Applications of Clustering Problems

Cecilia M. Procopiuc

April 30, 1997

1 Introduction

Clustering algorithms have been employed in many fields of human knowledge that require finding a "natural association" among some specific data. The way the term "natural association" is defined depends on the field and the particular application that uses it, and can vary quite a lot. For example, in the life sciences, the data to be clustered consists of life forms and the clusters themselves are species, or subspecies of a given species. The need for using clustering alghorithms is determined by the scientists' wish to study various characteristics of the life forms. [5] mentions a situation where mammals are clustered by similarities in the proportion of three chemicals in their milk. The "natural association" of the animals depends on the percentages of these chemicals in their milk. A number of other domains use clustering algorithms to extract information by grouping a large collection of objects, such as the medical sciences (clustering of symptoms yields information about the diseases), the earth sciences (to determine land use patterns, for example), marketing research (to target advertisements and promotions towards specific consumer groups that have a high probability of responding to them).

Because of the great variety of domains in which they appear, many variants of clustering problems have been studied. However, they all require a partition of a set of objects into clusters that optimizes a given objective function. We present below some of the most popular definitions.

Let S be a set of n objects, generally represented as points in a d–dimensional metric space. A partition of S into k sets $C_1, C_2, \ldots, C_k$ is called a *k–clustering*, and every C_i is called a *cluster*.

We define the *cluster size* of such a k-clustering to be the least value D for which all points in all clusters are either

(a) within distance D of each other, or

(b) within distance $D/2$ of some point called the cluster center.

The distance between two points is measured in the underlying metric (actually, one can define an arbitrary distance function $f : S \times S \rightarrow \mathbb{R}^+$ that does not necessarily satisfy the triangle inequality). The choice of the distance function is closely related to the particular application. We will present a few examples in the next section of the survey, when we discuss different applications of clustering.

Depending on whether we define the cluster size as in (a) or as in (b), the problem is called *pairwise clustering* or *center clustering*, respectively. Furthermore, center clustering problems can be divided into *general center clustering* problems, if the cluster centers can be any points in the d–dimensional space, and *discrete center clustering* problems, if the cluster centers are required to be points of S.

A clustering problem usually has one of the following forms:

(P1) Given S and k, find a k–clustering of S of the minimum cluster size.

(P2) Given S and a real number w, partition S into the minimum number of clusters so that the cluster size is at most w (i.e. find the smallest value of k so that S admits a k-clustering of cluster size $\leq w$).

For problems of type (P1), the objective function is the cluster size. If we define the value D_i for the ith cluster to be the maximum distance between two points in the cluster (or, alternatively, twice the maximum distance between a point in the cluster and the cluster center), then (P1) is the problem to *minimize* over all possible appropriate sets S the objective function

$$c(S) = \max_{1 \leq i \leq k} D_i.$$

This is why (P1) is sometimes referred to as the *min-max problem.*

Variations of (P1) and (P2) require to minimize the *sum* of all cluster sizes for a given S and k (problem (P1')), or give an upper bound on this sum, and require to find the clustering with minimum number of clusters which observes this upper bound (problem (P2')). The size of a cluster, D_i, is defined as the sum of distances between a point in the cluster and the cluster center. Formally, the objective function for (P1') can be written as

$$c(S) = \sum_{i=1}^{k} D_i.$$

In this case, problem (P1') is referred to as the *k–median problem* or the *min-sum problem.*

When the distance between two points is defined according to some metric L_q, $q \geq 1$, we will say that we have a *geometric clustering* problem. In a more general setting, one can define a complete graph with the input points as vertices, and with weighted edges, and define the distance between two points as the weight of the corresponding edge. In this case, we will call the problem *clustering on graph.* This is the approach taken by the applications we describe in the next section.

Most of the clustering problems are NP-complete [13, 33], even when an approximate solution is sought [11, 17, 26]. However, approximation algorithms exist for many of the geometric clustering problems, and for some clustering on graph problems, when the edge weights satisfy the triangle inequality. Unfortunately, the majority of applications we review below use distance functions that rarely have a nice geometric interpretation, and do not satisfy the triangle inequality. That is why heuristics are widely used in practice, although they can have significant drawbacks. Very little, if any, is known about the accuracy of these heuristics (in terms of the approximation factor), but their low time complexity makes them the primary choice in most of the situations we encountered.

We discuss a few applications of clustering in the next section of this survey, while in Section 3 we present the most popular heuristics currently in use.

2 Applications of Clustering

In this section, we review some applications of clustering algorithms in various fields of computer science. For each of them, we discuss to what extent the theoretical approximation algorithms are useful, and what are the problems that arise in practice.

2.1 Data Compression Applications

The min-sum center clustering has one important application in vector quantization, a method used for compression of digitized data such as images or speech [31, 30]. The problem consists in coding a continuous-value random vector by a vector from a finite set of *code vectors.* More specifically, given the space S of the d-dimensional random vector and a finite set $A = \{a_1, \ldots, a_k\}$ of d-dimensional vectors, the *vector quantizer* is a function

$$q : S \to A.$$

The set A is usually called a *code book* or reproduction alphabet, and its elements are the *code vectors.* The quantizer induces a partition of S into k disjoint regions $S1, \ldots, S_k$, where $S_i = \{X \mid q(X) = a_i\}$. For each vector quantizer q, we define the distorsion $D(q)$ to be the expected value of the dissimilarity between the random vector X and the code vector $q(X)$, i.e.

$$D(q) = E[d(X, q(X)].$$

The dissimilarity distance $d(X, q(X))$ is usually defined as the square-error

$$d(X, q(X)) = \sum_{i=1}^{d} (X_i - q(X)_i)^2,$$

although other distances such as the Euclidian or the L_∞ distance have also been considered [28].

When the distribution of X is known, $D(q)$ can be computed as

$$D(q) = \sum_{i=1}^{k} E[d(X, a_i)|X \in S_i] \cdot Pr[X \in S_i].$$

Many times, however, nothing is known about the distribution law of the random variable X. In these cases, it is considered that S consists of a finite sequence of n vectors $X^1, \ldots, X^n$, and $D(q)$ is approximated by

$$D(q) \approx \frac{1}{n} \sum_{i=1}^{n} d(X^i, q(X^i)).$$

Given S and k, the goal of vector quantization is to find the set A and the quantizer q with minimum distorsion. This is the min-sum problem for S and k with the square-error distance function. Adapting the argument of [17], Lin and Storer [27] proved that this problem is NP-complete. The problem was already known to be NP-complete for the Euclidian and the rectiliniar metrics [33], when $d \geq 2$. We are not aware of any approximation algorithm for the min-sum problem with the square-error metric. The current approach to solving this problem is by using heuristics (see Section 3). However, [27] proves that solutions obtained by the best known heuristic are not within any constant factor of the optimal. The authors suggest using the *Greedy Algorithm* of Gonzalez [17] for the min-max problem, starting from the observation that the min-max objective function upperbounds the min-sum objective function. The *Greedy Algorithm* is a theoretical approximation algorithm, guaranteed to achieve an approximation factor of 2 only when the distance satisfies the triangle inequality. Since square-error does not have this property, they are forced to use the Euclidian distance instead.

2.2 Information Retrieval Applications

The design of information retrieval systems must provide efficient methods for organizing and retrieving information from a collection of documents. Although in most of the cases it is considered that the collection consists of text documents, recent trends try to address the problem of multimedia collections [12, 32, 18].

Document clustering has been used in connection with a number of applications, such as query methods, browsing methods, text categorization, and hypertext analysis.

The first two methods (query and browsing) are closely related, as they are two paradigms for accessing a collection of documents (also known as a *corpus*). In the searching method, the user provides a query and asks the system to return a set of documents that most closely match his query. For text documents, the query is usually a word, and the answer should be the set of documents in which that word appears most frequently. Another possibility is that the query is a sequence of words, and the system should return either documents in which all the query words are present, or documents that contain any of the query words, ranked by the number of query words they contain. Document clustering has been used in order to reduce the query time, by matching the query against the clusters, rather than all the documents in the corpus. The intuition is that closely related documents will be relevant to the same queries (see [35] for a comprehensive survey).

When a browsing method is used, the system provides some general information about the corpus, organized in a small number of topics that are considered representative for the entire collection. The user can then select one or more of the topics, and ask the system to reorganize only the related documents into subtopics. When the user is satisfied with the level of detail of the topics, he can ask the system to display all documents in some given cluster. Cutting et al. have introduced the concept of *Scatter/Gather* browsing, and have given a number of clustering heuristics that achieve a trade-off between the precision of clustering and the running time. The slower, more precise algortihms are used off-line to organize the documents, while faster algorithms are used online for reclustering at user's request [9, 8, 19].

Two problems are central for implementing such a clustering algorithm:

1. How to represent a document as a d dimensional object?

2. What should the distance between two documents be?

The answer to the first question is the *vector space representation* of documents. Let d be the number of words in the vocabulary. Each document is represented as a d dimensional vector that has 0 on position i if and only if the ith word does not appear in the document. There are different approaches in the IR world regarding what value should be used on position i if the ith word does appear in the document. The easiest solution is to assign 1 if the word appears at least once in the document. A more accurate representation is to use the number of times word number i appears in the document as the value on position i of the vector. In more sophisticated methods, the value on position i is determined by some function of the relative frequency of word number i in the document and in the corpus.

An equally studied matter regards the answer to the second question. Although the Euclidian distance between the vectors may be occasionally used, more complex similarity measures are implemented in practice. One of the most popular measures seems to be the cosine of the angle between the two vectors (computed after applying a monotone function, usually logarithm or square root, on each element of the vector).

A third problem arises from the answer to the first two: how to create fast algorithms, given the fact that the dimension d is in the order of tens of thousands, and that computing the cosine between two vectors is time consuming, and cannot be assumed to take constant time? There are a number of approaches to reduce the dimension d, starting from the observation that most vectors are sparse. One idea is to use "stop word lists", that contain very frequently used words, which are considered irrelevant to the content of the document. When such a word is encountered, it is simply ignored, and no position is assigned for it in the vector. We mention here a related idea proposed by Bookstein et al. [4]: they argue that whether a word is relevant or not for a set of documents may depend on the documents themselves (i.e. some words may be relevant for a corpus, and irrelevant for another one). Starting from probabilistic arguments, they propose a serial clustering algorithm that estimates which words are irrelevant to a given corpus.

Another technique for reducing d is known as *Latent Semantic Indexing*. The method considers the $d \times n$ matrix A whose n columns are the vectors of all documents in the corpus and finds its singular value decomposition. This allows a projection of the vectors to a new, low-dimensional space.

However, the dimension of the vector space is still high enough after these techniques, and the number of documents can run in the millions, so it is very unlikely that any of the theoretic approximations algorithms can be used in this type of applications. Another impediment in using any of the geometric clustering algorithms is the peculiar form of the similarity measure for documents (the cosine measure), which is quite different of the metrics for which the approximation algorithms were given.

Another application in which clustering methods are useful is the text categorization problem, stated as follows: given a set of training documents with known categories, find the most likely categories that a test document can have. The straightforward approach is to compare the test document with all training documents, find its nearest neighbours among the training documents, and assign to the test document all the categories of its nearest neighbours. However, such an approach is very slow, and hence the need of speeding up the procedure by clustering the training documents first, and then comparing the test document with each cluster. Various clustering criteria have been used, such as: group all documents that are assigned the same category into a cluster labeled by that category; see the documents as vectors (in the sense presented above) and cluster them using the Euclidian distance as criterion of similarity; use a probabilistic algorithm, that estimates the most likely set of clusters from the given training data [22].

An interesting and recent application of clustering is identifying nodes of information on the Internet that are highly related to each other. In particular, hypertext links are clustered according to similarities in the information they contain. In [5], the hypertext is seen as a directed graph, which is then transformed into an undirected graph. The algorithm iteratively finds the maximal k for which the graph has a k-component, assigns all nodes of the k-component to a new cluster and deletes them from the graph (a k-component is a maximal subset of nodes A so that for any partition of A into two sets, the respective cut is traversed by at least k edges).

2.3 Data Bases Applications

The clustering paradigm appears in more than one aspect of VLDB applications, and so the objects and the criteria for clustering vary quite a lot. However, while the number of objects can be very large, the dimension of the space is usually $O(10)$, unlike the dimension for information retrieval algorithms, which can be $O(10^4)$.

An application domain that is receiving considerable attention is database mining. Organizations that have large amounts of data stored in their computers would like to use them to extract information that can improve the quality of their decision making. One typical example is targeting promotional mail according to the customers' response patterns and preferences, as observed from entries recording previous transactions. Based on this history, a classification is developed for identifying potential candidates for future promotions. The first step in developing such an application is clustering a sample set of objects, to obtain a number of groups that are assumed to be relevant for the entire data set (the sample is relevant for the database). After that, classification algorithms are employed in order to find a classification function for each group, that would allow efficient retrieval of all objects in the database that belong to that group. See [1, 37] for more information on classification algorithms.

One of the most common operations that database systems want to support efficiently is called the *merge/purge* problem. The task is to correlate information from different databases by identifying individuals that appear in a number of them. A typical application of this is correlating information from multiple magazine subscription databases that large businesses purchase in order to decide how to target their advertisement. One approach, called the *sorted neighbourhood method*, is to append all files, sort them by the most significant key, and then scan the resulting file and compare all items in a small window to find the matches. To speed up this process, [20] proposes applying a clustering algorithm prior to sorting the file, and then sorting the clusters separately. This has the double advantage of reducing the number of operations during sorting, and allowing a parallelization of the process. Various experiments presented in the paper show a significant speed-up using clustering, at the cost of a very small loss (less than one percent) of precision.

Recently, the advent of spatial databases has triggered studies in using clustering algorithms on the spatial attributes of the data [34, 25]. In a spatial database, objects of more complex types are allowed, such as points, lines, polygons. The user may want to receive as answer to a query all objects that are spatially close to each other. The answer can be farther used for a spatial join with different maps (i.e. the answer is overlaid with maps depicting various information on the same terrain). [34] discusses two clustering algorithms that can be used on objects with both spatial and nonspatial attributes. The spatial attributes are considered to be the x and y coordinates (each object is a point in space), and the metric is either the Euclidian or the L_∞ metric. The authors give a heuristic that estimates the optimal number of clusters, and then clusters the points accordingly. The method, called *CLARANS*, proceeds as follows: each cluster is represented by its *medoid* (the most centrally located data point in the cluster); the clustering process is formalized as searching a graph, in which each node corresponds to a k-clustering (a set of k medoids), and two nodes are adjacent if they differ by only one medoid; starting from a random node, the algorithm checks at most *maxneighbour* random neighbours of that node; if a better node is found, it moves to that node and continues, otherwise it records the current node as a local minimum and restarts with a new randomly selected node; the algorithm stops after a number of *numlocal* local minima have been found, and returns the best of them. The running time depends on the two parameters *maxneighbour* and *numlocal*, and is roughly $O(kn)$, where k is the number of clusters. However, the quality of the solution is estimated only from empirical results, on an input data that has a simple natural clustering.

A different approach to the same problem is proposed in [40]. The method, called *BIRCH*, is claimed to obtain significantly better results. An important contribution of the paper is that it takes into consideration the size of the physical pages, so that a cluster can fit into a disk page. It also allows dynamic clustering, as objects are inserted in the data base. The main idea is to maintain a hierarchy of clusters resembling a B-tree (each cluster is represented by a vector containing a small amount of characteristic information), and to insert a new object into its appropriate cluster by following a path to a leaf in the tree. When a cluster becomes too large to fit into a disk page, the cluster is split into two, and the split is propagated upwards in the tree. The strategy to split a cluster is in fact Gonzalez's method [17], but applied locally, rather than globally (choose the two points in the cluster that are farthest apart and split the cluster around them). Refining methods are proposed to improve the quality of the clustering obtained, at the cost of increased running time and multiple passes through data.

The authors of both papers seem to be unaware of the existing theoretical algorithms, which we believe could be very useful in this case.

At a different level, clustering in spatial databases is used to improve the disk layout of the data, which in turn reduces the access time and query response time. Because of the complexity of the objects in a database, only a few objects fit in a physical page frame, while typical search or range queries tend to return many objects that are close to each other. Thus, clustering the objects into pages, and the pages into larger consecutive disk pages can significantly improve the performance of the system.

The approach in [16] is to monitor the access pattern of the database, and cluster the data according to this pattern. A graph is constructed by the monitoring component: each node corresponds to an object in the database, and has a weight corresponding to the number of times it was accessed; an edge between two nodes has the weight equal to the number of times the corresponding two nodes were accessed consecutively (the edge can be either directed or undirected). Then, a heurisitic clustering algorithm is run on this graph. The authors implemented *Probability Ranking* [39], *Best First Traversal* [21], and *Greedy Graph Partitioning* [14, 15]. The last method achieves the best tradeoff between running time and performance. The number of page faults is reduced to up to 55%, and the running time for clustering and redistributing the data into disk pages is amortized if the application is run a small number of times on the database (between 2 and 12 times).

Another approach to improving the disk layout of the data is taken in [6]. Here, clustering relies heavily on the spatial access method used in the database, which is usually an R^*-tree. The objects are approximated by their *minimum bounding rectangle* (MBR), and a tree is constructed on these MBRs. The information of each node in the tree is stored in exactly one physical page. A non-leaf node stores pointers to its children, and, for each child, it also stores the minimum bounding rectangle of all rectangles stored in that child. A leaf node contains entries that consist at least of the MBRs of some of the objects. One possible organization of the R^*-tree, called *primary organization*, is to store both the objects and their MBRs in the leaves, and hence use only one disk access to transfer the object in the memory. An important drawback of this organization is that unnecessary transfers of the objects into memory will appear, when, many times, only the MBRs are needed. Also, very few MBRs and objects will fit in one node, so accessing neighbouring MBRs or objects will require a lot of disk pages to be transfered into memory. Another organization, called *secondary organization*, is to store only MBRs in the leaves, and have each MBR point to the physical pages that contain the actual objects. While this ensures a clustering within the same page of the MBRs that are close to each other, the actual objects are scattered on the disk. The solution proposed in the paper is to modify the secondary organization and create a cluster organization on 3 levels: the first level consists of the internal nodes of the R^*-tree, kept in the main memory; the second level is formed by the disk pages that correspond to the leaves of the R^*-tree, and only the MBRs of the objects are stored in them; the third level is formed by *cluster units*, and ensures the global clustering of the objects. A cluster unit consists of a fixed number of physical pages, and contains objects whose MBRs are stored in consecutive leaves (hence, objects in the same cluster are close to each other in space). Each leaf has a pointer to its corresponding cluster unit, in addition to the pointers from each MBR to its corresponding object (see Figure 1).

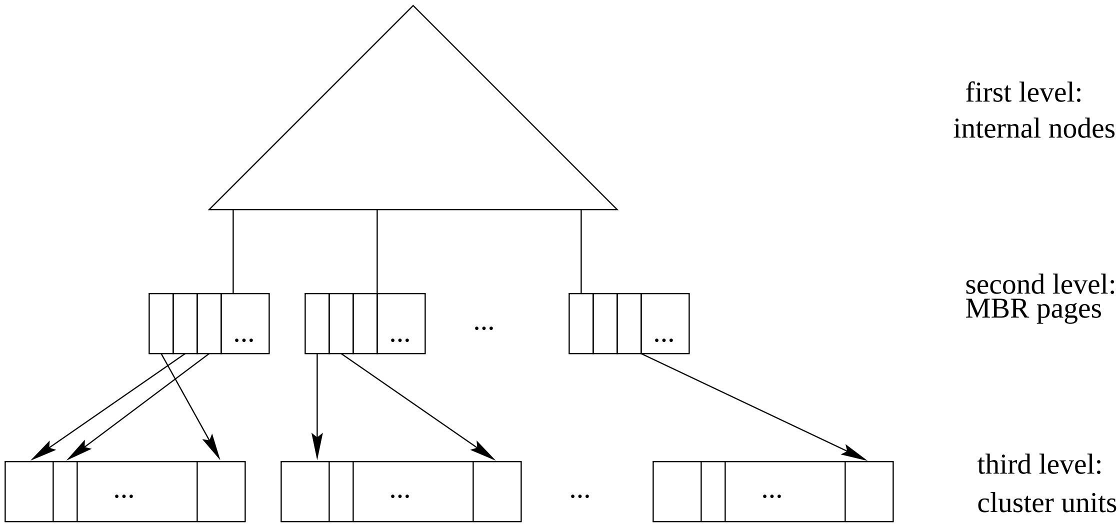

Figure 1: The Structure of the Cluster Organization

An insertion in this new structure differs from an insertion in the R^*-tree in the following ways: once the MBR of the new object is inserted into the corresponding page on the second level, the object itself must be appended to the corresponding cluster unit of that page; if either the maximum number of entries in a second level page, or the maximum size of a cluster unit is exceeded, split the second level page and the cluster unit into two, redistribute the objects, and then use the R^*-tree split algorithm.

We also refer the reader to [10], for a different clustering algorithm that minimizes the number of I/O's on the path from the root to the leaf in the R^*-tree.

2.4 Image Processing Applications

An image processing system transforms continuous images taken from a camera into digital data, and then processes that data to determine features in an image. The goal can be to improve image quality for the human eye (for example, to make satellite pictures easier to read), or to automatically classify various objects appearing in the image.

One important component of such a system is the image segmentation process, which partitions the two-dimensional picture into regions, so that pixels in the same region are more similar, in some sense, than pixels belonging to different regions. Each pixel is assigned a vector, whose components are called *features*. The choice of features depends on the particular application, and must give all the relevant information for the pixel with respect to the clustering criterion which is used. The similarity between two pixels is determined by a function of the corresponding two vectors.

If each cluster of the solution must be a connected set in the xy plane, the partitioning method is said to have spatial connectivity constraints (in this case, the features include the x and y coordinates of a pixel). The problem becomes more complex, due to the fact that clusters of the feature vectors are not necessarily connected in the xy plane. Various methods for finding an initial clustering and refining it to observe the spatial constraints have been studied [7, 24, 38, 36].

Jain and Dubes [23] discuss three types of images for which segmentation algorithms are employed, each of which determines a distinct feature vector. In the case of textured images, the vector of a pixel must reflect textural qualities such as coarseness or regularity. Various methods are used in image processing systems to measure these qualities. In range images, a three dimensional object is represented as a two dimensional picture in which each pixel has a gray value corresponding to the depth of that point (the z coordinate). The goal of the clustering algorithm is to partition the picture into regions that represent surface patches of the original object. This is a problem with spatial connectivity constraints, for which the features are the x, y and z coordinates of the pixel, and the unit normal vector representing the orientation of the surface at the pixel (for a total of 6 features). The last case they present is that of a multispectral image taken from a satellite by scanners that sense the energy reflected from the earth's surface in several wavelengths. There are 6 features, each representing an interval of wavelengths (band).

The approach to clustering is to use heuristics of one of the general types we present in the next section. These are easily adaptable to the specific similarity measure used in the application, as well as to other various conditions on the final clustering (such as having a defined validity measure above a certain threshold). However, given the fact that many times the dissimilarity measure is the square-error (see Subsection 2.1 for the definition), which is the square of the Euclidian distance, and that the dimension of the space is not very high, we think that some of the geometric approximation algorithms could also be used in this context.

2.5 Other Applications

A recent project of the Astrophysical Research Consortium and the Sloan Foundation is aimed to provide a much better map of the universe than is currently available [29].

A telescope containing 660 optical fibers will take a series of “snapshots” from different regions of the sky. The maximum number of galaxies for which data can be gathered in a single snapshot is 660.

Because a main factor in the cost of the survey will be the number of snapshots taken, the problem is to cover the galaxies (represented as points in the plane) by the minimum number of circles of given radius r, so that no more than $c = 660$ points are assigned to one cluster. This is the capacitated clustering problem, for which the best known approximation algorithm by Bar-Ilan et al. achieves an approximation factor of $\log n$. In this case, however, n is too large and the algorithm is not practical. The approach in [29] is heuristical, and it takes into account the fact that the points are distributed densely and somewhat uniformly in the plane. The plane is first covered by disks, so that the cover has (almost) minimum density. Then the covering is perturbed by moving disks towards dense areas, so that the number of points that cannot be assigned to any disk is minimized. The decision on how to move the disks is taken by reducing the problem to a maximum flow problem in graphs.

Numerical analysis methods that solve PDE's using finite differences on a uniform rectangular grid are often faced with the problem that the solution has sharp localized features, thus requiring very fine grids. Because such grids are not of practical size (the resulting matrices are too huge for an efficient computation), the solution adopted in the algorithms is to refine the grid locally: first use an error estimation procedure that “flags” the grid points needing to be refined, then put a finer grid on the rectangular patches that cover the flagged points. Because the cost of the numerical algorithm is proportional to the area of the rectangles, there should be as little unnecessarily refined area as possible. There is also a cost determined by boundary conditions for each rectangular subgrid, thus making it necessary to use as few rectangles as possible.

Two approaches are studied in [3]: a partitioning heuristic that chooses k initial points as cluster centers (k is the number of clusters), determines the clusters corresponding to them and then iteratively refines the cluster centers and the clusters (the general form of this method is detailed in the next section); and a divisive heuristic that iteratively divides each rectangular cluster whose ratio of flagged to unflagged points is too small, into two smaller rectangles (all rectangles are axis-parallel). Considerable attention is given to the way the initial k centers are selected, in the first case, and to the way the splitting place is decided, in the second case. The methods proposed use techniques from computer vision and pattern recognition.

3 Heuristic Algorithms

Unlike the theoretical approximation algorithms, the heuristics we describe here do not rely on a particular distance function, and do not make use of the “geometry” of the problem. These characteristics have made them largely popular in many applications where some kind of clustering is necessary, but the clustering criteria do not have an intuitive interpretation in the geometric space. Their main drawback, however, is that they do not guarantee "good" clusterings, or they may produce clusters where no "natural" clustering exists. Almost any application that uses such a heuristic has to adapt and improve it using knowledge specific to the problem it solves. And although such improvements make it behave reasonably well for most inputs, there is often the case when researchers report inputs for which the method performs very poorly. The various heuristics currently used can be classified into two categories: *hierarchical clustering* and *partitional clustering*. The next two subsections present a summary of these categories, which is largely based on [23] and [2].

3.1 Hierarchical Clustering Heuristics

If $\mathcal{C}$ and $\mathcal{C}'$ are two partitions (clusterings) of the input, we say that $\mathcal{C}$ is nested into $\mathcal{C}'$ if every component of $\mathcal{C}$ is a proper subset of $\mathcal{C}'$. The goal of a hierarchical clustering method is to produce a series of nested partitions of the input, so that at the lowest level the partition consists of n sets, each containing one of the input points, and at the highest level the partition consists of the entire set. The method can be pictured by a tree structure called *dendogram*, in which the nodes correspond to clusters generated during the algorithm, and there is an edge between clusters C and C' if C is included in C'. A layer of the dendogram represents the clustering of the input at some step of the algorithm (hence the name of hierarchical clustering).

Depending on whether the algorithm starts from the n individual clusters and merges them to obtain the entire set, or it starts with the entire set and divides it to get the individual clusterings, the method is called *agglomerative* or *divisive*. Clearly, one can be viewed as the reverse of the other, so in what follows, we will only consider agglomerative algorithms.

Assuming an initial matrix D of dissimilarities between the input points is given, the general form of an agglomerative algorithm is as follows:

Agglomerative Algorithm

1. Begin with n clusters, each containing one input point. Let the clusters be numbered 1 through n.

2. Find the most similar pair of clusters r, s (i.e. $D[r, s]$ is the minimum element of D). Merge clusters r and s, delete the rows and columns of D corresponding to r and s and put a new column and row for the new cluster. To save space, the new column and row can be written over the column and row of r, for example, and so the merged cluster will be labeled r.

3. Repeat step 2 a total of $n-1$ times, until only one cluster remains.

The differences among various agglomerative methods arise from the way step 2 is implemented, more precisely, from the way the dissimilarity distances of the new cluster are computed. We review here the most popular alternatives. In what follows, we will denote by t the cluster obtained in step 2 from merging clusters r and s, and by k any other cluster of the current clustering.

smallskip

(a) *Single-link clustering:* $D[t,k] = \min(D[r,k], D[s,k])$. One way to interpret the clustering at each step of this method is as follows: consider a graph whose nodes correspond to the input points, and whose edges are weighted by the values of the corresponding entries of D. If we look at the subgraph obtained by deleting all edges with weights higher or equal to $D[r,s]$, each maximal connected component of this subgraph corresponds to a cluster right before clusters r and s are chosen in step 2 of the algorithm. Moreover, clusters r and s correspond to the two connected components of the subgraph that can be connected to each other by a single link of minimum weight equal to $D[r,s]$ (hence the name of the method).

(b) *Complete-link clustering:* $D[t,k] = \max(D[r,k], D[s,k])$. If we consider the subgraph described above, the maximal cliques of this subgraph correspond to clusters that exist right before r and s are chosen during step 2. Moreover, the cliques corresponding to clusters r and s can be transformed into one single clique by adding an edge of minimum weight equal to $D[r,s]$ (the method is named like this because a clique is a complete (sub)graph).

(c) *Average linkage clustering:* $D[t,k] = n_r/(n_r+n_s) \cdot D[r,k] + n_s/(n_r+n_s) \cdot D[s,k]$ (unweighted case), or $D[t,k] = 1/2(D[r,k] + D[s,k])$ (weighted case). When computing the distance between two pairs, the average of the dissimilarities between elements of the two clusters is used. In the unweighted case, we take into consideration the number of elements in each cluster, while in the weighted case we ignore this, so elements in smaller clusters are weighted more heavily than elements in larger clusters.

(d) *Centroid clustering:* $D[t,k] = n_r/(n_r+n_s) \cdot D[r,k] + n_s/(n_r+n_s) \cdot D[s,k] - n_r n_s/(n_r+n_s) \cdot D[r,s]$ (unweighted case) or $D[t,k] = 1/2(D[r,k] + D[s,k]) - 1/4 D[r,s]$ (weighted case, also known as *median clustering*). Similar to (c), except that in this case a centroid is computed for each cluster, and the distance between clusters is given by the distance between their respective centroids.

(e) *Ward's clustering:* $D[t,k] = (n_r+n_k)/(n_r+n_s+n_k) \cdot D[r,k] + (n_s+n_k)/(n_r+n_s+n_k) \cdot D[s,k] - n_k/(n_r+n_s+n_k) \cdot D[r,s]$. This method is designed to merge at each step the pair of clusters for which the change in the variance of the clustering is minimized. The variance is defined as the sum of square-errors of the clusters, where for each cluster the square-error is computed by adding the square-error distance between a point in the cluster and the centroid of the cluster.

In practice, the dendogram of a single-link algorithm often tends to look like a chain, an undesirable result that gives little insight on the input set of points. On the other hand, clusters generated by complete-link methods may not be well separated. The other three methods described above try to alleviate these problems, by avoiding the extremes in the way dissimilarities are defined. Comparative studies seem to indicate that Ward's method outperforms all the other hierarchical clusterings.

3.2 Partitional Clustering Heuristics

While the goal of the heuristics described in the previous subsection was to create a hierarchy of clusterings that optimizes the nesting of one clustering into another (in the sense that the resulting dendogram should correspond to the embedding of the input points in space), the heuristics in this section generate a k-clustering of the input set and refine it successively, until a halting criterion is met. They are basically looking for an optimal, or a close to optimal solution, by searching the set of possible clusterings in some heuristic order. The hope is that for most of the problems, this order is good enough so that a clustering meeting the halting condition is found in a few iterations. Even so, however, it can be proved that a bad choice of the initial clustering can lead to a final clustering that approximates the optimal solution by an arbitrarily large factor.

In the following, we present three types of partitional clustering algorithms: *clustering by seed points*, *clustering by graphs* and *nearest-neighbour clustering*.

A clustering algorithm based on seed points proceeds as follows:

1. Choose k initial points, called seeds. Depending on the method employed for choosing them, the seeds may or may not be input points. To each seed there corresponds a cluster, which is initially empty.

2. Assign each input point to the cluster whose seed is closest to the point.

3. For each cluster, compute a new seed, according to some method that takes into account the input points in that cluster.

4. Repeat steps 2 and 3 until a halting condition is met.

Various heuristics can be used for choosing the initial k seeds, such as: randomly choose k of the input points (MacQueen's algorithm); take any partition of the input points into k clusters and choose the seeds to be the centroids of these clusters (Forgy's algorithm); use some hierarchical clustering algorithm to obtain an initial k-clustering (take the highest level in the dendogram that has at least k clusters, and merge the most similar clusters, if necessary, to obtain exactly k clusters).

A distance function must be specified in order to execute Step 2. The method by which a new cluster seed is computed in Step 3 is also closely related to the choice of the distance. The most popular distance in various applications seems to be the square-error, in which case the new seed of a cluster is its centroid.

The algorithm halts in Step 4 either when a maximum number of iterations have been executed, or when the "difference" between two clusterings generated by consecutive iterations is below a certain threshold (the definition of this difference depends on the distance function).

Another category of partitional algorithms uses graphs to divide the input points into clusters. First, a complete graph is defined so that its nodes correspond to the input points, and its edges are weighted by the distance between the corresponding points. The algorithm then proceeds as follows:

1. Construct some subgraph of the complete graph.
2. Remove the inconsistent edges from the subgraph, and call each resulting connected component a cluster.

Several subgraph structures have been considered for the implementation of Step 1: minimum spanning tree, the relative neighbourhood graph (points p_i and p_j are connected by an edge iff $d(p_i, p_j) \leq \max(d(p_i, p_k), d(p_j, p_k))$ for any input point $p_k \neq p_i, p_j$), the Gabriel graph (points p_i and p_j are connected by an edge iff $d^2(p_i, p_j) < d^2(p_i, p_k) + d^2(p_j, p_k)$ for any input point $p_k \neq p_i, p_j$), or the Delaunay triangulation of the input points.

An inconsistent edge is one whose weight is significantly larger than the average of nearby edge weights.

The last algorithm we present uses the nearest neighbour rule to cluster the points. The user provides a threshold parameter t, which will influence the number of clusters generated:

1. Let $S = \{p_1, \ldots, p_n\}$. Assign p_1 to cluster C_1.
2. For each p_k $(2 \leq k \leq n)$, find its nearest neighbour p_j among the points $\{p_1, \ldots, p_{k-1}\}$. If $d(p_j, p_k) \leq t$, assign p_k to p_j's cluster. Else, create a new cluster and assign p_k to this cluster.

4 Conclusions

In this survey, we have reviewed a number of applications of clustering in different fields of computer science. This wide range of applications has lead to various definitions and constraints on what clustering means. However, most such problems are NP-complete.

Theoretical approximation algorithms have been developed for geometric clustering problems, as well as for some clustering on graph problems, when the weights satisfy the triangle inequality. However, in most of the applications we presented, these algorithms cannot be applied, because the distance functions have no geometric interpretation, or do not satisfy the triangle inequality.

Another reason for this gap between theory and practice is that in many applications, the dimension of the space is quite large, making the running time of the theoretical algorithms (which often depends exponentially on the dimension) inacceptably high. This is why most of the implementations used in various fields rely on heuristics. However, there exist inputs for which a heuristic generates solutions that are very different from the optimal one, making the information obtained from clustering irrelevant or erroneous.

There are two main classes of heuristics: hierarchical and partitional methods. The latter seem to have a wider range of applications and to construct more "accurate" clusterings. However, methods of the first type are also used when not only a clustering, but also a hierarchy of clusterings is desired (in Information Retrieval, for example), or to obtain an initial set of seeds for a partitioning method.

References

[1] R. Agrawal, A. Ghosh, T. Imielinski, B. Iyer, and A. Swami, An interval classifier for database mining applications, *Proceedings of the 18th Conference on Very Large Databases*, Morgan Kauffman, August 1992.

[2] M. R. Anderberg, *Cluster Analysis for Applications*, Academic Press, New York, 1973.

[3] M. Berger and I. Rigoutsos, An algorithm for point clustering and grid generation, *IEEE Trans. Systems, Man and Cybernetics.*, 21 (1991), 1278–1286.

[4] A. Bookstein, S. T. Klein, and T. Raita, Detecting content-bearing words by serial clustering, *Proceedings of the Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1995, pp. 319–327. Extended Abstract.

[5] R. A. Botafogo, Cluster analysis for hypertext systems, *Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1993, pp. 116–125.

[6] T. Brinkhoff and H.-P. Kriegel, The impact of global clusterings on spatial database systems, *Proceedings of the International Conference on Very Large Databases*, Morgan Kauffman, September 1994.

[7] G. Coleman and H. Andrews, Image segmentation by clustering, *Proceedings of the IEEE.*, 1979, pp. 773–785.

[8] D. R. Cutting, D. R. Karger, and J. O. Pedersen, Constant interaction-time scatter/gather browsing of very large document collections, *Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1993, pp. 126–134.

[9] D. R. Cutting, D. R. Karger, J. O. Pedersen, and J. W. Tukey, Scatter/gather: A cluster-based approach to browsing large document collections, *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1992, pp. 318–329.

[10] A. A. Diwan, S. Rane, S. Seshadri, and S. Sudarshan, Clustering techniques for minimizing external path length., *Proceedings of the International Conference on Very Large Databases*, Morgan Kauffman, 1996.

[11] T. Feder and D. H. Greene, Optimal algorithms for approximate clustering, *Proc. 20th Annu. ACM Sympos. Theory Comput.*, 1988, pp. 434–444.

[12] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker, Query by image and video content: The QBIC system, *Computer*, 28 (1995), 23–32.

[13] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto, Optimal packing and covering in the plane are NP-complete, *Inform. Process. Lett.*, 12 (1981), 133–137.

[14] C. A. Gerlhof, A. Kemper, C. Kilger, and G. Moerkotte, Clustering in object bases, Tech. Rep. D-76050, University of Karlsruhe, 1992.

[15] C. A. Gerlhof, A. Kemper, C. Kilger, and G. Moerkotte, Partition-based clustering in object bases: from theory to practice, *Proceedings of the International Conference on Foundations of Data Organization and Algorithms*, 1993, pp. 301–316.

[16] C. A. Gerlhof, A. Kemper, and G. Moerkotte, On the cost of monitoring and reorganization of object bases for clustering, *SIGMOD Record*, 25 (1996), 22–27.

[17] T. Gonzalez, Clustering to minimize the maximum intercluster distance, *Theoret. Comput. Sci.*, 38 (1985), 293–306.

[18] K.-A. Han and S.-H. Myaeng, Image organization and retrieval with automatically constructed feature vectors, *Proceedings of the Nineteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*

(H.-P. Frei, D. Harman, P. Schäuble, and R. Wilkinson, eds.), ACM Press, August 1996, pp. 157–165.

[19] M. A. Hearst and J. O. Pedersen, Reexamining the cluster hypothesis: Scatter/gather on retrieval results, *Proceedings of the Nineteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (H.-P. Frei, D. Harman, P. Schäuble, and R. Wilkinson, eds.), ACM Press, August 1996, pp. 76–84.

[20] M. A. Hernandez and S. J. Stolfo, The merge/purge problem for large databases, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, 1995, pp. 127–138.

[21] S. E. Hudson and R. King, Cactis: A self-adaptive, concurrent implementation of an object-oriented database management system, *ACM Transactions on Database Systems*, 14 (1989), 291–321.

[22] M. Iwayama and T. Tokunaga, Cluster-based text categorization: A comparison of category search strategies, *Proceedings of the Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1995, pp. 273–280.

[23] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[24] J. Jolion, P. Meer, and S. Batauche, Robust clustering with applications in computer vision, *IEEE Trans. Pattern Analysis Mach. Intell.*, 13 (1991), 791–802.

[25] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, John Wiley & Sons, 1990.

[26] M. T. Ko, R. C. T. Lee, and J. S. Chang, An optimal approximation algorithm for the rectilinear m-center problem, *Algorithmica*, 5 (1990), 341–352.

[27] J. Lin and J. A. Storer, Geometric clustering to minimize the mean-square error, *Unpublished Manuscript*, (1997).

[28] Y. Linde, A. Buzo, and R. M. Gray, An algorithm for vector quantiser design, *IEEE Transactions on Communications*, COM-28 (1980), 84–95.

[29] R. Lupton, F. M. Maley, and N. Young, Data collection for the sloan digital sky survey: A network-flow heuristic, *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, ACM/SIAM, January 1996, pp. 296–303.

[30] J. MacQueen, Some methods for classification and analysis of multivariate observations, *Proceedings of the Fifth Berkeley Symposium on Math., Stat. and Prob.*, 1967, pp. 281–296.

[31] J. Makhoul, S. Roucos, and H. Gish, Vector quantization in speech coding, *Proceedings of the IEEE*, 73 (1985), 1551–1588.

[32] C. Meghini, An image retrieval model based on classical logic, *Proceedings of the Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1995, pp. 300–308.

[33] N. Megiddo and K. J. Supowit, On the complexity of some common geometric location problems, *SIAM J. Comput.*, 13 (1984), 182–196.

[34] R. T. Ng and J. Han, Efficient and Effective Clustering Methods for Spatial Data Mining, *Proceedings of the Twentieth International Conference on Very Large Databases*, 1994, pp. 144–155.

[35] P. Raghavan, Information retrieval algorithms: A survey, *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, January 1997, pp. 11–18.

[36] P. Schroeter and J. Bigün, Hierarchical image segmentation by multi-dimensional clustering and orientation-adaptive boundary refinement, *Pattern Recognition.*, 28 (1995), 695–709.

[37] J. Shafer, R. Agrawal, and M. Mehta, Sprint: A scalable parallel classifier for data mining., *Proceedings of the International Conference on Very Large Databases*, Morgan Kauffman, 1996.

[38] M. Spann and R. Wilson, Quadtree approach to image segmentation which statistical and spatial information, *Pattern Recognition*, 18 (1985), 257–269.

[39] P. C. Yue and C. K. Wong, On the optimality of the probability ranking cheme in storage applications, *Journal of the ACM*, 20 (1973), 624–633.

[40] T. Zhang, R. Ramakrishnan, and M. Livny, Birch: An efficient data clustering method for very large databases, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, 1996, pp. 103–115.