Appl Intell (2011) 35:110–122
DOI 10.1007/s10489-009-0206-7

Integrating multi-objective genetic algorithm based clustering and data partitioning for skyline computation

Tansel Özyer · Ming Zhang · Reda Alhajj

Published online: 24 December 2009

Abstract Skyline computation in databases has been a hot topic in the literature because of its interesting applications. The basic idea is to find non-dominated values within a database. The task is mainly a multi-objective optimization process as described in this paper. This motivated for our approach that employs a multi-objective genetic algorithm based clustering approach to find the pareto-optimal front which allows us to locate skylines within a given data. To tackle large data, we simply split the data into manageable subsets and concentrate our analysis on the subsets instead of the whole data at once. The proposed approach produced interesting results as demonstrated by the outcome from the conducted experiments.

Keywords Skyline computation · Multi-objective clustering · Genetic algorithm · Cluster validation

1 Introduction

Recently, the database community has realized the need for a new operator capable of returning tuples/values that dominate the rest of the tuples/values in the database, e.g. [1, 27, 44]. This new operator named skyline has already attracted the attention in the research community for its growing set of practical applications, like dining, accommodation, flight, etc. For instance, in selecting a flight, we mainly consider fare, number of stopovers, waiting time between flights, etc. as the main objectives to be satisfied; these are mostly conflicting. Accordingly, the skyline operator mainly suggests a baseline for multi-criteria decision making, data mining and data visualization, among others.

Canonically, given a set of n objects $\{p_1, p_2, \ldots, p_n\}$, the skyline operator returns all objects p_i such that p_i is not dominated by any other object p_j. Objects are retrieved with respect to some criteria. Assume there are m criteria that should be evaluated; one way of handling the problem is merging the m criteria to form the composite criteria of the queries to be evaluated. The queries can be any function such as $f_1, f_2, \ldots, f_m$, and they are minimized (maximized) with respect to a monotone function f_q.

There are many real life examples that have been described in the literature as good applications of skyline queries; these include finding appropriate route for flights, hotels for accommodation, restaurant for dinning, etc. The main issue here is having multiple criteria to be considered in finding a solution and hence query evaluation results in more than one solution because each criterion may favor different record(s) as suggestion since it is hard to satisfy all optimal attribute values in one record. For instance, finding an appropriate route for the flight is not always the shortest route. Passengers must take other criteria into account such as waiting times, stopovers, fare, service quality of airline, benefits, etc. A hotel room can be the cheapest but its proximity to facilities, living standards, and meals are other doubts to take care of. While shopping, quality versus price is the basic concern that people generally consider. There are many application areas such as top-k queries, nearest

T. Özyer
Department of Computer Engineering, TOBB ETU Economics and Technology University, Sogutozu Cad No. 43, Sogutozu, 06560 Ankara, Turkey

M. Zhang · R. Alhajj (✉)
Department of Computer Science, University of Calgary, Calgary, Alberta, Canada
e-mail: alhajj@ucalgary.ca

R. Alhajj
Department of Computer Science, Global University, Beirut, Lebanon

neighbor search, and particularly the convex hull problem as defined in [35] for finding the optimal subset of skyline objects as the optimal silhouette for linear preferences.

Borzsonyi et al. [5] described the following SQL syntax for the skyline computation for route suggestion of flights from Istanbul to New York:
Select *, From Routes, Skyline of [no of stopovers] min, Distance min, Price min, [Total Waiting Time].

The area is very promising and computationally intensive; researchers are investigating effective methods for efficiently computing the skylines for a give domain. This motivated for the work described in this paper; we describe a novel approach that incorporates our achievements in multi-objective genetic algorithms based clustering in order to find skylines within the data. The produced alternative clustering results are evaluated using validity indexes and majority voting to decide on the most appropriate number of clusters. The latter is used in producing the skylines. The proposed approach reflects into the solution set all insertions and deletions done on the data. Hence it is a dynamic approach. The reported test results are promising, and demonstrate the effectiveness and applicability of the proposed approach.

The rest of this paper is organized as follows. Section 2 is an overview of the necessary background and the related work. Section 3 describes the employed multi-objective genetic algorithm based clustering approach and how it facilitates identifying skylines. Section 4 reports experimental results. Section 5 is conclusions.

2 Necessary background and related work

Before presenting an overview of some of the basic approaches that have been developed for skyline computation, we briefly describe the background necessary for understanding the proposed approach.

2.1 Multi-objective optimization

Skylines are directly linked to the concept of pareto front that has been widely researched especially in multi-objective genetic algorithms. A multi-objective optimization problem, e.g., [6, 15] has n decision variables, k objective functions, and m constraints. Objective functions and constraints are functions of the decision variables. The optimization goal may be described as follows:

$$\text{Max/min} \quad y = f(x) = (f_1(x), f_2(x), \ldots, f_k(x))$$

$$\text{subject to} \quad e(x) = (e_1(x), e_2(x), \ldots, e_m(x)) \leq 0$$

$$\text{where } x = (x_1, x_2, \ldots, x_n) \in X$$

$$y = (y_1, y_2, \ldots, y_k) \in Y$$

Where x is the attribute vector, y is the objective vector, X denotes the attribute space, and Y is called the objective space. The constraints $e(x) \leq 0$ determine the set of feasible solutions [48].

Solutions to a multi-objective optimization problem are mathematically expressed in terms of non-dominated or superior points. In a minimization problem, a vector $y(1)$ is partially smaller than another vector $y(2)$, denoted $y(1) \prec y(2)$, when no value of $y(2)$ is smaller than $y(1)$ and at least one value of $y(2)$ is strictly greater than $y(1)$. If $y(1)$ is partially smaller than $y(2)$, we say that $y(1)$ dominates $y(2)$ or the solution $y(2)$ is inferior to $y(1)$. Any vector which is not dominated by any other vector is said to be non-dominated or non-inferior. A multi-objective optimization problem has a set of alternative solutions based on the different combinations of the objectives. The optimal solutions are referred to as non-dominated solutions [38]. Conflicting objectives are the basis for finding alternative solutions. As conflict increases, it would be hard to produce only one single feasible solution that satisfies all objectives and the nature of a problem may require more than one objective. One traditional multi-objective optimization, multiple objectives may be combined to form one objective function. One of the traditional methods being used is weighting each objective and scalarizing the result. However, the process is subjective.

There are many application areas of multi-objective genetic algorithms (MOGA) in the literature such as flowshop scheduling [3, 30], telecommunication systems [21, 22], reliability optimization [14, 46], finance, covering tour problem, Aerodynamic design, fluid power systems, manufacturing systems, DNA sequences, attribute selection [12], and clustering [31–33].

One of the suggestions proposed by Deb et al. [13] is to rank the solution in MOGAs, where Pareto front extraction automatically groups the set of solutions in layers. The first layer (the closest to the optimal) presents the best solutions found; and moving away from the optimal layer each of the other layers contains one set of solutions that might turn into optimal shall the dominating solutions be removed from the database.In other words, the solution set is ranked and categorized in layers; each object p_i has a rank as layer number; deletions and insertions may affect the dominance by eliminating some of the dominating solutions or by introducing some new dominating solutions.

2.2 Clustering with multi objective genetic algorithm

We have already studies various aspects of multi-objective clustering, e.g., [2, 13, 31–34] and the reported results are satisfactory. This experience is to be invested in developing the effective approach for skyline computation. Genetic algorithms mimic the biological life that goes through the initialization of population, reproduction and selection that ends up with the final population as the result set. We have adopted the idea of utilizing multi-objective genetic algorithm for clustering the points according to the attribute

space. This algorithm employs the idea of clustering with multiple objectives as preferred to single objective and results in alternative solutions leading to the ideal number of clusters. This has been proven to work on several popular datasets from different domains.

Employing the idea of pareto front does not only keep the skylines; it will also keep all the layers leading to a flexible model that facilitates dynamic handling of skylines. The study in [13] addresses the problem of pareto front ranking that has also been used in our multi-objective clustering algorithms [2, 13, 31–34].

Our idea depends on the multi-objective clustering that puts data into different clusters such that all data are clustered according to three different criteria. These are minimizing the intra-cluster distance; maximizing inter-cluster distance; and minimizing the centroid average. Previously, different cases have been investigated under different circumstances such as large scale data, memory scalable data, attribute weighting, etc. One of the produced alternative solutions is to be preferred to others. This process depends on the result from cluster validation as described next.

2.3 Cluster validation

Cluster validation is used to investigate the best partitioning fitting the underlying data. Here, the target is to find the optimal number of clusters. If the dataset is already known, the method for checking the optimal number of clusters is by validating each of the alternative candidates with different parameters to find out the one that has more true labels of the classes. This is achieved by employing validity indices, which are classified as external, internal and relative. The first two categories belong to statistical tests with high computational cost. Internal validity indices use quantities and features of the data. External validity indices use priori known true class labels as reference partitioning; they use the agreement between the clustering algorithm and the prediction with respect to the reference partition F&M, Rand, Jaccard and Hubert are known external validity indices. Relative validity indices use heuristically selected parameters favoring the clustering criteria. They are used as sort of metrics for clustering assessment. Studies described in [31, 32] and [31, 33, 34] summarize cluster validation methods.

2.4 Skyline computation

Skyline computation in memory can date back to the works described in [23, 28]. To the best of our knowledge, [5] is the first work to address the skyline problem in the database environment. Several algorithms were proposed in [5]: The D&C (*Divide-and-Conquer*) algorithm divides the data set into multiple sections such that each section could fit in the main memory, applies main-memory algorithm like [28] to each individual section (in memory), and merges the partial results to obtain the skylines of the whole data set. The BNL (*Block Nested Loop*) algorithm scans the data set and maintains a candidate skyline set in memory; each visited point is compared with the points in the candidate set which is updated accordingly. Both algorithms do not need to preprocess the data set and do not use index, thus have wide applicability. However, both algorithms require scanning the whole database at least once. For the BNL algorithm, if the candidate set can not fit in the main memory, multiple scans are required. In addition, neither of the two algorithms supports progressive processing as a skyline can not be determined before scanning the whole database.

The *SFS* (*Sort-First-Skyline*) [11] is an improved version of BNL by pre-sorting the database using a monotonic function. As the dataset is sorted, a point can not be dominated by points behind it, thus *SFS* can determine if a point is a skyline by comparing it with the points visited before it. *SFS* is progressive but still need to scan the whole database. The *bitmap* method proposed in [39] represents each point by binary bit string, which contains the information of the relative position of each dimension of the point. Consequently, skyline testing is performed by bit operations. However, to decide whether a point is a skyline, it needs to retrieve the bit-string representation for all the data points. Furthermore, bitmap is only suitable for static databases because insertions may change the bit-string representation of all the existing points.

Tan *et al.* [39] introduced another approach to avoid scanning the whole database. This method divides a set of d-dimensional points into d sorted lists; the ith list contains points with ith coordinate as the minimum coordinate among all the dimensions; and the search is perform among the d sorted lists iteratively; at each step, it searches the list with the first unvisited point having the smallest sorting key among all the lists. The search terminates when it finds a point with all coordinates larger than the next point to be visited. This method does not work for sub-space skyline queries because the lists change under different sub-spaces.

Kossmann *et al.* [20] proposed the *NN* algorithm using R-tree index. Using R-tree, it finds the point p that is nearest to the origin; p must be a skyline point. The space that is dominated by p can be pruned and the remaining space is partitioned into multiple spaces and inserted into a *to-do* list. The same process is applied recursively to each item in the *to-do* list until it is empty. The main disadvantage of *NN* is that the spaces in the *to-do* list overlap when the dimensionality is larger than 2; thus large part of the data points would be accessed multiple times. The *BBS* (*branch-and-bound skyline*) algorithm [35] also uses R-tree and progressively produces the nearest neighbors of the origin by best-first search as described in [16]. It uses another in-memory R-tree S to hold the skylines, the first nearest neighbor must

be a skyline and it is inserted to S. In the search process, only the nodes that are not dominated by points in S are expanded for further search. The efficiency of *BBS* depends on the efficiency of the R-tree to find the nearest neighbor. It is well known that R-tree is not efficient in high-dimensional space (e.g., dimensionality > 7); it requires almost exhaustive search for nearest neighbor search. Thus, *BBS* is only suitable for low-dimensional space.

Recently, Lee *et al.* [24] proposed a method that maps the multi-dimensional data points into the positions, called the *Z-address*, on the one-dimension space filling curves (i.e., *Z-curve*) and introduced a new data structure, called *ZB*-tree, to index the points. A *ZB*-tree is a B-tree using the *Z-address* as key and equipped with bounding regions (called *RZ*-region) for region-based dominance test as done by *BBS*. The main disadvantage of *ZB*-tree is that it does not support sub-space skyline queries because the order of the Z-address does not hold in sub-spaces. In addition, to maintain the clustering property of each node, the space utilization is low.

Recently, researchers start to shift their attention from full-space skyline to sub-space skyline computation, e.g., [36, 37, 41, 45, 47]. The main motivation is the fact that in high-dimensional space, full-space skyline queries may return too many data points for users to make decision. The sub-space skyline methods could be roughly classified into two categories: (1) pre-materializing the sub-space skylines, e.g., [36, 37, 45, 47]; and (2) use index, e.g., [41]. The first category pre-computes the skylines for all the sub-spaces and organizes them into data cubes, called skycube or skyline cubes. Thus, given any sub-space skyline query, the data cubes can return the result immediately. Generally speaking, pre-materialization methods are storage-inefficient because a d-dimensional space may have $2^d - 1$ different non-empty sub-spaces. The works in this category mainly focus on how to efficiently compute the skyline cubes, e.g., [36, 37, 47] or how to efficiently organize the skycube to save storage space [45] (strictly speaking, [45] is not fully pre-materializing the skylines; each point p is only stored in cuboids of its minimum subspaces in which p is a skyline). The works described in [36, 37, 47] did not provide the operations to update the skyline cubes when the database is updated; thus they are not suitable for dynamic databases.

The second category does not pre-compute the skylines. Tao *et al.* [41] proposed a method, called *SUBSKY*, to compute skylines in subspace. Assume the data domain on all dimensions is [0,1]. It defines, for each data point p, a function $f(p) = MAX_{i=1..d}(1 - p[i])$, where $p[i]$ denotes the ith coordinate of point p; $f(p)$ is the L_∞-distance from p to the maximal corner of the data space. A B-tree is employed to index the data points with $f(p)$ being the key. Thus, the points are sorted by f-value in the leaf nodes. The proposed algorithm is based on the following property: for any point p, p is not a skyline (in the queried sub-space) if $f(p) < MIN_{i \in SUB}(1 - p_{sky}[i])$, where SUB specifies the queried sub-space and p_{sky} is a skyline point is the sub-space. The main disadvantages of SUBSKY are that it does not work for un-normalized data set as it's performance largely depends on the relationship between different dimensions; in addition, it is not progressive.

Many other researchers designed skyline algorithms for data sets with some specific properties. For example, Chan *et al.* [7] proposed skyline algorithms for data set with partially-ordered attributes; Morse *et al.* [29] presented skyline algorithms for data set with attributes from low-cardinality domains; Wong *et al.* [43] introduced algorithms for skyline querying with nominal attributes, which do not have a pre-defined full order on them. In a skyline query involving nominal attributes, the user would specify his/her preferences by giving partial order to some values of the nominal attributes. That is, the partial order on the nominal attributes is not fixed. Kalefa [19] designed skyline algorithms for incomplete data sets, i.e., data sets in which some values (of certain points) are missing or unknown. In this case, the dominance relation is not transitive. Transitivity of dominance is the basis of many previous skyline algorithms for the complete data set.

Some researchers studied the skyline queries under modified definition. For instance, Chan *et al.* [8] proposed the concept of *k-dominant skyline* and gave algorithms to find k-dominant skylines. A point p is said to *k-dominant* another point q if p is not worse than q in at least k dimensions and p is better than q in at least one dimension. A *k-dominant skyline* is a point that is not k-dominated by any other points. Lin *et al.* [18] designed algorithms for finding k *most representative skylines*, which are the k skylines that maximize the number of points being dominated.

Other works designed skyline algorithms under various specific environments, such as skylines over data streams [40], skylines for moving objects [17], parallel skyline computation [42], etc. Another work that worth mentioning is [9], which proposed methods to estimate the cardinality of the result of the skyline operator for query optimization.

3 Skyline computation with multi-objective based clustering

3.1 Overview

The overall proposed process for computing and maintaining skyline consists of stages. First, we do indexing with multi-objective genetic algorithm for clustering. We have specified objectives for this purpose and we propose an algorithm that performs the task. Here, we try to suggest a parameterless and scalable algorithm; thereafter indexed clusters are used for skyline computation.

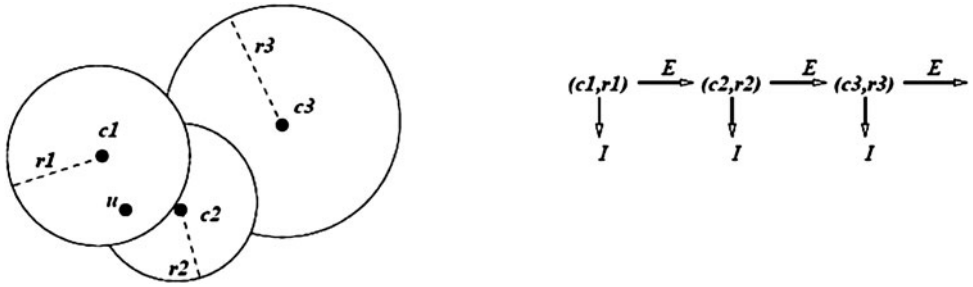

Fig. 1 Tail recursive and iterative space decomposition algorithm [10]

3.2 Indexing with multi-objective genetic algorithm

3.2.1 Objectives

Our purpose is to extract clusters in hierarchical manner. This employs the idea of clustering data with top down approach. A dataset D can be clustered in memory if it is of manageable size. Otherwise, D is split into p partitions as $D_1, D_2, \ldots, D_p$. The whole data (in case could fit in memory) or each portion of the partitioned data is clustered by using multi-objective clustering algorithm.

As illustrated in Fig. 1, the list of clusters approach developed by Chavez and Navarro [10] is a compact space partitioning for effective metric indexing. Basically, it does clustering iteratively and by using some heuristics. At each step, it obtains one cluster (I-nterior) and then it obtains the next from the remaining dataset (E-xterior) and so on. Apparently, data spread is expected to get larger as the number of iterations increases. Insert and delete operations can be done by inserting and deleting the instances but their main concern is the optimality of the radius value. Insertion is done by simply inserting the instance to the cluster having minimum distance to the centroid. In other words, they do clustering with constant radius r_c and m elements in order to produce buckets. In general, the clustering process is repeated recursively and a tree structure is constructed where each bucket has a triplet $\langle I_i, c_i, r_i \rangle$, where I_i is the internal bucket that has elements; but our idea [10] does not necessarily consider the precedence for an instance to become a member. Once an instance has been clustered (less than radius constraint), it cannot become member of any subsequently generated cluster even though it fits better within the latter cluster (its distance from the center of the latter cluster is less than the radius specified for the latter cluster). Conventional nearest neighbor to cluster centroid approach is enough for making an instance a member of a cluster in our work. Our study is different from the work described in [10] in the following aspects:

1. There is no strict radius and\or number of objects value related to a cluster; that is, our algorithm can have different radius and number of objects value for each data partition i.
2. Each partition can have different number of clusters.
3. Index regions are determined automatically.
4. It does not extract clusters iteratively, it considers the whole structure at once and we split also the data into partitions.

Our work designates the radius and number of objects per cluster automatically by using a number of cluster validation techniques. Basically, by employing more cluster validation metrics, we decide on the ideal number of clusters with majority voting. For each partition i, the number of clusters can vary accordingly. In [10], the clustering process starts with a center selection and several heuristics are proposed for radius specification. On the other hand, we just give an upper bound for m and radius r_c; further, both m and r_c can be less than the specified criteria for optimum number of clusters.

While doing multi-objective clustering, we have specified the main objectives o_1 and o_2 as minimize intra-cluster distance and maximize the inter-cluster distances, respectively:

$$\text{minimize} \quad o_1 = \sum_{i=1}^{k} \sum_{\forall p_j \in D}^{|D|} \sum_{f=1}^{F} \left| c_{if} - p_{jf} \right| \times I(p_j \in C_i) \tag{1}$$

$$\text{maximize} \quad o_2 = \sum_{i=1}^{k} \sum_{j=1}^{i-1} \text{distance}(c_i, c_j) \tag{2}$$

where dataset D is clustered in k partitions as $C_1, C_2, \ldots, C_k$ which have representative points (centroids) as $c_1, c_2, \ldots, c_k$, respectively. Instances have F features; every instance p_j is an element of the given data set D and p_{jf} is the value of f-th attribute for p_j; and $I(p_j \in C_i)$ is a function that returns 0 or 1 to determine how p_j belongs to C_i.

We decided to incorporate in the process two more objectives (namely, o_3 and o_4, given next) in order to minimize incomparable regions between clusters and minimize the radius, respectively. Objective o_3 can be further explained. Basically, o_3 is explained with three premises: (1) A cluster is represented by its centroid and radius value (Fig. 1). This helps decide whether all instances of a cluster dominate those of another cluster. (2) A cluster can dominate another cluster relatively. (3) By employing Algorithm 2,

the skyline computation of clusters would help eliminate instances of another cluster directly. This can be checked with radius and centroid characteristics of clusters. To sum up, for a cluster to dominate another cluster, all instances of clusters should be dominating instances of that cluster. Objective o_3 encourages the size of dominating cluster to be smaller than the dominated cluster. We expect dominating cluster's size to be larger than the dominated cluster's size and as the distance between clusters decreases, this becomes more valuable and critical.

$$\text{maximize} \quad o_3 = \sum_{i=1}^{k} \sum_{j=1}^{k} \Xi(C_i, C_j) \times I(C_i \neq C_j)$$

where $I(c_i \neq c_j)$ is the function that returns 0 or 1 to indicate when C_i and C_j are the same; $\Xi(.)$ is the function that measures the pairwise dominance effect between clusters C_i and C_j. A cluster that dominates another and is smaller in size has more effect. Dominance has been normalized by distance; it is inversely proportional to the distance. It has more effect as distance gets smaller. $\Psi(C_i \prec C_j)$ is the function that returns 0 or 1 when the region of cluster C_i dominates the region of C_j.

$$\Xi(C_i, C_j) = \frac{1}{d(c_i, c_j)} \times \frac{size(C_j)}{size(C_i)} \times \Psi(C_i, C_j)$$

This conforms to the graph-dominance for directed graphs, and establishing the matrix M that has pairwise node relationship as 1 when cluster C_i dominates C_j and 0 otherwise. Poser of dominance is calculated as $M^2 + M$ [4]. We added the strength as the ratio of dominated clusters over dominating clusters, and the distance as the coefficient. We added the distance as clusters are close; this will contribute more since they likely co-occur at adjacent layers. If more than one cluster dominate the same cluster c, o_3's score will mostly increase; and in case one cluster (from the dominating clusters) disappears after deletion, the other cluster(s) will still dominate cluster c. This will reduce the overhead of deletion. The diameter of a group of clusters is determined in terms of the maximum radius of the clusters in the group.

$$\text{minimize} \quad o_4 = \max_{i=1..k} (\text{diameter}(C_i))$$

3.2.2 *The overall process: multi-objective genetic algorithm for clustering from k_{min} to k_{max}*

We assume we partition the dataset into manageable sizes to be able to do in memory clustering of individual partitions. For each partition $D[i]$, we apply multi-objective genetic algorithm for clustering. For each partition, we end up with different number of clusters with different radius values in a while loop where the number of clusters (k) takes its values starting from 2 to K (rule of thumb, $K = \sqrt{|D[i]|}$).

Our entire multi-objective clustering algorithm applies clustering followed by validation steps. For the validation step, relative indices have been used to detect the natural clustering. For this purpose, we have used a number of indices, namely Silhouette, dunn, Davies Bouldin, calinski, hartigan, ratkowsky, scott, friedman, ball, mariott, traceCovW, traceW, rubin, and db.

Bio-inspired genetic algorithm model has the population as a solution set, and at the end it produces an optimal solution set. The final population is used for the assessment. Each individual's similarity to other population members is calculated and the most similar individual to other population members in the solution set is accepted as the final solution for the specified number of clusters. Similarity is calculated with the F&M external validity index as follows.

$$\text{Similarity}(i,\ P) = \frac{1}{|P| - 1} \sum_{\substack{j \in P \\ \wedge i \neq j}} F\&M(i, j)$$

Simply the F&M index measures the pairwise co-occurrence of instances in the same cluster. At the end, we obtain a score for pairwise co-occurrence of instances.

The above mentioned relative cluster validity indices are used as metrics for deciding on the best natural clustering. The metrics will assess a value for $k = k_{min}$ to k_{max} and majority voting is employed thereafter.

Our entire multi-objective clustering process is outlined in Algorithm 1.

Algorithm 1 Multi objective clustering algorithm

```
Input: dataset D = D[1] ∪ D[2] ∪ ··· ∪ D[d]
  //Dataset D with d partitions
Output: partition P = P[1], P[2], ..., P[d]
  // Each dataset has partition solutions.
P = φ
for (i = 1; i ≤ d; i++) do
    scores = φ
    for (k = k_min; k ≤ k_max; k++) do
        S = multi_objective_clustering(D[i], k),
        /*perform multi-objective clustering and find the
        most similar individual with F&M index*/
        scores[k] = Best_Score(S);
        //relative index scores are assigned.
    end for
end for
P[i]=arg best (P, scores)      // for each metric
```

Regardless of what objective is more prominent, we use all the objectives discussed above in the clustering process and we have more than one solution that can be screened.

Also, it would be better to mention that we do have clustering from k_{min} to k_{max} and these individuals reported when the optimal number of clusters was k are tested using what-if type of analysis to decide on making one of the instances an additional new cluster centroid, i.e., for incrementing the number of clusters by 1. Here, the error is the difference between the number of clusters proposed and the actual number of clusters in the outcome [25]. The assumption given in [25] is that if k (the upper bound) is detected as the best number of clusters then on the basis of k clusters $(k+1)$ clusters is checked as the possible best number of clusters. This process of expanding the upper bound up continues until the preferred number of clusters falls inside the interval (it is not the upper bound).

3.3 Multi-objective clustering for k

The same multi-objective genetic algorithm based clustering approach described in Sect. 3.2 for identifying the most appropriate number of clusters is employed in this section to determine the optimal clustering for the identified number of clusters k; we apply almost the same operators/stages with slight differences.

3.3.1 Encoding of chromosomes

Individual coding in the population is a chromosome of length N. Every gene is represented by an allele where allele I is the corresponding cluster of instance i. In other words, each allele in the chromosome takes a value from the set $\{1,2,\ldots,k\}$, where k is the maximum number of clusters for which we try to get an optimal partitioning for every optimal number of clusters value tried in ascending order.

3.3.2 The clustering process

Initially, current generation is assigned to zero. Each chromosome takes number of clusters parameter within the range 1 to the current number of clusters k. For the lowest number of clusters, a population with the specified number of chromosomes is created with an equal probability. To do this, we use the ordered initialization first. In round order, each allele takes values 1 to k in order, and then those allele value assignments are shuffled within the chromosome randomly by processing the random pairwise swap operation inside the chromosome. This way, we can avoid generating illegal strings, which means some clusters do not have any pattern in the string. The next generation is the selection using pareto domination tournament. In this step, two candidate items picked among (*population size-*t_{dom}) individuals participate in the pareto domination tournament against the t_{dom} individuals for the survival of each in the population. In the selection part, t_{dom} individuals are randomly picked from the population. With two randomly selected chromosome candidates in (*population size-*t_{dom}) individuals, each of the candidates is compared against each individual in the comparison set, t_{dom}. If one candidate has larger total within cluster variation fitness value, larger number of clusters, radius and less domination score than all of the chromosomes in the comparison set, this means it is dominated by the comparison set already and will be deleted from the population permanently. Otherwise, it resides in the population. After the pareto domination tournament, two-point crossover operator is applied on randomly chosen two chromosomes. Crossover is carried out with probability p_c. The mutation operator replaces each gene value a_n by a'_n with respect to the probability distribution; for $n=1$ to N; a'_n is a cluster number randomly selected from $\{1,2,\ldots,k\}$ with probability distribution $\{p_1, p_2, \ldots, p_k\}$ defined as:

$$p_{ij} = \frac{e^{-d(X_n,C_j)}}{\sum_{r=1}^{k} e^{-d(X_n,C_r)}}$$

where $i \in [1..k]$ and $d(X_n, C_k)$ denotes the Euclidean distance between pattern X_n and the centroid C_k of the k-th cluster; p_i represents the probability interval of mutating gene assigned to cluster i (e.g., RouletteWheel). Eventually the k-means operator [26] is applied to reorganize each object's assigned cluster number.

After all the operators are applied, we have twice the number of individuals after having the pareto dominated tournament. We cannot give an exact number as equal to the number of initial population size, l, because at each generation candidates are randomly picked for the survival test leading to deletion of one, in case dominated. To halve the number of individuals, the ranking mechanism proposed in [13] is employed. So, the individuals obtained after applying crossover, mutation and k-means operator are ranked, and we pick the best individuals among the m to place in the population for the next generation. Our approach picks the first l individuals by considering the elitism and diversity among $2l$ individuals; pareto fronts are ranked. Basically, after recombination, parent and child chromosomes ($2l$ chromosomes) are ranked with respect to pareto optimal front and we pick and place the first l chromosomes after ranking into the population to be run in the next generation. Placement of l individuals occurs in the following way, until l individuals capacity is filled, pareto optimal fronts are picked: We pick the first pareto-optimal front, and we put it in the population and so on until we have l individuals. The last pareto-optimal front may have more individuals than required to complete the number of individuals to l; we handle the diversity automatically. We rank the m individuals and reduce the objective dimensions into one. Then, we sum the normalized values of the objective functions of each individual. We sort the m individuals in increasing order and find each individual's total difference from its individual pairs:

the one with the closest smaller summed values and the one with the closest greater summed values. After sorting the individuals, in terms of each one's total difference in decreasing order, we keep placing from the top as many individuals as we need to complete the population to l. Finally, if the maximum number of generations is reached, or the pre-specified threshold is satisfied, exit; otherwise the next generation is produced.

3.4 Pareto front algorithm

The pareto front algorithm finds all layers in a database and the outer layer is the pareto front which is the skyline of the database. Given a *resultset* as the result of a database query on database and skylines, all layers sorted in dominating order are computed in $O(C * N^3)$ where C is the number of attributes specified as criteria and N is the number of instances in our database.

When the skyline is computed only, the outmost while loop will iterate only once, so the cost becomes $O(C * N^2)$.

3.5 Skyline computation

The proposed skyline computation depends on the extension of space decomposition for indexing in metric space [10]. After multi-objective clustering with genetic algorithm, we obtain a list of clusters. Assume we have a dataset $D = D[1] \cup D[2] \cup \cdots \cup D[d]$ and each $D[i]$ is clustered with the corresponding partitioning $P[i]$ specified as:

$$P[i] = \langle I_{i1}, c_{i1}, r_{i1} \rangle, \langle I_{i2}, c_{i2}, r_{i2} \rangle \ldots \langle I_{ik_i}, c_{ik_i}, r_{ik_i} \rangle$$

Each partitioning will have different number of clusters k_i, and all radius values will have upper bound value as r_i because of the 4th objective (o_4): $r_{ij} \leq r_i$ where $1 \leq j \leq k_i$.

Initially, we assume that we have well separated and compact (homogeneous) clusters with radius as small as possible; also the clusters dominating other clusters are as many as possible. Next, our algorithm undergoes three stages; we first skip dominated regions that are represented as clusters. Algorithm 2 finds all layers as pareto fronts and as we indicated, the pareto front algorithm does all computations as layers of silhouettes (the first layer named as skyline). *ParetoFront_FirstLayer(.) just* calculates the skyline function, so Algorithm 2 will be invoked only once, and we gather dominating regions at each partition. At the second stage, clusters which are dominated as a region are eliminated in turn; clusters that are in the same partition are not compared. The *remove* function removes clusters from dominating set and the maximum radius value (r_k) is updated. At the final stage, we just repeat *ParetoFront_FirstLayer(.)* for each partition and collect the outcome for final skyline computation. Here, we assume that the input to step 3 is of manageable size to compute

Algorithm 2 Pareto front algorithm

```
1: Input: resultset (1..count)
2: Output: int L[1..count]        //layer number for each individual is held
   int R[1..count]      // number of objects in layer i
   int O[1..count]      //order of objects in overall
   int maxlayer      //maximum layer
3: currentitem = 0;
4: currentlayer = 0;
5: maxlayer = 0;
6: for (i = 1; i ≤ count; i++) do
7:    resultset[i].status = unprocessed;
8: end for
9: while (∃i : i = 1.count ∧ resultset[i].status = unprocessed) do
10:   currentlayer = currentlayer + 1;
11:   for (i = 1; i ≤ count; i++) do
12:      if (resultset[i].status==unprocessed) then
13:         continue;
14:      end if
15:      for (j = 1; j ≤ count; j++) do
16:         if ((i == j) or (resultset[j].status == processed)) then
17:            continue;
18:         end if{ }
           /*resultset[i] is compared against resultset[j] and the value is returned as dominated, dominating, or incomparable*/
19:         comparison = Compare(resultset[i], resultset[j]);
20:         if (comparison==dominated) then
21:            resultset[i].state = dominated;
22:            break;
23:         else if (comparison==dominating) then
24:            resultset[j].state=dominated;
25:         else if (comparison==incomparable) then
26:            if (resultset[i].state ≠ dominated) then
27:               resultset[j].state = incomparable;
28:            end if
29:         end if
30:      end for
31:      if (resultset[i].state ≠ dominated) then
32:         resultset[i].status = processed;
33:         L[i] = currentlayer;
34:         O[i] = currentitem;
35:         currentitem++;
36:      end if
37:   end for
38: end while
39: maxlayer = currentlayer;
```

Algorithm 3 Skyline Computation

```
1: Input: D = D[1] ∪ D[2] ∪ ··· ∪ D[d]
   P = P[1] ∪ P[2] ∪ ··· ∪ P[d] and
   P[i] = ⟨I_i1, c_i1, r_i1⟩, ⟨I_i2, c_i2, r_i2⟩ ... ⟨I_ik, c_ik, r_ik⟩
2: Output: skyline[1..count]      //silhouette of dataset D
3: Step 1
4: find skyline for each P[i] in terms of centroid
5: for (i = 1; i ≤ d; i++) do
6:    S[i] = ParetoFront_FirstLayer(P[i]);
7: end for;
8: Step2:
9: for (i = 1; i ≤ d; i++) do
10:   for (j = 1; j ≤ size(P[j]); j++) do
11:      c_ij_dominated = false;
12:      for (k = 1; k ≤ d; k++) do
13:         if (c_ij_dominated == true) then
14:            i = i − 1;
15:            break;
16:         else if (i == k) then
17:            continue;      //already proven to be incompa-
               rable
18:         end if
19:         for (l = 1; l ≤ size(P[l]); l++) do
20:            if Ψ(c_ij ≺ c_kl) then
21:               //cluster c_1's region dom. cluster c_2
22:               remove(< I_kl_i, c_kl_i, r_kl_i >, S[l]);
23:               l = l − 1;
24:            else if Ψ(c_kl ≺ c_ij)      //cluster c_2's region-
               dom.cluster c_1 then
25:               remove(< I_ij_i, c_ij_i, r_ij_i >, S[i]);
26:               c_ij_dominated = true;
27:               break;      //c_ij is dominated and subseq is
                  tested
28:            end if
29:         end for
30:      end for
31:   end for
32: end for
33: Step 3:
34: for (i = 1; i ≤ d; i++) do
35:   S[i] = ParetoFront_FirstLayer(S[i]);
36: end for
37: skyline = ParetoFront_FirstLayer(S[1] ∪ S[2] ∪ ··· ∪
   S[d]);
```

the skyline. In case not, we repeat step 1 and step 2 on new clusters having only the first layer until it is of manageable size. When we only get the first layer, if the cluster's radius is determined by the instances in the first layer of clusters then the radius will get smaller to cause more dominance and to repetitively eliminate more clusters and instances as well.

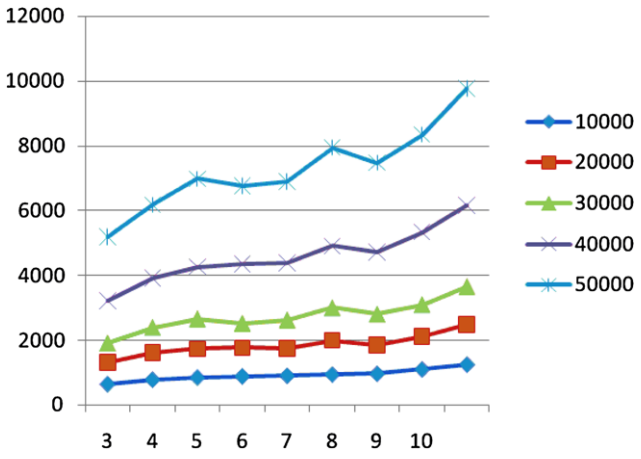

Fig. 2 Multi-objective clustering processing time (seconds) for independent data (# of features = 2..10)

Insertion will cause no problem. Among partitions a new instance will be classified into the closest cluster based on radius constraint, otherwise it will be a new cluster with one instance. If the cluster is in the first layer in the partition then it will be checked against the completed skyline computation. We merge clusters that grow into one cluster. When deletion occurs, if deletion is not at the first layer then there will be no problem. Otherwise, instances dominated at the immediate layer are tested against the current skyline.

The entire process can be summarized as below:

Repeat

1. Do MOGA for clustering over dataset for number of clusters value $k = 2$ to k_{max} and assess the number of clusters to come up with the optimal clustering.
2. Do ParetoFront_FirstLayer over clustering results for elimination and collect all results and obtain the resulting set as the new dataset

Until the obtained pareto front is at manageable size.

4 Experiments

To demonstrate the effectiveness and applicability of the proposed approach, we generated independent and anti-correlated data sets by using the same way that has been used to generate the datasets in [5]. Each instance has been transformed into 100 bytes after one character type garbage variable insertion. We have generated datasets with size from 10,000 to 50,000, incrementing by 10,000 instances at a time (1 MB to 5 MB); and we set the partitions as $P = 100$ K each with varying dimension size ($d = 2..10$).

We employed multi-objective clustering by genetic algorithm from 1 MB to 5 MB for both independent and anti-correlated data with population size 100; crossover rate 0.9;

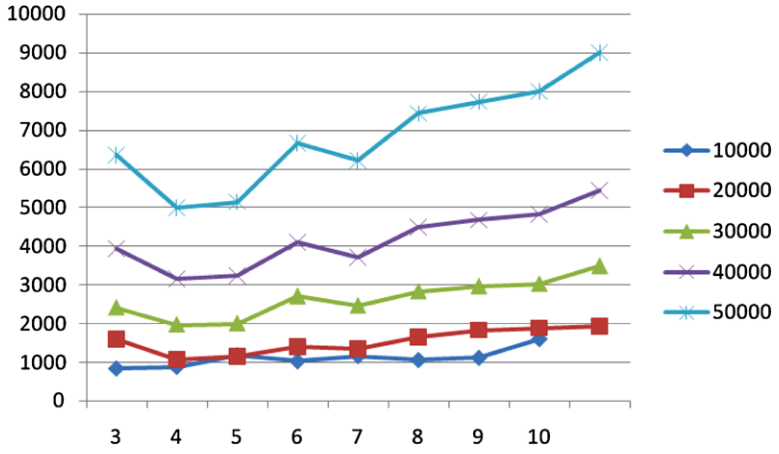

Fig. 3 Multi-objective clustering processing time (seconds) for anti-correlated data (# of features = 2..10)

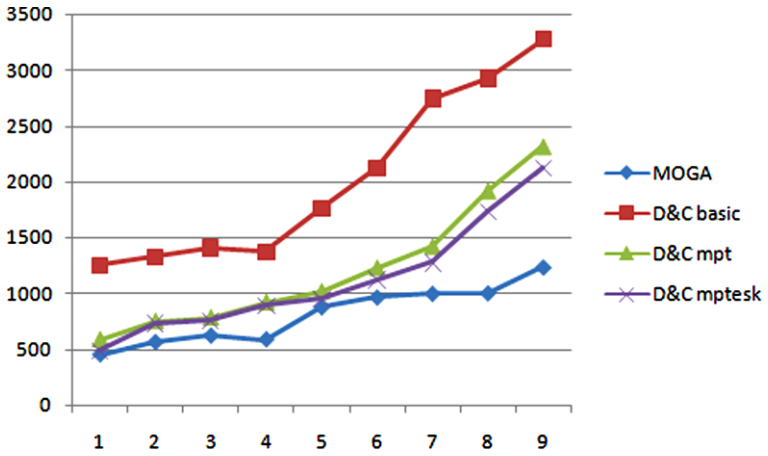

Fig. 4 Comparative results of multi-objective clustering processing time and divide and conquer (seconds) for 1 MB independent data (# of features = 2..10)

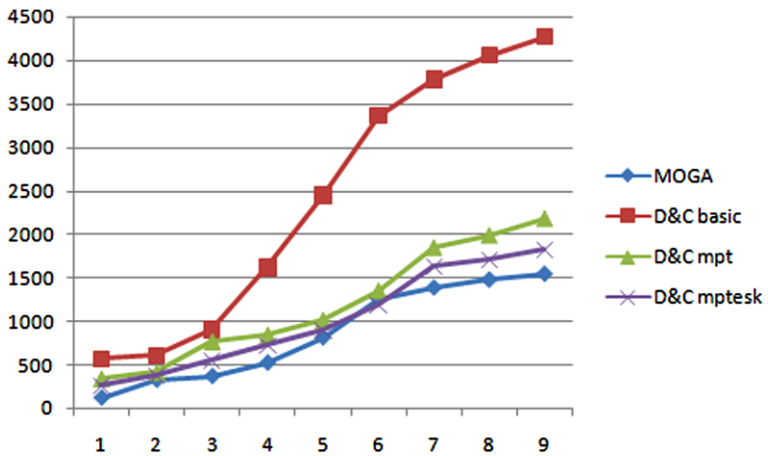

Fig. 5 Comparative results of multi-objective clustering processing time and divide and conquer (seconds) for 1 MB anti-correlated data (# of features = 2..10)

and mutation rate 0.05. We performed different experiments comparing our approach and divide and conquer [5] techniques: Basic divide and conquer($D\&C$ basic), Divide and Conquer with multi-way partitioning and Divide and Conquer with multi-way partitioning and early skyline. The term early skyline has been coined in order to pre-filter dominated ones. Figures 2 and 3 gives the results of total elapsed time for multi-objective clustering by using genetic algorithm. Mainly, we start from $k = 2$ to $\sqrt{P}$. It is quite obvious that as the data size increases, the total processing time increases linearly and this is also valid as dimension size increases.

Figures 4 and 5 summarize the total time elapsed for skyline computation. They give the comparative results of skyline computation for multi objective genetic algorithm against divide and conquer based skyline computation. For MOGA, Up to the 6th dimension, anti-correlated

Table 1 Number of skylines for 1 MB data (# of features = 2..10)

	indep.	anti-corr
2	348	653
3	983	5037
4	3383	43251
5	11034	128303
6	17176	267148
7	57325	413127
8	97043	598634
9	187915	670153
10	259912	780715

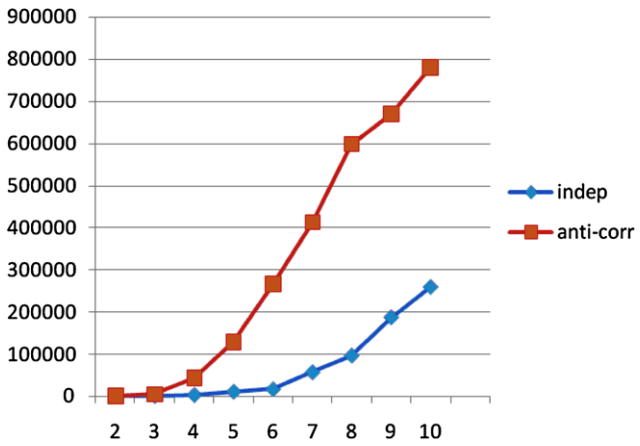

Fig. 6 Number of skylines for 1 MB data (# of features = 2..10)

data demonstrates better performance as compared to independent data; however anti-correlated data performs worse as dimension size increases. But the increase in computation time follows almost linear trend. As dimensionality increases, skyline computation takes longer time. Our method outperforms divide and conquer based techniques as evident from the curves plotted in the two figures.

Table 1 and Fig. 6 summarize the number of skyline points for both independent and anti-correlated data. It can be easily seen that independent data has less number of skyline points compared to anti-correlated data. The skyline computation increase is also dependent on the conflicting attributes and as d increases, it is ordinary to observe the increase in skyline computation time, especially for the anti-correlated data.

We also performed experiments on real data set. We used the statistics on nba.players from databaseBasketball.com; football.seasons and football.games tables have been used from pro-football-reference.com.

nba.players have 2 numeric attributes (height and weight) and 3572 instances with 4 skyline instance; football.seasons have numeric10 attributes and 22244 instances with 249 skyline instances and football.games have 7 numeric attributes and 62887 instances with 306 skyline instances.

Table 2 Number of skylines for nba.players, football.games and football.seasons

	MOGA	D&C basic	D&C mpt	D&C mptesk
nba.players	26	49	26	26
footbal.games	43	118	44	43
football.seasons	55	132	67	63

Based on the performed tests on the above enumerated real data sets, our results reported in Table 2 indicate that as number of features and number of instances increase, our method gives significantly better results in skyline computation.

5 Conclusions

We applied data mining techniques for skyline computation. The basic contribution reported could be articulated as follows: our previously introduced multi-objective genetic algorithm based clustering approach has been adapted and applied for skyline computation. The proposed approach for identifying skylines utilizes indexing with automatic and dynamic radius detection. We have done all indexing at once on several partitions. This work uses some objectives as well separated and compact clusters with minimal radius and more pareto dominance relationship. To the best of our knowledge, this is the first study of using multi-objective genetic algorithm for clustering (hence machine learning and data mining techniques) in order to obtain skyline instances. Our current efforts concentrate on a more comprehensive approach for dealing with insertions and deletions to turn the process into fully dynamically.

References

1. Aksoy Y, Butler TW, Minor ED (1996) Comparitive studies in interactive multiple objective mathematical programming. Eur J Oper Res 89:408–422
2. Alhajj R, Kaya M (2008) Multi-objective genetic algorithms based automated clustering for fuzzy association rules mining. J Intell Inf Syst 31(3):243–264
3. Armentano VA, Claudio JE (2004) An application of a multi-objective tabu search algorithm to a bicriteria flowshop problem. J Heuristics 10(5):463–481
4. Berge C (1962) The theory of graphs and its applications. Methuen, London
5. Borzsonyi S, Kossmann D, Stocker K (2001) The skyline operator. In: Proceedings of IEEE international conference on data engineering, Heidelberg, Germany, pp 421–430
6. Buchanan J, Daellenbach HG (1987) A camparative evaluation of interactive solution methods for multiple objective decision models. Eur J Oper Res 24:353–359

7. Chan C-Y, Eng P-K, Tan K-L (2005) Stratified computation of skylines with partially-ordered domains. In: Proceedings of ACM-SIGMOD international conference on management of data, 2005
8. Chan C-Y, Jagadish HV, Tan K-L, Tung AKH, Zhang Z (2006) Finding k-dominant skylines in high dimensional space. In: Proceedings of ACM-SIGMOD international conference on management of data, 2006
9. Chaudhuri S, Dalvi N, Kaushik R (2006) Robust cardinality and cost estimation for skyline operator. In: Proceedings of IEEE international conference on data engineering, 2006
10. Chavez E, Navarro G (2005) A compact space decomposition for effective metric indexing. Pattern Recogn Lett 26(9):1363–1376
11. Chomicki J, Godfrey P, Gryz J, Liang D (2003) Skyline with presorting. In: Proceedings of IEEE international conference on data engineering, 2003
12. Coello CA, Lamont GB (2004) Recent developments and future research directions. In: Applications of multi-objective evolutionary algorithms. World Scientific, Singapore
13. Deb K, Agrawal S, Pratab A, Meyarivan T (2000) A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-II. In: Proceedings of the parallel problem solving from nature, number 1917, Paris, France
14. Elegbede C, Adjallah K (2003) Availability allocation to repairable systems with genetic algorithms: a multi-objective formulation. Reliab Eng Syst Saf 82(3):319–330
15. Fonseca CM, Fleming PJ (1998) Multiobjective optimization and multiple constraint handling with evolutionary algorithms—Part I: a unified formulation. IEEE Trans Syst Man Cybern, Part A 28:26–37
16. Hjaltason G, Samet H (1999) Distance browsing in spatial databases. ACM Trans Database Syst 24(2):265–318
17. Huang Z, Lu H, Ooi BC, Tung AK (2006) Continuous skyline queries for moving objects. IEEE Trans Knowl Data Eng 18(12):1645–1658
18. Lin X, Yuan Y, Zhang Q, Zhang Y (2007) Selecting stars: The k most representative skyline operator. In: Proceedings of IEEE international conference on data engineering, 2007
19. Kalefa ME, Mokbel MF, Levandoski JJ (2008) Skyline query processing for incomplete data. In: Proceedings of IEEE international conference on data engineering, 2008
20. Kossmann D, Ramsak F, Rost S (2002) Shooting stars in the sky: An online algorithm for skyline queries. In: Proceedings of the international conference on very large databases, 2002
21. Kumar R, Banerjee N (2003) Multicriteria network design using evolutionary algorithm. In: Proceedings of genetic and evolutionary computations conference (GECCO-03). LNCS, vol 2023. Springer, Berlin, pp 2179–2190
22. Kumar R, Parida PP, Gupta M (2002) Topological design of communication networks using multiobjective genetic optimization. In: Proceedings of the congress Evolutionary Computation (CEC-2002). IEEE Press, New York, pp 425–430
23. Kung HT, Luccio F, Preparata FP (1975) On finding the maxima of a set of vectors. J ACM 22(4):469–476
24. Lee KCK, Zheng B, Li H, Lee W-C (2007) Approaching the skyline in Z order. In: Proceedings of the international conference on very large databases, 2007
25. Likas A, Vlassis N, Verbeek J (2003) The global k-means clustering algorithm. Pattern Recogn 36(2):451–461
26. Lu Y, Lu S, Fotouhi F, Deng Y, Brown S (2004) FGKA: A fast genetic k-means clustering algorithm. In: Proceedings of ACM symposium on applied computing, pp. 162–163, 2004
27. Mamede M (2005) Recursive lists of clusters: A dynamic data structure for range queries in metric spaces. Proceedings of the 20th international symposium on computer and information sciences, pp. 843–853
28. Matousek J (1991) Computing dominances in En. Inf Process Lett 38(5):277–278
29. Morse M, Patel JM, Jagadish HV (2007) Efficient skyline computation over low-cardinality domains. In: Proceedings of the international conference on very large databases, 2007
30. Murata T, Ishibuchi H, Tanaka H (1996) Multi-objective genetic algorithm and its applications to flowshop scheduling. Comput Ind Eng 30(4):957–968
31. Özyer T, Alhajj R (2006) Clustering by integrating multi-objective optimization with weighted k-means and validity analysis. In: Proceedings of the international conference on intelligent data engineering and automated learning, 2006
32. Özyer T, Alhajj R (2006) Achieving natural clustering by validating results of iterative evolutionary clustering approach. In: Proceedings of the IEEE conference on intelligent systems, 2006
33. Özyer T, Alhajj R (2008) Deciding on number of clusters by multi-objective optimization and validity analysis. J Multi-Valued Log Soft Comput 14(3–5):457–474
34. Özyer T, Alhajj R (2009) Parallel clustering of high dimensional data by integrating multi-objective genetic algorithm with divide and conquer. J Appl Intell 31(3):318–331
35. Papadias D, Tao Y, Fu G, Seeger B (2005) Progressive skyline computation in database systems. ACM Trans Database Syst 30(1):41–82
36. Pei J, Jin W, Ester M, Tao Y (2005) Catching the best views of skyline: A semantic approach based on decisive subspaces. In: Proceedings of the international conference on very large databases, 2005
37. Pei J, Fu AW-C, Lin X, Wang H (2007) Computing compressed multidimensional skyline cubes efficiently. In: Proceedings of IEEE international conference on data engineering, 2007
38. Tamura K, Miura S (1979) Necessary and sufficient conditions for local and global non dominated solutions in decision problems with multi-objectives. J Optim Theory Appl 28(4):501–523
39. Tan KL, Eng PK, Ooi BC (2001) Efficient progressive skyline computation. In: Proceedings of the international conference on very large databases, 2001
40. Tao Y, Papadias D (2006) Maintaining sliding window skyline on data streams. IEEE Trans Knowl Data Eng 18(2):377–391
41. Tao Y, Xiao X, Pei J (2006) SUBSKY: Efficient computation of skylines in subspaces. In: Proceedings of IEEE international conference on data engineering, 2006
42. Vlachou A, Doulkeridis C, Koidis Y (2008) Angle-based space partitioning for efficient parallel skyline computation. In: Proceedings of ACM-SIGMOD international conference on management of data, 2008
43. Wong RC-W, Fu AW-C, Pei J, Ho YS, Wong T, Liu Y (2008) Efficient skyline querying with variable user preference on nominal attributes. In: Proceedings of the international conference on very large databases, 2008
44. Wu P, Agrawal D, Egecioglu O, Abbadi AE (2007) DeltaSky: Optimal maintenance of skyline deletions without exclusive dominance region generation. In: Proceedings of IEEE international conference on data engineering, pp 486–495
45. Xia T, Zhang D (2006) Refreshing the sky: The compressed skycube with efficient support for frequent updates. In: Proceedings of ACM-SIGMOD international conference on management of data, 2006
46. Yang J-E, Hwang M-J, Sung T-Y, Jin Y (1999) Application of genetic algorithm for reliability allocation in nuclear power plants. Reliab Eng Syst Saf 65(3):229–238
47. Yuan Y, Lin X, Liu Q, Wang W, Yu JX, Zhang Q (2005) Efficient computation of the skyline cube. In: Proceedings of the international conference on very large databases, 2005
48. Zitzler E (1999) Evolutionary algorithms for multiobjective optimization: Methods and applications. PhD thesis, Zurich: Swiss Federal Institute of Technology (ETH), Aachen, Germany

Tansel Özyer obtained his bachelor's degree and M.Sc. degree from the Department of Computer Engineering, Bilkent, Ankara (1996) and Middle East Technical University, Ankara (2000). After spending several years in the industry in North America, he joined the PhD program in the Department of Computer Science at the University of Calgary, Canada in 2002 and received his PhD degree in 2006. He published over 35 refereed conference papers and journal articles at prestigious venues. He served on the program committee of several international conferences including IEEE IRI, ASONAM, CaSoN, ADMA. He is also on the editorial board of some journals. He is currently the publicity chair of ASONAM 2010 and MS 2010. His research interests are data mining, bioinformatics, social networks and XML.

Ming Zhang received the Master's degree in computer science from University of Regina, Canada, in 2005. He recently defended his PhD dissertation in the Department of computer science at the University of Calgary. He published over 10 papers in reputable international conferences and journals. His research interest includes database, data mining, XML, indexing of high dimensional data and image processing. He is a recipient of NSERC Postgraduate Scholarship and iCore Postgraduate Scholarship.

Reda Alhajj received his B.Sc. degree in Computer Engineering in 1988 from Middle East Technical University, Ankara, Turkey. After he completed his BSc with distinction from METU, he was offered a full scholarship to join the graduate program in Computer Engineering and Information Sciences at Bilkent University in Ankara, where he received his M.Sc. and Ph.D. degrees in 1990 and 1993, respectively. Currently, he is Professor in the Department of Computer Science at the University of Calgary, Alberta, Canada. He is also affiliated with the Department of Computer Science at Global University, Beirut, Lebanon. He published over 300 papers in refereed international journals and conferences. He served on the program committee of several international conferences including IEEE ICDE, IEEE ICDM, IEEE IAT, SIAM DM; He is currently serving as the program chair of OSINT-WM 2010, SONAM 2010, IEEE IRI 2010. He is editor in chief of International Journal of Social Networks Analysis and Mining, International Journal of Data Mining and Bioinformatics, associate editor of IEEE SMC- Part C and he is member of the editorial board of the Journal of Information Assurance and Security; he has been guest editor for a number of special issues and edited a number of conference proceedings. Dr. Alhajj's primary work and research interests are in the areas of biocomputing and biodata analysis, data mining, multiagent systems, schema integration and reengineering, social networks and XML. He currently leads a research group of 10 Ph.D. and 6 M.Sc. candidates working on different aspects of his research program.