International Journal of Engineering, Mathematical and Physical Sciences
ISSN: 2517-9934
Vol:3, No:12, 2009

Grid-based Supervised Clustering - GBSC

Pornpimol Bungkomkhun and Surapong Auwatanamongkol

***Abstract*—**This paper presents a supervised clustering algorithm, namely Grid-Based Supervised Clustering (GBSC), which is able to identify clusters of any shapes and sizes without presuming any canonical form for data distribution. The GBSC needs no pre-specified number of clusters, is insensitive to the order of the input data objects, and is capable of handling outliers. Built on the combination of grid-based clustering and density-based clustering, under the assistance of the downward closure property of density used in bottom-up subspace clustering, the GBSC can notably reduce its search space to avoid the memory confinement situation during its execution. On two-dimension synthetic datasets, the GBSC can identify clusters with different shapes and sizes correctly. The GBSC also outperforms other five supervised clustering algorithms when the experiments are performed on some UCI datasets.

***Keywords*—**supervised clustering, grid-based clustering, subspace clustering

I. Introduction

CLUSTERING analysis is one of the primary methods to understand the natural grouping (or structure) of data objects in a dataset. The main objective of clustering is to separate data objects into high quality groups (or clusters), based on similarities among the data objects. Due to the acknowledgment that no single clustering method can adequately handle all sorts of cluster structures [1], and that different clustering approaches often define different definitions for clusters, it is impossible to define a universal measure of clustering quality [12].

Traditional clustering is performed in unsupervised learning manner. No class label attribute of data objects is used to guide clustering them into groups. Since the problem of finding the optimal clustering of data objects was proved to be NP-complete [13], many heuristic methods have been developed to solve the problem. Ref. [7] categorized traditional clustering algorithms into partitioning methods, hierarchical methods, density-based methods, grid-based methods, and model-based methods.

Unlike the goal of traditional clustering, [23] and [10] proposed that the goal of supervised clustering was to identify class-uniform clusters that had high data densities. According to them, not only data attribute variables, but also a class variable, take part in grouping or dividing data objects into clusters in the manner that the class variable is used to supervise the clustering. At the end, each cluster is assigned with specific class label corresponding to the majority class of data objects inside the cluster.

As reviewed by [12], the objective of subspace clustering is to find clusters in different subspaces of a dataset. Localizing search space by considering only the relevant dimensions allows subspace clustering to find clusters that exist not only in all of the original dimensions of an input dataset, as in traditional clustering algorithms, but also in various combinations of relevant dimensions.

This paper proposes grid-based supervised clustering (GBSC) that performs supervised clustering based on grid-based clustering method, density-based clustering method, and bottom-up subspace clustering method. The GBSC algorithm relies on grid-based clustering method which gradually divides data space into grid cells in the bottom-up fashion. It begins with the subspace division based on all one-dimensional spaces then proceeds by adding dimensions to be considered for the subspace division one dimension at a time until all dimensions have been considered. Finally, all adjacent dense cells whose classes are identical are merged into the same cluster using density-based clustering techniques. The paper is organized as followed. Section 2 presents backgrounds and related works in developing the GBSC algorithm. Section 3 marks on the problem statements of the GSBC algorithm. The detail of the GBSC algorithm is described in section 4. Finally in section 5, two sets of experiments designed to evaluate the effectiveness of GBSC are presented.

II. Backgrounds and Related Works

In this section, essential backgrounds on subspace clustering and supervised clustering are provided. Reviews on clustering algorithms relevant to the GBSC algorithm are also given.

A. Subspace Clustering

Data objects may be related in different ways when different subsets of dimensions are considered. Thus different clusters exist when different sets of dimensions of the data objects are used for clustering [17]. Subspace clustering aims to reveal clusters lying in various subspaces of the dataset. Ref. [12] classified subspace clustering algorithms into two major groups with regard to the search technique employed: the bottom-up search method and the top-down search method.

A number of subspace clustering algorithms were categorized and reviewed in [12]. Among them, the one that the GBSC algorithm is based upon is CLIQUE [1]. CLIQUE is one of very first subspace clustering algorithms. It is a grid-

P. Bungkomkhun is a Ph.D student, School of Applied Statistics, National Institute of Development Administration, Bangkok, Bangkok 10240, Thailand (e-mail: pornpimolb@ yahoo.com).

S. Auwatanamongkol is with School of Applied Statistics, National Institute of Development Administration, Bangkok, Bangkok 10240, Thailand (e-mail: surapong@as.nida.ac.th).

based clustering algorithm that provides an efficient approach for bottom-up subspace clustering. It uses an APRIORI style technique to find clusters in subspaces, based on the observation that dense areas in a higher-dimensional space imply the existence of dense areas in a lower-dimensional space.

CLIQUE identifies dense clusters in a subspace of maximum dimensionality by automatically identifying projections of the data objects onto various subsets of dimensions where regions of high density with respect to those objects reside. The algorithm uses a bottom-up approach in generating grid cells and identifying dense cells. It begins by finding dense units in all one-dimensional spaces. The algorithm then proceeds level-by-level, in the manner that the candidate *k*-dimensional dense cells can be determined using already determined *(k-1)*-dimensional dense cells. Hence, the set of candidate *k*-dimensional cells that might possibly be dense can be found inside dense *(k-1)*-dimensional cells only. The algorithm terminates when no more candidates are discovered. To form clusters, CLIQUE uses a depth-first search algorithm to find the connected dense cells then creates cluster descriptions in the form of DNF expression.

B. Supervised Clustering

Supervised clustering aim is to identify clusters that have high data densities and have minimal impurity, with respect to majority classes of the clusters. The clustering is performed on attribute variables under the supervision of a target class variable. As a consequence, each generated cluster is labeled with only one specific class that has majority of data objects inside the cluster. Supervised clustering procedure is therefore used not only for knowledge discovery, but also for data classification, as the cluster structure with class information can be used as a classification function [20].

Ref. [18], [16], and [2] proposed supervised clustering algorithms based on bottom-up agglomerative approach. The algorithm proposed in [15] is intended to find clusters that are homogenous in the target class variable using a probabilistic approach based on discriminative clustering to minimize distortion within clusters. Ref. [19] introduced supervised model-based clustering algorithms that were based on multivariate Gaussian mixture model which employs EM algorithm to estimate model parameters.

Ref. [4] proposed that supervised clustering can be achieved by training a clustering algorithm to produce desirable clusters. They presented SVM algorithm that learned an item-pair similarity measure to optimize clustering performance based on a variety of performance measures. Ref. [3] introduced supervised K-mean algorithm that combined Simulated Annealing with K-mean algorithm.

CCAS algorithms were developed for detecting intrusions into computer network system, through intrusion signature recognition. The algorithms starts by learning data patterns based on supervised clustering procedure, and afterwards use these patterns for data classification. The original version of CCAS [21] starts with two dummy clusters and allows clusters of each individual class to spread over the entire data space regardless of the training sequence of data objects. Ref. [8] modified the original CCAS with grid-based method to limit the search space in splitting training data objects into smaller size clusters. The algorithm begins with dividing data space into equal size grid cells. It then performs dummy-based clustering only on data objects lying in the same cell.

Ref. [9] enhanced the robustness of CCAS by strengthening the algorithm with three post-processing steps: data redistribution, supervised grouping of clusters, and removal of outliers. ECCAS [10] enabled CCAS to handle data of mixed types, by introducing two methods for combining numerical and nominal variables in calculating distance measure. The first method combines different distance measures for each type of variables into a single distance measure ranging between 0 and 1. The second method is based on conversion of nominal variables to binary variables, and then treats these binary variables as numeric variables.

Three representative-based supervised clustering algorithms were introduced in [23]: Supervised Partitioning Around Medoids (SPAM), Single Representative Insertion/Deletion Steepest Decent Hill Climbing with Randomized Start (SRIDHCR), and Supervised Clustering using Evolutionary Computing (SCEC). In their paper, a new fitness function used for measuring the performance of supervised clustering algorithms was proposed. Instead of relying only on the tightness of data objects in each cluster, like most of the traditional clustering algorithms, the three algorithms weights cluster purity against the number of generated clusters in the proposed fitness function.

SPAM, aimed to be the variation of PAM (Partitioning Around Medoids) that uses the proposed fitness function, starts by randomly selecting a medoid from the most frequent class data objects as the first representative. The algorithm then fills up the initial set of representatives with non-representative objects. The number of representatives is fixed by a pre-defined figure. SPAM later on repeatedly explores all possible replacements of a single representative of the most current solution by a single non-representative, provided that the new set of representatives induces minimum fitness function value. The algorithm terminates if none of the replacements can provide lower fitness function value.

In order to eliminate the limitation of SPAM that the number of representatives is fixed by a pre-defined parameter, SRIDHCR algorithm permits either adding or removing any representatives into or from the current set of cluster representatives. The algorithm terminates when there is no significant improvement in the solution quality (measured by the value of the fitness function).

Besides the above two greedy algorithms, [23] also proposed an evolutionary computing algorithm called SCEC. The algorithm evolves a population of solutions, each is a set of representatives, over a pre-defined number of generations. The best solution of the last generation is chosen to be the set of representatives for the clustering. Each solution in the initial population is randomly created. Populations of the

subsequent generations are generated through three genetic operators: mutation, crossover, and copy. SCEC uses K-tournament selection method (with tournament size of K = 2) in selecting potential solutions to participate in creating new population. Different adaptive values are used to control the probabilities of applying each of the three genetic operators to generate new solutions for the subsequent generations.

Ref. [5], [6] proposed two supervised clustering algorithms based on prototype-based clustering methodology: Supervised Growing Neural Gas (SGNG) and Robust Supervised Growing Neural Gas (RSGNG). The SGNG incorporates Growing Neural Gas network with various techniques such as Type Two Learning Vector Quantization (LVQ2), adaptive learning rates, and cluster repulsion mechanisms. The SGNG also proposed a new validity based on geometry measurement paradigm in order to determine the optimal number of prototypes. Due to drawbacks of the SGNG of being sensitive to the prototype initialization, the sequence of input data objects, and the presence of noises, the RSGNG is intended to be the robust version of SGNG. The RSGNG incorporates SGNG learning schema with the outlier resistant strategy. Moreover, to determine the optimal number of prototypes where data objects may include some outliers, a modified validity index was proposed. The index is based on MDL() or Minimum Description Length technique.

III. Definitions

The definitions of major terms to be used through out this paper are defined in this section.

A. Data Objects

A data object is considered a data point in a *d*-dimensional space. Formally, each data point is a $(d+1)$-tuple in the form $\{a_1, a_2, ..., a_d, T\}$, where a_i represents value of the i^{th} predictor variable (or attribute) and T represents the value of the target variable (or class label) of the data point [22].

B. Grid Cells

Let $A_1, A_2, ..., A_d$ be sets of dimensions (or attributes, or predictor variables) of any datasets, and let $A = A_1 \times A_2 \times ..... \times A_d$ be the *d*-dimensional data space. The problem is to divide the data space A into $\prod_{i=1}^{d} P_i$ non-overlapping hyper-rectangular grid cells, where P_i represents the number of intervals in the i^{th} dimension of *d*-dimensional data space. A cell is defined by a set of *d*-dimensional hyperplanes, all of which are parallel to $(d-1)$ coordinate axes.

To accomplish this, the range of the value domain of each dimension A_i is partitioned into P_i number of mutually exclusive equal-size right-opened intervals $I_i^j = [l_i^j, h_i^j)$, $1 \le j \le P_i$, where l_i^j and h_i^j respectively denotes the start value and end value of the j^{th} interval in the i^{th} dimension, and hence each cell is represented in the form $U = \{I_1, I_2, ..., I_d\}$ [11].

C. Dense Cells

An object $a = \{a_1, a_2, ...a_d\}$, where a_i is the value of the i^{th} dimension, is said to lie in a cell U only if $l_i \le a_i < h_i$ for all I_i. Since each cell U is formed upon the intersection of one interval from each of all attributes, obtained by partitioning each dimension into P_i intervals of equal length, the volume of all grid cells are equal. As a consequence, the density of each cell can be measured in term of the number of data objects lying within the region of the cell. A cell U is called dense if the density of the majority class data objects in the cell is at least equal to the density threshold, the only input parameter for the GBSC algorithm.

D. Clusters

As defined by [1], a cluster is a maximal set of connected dense cells in *d*-dimensions. The problem is to separate all identified dense cells D into $D_1, D_2, ..., D_k$ sets, such that all cells in the set D_i are connected, and no two cells $U_i \in D_i$, $U_j \in D_j$ with $i \ne j$ are connected. Two *d*-dimensional cells U_1, U_2 are declared connected, either in case they share at least one common corner point, or their exists another *d*-dimensional cell U_s to which both U_1 and U_2 are connected.

If a running number is assigned to each interval in all dimensions, starting from 1 to P_i, where P_i is the number of intervals in the i^{th} dimension, each cell can be represented in the form $U_j = \{I_{1j}, I_{2j}, ..., I_{dj}\}$, where I_{ij} is the interval number of the cell j in the i^{th} dimension. Cells $U_1 = \{I_{11}, I_{21}, ..., I_{d1}\}$ and $U_2 = \{I_{12}, I_{22}, ..., I_{d2}\}$ are claimed connected if all $|I_{i1} - I_{i2}| \le 1$, where I_{i1} and I_{i2} are the interval numbers of the i^{th} dimension of U_1 and U_2 respectively.

IV. Grid-based Supervised Clustering

The GBSC algorithm is a bottom-up supervised clustering algorithm relying on the combination of the concepts of grid-based clustering, density-based clustering, and the downward closure property of density used in subspace clustering. The algorithm uses heuristics to partition data spaces into grid cells, and defines a cluster as a set of adjacent hyper-rectangle grid cells with the same-class label.

The GBSC algorithm possesses all of the good clustering properties mentioned in [14]. The algorithm has ability to produce identical results regardless of the order of data objects to be processed. It can automatically determine the optimal number of clusters. Moreover, the GBSC algorithm is resistant to noises, can handle clusters of arbitrary shapes and sizes

without making any assumption about the distribution of data objects. Moreover, the GBSC algorithm is strengthened with the ability to automatically suggest the number of intervals to be used in partitioning each dimension. .

A. Fitness Function

The objective of supervised clustering is to identify groups of data objects, that posses low impurities, and at the same time the clustering wants to keep the number of groups as low as possible. To accomplish this, [23] proposed the following fitness function, *q(x)*, as a validity measurement to evaluate the performance of a supervised clustering algorithm.

$$q(x) = Impurity(x) + \beta * Penalty(k)$$

$$where \quad Impurity(x) = \frac{number\ of\ \text{min}\ ority\ objects}{n}$$

$$Penalty(k) = \begin{cases} \sqrt{\frac{k-c}{n}}, & k > c \\ 0, & k \le c \end{cases}$$

k = *number of generated clusters*

c = *number of classes*

n = *number of data objects*

The proposed fitness function consists of 2 contradictory parts, *Impurity()* and *Penalty()*. Due to the objective of supervised clustering, the *q(x)* value must be kept as low as possible. Further split of data objects into more clusters may cause a decrease on *Impurity()* value but an increase in *Penalty()* value. The parameter β can be selected between (*0, 5.0*] to put a weight on the significance of the *Penalty()* part against the *Impurity()* part, i.e. the higher the β value, the higher the significance of the *penalty* part.

Under the thorough consideration that the above fitness function can certainly lead supervised clustering to yield the most effective solution, this *q(x)* function is chosen to be the fitness function for the GBSC algorithm.

B. GBSC Algorithm

The basic idea of the GBSC algorithm is to create uniform-size grid cells over the whole data space, and afterwards define clusters by merging together all adjacent dense cells with regard to the data objects' classes. In consequence, the data objects lying inside the region of such adjacent dense cells are claimed to be in the same cluster. For the GBSC algorithm, each dimension is partitioned into same-size intervals, although the numbers of intervals of different dimensions are allowed to be different.

The numbers of intervals for each dimension must be carefully selected so they produce the smallest value of the fitness function *q(x)*. Such numbers are automatically determined by the GBSC algorithm using a greedy method. Fig. 1 shows the framework of the GBSC algorithm.

```
- noi = noc
- min-q(x) = 1.0,  spec-q(x) [d] = 1.0
- continue = 'y'
- while (continue)
  {
     - STEP 1: DIMENSIONS SEQUENCING
     - STEP 2: GRID CELLS IDENTIFYING

     if (mutual-q(x) < min-q(x))
        { - min-q(x) = mutual-q(x)
          - keep copy of grid cells file
          - noi = noi ++ }
     else
        { - continue = 'n' }
  }

- STEP 3: CLUSTERS FORMATION
```

Fig. 1 The GBSC algorithm Framework

Let *noi* represent the maximum number of intervals each attribute is allowed to be divided into. The GBSC comprises the following three main steps :

1) Dimensions Sequencing

The objective of this step is to find the order of dimensions for which they are to be processed sequentially in the subspace clustering. The dimensions are ordered according to their anticipated potentials in generating clusters of possible smallest fitness function *q(x)*. The detail of the process in the step is shown in Fig.2.

```
1   for (i from 1 to d)
2   { compute ranges of intervals when partitioning
        i^th dimension into noi intervals
3     divide data objects into groups based on
        the intervals defined by line 2
4     for (each interval)
5     { if (the interval is dense)
6       { define cell class, and count
            impurity objects } }
7     form intermediate clusters
8     compute q(x) of the intermediate clusters
9     if (q(x) < spec-q(x[i]))
10    { spec-q(x)[i] = q(x)
        spec-interval[i] = noi }
11  }
12  create d-node link list
      storing values of i and spec-interval[i]
```

Fig. 2 Dimensions Sequencing Step

The first task of this step (*line 1-11*) is to determine the smallest *q(x)* value that each specific dimension could produce (represented by *spec-q(x)*) if that dimension is solely used for the subspace clustering. Let i be an attribute or dimension number. The i^{th} iteration of the step starts with the calculation

of the range of each interval as the i^{th} attribute's value domain is divided into *noi* intervals (*line 2*). The data objects are then partitioned into groups in accordance with these ranges (*line 3*). Upon visiting each group (*line 4*), only data objects lying in the ranges of the intervals (in other words, *1*-dimensional cells) that are dense (their densities are above the pre-defined threshold) are investigated (*line* 5). Since each dimension is divided into equal-size intervals, all generated grid cells cover the same amount of spaces, and hence the density of each cell can be calculated by simply counting the number of data objects in that cell. Each dense cell is labeled with the majority class of the objects belonging to that cell, whereas the minority class objects are counted as impurities (*line 6*), and finally all objects lying in the region of the adjacent cells with same class label are merged into the same clusters (*line 7*) forming the intermediate clustering. The fitness function *q(x)* of that clustering is afterward computed (*line* 8), and compared with the current lowest *q(x)* of the i^{th} attribute stored in *spec-q(x)[i]* (*line* 9), and the lower one is kept in *spec-q(x)[i]* whereas its corresponding *noi* value is kept in the *spec-interval [i]* (*line 1*0).

The last task of the first step is to find the sequence of dimensions for subspace clustering during the second step. Using a greedy approach, a dimension which generates smaller *q(x)* values when it is solely used for the subspace clustering should be given higher priority for the subspace clustering. Hence, to create the sequence, the dimensions accompanied with their specific *noi* values are sorted in the *d*-node link list in ascending order according to their *spec-q(x)* values (*line 1*2).

2) Grid Cells Identifying

The intention of this step is to find out the delineation of grid cells that would produce the possible smallest q(x) value resulting from the mutual performance of all dimensions working together (represented by *mutual-q(x)*), within each *noi* limitation. The GBSC algorithm creates grid cells in the bottom-up fashion, by gradually and repeatedly adding dimensions into the grid space one at a time, in accordance with the order specified in the *d*-node link list. Starting from the dimension whose corresponding number of intervals renders the smallest q(x) value as the first dimension, the subsequent dimensions are added into the cells with the grid length that make the *mutual-q(x)* of the new clustering less than or at least equal to the former one.

After each new dimension is added, the information about current grid cells including their data objects are written onto the external storage. The information will be to be used in the next iteration. This external storage allows the GBSC algorithm to avoid the memory confinement problem, and hence enables the algorithm to cope with datasets of any size.

Refer to the fact that the number of generated grid cells can be computed as $\prod_{i=1}^{d} P_i$, where *d* represents the number of dimensions and P_i represents the number of intervals in the i^{th} dimensions of *d*-dimensional space, the number of created cells increases dramatically whenever the number of dimensions increases. In the occasion when the number of dimensions is rather high, not all grid cells contain data objects and the number of grid cells containing data objects is usually tremendously low when compared with the number of created cells.

```
1   compute ranges of intervals of dimension defined in
        the first node of the d-node link list
2   divide data objects into groups based on
        the intervals defined by line 1
3   for ( each interval )
4   { if (the interval is dense)
5     { write the interval as 1-level cell in the external file  } }
6   mutual-q(x) = spec-q(x) [attribute# in the first node]
7   for ( i from 2 to d )
8   { min-interval = 1
9     for ( j from 1 to noi )
10    { compute j ranges of intervals as
              the i^th dimension of the cells
11      for ( each (i-1) level cell stored in the external file )
12      { divide the cell into sub cells based on
                intervals defined by line 11  }
13        for ( each sub cells )
14        { if ( the sub cell is dense )
15          { define the sub cell class,
                  and count impurity objects in the cell  } }
16        form intermediate clusters
17        compute q(x) of the intermediate clusters
18        if  ( q(x) < mutual-q(x) )
19        {  mutual-q(x) = q(x)
             min-interval = j   }
20    }
21    compute min-interval ranges of intervals as
          the i^th dimension of the cells
22    for (each (i-1) level cell stored in the external file )
23    {  divide the cell  into sub cells based on
            intervals defined by line 22
24      if ( the sub cell is dense)
25      { write the sub cell as i-level cell in the external file }}
26  }
```

Fig. 3 Grid Cells Identifying Algorithm

To reduce the search space, only dense cells in the current-dimensional-level grid space are kept for the processing of the subsequent higher dimensional level. This procedure results in saving a lot of processing time, since majority parts of search space are discarded and/or marked as outliers. As a consequence, the GBSC algorithm allows only dense cells in (*i-1*) dimension to be candidates into the generation of *i*-dimensional cells, based on the perception that only (*i-1*)-dimensional cells that are dense can breed dense cells in their corresponding *i*-dimensional space. The algorithm elucidating the processes of this step is shown in Fig.3.

3) Clusters Formation

To create final clusters, adjacent same-class labeled dense cells are merged into a same cluster. The input for this step is a set of dense cell blocks *D*, each of which consists of cell's structure, cell's class, and data objects belonging to that cell. Starting from any cell $U \in D$ as the first cell in the being generated cluster C_j, the GBSC algorithm first searches in *D* looking for all of U's connected cells, $U_j \in D$, which are of the same class as U, appends U_j into C_j, and removes U_j from *D*. The algorithm iteratively picks up the subsequent cell U in C_j, searches in *D* for all of its connected cells $U_j \in D$ whose classes are the same as those of cell *U*, appends U_j into C_j, and removes U_j from *D*. The iteration stops when all dense cells in C_j have been visited.

The process is equivalent to the construction process of breadth-first-search (BFS) tree. One of the dense cells $U_j \in D$ can be used as a starting node or root node of the tree. All nodes in the same tree represent all dense cells that are either directly or indirectly connected in one way or another to the root node cell. The GBSC algorithm claims all dense cells representing nodes in the same BFS search tree are in the same cluster. With this cluster formation procedure, the GBSC algorithm can generate clusters of any shapes and sizes without presuming any specific mathematical form for data distribution, and can produce identical results regardless of the order in which input data objects are presented.

V. Experimental Results

Two experiments were performed to evaluate the effectiveness of the GBSC algorithm. The first experiment was performed on two-dimension synthetic datasets under the permission of the author of [5], and the second one was performed on datasets obtained from University of California at Irving Machine Learning repository [20].

A. The First Experiments on 2D synthetic datasets

The experiment is intended to affirm the effectiveness of the GBSC algorithm under various proclaimed situations through *2-D* geometrical presentations. The GBSC algorithm was performed on four 2-D syntactic datasets [5] representing 4 different scenarios : *Test-1* (1,250 records, 2 classes), *Test-2* (1,000 records, 6 classes), *Test-3* (4,185 records, 4 classes), and *Test-4* (3,421 records, 4 classes). The results of the experiments are graphically displayed in Fig. 4. Objects that are claimed as impurities are encircled with the dark edge.

Fig. 4(a), illustrates the result of applying the GBSC on *Test-1* dataset. It shows two pure cross-board shape clusters: *A* and *B*, one cluster per one individual class. This result confirms that the GBSC algorithm has ability to identify any irregular shape clusters. The result on *Test-2* dataset is shown in Fig. 4(b). The GBSC algorithm depicts 14 sparse various shape and density clusters: *A1, A2, A3, B1, B2, C1, C2, C3, D1, D2, E1, E2, E3*, and *F*, with sparse-and-scattered impurity objects. There are two clusters which contain only one objects: *A3* and *E3*, and hence may be counted as outliers.

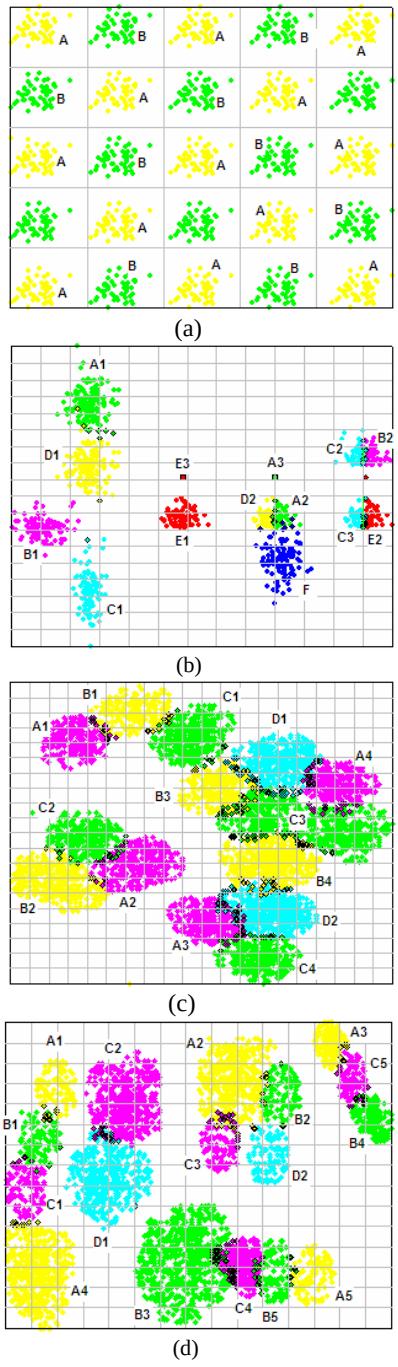

Fig. 4 Results on four 2-D synthetic datasets

The set of clusters shown in Fig. 4(c) is the result from running the GBSC algorithm on *Test-3* dataset. Fourteen crowded similar shape, size, and density clusters are delineated: *A1, A2, A3, A4, B1, B2, B3, B4, C1, C2, C3, C4,*

D1, and *D2* , most of which are overlapped, as can be seen surrounded by considerable number of impurity objects. The results on *Test-4* datasets is shown in Fig. 4(d). The GBSC algorithm defines 17 various size and density clusters: *A1*, *A2*, *A3*, *A4*, *A5*, *B1*, *B2*, *B3*, *B4*, *B5*, *C1*, *C2*, *C3*, *C4*, *C5*, *D1*, and *D2*. Some contain a small number of impurity objects in various locations. The results from this experiment endorse the ability of the GBSC in identifying clusters of any shapes and sizes without presuming any canonical form of data distribution, as well as the ability in handling outliers.

B. The Second Experiment on UCI datasets

The objective of this experiment is to evaluate the performance of the GBSC algorithm in a comparative manner with some other supervised clustering algorithms. The experiments were performed on four datasets obtained from University of California at Irving Machine Learning repository [20]: *Iris-Plans* (150 records, 4 attributes, 3 classes), *Pimma-Indian Diabetes* (768 records, 8 attributes, 2 classes), *Vehicle Silhouettes* (846 records, 18 attributes, 4 classes), and *Image-Segmentation* (2100 records, 19 attributes, 7 classes). The results of the best fitness values *q(x)* from the experiment are compared with those results from SPAM, SREDHCR, and SCEC reported in [23], and the best solutions selected from SGNG and RSGNG in [5].

TABLE I
EXPERIMENTAL RESULTS (ON $\beta = 0.1$)

algorithm	# of cluster	cluster purity	q(x)
Iris-Plants (150/4/3)			
SCEC	5	0.993	0.018
SREDHCR	3	0.980	0.020
SPAM	3	0.973	0.027
SGNG	5	0.986	0.026
RSGNG	5	0.986	0.026
GBSC	**3**	**0.987**	**0.013**
Pima-Indian Diabetes (768/8/2)			
SCEC	64	0.893	0.135
SREDHCR	45	0.859	0.164
SPAM	45	0.822	0.202
SGNG	75	0.941	0.090
RSGNG	75	0.911	0.120
GBSC	**30**	**0.974**	**0.045**
Vehicle Silhouettes (846/18/4)			
SCEC	132	0.923	0.116
SREDHCR	65	0.835	0.192
SPAM	65	0.764	0.263
SGNG	132	0.946	0.093
RSGNG	132	0.955	0.084
GBSC	**11**	**0.999**	**0.010**
Image-Segmentation (2100/19/7)			
SCEC	60	0.989	0.026
SREDHCR	53	0.980	0.035
SPAM	53	0.944	0.071
SGNG	60	0.977	0.039
RSGNG	60	0.969	0.047
GBSC	**20**	**0.994**	**0.014**

TABLE II
EXPERIMENTAL RESULTS (ON $\beta = 0.4$)

algorithm	# of cluster	cluster purity	q(x)
Iris-Plants (150/4/3)			
SCEC	3	0.987	0.013
SREDHCR	3	0.987	0.013
SPAM	3	0.973	0.027
SGNG	3	0.986	0.027
RSGNG	3	0.986	0.027
GBSC	**3**	**0.987**	**0.013**
Pima-Indian Diabetes (768/8/2)			
SCEC	9	0.819	0.219
SREDHCR	2	0.776	0.224
SPAM	2	0.772	0.227
SGNG	75	0.941	0.182
RSGNG	75	0.911	0.212
GBSC	**5**	**0.905**	**0.120**
Vehicle Silhouettes (846/18/4)			
SCEC	61	0.857	0.247
SREDHCR	56	0.835	0.265
SPAM	56	0.754	0.345
SGNG	132	0.946	0.210
RSGNG	132	0.955	0.201
GBSC	**4**	**0.993**	**0.007**
Image-Segmentation (2100/19/7)			
SCEC	28	0.969	0.069
SREDHCR	32	0.970	0.074
SPAM	32	0.940	0.103
SGNG	42	0.977	0.085
RSGNG	42	0.969	0.093
GBSC	**7**	**0.956**	**0.044**

TABLE I and TABLE II show the experimental results at β value 0.1 and 0.4 respectively. The results in TABLE I and TABLE II does show that the GBSC yields the best solutions (the smallest *q(x)* value) among the six algorithms in both values of β. Furthermore, the numbers of clusters generated by the other five algorithms in the last three datasets are remarkably higher than those by the GBSC, due to the nature of representative-based clustering algorithms that incline to create global-shape clusters. As the conclusion, the GBSC can cope with datasets of any shapes, and still outperforms on irregular-shape datasets.

REFERENCES

[1] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, "Automatic subspace slustering of high dimensional data for data mining applications," In Proc. ACM SIGMOD International Conference on Management of Data, Seattle, Washington, June 1998, pp. 94-105.

[2] J. S. Aguilar, R. Ruiz, J. C. Riquelme, and R. Giraldez, "SNN: A supervised clustering algorithm," In Proc. 14th International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems (IEA/AIE 2001)

[3] S. H. Al-Harbi, and V. J. Rayward-Smith, "Adaptive k-means for supervised clustering," Applied Intelligence, Volume 24, Number 3, pp. 219-226(8), June 2006.

[4] T. Finley and T. Joachims, "Supervised clustering with support vector machines," In Proc. International conference on Machine learning, Bonn, Germany, August 07 - 11, 2005, pp. 217-224.

[5] A. Jirayusakul, "Supervised growing neural gas algorithm in clustering analysis," Ph.D. dissertation, School of Applied Statistics, National Institute of Development Administration, Thailand, 2007, unpublised.

[6] A. Jirayusakul, and S. Auwatanamongkol, "A supervised growing neural gas algorithm for cluster analysis," International Journal of Hybrid Intelligent Systems, Vol. 4, No.2, 2007.
[7] S. B. Kotsiantis and P. E. Pintelas, "Recent Advances in Clustering: A Brief Survey," Transactions on Information Science and Applications, 2004,1(1):73-81.
[8] X. Li and N. Ye, "Grid-and Dummy-Cluster-Based Learning of Normal and Intrusive Clusters of Computer Intrusion Detection," Journal of Quality and Reliability Engineering International, Vol. 18, No. 3, pp. 231-242.
[9] X. Li and N. Ye, "A Supervised Clustering Algorithm for Computer Intrusion Detection," Knowledge and Information Systems, Vol. 8, No.4, pp. 498-509.
[10] X. Li and N. Ye, "A Supervised Clustering and Classification Algorithm for Mining Data With Mixed Variables," IEEE Transactions on Systems, Man, and Cybernetics-Part A, Vol. 36, No. 2, pp. 396-406.
[11] N. H. Park and W. S. Lee, "Statistical Grid-based Clustering over Data Streams," SIGMODD Record, Vol.33, No.1, March 2004.
[12] L. Parsans, E. Haque, and H. Liu, "Subspace Clustering for High Dimensional Data: A Review," ACM SIGKDD Explorations Newsletter, 6(1):90–105, June 2004.
[13] C. M. Procopiuc, 1997, "Clustering Problems and their Applications (a Survey)," Available: http://www.cs.duke.edu/~magda.
[14] G. Sheikholeslami, S. Chatterjee, and A. Zhang, "WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases," In Proc. International Conference on Very Large Databases, New York City, August 24-27, 1998.
[15] *J. Sinkkonen, S. Kaski, and J. Nikkila, "Discriminative Clustering: Optimal Contingency Tables by Learning Metrics," In Proc. European Conference on Machine Learning (ECML'02), Springer-Veriag, London, pp. 418-430.*
[16] N. Slonim and N. Tishby, "Agglomerative Information Bottleneck," In Proc. Neural Information Processing Systems, pp. 617–623, 1999.
[17] P-N. Tan, M. Steinbach, and V. Kumar, "Introduction to Data Mining. Boston: Pearson Education, Inc., pp. 604-608.
[18] N. Tishby, F. C. Pereira, and W. Bialek, "The Information Bottleneck Method," In Proceedings of the Allerton Conference on Communication and Computation, 1999.
[19] Y. Qu and Z. Xu, "Supervised Clustering Analysis for Microarray Data Based on Multivariate Gauussian Mixture," Bioinformatics. 20 (Aug) : 1275-1288.
[20] University of California at Irving, Machine Learning Repository. Available: http:/www.ics.edu/~mlearn/MLRepository.html
[21] N. Ye and X. Li, "A Scalable Clustering Technique for Intrusion Signature Recognition," In Proc. The 2001 IEEE Workshop on Information Assurance and Security United States Military Academy, West Point, NY, 5-6 June, 2001.
[22] N. Ye and X. Li, 2005, "Method for Classifying Data using Clustering and Classification Algorithm Supervised," Available: http://www.patentstorm.us/patents/6907436/fulltext.html.
[23] N. Zeidat, C. F. Eick, and Z. Zhao, "Supervised Clustering: Algorithms and Applications," Available: http://www2.cs.uh.edu/~ceick/kdd/ZEZ06.pdf.