. Our method to do this is to transform the capacities, let the lower capacities still stay low, the high capacities are very high.

$Capacity_{new} = Capacity_{original}^n$, n could be 4 or 5, or another integer as long as the maximum flow much lower than the high capacities. If the full capacity is 1000, when $n=4$, after transform, the capacity is 1,000,000,000,000. At the same time, the low capacities will not increase very much. Transforming the capacities will not hurt the algorithm. On the contrary, it helps us expose weakest edges soon.

The graph initially is undirected. Network flow algorithm works on directed graph. The undirected graph may be converted to a directed graph in the usual way, by replacing each edge by two oppositely directed edges.

3.2 Outlier Detection

Our data is the set of points of a single cluster C . In our current method of outlier detection, we begin at a random vertex as the source S of a flow. For this source S we use the farthest vertex t_s from S as sink. The Single-Source Shortest Path algorithm is used to find t_s , Dijkstra's algorithm is used here[1]. Let f_s be the amount of flow in a maximum flow from S to t_s . Let $(X_s, \overline{X_s})$ be a minimum cut separating S and t_s . If there are outliers in this cluster, usually, this t_s is an outlier, because it is the farthest from a random vertex.

The algorithm is:

1. clustering the data.
2. for the data of a cluster, set up the graph.
3. begin at a random vertex as source s , find its farthest vertex as the sink t .
4. using the Maximum-Flow/Minimum-Cut algorithm to find the flow from source to sink, get the cut separate s and t .
5. select the next source, go back to 3 until the stop criteria.

3.3 How to select the next source(sink)?

If the next source is an outlier or one of a outlier group, the Minimum Cut would be very likely happen at the source or sink. Before we select the next source, let us look at the process of finding a path from source to sink.

Ford and Fulkerson suggested the use of augmenting paths to change a given flow function in order to increase the total flow. An augmenting path is a simple path from s to t , which is not necessarily directed, but it can be used to advance flow from s to t . If on this path, e points in the direction from s to t , then in order to be able to push flow through it, $f(e)$ must be less than $c(e)$. If e points in the opposite direction, then in order to be able to push through it additional flow from s to t , we must be able to cancel some of its flow. Therefore, $f(e) > 0$ must hold[2].

We begin with any feasible flow (e.g., $f=0$). In general, a node is in one of three states: unlabeled, labeled and unscanned, or labeled and scanned. Upon entering Routine A, all nodes are unlabeled. The first step renders the source labeled and unscanned.

Routine A (labeling process). Initially, label the source ($-, \mathcal{E}(s) = \infty$).

General step: select any node, x , that is labeled and unscanned, and let $(z^\pm, \mathcal{E}(x))$ be its label. To all unlabeled successor nodes, y , such that $f(x, y) < c(x, y)$, assign the label $(x^+, \mathcal{E}(y))$, where

$$\mathcal{E}(y) = \min\{\mathcal{E}(x), c(x, y) - f(x, y)\} \quad (1)$$

(Such y are now labeled and unscanned.) To all predecessor nodes, y , that are unlabeled, such that $f(y, x) > 0$, assign the label $(x^-, \mathcal{E}(y))$, where

$$\mathcal{E}(y) = \min\{\mathcal{E}(x), f(y, x)\} \quad (2)$$

(Such y are now labeled and unscanned.) Now define x to be labeled and scanned. Repeat the general step until the sink is labeled and unscanned, or until no more labels can be assigned. In the former case, go to Routine B; in the latter case, terminate. (f is a maximum flow.)

Routine B (flow change). The sink has label $(y^\pm, \mathcal{E}(t))$. If the first part of the label is y^+ , replace $f(y, t)$ with $f(y, t) + \mathcal{E}(t)$; otherwise, replace $f(t, y)$ with $f(t, y) - \mathcal{E}(t)$. Go to node y and treat it the same way: if its label is $(x^+, \mathcal{E}(y))$, replace $f(x, y)$ with $f(x, y) + \mathcal{E}(t)$; if its label is $(x^-, \mathcal{E}(y))$, replace $f(y, x)$ with $f(y, x) - \mathcal{E}(t)$. In either case, go to node x and repeat until the source is reached. Then, discard all labels and return to Routine A[3].

We will call the last labeling process – the one that does not reach t – the *last wave*, and let S be the set of vertices labeled in the last wave. The *last wave value* of each vertex y in S is the second component $\mathcal{E}(y)$ of its label. If the sink is in an outlier or in an outlier group, most of the vertices will be in the source side. Then the vertex we choose as next source is the vertex y in S that minimizes the sum of the last wave value plus the flow passing through y .

From formulas (1) and (2), we can see that the amount of additional flow pushed to t equals the maximum amount that can be pushed to t by a single path. For example, in Fig. 2, the first time, from s to t , the flow passed through $s \rightarrow a \rightarrow d \rightarrow t$; the second time, the flow passed through $s \rightarrow b \rightarrow c \rightarrow t$. At this moment, the edges $d \rightarrow t$ and $c \rightarrow t$ are already saturated, but this algorithm will continue search for a path from s to t . In the last wave, the minimum increase is 4 at vertex f , using path $s \rightarrow a \rightarrow d \rightarrow f$. After we add the flow already passed, f has the minimum value 4, and e as well. Next time, minimum cut will keep e and f together because the maximum flow will be 8, which is less than 15 (capacity of the high capacity edge near to the sink).

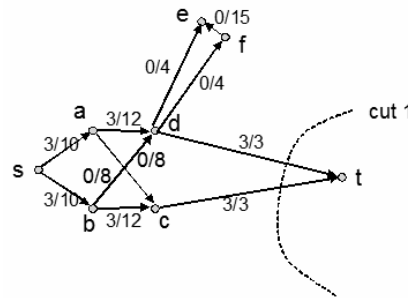


Fig. 2 Each edge was originally undirected; the reverse edge has the same capacity but is not shown, And has zero flow. The two numbers on each edge are flow/capacity.

Do not forget that the whole graph is a directed graph. After cutting, we use the side with more vertices to continue the process. The larger side may be the source side, and may be the sink side. If the larger side is the source side, we use the way already described to select the next source; if the larger side is the sink side, there is no last wave of the sink side, so the previous method won't work. A simple way to solve this is to change the source and sink, using source as the sink, and the sink as the source. The cut will be at the same place and the total flow won't change. Now, the source side has the last wave, and we can still use this method to select the next source(sink). By experience, when the source is selected as described (and the sink is the vertex farthest from the selected source), it is very common that the source side of the cut in the next flow is smaller than the sink. So, we reverse this procedure and use the selected vertex as the sink, its farthest vertex as the source to compute the network flow. If the source is a stronger outlier, then switch the source and sink. Experience indicates that this way decreases the time spent in switching source and sink.

Another way to select the next source is to get each vertex's average distance to other vertices. Then order these distances from maximum to minimum. Begin this process from vertex with maximum average distance. After the minimum cut, some vertices are cut. Then, the next source is the next vertex with bigger average distance which is not cut. Especially, for previous method, we can use this method to select the first sink instead of a random vertex for the first time. Please see the graph in Fig. 6 (a).

4. EXPERIENCE

4.1 Outlier (groups) detection

In order to illustrate the process, we use the data set mentioned in the Chameleon paper[4]. In the t4.mat data set, there are 8000 points. We use 'vcluster' method to cluster into 20 groups, such as in Fig. 3. Clustering is performed on this graph. In particular, we have used hMETIS.

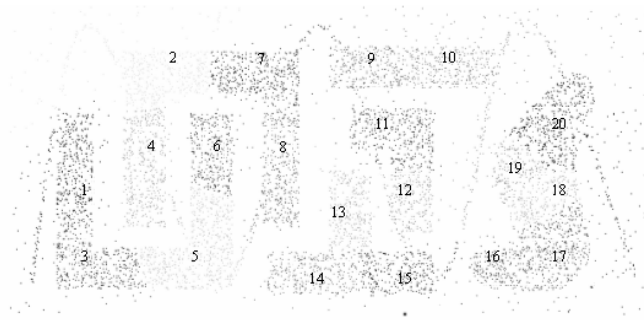


Fig. 3 For the data set t4.mat, we use 'vcluster' to cluster them into 20 clusters.

(The command and parameters are: `vcluster t4.mat 20 -clmethod=graph -sim=dist -agglofrom=30`)

For the No. 20 cluster, in Fig. 4 (a), there are 591 points in it. We want to find the outliers and outlier groups in this cluster. First of all, we set up the graph by connecting each vertex with its 7 nearest neighbors, giving the graph in Fig.4 (b). All points are connected, and there is a total of 5028 edges. (Edges are in both directions; here, we just use one undirected edge to represent a pair of directed edges.)

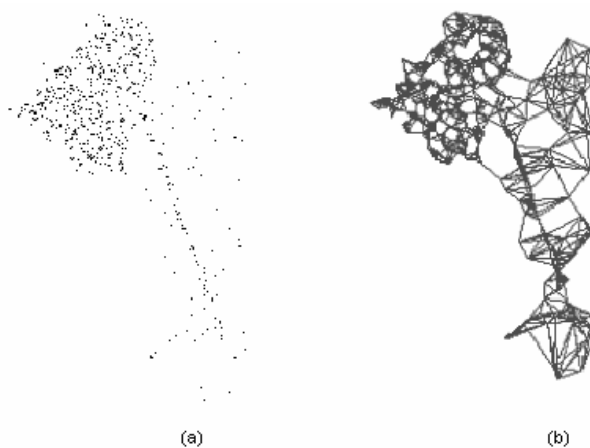


Fig. 4, (a) 591 points in No.20 cluster; (b), 7 nearest neighbors graph.

For this graph, we use $c = \frac{1}{(1 + dist)} * 100$ as the basic capacity, and then define capacity as the fourth power of

this, $capacity = c^4$. So the maximum capacity is $100^4 = 100,000,000$. Because the total flow from source to sink is much lower than 100,000,000, this is enough to expose the weak edges. We call the edges with low capacities are *weak edges*. So, the total flow from source to sink will not saturate the high capacity edges around the source or sink. It uses up the capacities of the weak edges first.

In Fig. 5, the left graph shows the results after 17 times of Maximum-flow/Minimum-cut for outlier detection. We list the total flow in (b), it ordered according to flow value. From that, we can see the relationship the outlier or outlier group with the majority data. If the flow is low, the small side of the cut is a strong outlier (group); If the flow is high, maybe it is not an outlier. The maximum flow give a measure of outliers (groups) at multiple levels of granularity.

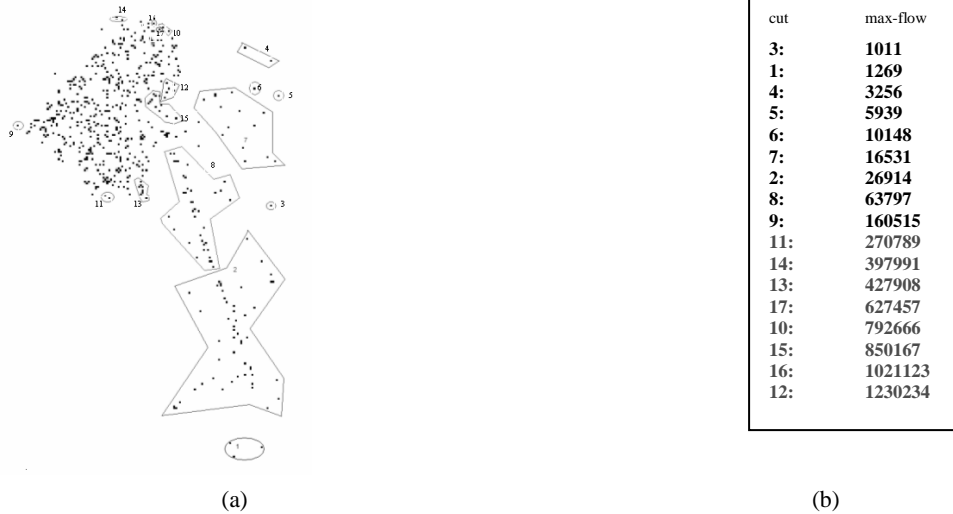


Fig. 5, (a), 17 times of maxi-flow/min-cut for finding outliers and outlier groups;
 (b), the cut and its corresponding flow order by the maximum flow from low to high.

4.2 K nearest neighbors

In previous experience, we have used 7 nearest neighbor graph to set up the network. If we use 5 or lower nearest neighbors, some small groups of points will become disconnected from the rest of the data, even though they are not outlier groups. This happens to a small group (of at least $k+1$ points) because points in the group are near the rest of the points in the cluster, but are extremely near to each other. We want to connect all the points, which puts a lower bound on k . That means, k nearest neighbors should connect all points.

If we increase the number k , the graph will have more edges. That means, it will take more time to find a flow. We did experiments with k being 10 and 15. In Fig. 6, (b) is for $k=10$, (c) is for $k=15$.

On this data, we can see, with the increase of k nearest neighbors, some outlier groups are smashed into small pieces. Total maximum flow can still reflect the relationship of this outlier or outlier group with the main data. The larger k is, the more edges in the network, and more time is needed. For, $k=7$, it uses 14 minutes of maximum flow; for $k=10$, it uses 20 minutes; for $k=15$, it uses 25 minutes. (Of course, it includes several times of switching source and sink because to find the last wave, the source side has to be the larger side.) As long as k nearest neighbors can connect all points, there is no other strict conditions about k . By experience, we recommend a little bit more than the minimum k . Because, sometimes, outliers connect the points which are not outliers, and those points connect with their k neighbors which don't connect with other majority points. For this case, the minimum cut will cut the outliers and points which are part of the majority data together. But the increase of k by one or two can solve this problem.

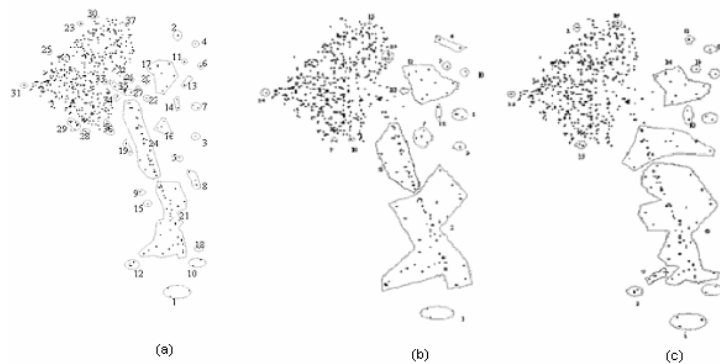


Fig. 6, (a), select the next source by the maximum average distance of vertices;
 (b), $k=10$, edge=7018; (c), $k=15$, edge=10372.

4.3 Stop criteria

So far, there are no complicated parameters of our algorithm. We use the properties of data themselves to find the useful information. From Fig. 5 you can see, in the first nine times, the minimum cut found outliers and outlier groups; after the ninth time, it only cut the points near the periphery of the majority points, they are not outliers any more. We can ask the users to set how many outlier groups they want. Or, let the user set a minimum distance (if lower than this distance, stop cutting.), convert the distance to capacity, this capacity as the threshold. When the minimum_(last wave + flow passed) \geq threshold, stop.

From Fig.5, maybe you will ask why it didn't cut the two points on left side of main data. That's because from Fig. 4, we can see, there are some gaps inside the majority data. The vertex with minimum (last wave + flow already passed) is in the middle somewhere, and its farthest is always along the longest axis at the edge of the majority data. If we use vertex's average distance as the next sink, it could solve this problem.

4.4 Complexity

The complexity of network flow is based on the number of vertices ($|V|$) and edges ($|E|$) of the network. The complexity of Ford and Fulkerson is $O(|V|^3|E|)$. Dijkstra's algorithm for single source shortest path, complexity is $O(|V|^2)$. Other algorithms for Maximum Flows, such as Dinic, complexity is $O(|V|^2|E|)$; Malhotra, Pramodh Kumar and Maheshwari, complexity is $O(|V|^3)$ and $O(|V|^2|E|^{1/2})$. [2]

5. FUTURE WORK

Future work includes different ways to set up the graph, and we also want to give the user more hints for the stop criteria.

Reference

- [1] A. Aho, J. Hopcroft, and J. Ullman. *Data Structures and Algorithms*. Addison-Wesley Publishing Company, 1982.
- [2] S. Even. *Graph Algorithms*. Computer Science Press, 1979.
- [3] H. Greenberg. *Ford-Fulkerson Max Flow labeling Algorithm* <http://carbon.cudenver.edu/~hgreenbe/>
- [4] G. Karypis, E. Han, and V. Kumar. Chameleon: Hierarchical Clustering Using Dynamic Modeling.
- [5] J. Han and M. Kamber. *Data Mining Concepts and Techniques*. Morgan Kaufmann Publishers, 2000.
- [6] M. M. Breunig, H. Kriegel, R. T. Ng, and J. Sander. *LOF: Identifying Density-Based Local Outliers*. Proc. ACM SIGMOD 2000 Int. Conf. On Management of Data, Dalles, TX, 2000.
- [7] E. Knorr and R. Ng. A Unified Notion of Outliers: and Computation. American Association for Artificial Intelligence.
- [8] Y. Liu and A. P. Sprague. *Network Flow for Outlier Detection*. ACMSE'04 Conference
- [9] S. Guha, R. Rostogi, and K. Shim. *ROCK, A Robust Clustering Algorithm for Categorical Attributes*. Information Systems Vol. 25, No. 5, pp. 345-366, 2000