U N I V E R S I T Y O F T A R T U
Faculty of Mathematics and Computer Science
Institute of Computer Science

Meelis Kull

Fast Clustering in Metric Spaces

Master Thesis

Supervisor: Jaak Vilo, PhD

TARTU 2004

Contents

Chapter 1

Introduction

1.1 Motivation and Background

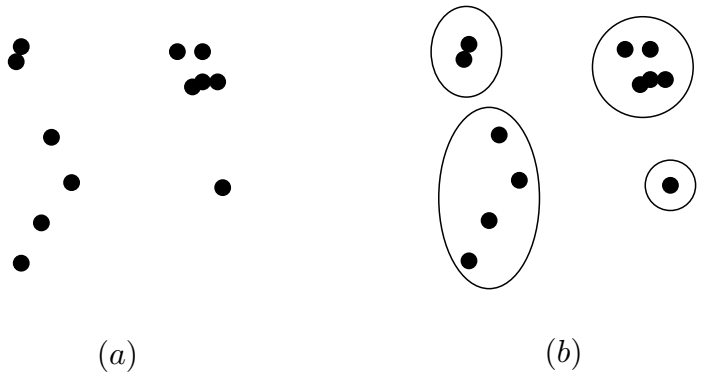
Data analysis is a process that involves describing, summarizing, and comparing data. It has traditionally been a branch of statistics. However, it has now also spread to the fields of computer science, for example machine learning, artificial intelligence, data management, and data mining.

An important technique frequently used in data analysis is clustering. The aim of clustering is to organize data based on similarity. It can be used on any dataset consisting of items for which we have a measure of similarity.

The organization of data can be represented in different ways. The simplest one is to split the data into disjoint *clusters*, *i.e.* groups of items, such that the items within any group are similar, and the items in different groups are dissimilar. For example, Figure 1.1a depicts a dataset with 12 items, and Figure 1.1b gives one possible clustering with 4 clusters. Here, similarity is defined as the geometric distance between the points representing the items.

Fuzzy clustering is another way of representing the similarities found in data. Here, every item may belong to many clusters with some partial amount of membership. This enables to capture the situation where some items are between clusters.

Hierarchical clustering represents data as a tree with the data items as leaves. The tree is generated so that the similarity of any two items is represented by the distance to their nearest common branching.



(a) (b)

Figure 1.1: A dataset with 12 data items (a) and one possible clustering of this dataset (b).

Figure 1.1 shows data items as points in the plane. Similarity can be measured as the usual Euclidean distance. But it is just one possibility. Actually, for any kind of data one can define infinitely many different measures of similarity. Which is the best always depends on the application. Therefore, the measure of similarity must be fixed before clustering.

Most of the clustering algorithms can be applied to only certain kinds of data and some particular measures of similarity. For example, many algorithms require the data to be in the form of vectors of real numbers. These algorithms make use of the operations of the vector field, *i.e.* multiplication by scalars and addition.

A different set of algorithms can be applied in the case of *metric spaces*. Here, restrictions on the similarity measure, rather than on the form of data, are imposed. First, dissimilarity is measured instead of similarity. In the setting of metric spaces, the dissimilarity is referred to as *distance*. In addition, the distance must obey several conditions. The first condition is that the distance must be a non-negative number. The value zero means that the items are identical with respect to this measure. If the distance grows, then the items get less similar. The second condition is that the distance from x to y must be the same as the distance from y to x. Finally, the *triangle inequality* must hold. It means that the distance between any two data items via a third item is greater than or equal to the direct

distance. Mathematically written,

$$d(x, z) + d(z, y) \geqslant d(x, y),$$

where $d(x, y)$ denotes the distance between items x and y.

If we would omit the triangle inequality, we would be in the setting of *semi-metric spaces*. However, the triangle inequality can be useful in the case of large datasets to help speeding up the clustering algorithms (Elkan 2003).

Clustering is extensively applied in biology and ecology as an important technique of data analysis (Legendre & Legendre 1998). The development of technologies for gaining huge amounts of data has increased the need for fast clustering methods. A good example is microbiology, which provides large sets of numerical and textual data about living organisms.

The topic of this thesis was inspired by the author's contribution developing Expression Profiler — a web-based software package for gene expression data analysis (Vilo *et al.* 2003; Kapushesky *et al.* 2004 accepted for publication). Expression Profiler has a clustering module with several clustering algorithms implemented. The user can submit data and perform clustering via a web interface. Therefore, it is desirable that the clustering is fast. The aim of this thesis is to find a method for fast clustering in metric spaces.

1.2 State of the Art

One of the most commonly used clustering algorithms is the k-*means* method. It is meant to be used for splitting the data items into k disjoint clusters, where k is priorly fixed. This method first chooses k items to be the centers of the clusters. Second, it assigns each item to the cluster, whose center is closest to the item. Third, it recalculates each center to be the *mean* of the cluster. The second and third step are repeated until convergence.

Unfortunately, k-means can only be applied to vector data. The reason is that the mean is defined using addition and multiplication by scalars. For other kinds of data, the k-*medoids* method is used instead of k-means. The k-medoids method uses *medoids* as centers of the clusters. The medoid is defined only based on the distances. Therefore, k-medoids can be used in any metric space.

Another kind of algorithms that can be used in arbitrary metric spaces are some of the hierarchical clustering methods. We will introduce three of these in the third chapter. We will also give a more detailed description of the k-means and k-medoids clustering methods.

The common problem with the k-medoids and hierarchical clustering methods is that they calculate $\Omega(n^2)$ distances between the data items, where n is the number of items in the dataset. For many applications this is too slow.

One way to overcome the problems caused by the large size of data is summarizing (Ganti *et al.* 1999; Breunig *et al.* 2001; Zhou & Sander 2003). It is a technique which summarizes each group of very similar items into a *microcluster*. For every microcluster, only a summary is stored. The summary contains of a small number of data items, called *representatives*. The rest of the items are described only by a few statistical parameters, such as the average and maximum distances from the representatives.

Next, some distance measure between the microclusters is defined. This is typically based on the distance between the representative items and in a way takes into account the parameters of the microclusters. Now, we can take the set of microclusters and run any clustering algorithm that is working in metric spaces in order to cluster the microclusters. Finally, the clustering of microclusters can be converted to the clustering of the original data.

An important problem is to decide, which items are summarized into which microclusters. The naïve way of obtaining the microclusters is by random sampling (Breunig *et al.* 2001). First, a sample of items is randomly drawn from the dataset. Next, all the other items are divided into microclusters based on the sampled item they are closest to. During this process some statistical parameters can be incrementally calculated in order to generate the summary. This approach is developed into a fast hierarchical clustering method in (Zhou & Sander 2003). More complicated methods for constructing microclusters have been given in (Ganti *et al.* 1999).

The common problem with data summarization is that two very similar data items may incidentally fall into different microclusters and therefore be clustered apart.

A different idea is to map the data items into some vector space so that the distances between the items remain approximately the same. Such mappings are

called *distance preserving transformations*. Now, we can cluster the items in the vector space instead. The point of doing this is that clustering can be more efficient in vector spaces (Ganti *et al.* 1999). The mapping can be performed in reasonable time, a well-known method for this is FastMap (Faloutsos & Lin 1995).

The downside of this idea is that some datasets cannot be mapped so that the distances remain the same. Or, the dimensionality needed is very large (comparable to the number of data items). In the first case our clustering will not be correct and in the second case the clustering method used in the vector space will be very slow because of the high dimensionality. Therefore, we must map the data into a low dimensional vector space, the distances may get distorted, and a good clustering in the vector space may be of bad quality in the original space. Experimental evidence has been presented in (Ganti *et al.* 1999).

1.3 Contribution and Overview

The key idea of this thesis can be stated simply — the fact that two items are very similar to each other is important information for any clustering algorithm. In some sense, it is more important than the fact about dissimilarity. For example, if x and y are very similar and we know that x and z are not similar, then we can infer that probably y and z are not similar either. We cannot do the same if x and y are not similar. Actually, we were using the triangle inequality here.

Inspired by this key idea, a method for finding all pairs of items that have distance less than some fixed ε is developed in the thesis. Next, an approximate version of hierarchical clustering is developed. It takes a set of pairs of items as input and performs the clustering based on only these pairs. Finally, the similar pairs found by the first method are used as input to the approximate hierarchical clustering. The experiments confirm that similar pairs induce a clustering with better quality than random pairs.

We now give the overview of the thesis. Chapter 2 gives the definitions and notations used throughout the thesis. Chapter 3 introduces the k-medoids and k-means clustering algorithms as well as three variants of agglomerative hierarchical clustering. Chapters 4 and 5 describe the developed methods for finding pairs of similar items and performing approximate hierarchical clustering in metric spaces. Chapter 6 concludes the work and brings out points for further work.

Chapter 2

Definitions and Notations

In this chapter, we first list all the limitations we place on the data. Next, we give the definitions of some metric spaces used in the applications. After that, we discuss the concept of a *clustering criterion*. Finally, we list the datasets used in this thesis in examples and experiments.

2.1 Limitations on Data

Clustering assumes that the data can be represented as a finite list of data items. We call this list the *dataset* and denote it by X. Most of the clustering algorithms are not sensitive to the order of items in the dataset, but as some are, we define X as an ordered list. Data items, *i.e.* the elements of the list X will be denoted as $x_1, x_2, \ldots, x_n$. The list may contain duplicates as this may be the case with the data.

The only thing we are interested in about data is similarity. Actually, for the sake of mathematical convenience, we will use the reversed scale, *i.e.* we talk about dissimilarity instead of similarity. The measure of dissimilarity must be provided together with the data. The reason is that there are many ways of measuring dissimilarity, and which method is the best depends on the application. So we assume that there is a predefined way of calculating the dissimilarity of any two items x_i and x_j. We denote the dissimilarity value by $d(x_i, x_j)$. We want all our methods to work on any kind of data. Therefore, calculating dissimilarity is the only operation we are allowed to perform on data.

So far, we have put only inevitable restrictions on the data. Now we make our main assumption that restricts the area of application. We require the dissimilarity measure to be a *metric*. It means that it is a real-valued function with the following three properties:

1. $\forall i, j \in \{1, \ldots, n\}, \quad d(x_i, x_j) = 0 \Leftrightarrow x_i = x_j$ — *reflexivity*;
2. $\forall i, j \in \{1, \ldots, n\}, \quad d(x_i, x_j) = d(x_j, x_i)$ — *symmetry*;
3. $\forall i, j, k \in \{1, \ldots, n\}, \quad d(x_i, x_j) \leqslant d(x_i, x_k) + d(x_k, x_j)$ — *triangle inequality*.

Triangular inequality is analogous to the geometrical fact that any side of the triangle is always shorter than the sum of the other two sides. Reflexivity states that duplicates are the only source of zero dissimilarity. Suppose two inherently different items have dissimilarity zero. Then they are still equivalent for us because we cannot distinguish them by only calculating dissimilarities. Therefore, we can safely assume reflexivity.

From these three conditions the fourth follows:

4. $\forall i, j \in \{1, \ldots, n\}, \quad d(x_i, x_j) \geqslant 0$ — *non-negativeness*.

If the dissimilarity measure is a metric, we refer to dissimilarity as *distance*. This is by analogy with the geometrical distance which is also a metric.

Often the distance measure d is defined on a larger set than just the data items X. This encourages the introduction of $\mathcal{D}$ as the set of all items between which d can measure distances such that it remains a metric. The set $\mathcal{D}$ is to be taken as the domain where we are working. Most of the time the domain will be $\mathcal{D} = \{x_i \,|\, i = 1, \ldots, n\}$ — the dataset X without duplicates. Hence, the methods do not assume more domain knowledge than explicitly given in the data.

To sum up, we are going to assume that:

- Data is given in the form of a list, $X = (x_1, \ldots, x_n)$;
- All data items belong to a specified domain, $x_i \in \mathcal{D}$ for all $i \in \{1, \ldots, n\}$;
- A metric distance measure d is defined on the domain, $d : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$. In other words, the pair $(\mathcal{D}, d)$ is a *metric space*.

2.2 Metric Spaces

A metric space is a set (the domain) together with a metric distance measure defined between its elements. In this section we give the definitions of some metric spaces met in the applications. The metric spaces will be from two important domains. The first is the set of m-tuples of real numbers, $\mathbb{R}^m$. We will give the metric spaces by defining the distance measures only, because we have fixed the domain.

Definition 1 *Let $\mathcal{D} = \mathbb{R}^m$. The distance measure defined as*

$$d_{L_2}(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_{i=1}^{m} (a_i - b_i)^2}$$

for any m-tuples $\mathbf{a} = (a_1, \ldots, a_m)$ and $\mathbf{b} = (b_1, \ldots, b_m)$ is called the Euclidean distance. *The metric space $(\mathbb{R}^m, d_{L_2})$ is called the* Euclidean space.

The Euclidean distance is the geometric distance we are used to in our 3-dimensional space. We will prove that this and the next two distance measures are metrics after definition 3.

Definition 2 *Let $\mathcal{D} = \mathbb{R}^m$. The distance measure defined as*

$$d_{L_1}(\mathbf{a}, \mathbf{b}) = \sum_{i=1}^{m} |a_i - b_i|$$

for any m-tuples $\mathbf{a} = (a_1, \ldots, a_m)$ and $\mathbf{b} = (b_1, \ldots, b_m)$ is called the Manhattan distance *or the* city-block distance.

The Manhattan distance is the traveling distance one must cover, if only allowed to move in the directions of the coordinate axes. If the distance has to be measured in a city with rectangular blocks, the Manhattan distance is probably the most appropriate one.

Definition 3 *Let $\mathcal{D} = \mathbb{R}^m$ and $p > 0$. The distance measure defined as*

$$d_{L_p}(\mathbf{a}, \mathbf{b}) = \left(\sum_{i=1}^{m} |a_i - b_i|^p \right)^{\frac{1}{p}}$$

*for any m-tuples $\mathbf{a} = (a_1, \ldots, a_m)$ and $\mathbf{b} = (b_1, \ldots, b_m)$ is called the L_p-*distance *or the* Minkowski distance*.*

In the case of $p = 1$ we obtain the Manhattan distance and in the case of $p = 2$ the Euclidean distance. We now prove that L_p-distance is a metric. Reflexivity and symmetry are trivial. The triangle inequality can be proved using the *Minkowski inequality*,

$$\left(\sum_{i=1}^{m} |x_i + y_i|^p\right)^{\frac{1}{p}} \leqslant \left(\sum_{i=1}^{m} |x_i|^p\right)^{\frac{1}{p}} + \left(\sum_{i=1}^{m} |y_i|^p\right)^{\frac{1}{p}},$$

which holds for any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^m$. For proof of the Minkowski inequality (the integral version), see (Yosida 1995). Now, we just have to substitute $\mathbf{a} - \mathbf{c}$ for $\mathbf{x}$ and $\mathbf{c} - \mathbf{b}$ for $\mathbf{y}$ to obtain the triangle inequality $d_{L_p}(\mathbf{a}, \mathbf{b}) \leqslant d_{L_p}(\mathbf{a}, \mathbf{c}) + d_{L_p}(\mathbf{c}, \mathbf{b})$.

Definition 4 *Let $\mathcal{D} = \mathbb{R}^m$. The distance measure defined as*

$$d_{L_\infty}(\mathbf{a}, \mathbf{b}) = \max_{i=1}^{m} |a_i - b_i|$$

*for any m-tuples $\mathbf{a} = (a_1, \ldots, a_m)$ and $\mathbf{b} = (b_1, \ldots, b_m)$ is called the L_∞-*distance*.*

This distance measure is obtained from the L_p-distances by taking the limit as p approaches infinity. Therefore, it is also a metric.

The second domain contains textual data, *i.e.* strings of letters of a finite alphabet. We denote the alphabet by Σ, so the domain is the set Σ^*.

Definition 5 *Let $\mathcal{D} = \Sigma^*$. We define the distance between any two strings $\mathbf{a} = a_1 a_2 \ldots a_u$ and $\mathbf{b} = b_1 b_2 \ldots b_v$ as the number of positions $i \in \{1, \ldots, \max(u, v)\}$ where the strings $\mathbf{a}$ and $\mathbf{b}$ differ, i.e. $a_i \neq b_i$ or $u < i$ or $v < i$. We call this distance the* Hamming distance *and denote it by $d_{\mathrm{Hamming}}(\mathbf{a}, \mathbf{b})$.*

The Hamming distance is well-known for bit-strings, *i.e.* for $\Sigma = \{0, 1\}$. It is a metric because, first, it is trivially reflexive and symmetrical. And second, the triangle inequality also holds: if strings $\mathbf{a}$ and $\mathbf{b}$ differ at some location then for any string $\mathbf{c}$ either the strings $\mathbf{a}$ and $\mathbf{c}$ or the strings $\mathbf{c}$ and $\mathbf{b}$ (or both) must differ at the same location.

Definition 6 *Let $\mathcal{D} = \Sigma^*$. We define the distance between any two strings $\mathbf{a} = a_1 a_2 \dots a_u$ and $\mathbf{b} = b_1 b_2 \dots b_v$ as the number of operations needed to obtain $\mathbf{b}$ from $\mathbf{a}$. The allowed operations are*

- *Replacement — replacing one letter with some other letter from the same alphabet Σ.*
- *Deletion — deleting one letter from the string.*
- *Insertion — inserting one letter into the string at some location.*

This distance is known as the edit distance *or the* Levenshtein distance *and we denote it by $d_{\text{edit}}(\mathbf{a}, \mathbf{b})$.*

The edit distance is also a metric. It is trivially reflexive and symmetrical. The triangle inequality also holds, because if a string $\mathbf{c}$ can be obtained from another string $\mathbf{a}$ by $d_{\text{edit}}(\mathbf{a}, \mathbf{c})$ operations and a string $\mathbf{b}$ from the string $\mathbf{c}$ by $d_{\text{edit}}(\mathbf{c}, \mathbf{b})$ operations then these same operations can be used to obtain the string $\mathbf{b}$ from $\mathbf{a}$.

2.3 Clustering

The aim of clustering is to group data according to similarity. That is, similar items have to be in the same cluster and dissimilar items in different clusters. Of course, this is not a definition, but just an intuitive description of the task. For example, in Figure 2.1 there are two results of clustering for the same dataset. It is not clear why one should be better than the other. We could even say that it is a matter of taste.

In order to define the clustering task well, many formal criteria have been suggested. In some sense they all match the description of the aim of clustering given above.

Let the data $X = (x_1, \dots, x_n)$ be split into k clusters C_1,…,C_k, with n_1,…,n_k items, respectively. Let the cluster C_i consist of items $C_i = (x_1^{(i)}, x_2^{(i)}, \dots, x_{n_i}^{(i)})$. The clusters must be disjoint and must cover the whole dataset X. It means that the list of items of all clusters must be the same as the original list X but possibly in some other order. A clustering criterion gives an evaluation value (or *score*) to any such division of data into clusters.

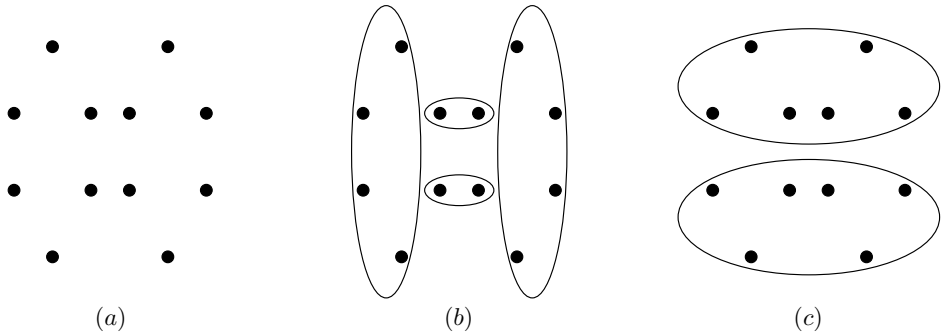


Figure 2.1: A dataset with 12 items (a) and two different clusterings for this dataset (b), (c).

According to (Jain, Murty, & Flynn 1999), the most frequently used criterion is the *squared error criterion*. The squared error is given by the following formula:

$$e(C_1, \ldots, C_k) = \sum_{j=1}^{k} \sum_{i=1}^{n_j} \left(d(x_i^{(j)}, c_j) \right)^2,$$

where c_j is the centroid of the cluster C_j. The centroid is an element of the cluster minimizing the following sum:

$$\sum_{i=1}^{n_j} \left(d(x_i^{(j)}, c_j) \right)^2.$$

The squared error criterion can be applied for the fixed number of clusters k, and according to this criterion it is best to choose the clustering with minimal squared error. In Figure 2.1, the preferred clustering would be 2.1c. One clustering method optimizing this criterion, the k-medoids method, is further described in the next chapter.

The problem with the squared error criterion is that using it for comparing clusterings with different numbers of clusters is not possible. It would most probably choose the clustering with more clusters. For example, a clustering with every item as a separate cluster has the best score (zero) with respect to this criterion. Therefore special methods need to be used for estimating a good number of clusters.

i	x_i
1	$(3.5, 0.9)$
2	$(1.5, 3.0)$
3	$(1.7, 5.5)$
4	$(2.1, 8.0)$
5	$(3.0, 8.4)$
6	$(5.1, 5.2)$
7	$(8.4, 6.5)$
8	$(8.3, 2.2)$
9	$(8.3, 1.4)$

$d_{L_2}(\cdot,\cdot)$	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
x_1	0.0	2.9	4.9	7.2	7.5	4.6	7.4	5.0	4.8
x_2	2.9	0.0	2.5	5.0	5.6	4.2	7.7	6.8	7.0
x_3	4.9	2.5	0.0	2.5	3.2	3.4	6.8	7.4	7.8
x_4	7.2	5.0	2.5	0.0	1.0	4.1	6.5	8.5	9.1
x_5	7.5	5.6	3.2	1.0	0.0	3.8	5.7	8.2	8.8
x_6	4.6	4.2	3.4	4.1	3.8	0.0	3.5	4.4	5.0
x_7	7.4	7.7	6.8	6.5	5.7	3.5	0.0	4.3	5.1
x_8	5.0	6.8	7.4	8.5	8.2	4.4	4.3	0.0	0.8
x_9	4.8	7.0	7.8	9.1	8.8	5.0	5.1	0.8	0.0

Table 2.1: The items of the dataset 1 (a) and the distance matrix corresponding to the Euclidean distance (b).

Another kind of solution is hierarchical clustering which, intuitively, generates clusterings for all possible numbers of clusters. Hierarchical clustering is also described in the next chapter.

2.4 Datasets used in the Thesis

As the last thing in this chapter we describe the datasets used in this thesis. The first dataset is used as a working example to illustrate most of the methods throughout the thesis. The other two datasets are used for experiments in the fifth chapter.

Dataset 1 Let $\mathcal{D} = \mathbb{R}^2$ and $n = 9$. Thus, we have a nine-item dataset of two-dimensional vectors. The dataset itself is given in Table 2.1a and the plot is given in Figure 2.2. The distance measure we are going to use on this dataset will be the Euclidean. The distance matrix containing all the pairwise distances is given in Table 2.1b.

Dataset 2 The second dataset contains gene expression data of the baker's yeast (*Saccharomyces cerevisiae*) from (Eisen *et al.* 1998), available from (Eisen

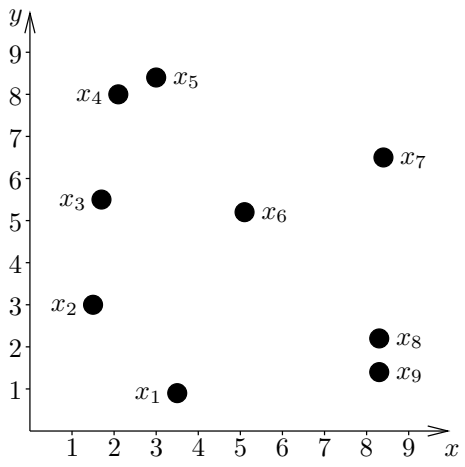


Figure 2.2: The items of the dataset 1 plotted in the plane.

2004). The dataset has expression levels for 6221 genes in 80 experiments. We will cluster the genes, therefore the number of items will be $N = 6221$, and every item is a vector of length 80. We will experiment with two distance measures:

(a) the Euclidean distance;

(b) the Manhattan distance.

Dataset 3 The third dataset contains the first 50000 basepairs in the genome of the baker's yeast (*Saccharomyces cerevisiae*), available from (Eisen 2004). The basepairs are split into $N = 1000$ strings of length 50 and clustered using the edit distance.

Chapter 3

Clustering in Metric Spaces

In this chapter we describe two commonly used clustering methods that can be applied in the setting of metric spaces. First, we explain the k-medoids method, which is a modification of the well-known k-means clustering algorithm. After that we introduce three variants of agglomerative hierarchical clustering.

3.1 k-Medoids Clustering

The k-medoids clustering method is a modification of the k-means method, which is one of the most widely used clustering methods. The reason for modification is the fact that k-means makes use of the vector space operations, *i.e.* addition of vectors and multiplication of vectors by some scalar. Therefore it cannot be used in the setting of arbitrary metric spaces, where we are only allowed to use distances between items. In the following we introduce the k-medoids method and also explain what the original k-means method is.

k-Medoids is a method for splitting the dataset into k clusters. The algorithm is as follows.

Algorithm 1: k**-Medoids clustering**

1. Choose k data items to be the initial centers of clusters, let the centers be $c_1, \ldots, c_k$. The choice may be random, but other strategies can also be used.

2. For each data item x_i ($i = 1, \ldots, n$) find the center c_j which is nearest to it, *i.e.* $d(x_i, c_j)$ is minimized. Assign x_i to cluster C_j.

3. Recompute the cluster centers, i.e. for each $j \in \{1, \ldots, n\}$ find a new c_j in cluster C_j which minimizes the squared error in the cluster:

$$\sum_{i=1}^{n_j} \Big(d(x_i^{(j)}, c_j) \Big)^2 .$$

4. If the centers have changed compared to the last iteration, go to step 2.

Most of the time this method converges quite fast. For the rare cases when the method loops infinitely, some threshold should be set for the number of iterations.

Figure 3.1 depicts the 3 iterations that were made in the case of the 9-item dataset described in the previous chapter. Here we assume that in step 3 of Algorithm 1 the first c_j is chosen in the case of several equally good candidates.

Let us now give an estimation for the time complexity of the k-medoids method. Step 2 has complexity $\mathcal{O}(nk)$, because the distances from n items to k pivots are calculated. Here, as well as later in the text, we assume the calculation of a distance to take a constant time. In step 3 we have to calculate all the pairwise distances within each cluster. In the best case all the clusters are of the same size. Hence, the time complexity is $\mathcal{O}(\frac{n^2}{k^2})$. In the worst case we have one big cluster and many small clusters. Thus, the complexity of step 3 is $\mathcal{O}(n^2)$. This is dominating over the second step. Let us denote the number of iterations by l. Then the best and worst case complexities of k-medoids are $\mathcal{O}(nkl + \frac{n^2}{k^2}l)$ and $\mathcal{O}(n^2 l)$, respectively.

k-Means clustering assumes that the domain is $\mathcal{D} = \mathbb{R}^m$ for some $m \in \mathbb{N}$ and the distance used is the Euclidean. It differs from k-medoids at the third step. Instead of choosing the minimizing item from the dataset, it looks for it in the

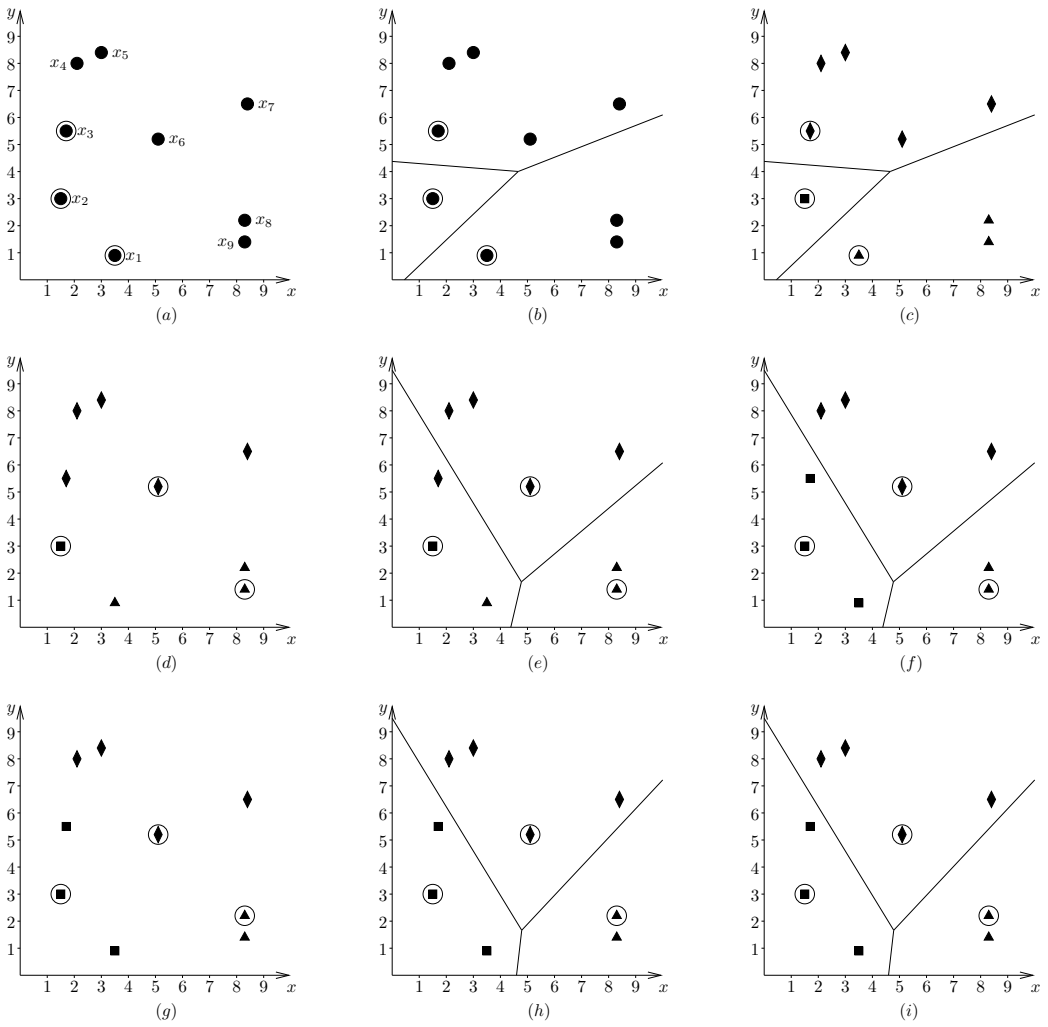


Figure 3.1: k-Medoids clustering with $k = 3$. First, three centers are chosen (a). Next, for each data item the nearest center is determined (b) and each data item is put into the cluster with the nearest center (c). Then the new centers are calculated (d) and the new iteration starts. Iterating is finished when the centers would not change (i). The final clustering is in the last figure (i).

whole domain $\mathbb{R}^m$. The minimizing element is the *mean*, which is defined as follows (Estivill-Castro 2002):

$$c_j = \frac{1}{n_j} \cdot \sum_{i=1}^{n_j} x_i^{(j)}. \tag{3.1}$$

This is easy to calculate and it makes the k-means method very fast. The time complexity of k-means is $\mathcal{O}(nkl)$.

The drawback of k-means is that it assumes the domain $\mathcal{D} = \mathbb{R}^m$ and the Euclidean distance. It can be used with other distance measures but then formula (3.1) may not give the center with respect to the used distance measure.

Both k-medoids and k-means clustering are *hill-climber* type of methods for optimizing the squared error criterion (Estivill-Castro 2002). They may not give the globally optimal answer.

3.2 Hierarchical Clustering

Hierarchical clustering does not simply split the data into groups, it generates a nested grouping of data items. Figure 3.2 depicts the results of three different hierarchical clustering methods. Every such tree is called a *dendrogram*. The dendrogram can be cut at some distance from the leaves to obtain a clustering with a fixed number of clusters. For example, in Figure 3.2d the dendrogram is cut at distance 5.0. We obtain three subtrees, which represent clusters with items (x_2, x_3, x_1), (x_4, x_5) and (x_8, x_9, x_6, x_7). Thus, dendrograms contain clusterings for all possible numbers of clusters $k = 2, 3, \ldots, n-1$. Dendrograms are also good for the visualization of the similarities in the data.

Hierarchical clustering can be performed either *agglomeratively* or *divisibly*, i.e. the dendrogram can be built starting from the leaves or from the root, respectively. Most of the time agglomerative hierarchical clustering is used, hence we focus on this. We will refer to it as simply *hierarchical clustering*. There are three main types of hierarchical clustering — single linkage, average linkage, and complete linkage. They all work according to the same schema.

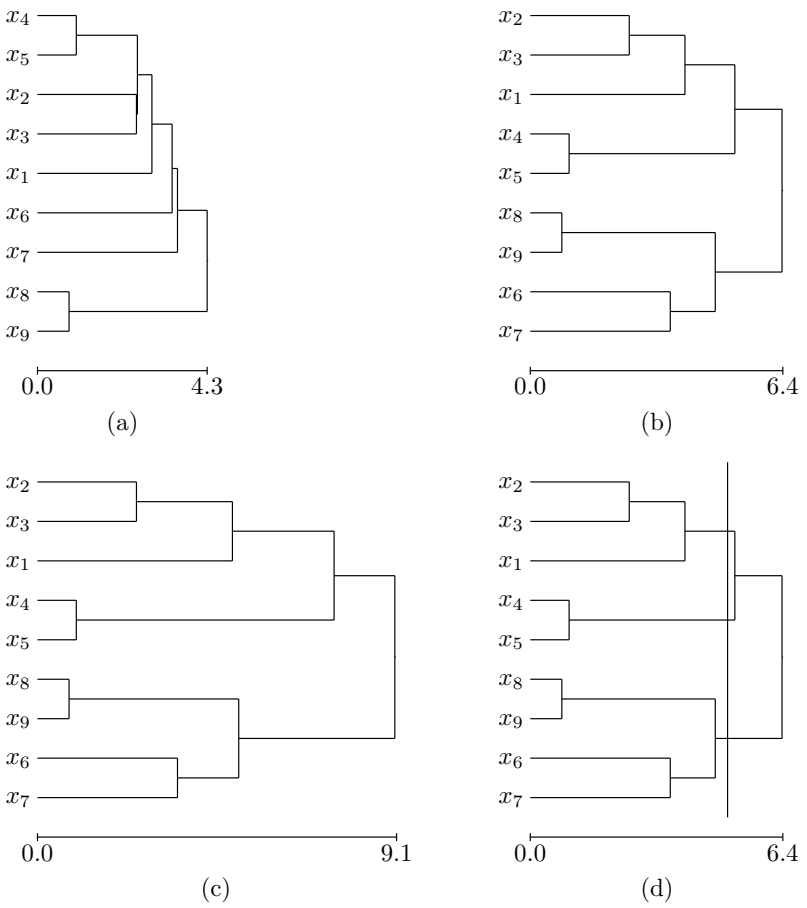


Figure 3.2: Agglomerative hierarchical clustering using single linkage (a), average linkage (b), and complete linkage (c) methods. Cutting the dendrogram of average linkage at distance 5.0 (d) yields the clustering $C_1 = (x_2, x_3, x_1)$, $C_2 = (x_4, x_5)$ and $C_3 = (x_8, x_9, x_6, x_7)$.

Algorithm 2: Hierarchical clustering

1. Start with each data item as a *singleton cluster* (cluster with a single item).
2. Join the two nearest clusters into one. In the dendrogram, join the corresponding subtrees at the distance (measured from the leaves) which is equal to the distance between the clusters.
3. While the number of clusters is more than one, repeat step 2.

The difference between the three methods is how they measure the distance between clusters. In the case of both clusters being singletons, the distance is just the distance between the data items. Therefore, in this case the three methods work the same. The difference appears when at least one of the clusters is not a singleton.

- The single linkage method defines the distance between two clusters C_u and C_v ($u, v \in \{1, \ldots, k\}$) as follows:

  $$d(C_u, C_v) = \min_{i=1}^{n_u} \min_{j=1}^{n_v} d(x_i^{(u)}, x_j^{(v)}),$$

  where n_u denotes the size of the cluster C_u. It means that the distance between two clusters is equal to the distance between the nearest pair of items, where the items are from different clusters.

- The complete linkage method defines the distance between two clusters C_u and C_v ($u, v \in \{1, \ldots, k\}$) as follows:

  $$d(C_u, C_v) = \max_{i=1}^{n_u} \max_{j=1}^{n_v} d(x_i^{(u)}, x_j^{(v)}).$$

  So in this case the distance between two clusters is the distance between the most distant pair of items.

- The average linkage method defines the distance between two clusters C_u and C_v ($u, v \in \{1, \ldots, k\}$) as follows:

  $$d(C_u, C_v) = \frac{1}{n_u \cdot n_v} \sum_{i=1}^{n_u} \sum_{j=1}^{n_v} d(x_i^{(u)}, x_j^{(v)}).$$

Average linkage is somewhere in between the single and complete linkage methods. Here the distance between two clusters is the average of all distances between pairs.

In Figure 3.2 there are dendrograms for all three methods applied to our sample 9-item dataset. All three methods start by joining the items x_8 and x_9, since this is the nearest pair in the dataset. Next two steps are also the same for all methods, x_4 is connected with x_5 and x_2 is connected with x_3. After that we have the situation given in Figure 3.3a. At fourth step, single linkage would join clusters (x_2, x_3) and (x_4, x_5) as the next step. The distance between these clusters with respect to single linkage is $d(x_3, x_4) \approx 2.5$, see Figure 3.3b. The average linkage method differs at this point. It joins clusters (x_6) and (x_7) because $d(x_6, x_7) \approx 3.5$, but the distance between clusters (x_2, x_3) and (x_4, x_5) is

$$\frac{1}{2 \cdot 2} \left(d(x_2, x_4) + d(x_2, x_5) + d(x_3, x_4) + d(x_3, x_5) \right) \approx 4.1,$$

see Figure 3.3c. Complete linkage makes the same choice as average linkage, as the distance between clusters (x_2, x_3) and (x_4, x_5) is equal to $d(x_2, x_5) \approx 5.6$, see Figure 3.3d.

Figure 3.2 shows that the results of different methods can be quite different. Single linkage quite typically builds some kind of a cascade. It means that quite often a singleton (or very small) cluster is connected with a huge cluster. This can also be seen in Figure 3.2a. Complete linkage on the contrary tends to join clusters of approximately same size. Average linkage is somewhere in between, but this is not apparent in such a small example.

These three methods of hierarchical clustering can be carried out in time $\mathcal{O}(n^2 \log n)$ by using priority queues to determine the nearest neighbour of each cluster. First, we build for each cluster C_u a priority queue of all other clusters in the order of distances from C_u. During every iteration we find the nearest pair of clusters by finding the nearest neighbour of each cluster (using the priority queues). After joining the nearest clusters C_u and C_v, distance from the new cluster to every other cluster C_w is calculated. This can be done in constant time for all three methods as follows:

(a) Single linkage — $d(C_u \cup C_v, C_w) = \min\left(d(C_u, C_w), d(C_v, C_w)\right)$;
(b) Complete linkage — $d(C_u \cup C_v, C_w) = \max\left(d(C_u, C_w), d(C_v, C_w)\right)$;
(c) Average linkage — $d(C_u \cup C_v, C_w) = \frac{n_u \cdot d(C_u, C_w) + n_v \cdot d(C_v, C_w)}{n_u + n_v}$.

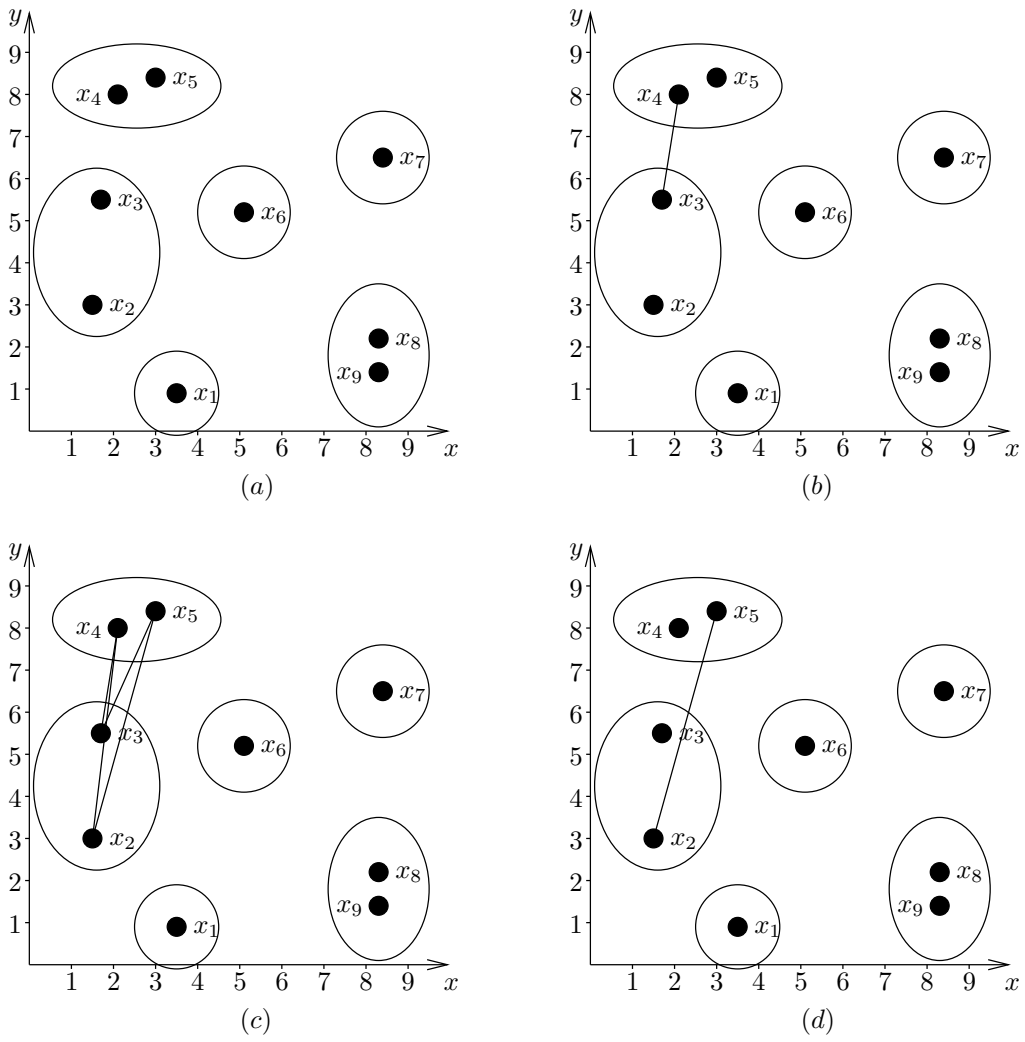


Figure 3.3: Situation after three steps of hierarchical clustering of the sample dataset (a). The pairs making up the distance between clusters (x_2, x_3) and (x_4, x_5) are joined with lines for single linkage (b), average linkage (c), and complete linkage (d).

These formulas follow trivially from the definitions of distances between clusters. As the last thing of the iteration, the priority queues are updated. When using heaps as the implementation of priority queues, every operation can be performed in $\mathcal{O}(\log n)$ time. During the iteration $\mathcal{O}(n)$ operations are performed, therefore taking $\mathcal{O}(n \log n)$ time. The number of iterations is $n - 1$, yielding the time complexity of $\mathcal{O}(n^2 \log n)$.

It appears that this complexity can be lowered to $\mathcal{O}(n^2)$ by using nearest neighbour chains. For details, see for example (Olson 1995). This is also the best we can achieve because we have to calculate all pairwise distances anyway.

Chapter 4

Finding Pairs of Similar Data Items

In this chapter we have the task to find all the pairs of ε-similar items, *i.e.* items with distance less than some fixed $\varepsilon > 0$. The trivial way would be to calculate all the pairwise distances. However, it is possible to avoid this. Here, we give one possible solution. This will be used in the next chapter for fast approximate hierarchical clustering.

4.1 Technique of Pivots

How can we be sure that we have found all the pairs of ε-similar items, if we have not calculated the distance between some items x_i and x_j? The only way is to prove that either $d(x_i, x_j) < \varepsilon$ or $d(x_i, x_j) \geqslant \varepsilon$. The only means we have for that in the metric space is the triangle inequality. Let us have some item x_k with distances $d(x_i, x_k)$ and $d(x_k, x_j)$ calculated. Now we try to infer something from the three different triangle inequalities that can be written using distances between the items x_i, x_j, and x_k:

$$\begin{aligned} d(x_i, x_j) &\leqslant d(x_i, x_k) + d(x_k, x_j), \\ d(x_i, x_k) &\leqslant d(x_i, x_j) + d(x_j, x_k), \\ d(x_k, x_j) &\leqslant d(x_k, x_i) + d(x_i, x_j). \end{aligned}$$

By using the symmetry of distance (i.e., $d(x, y) = d(y, x)$), we can convert these inequalities into the following three:

$$\begin{aligned} d(x_i, x_j) &\leqslant d(x_i, x_k) + d(x_j, x_k), \\ d(x_i, x_j) &\geqslant d(x_i, x_k) - d(x_j, x_k), \\ d(x_i, x_j) &\geqslant d(x_j, x_k) - d(x_i, x_k). \end{aligned}$$

Since the left side is positive anyway, the two last inequalities can be taken together, obtaining the following two inequalities:

$$d(x_i, x_j) \leqslant d(x_i, x_k) + d(x_j, x_k), \tag{4.1}$$

$$d(x_i, x_j) \geqslant |d(x_i, x_k) - d(x_j, x_k)|. \tag{4.2}$$

They give respectively the upper and lower bound to the distance $d(x_i, x_j)$. Now we are able to prove the inequalities $d(x_i, x_j) < \varepsilon$ and $d(x_i, x_j) \geqslant \varepsilon$ using the two implications following from (4.1) and (4.2):

$$d(x_i, x_k) + d(x_j, x_k) < \varepsilon \implies d(x_i, x_j) < \varepsilon, \tag{4.3}$$

$$|d(x_i, x_k) - d(x_j, x_k)| \geqslant \varepsilon \implies d(x_i, x_j) \geqslant \varepsilon. \tag{4.4}$$

It would also be possible to use chains of triangle inequalities, like this one:

$$d(x_i, x_j) \leqslant d(x_i, x_k) + d(x_k, x_j) \leqslant d(x_i, x_k) + d(x_k, x_l) + d(x_l, x_j).$$

However, we are not going to use such inequalities for the sake of simplicity. We are even going to discard the implication (4.3), and only make use of implication (4.4). The reason for this is that we will need the exact values for distances less than ε for clustering.

In order to find all pairs of ε-similar items, we now propose the following strategy.

1. Calculate the distances between some chosen pairs of items. Among these we may already find some ε-similar pairs. All other pairs, for which we did not calculate the distance, are set as candidate pairs for being ε-similar.

2. For each candidate pair (x_i, x_j), do the following. For all items x_k such that $d(x_i, x_k)$ and $d(x_j, x_k)$ are calculated, check if $|d(x_i, x_k) - d(x_j, x_k)| \geqslant \varepsilon$. If

this is the case, we do not have to calculate the distance $d(x_i, x_j)$, because it is not less than ε by (4.4). Thus we drop the pair (x_i, x_j) from the candidate list.

3. Calculate the distance $d(x_i, x_j)$ for all pairs (x_i, x_j) that are still in the candidate list.

The main question here is how to choose the pairs in the first step. We want to keep the number of distance calculations low, so we should not choose too many pairs. For example, we should probably avoid calculating all three pairwise distances for some items x_i, x_j, and x_k because maybe we could have calculated only two and used implication (4.4) for the third distance. On the other hand, we should probably try to have for each pair (x_i, x_j) some item x_k such that $d(x_i, x_k)$ and $d(x_j, x_k)$ are calculated. Otherwise we cannot use implication (4.4).

One way to have these requirements fulfilled is to calculate in the first step all distances from some fixed x_k. We call such an item x_k a *pivot*. Now, for every candidate pair (x_i, x_j) we know the distances $d(x_i, x_k)$ and $d(x_j, x_k)$. Therefore we obtain a lower bound for $d(x_i, x_j)$ from (4.2).

Example 1 Let us take the sample dataset with 9 items and choose x_1 to be the pivot (Figure 4.1a). Suppose we want to find all pairs with distance less than 1.5. In step 1 we calculate the distances from x_1 to all other items. The resulting values are listed in Table 4.1a. In step 2 we have to go through all the candidate pairs, which are in this case all pairs between items $x_2, x_3, \ldots, x_9$. The only item x_k for which $d(x_i, x_k)$ and $d(x_j, x_k)$ are both calculated is x_1. Thus, we drop the pair (x_i, x_j) if and only if $|d(x_i, x_1) - d(x_j, x_1)| \geqslant \varepsilon$. In step 3 we calculate the distances between the remaining candidate pairs, which are (x_3, x_6), (x_3, x_8), (x_3, x_9), (x_4, x_5), (x_4, x_7), (x_5, x_7), (x_6, x_8), (x_6, x_9), and (x_8, x_9). Among them, (x_4, x_5) and (x_8, x_9) are ε-similar. This can be seen in the distance matrix given in Table 2.1b on page 15. Altogether in steps 1 and 3 we had to calculate $8 + 9 = 17$ distances, while in total there are $\binom{n}{2} = \frac{9 \cdot 8}{2} = 36$ distances.

As could be seen in the example, the lower bound is not very tight. Therefore still quite a lot of distances had be calculated. The number of calculated distances might be lower when taking more than one pivots. Then each pivot sets a lower bound for every candidate pair and altogether we might get a greater lower bound.

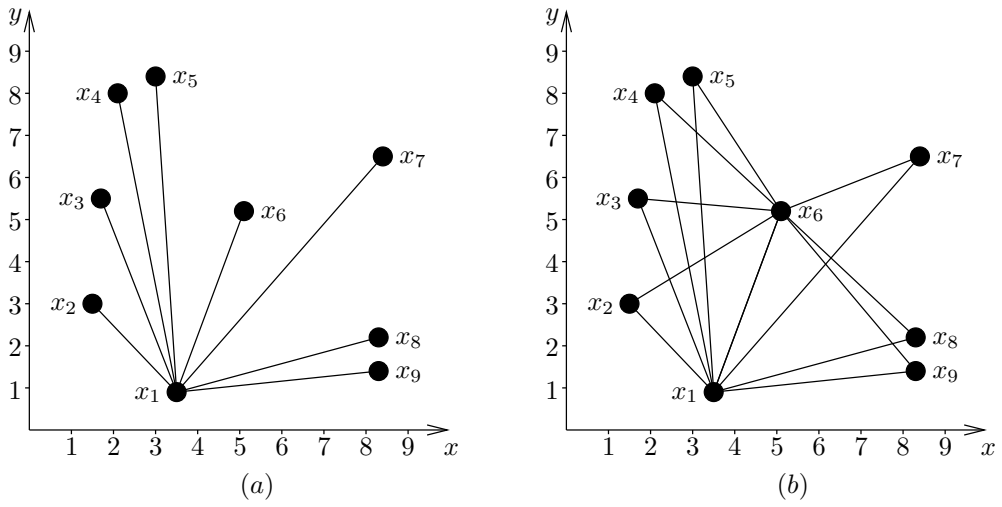


Figure 4.1: Situation after step 1 if item x_1 is the pivot (a) and if items x_1 and x_6 are the pivots (b).

i	2	3	4	5	6	7	8	9
$d(x_i, x_1)$	2.9	4.9	7.2	7.5	4.6	7.4	5.0	4.8

(a)

i	1	2	3	4	5	6	7	8	9
$d(x_i, x_1)$	0.0	2.9	4.9	7.2	7.5	4.6	7.4	5.0	4.8
$d(x_i, x_6)$	4.6	4.2	3.4	4.1	3.8	0.0	3.5	4.4	5.0

(b)

Table 4.1: Distances from item x_1 to all other items (a). Additionally, distances from item x_6 to all other items (b).

Example 2 Let us repeat the previous example with two pivots x_1 and x_6. In step 1 we have to calculate additionally the distances from x_6 to all other items (Figure 4.1b). The results are given in Table 4.1b. In step 2 we can throw out the candidate pairs that have either $|d(x_i, x_1) - d(x_j, x_1)| \geqslant \varepsilon$ or $|d(x_i, x_6) - d(x_j, x_6)| \geqslant \varepsilon$. As a result the remaining candidate pairs for step 3 are (x_3, x_8), (x_4, x_5), (x_4, x_7), (x_5, x_7), and (x_8, x_9). As mentioned in the previous example, (x_4, x_5) and (x_8, x_9) are ε-similar. Altogether in steps 1 and 3 we had to calculate $15 + 5 = 20$ distances. Although it turned out to be more than in the previous example, the situation may be better with larger datasets.

In the general case we have q pivots $p_1, p_2, \ldots p_q$. The three steps in the algorithm given above can now be written as follows (in step 2 we have the maximum instead of checking one by one):

Algorithm 3: Finding ε-similar items

1. For every item x_i ($1 \leqslant i \leqslant n$) and every pivot p_t ($1 \leqslant t \leqslant q$) calculate the distance $d(x_i, p_t)$. Add all pairs with the distance not calculated to the list of candidates.

2. For every candidate pair (x_i, x_j) check if $\max\limits_{t=1}^{q} |d(x_i, p_t) - d(x_j, p_t)| \geqslant \varepsilon$. If it is the case, drop the pair (x_i, x_j) from the list of candidates.

3. Calculate distance $d(x_i, x_j)$ for all pairs (x_i, x_j) that are still on the candidate list.

Altogether in steps 1 and 2 all the pairs of items are considered. Therefore, this algorithm has time complexity at least $\Omega(n^2)$. However, in the next section we will find a way to filter the candidates more efficiently.

A similar technique of pivots is being used when searching in metric spaces (Chavez *et al.* 2001). In that area, the typical problems of interest are the following:

- *Range query* — find all items which are within some distance ε from a fixed query item x_i.

- *Nearest neighbour query* — find the closest item from a fixed query item x_i.
- *K-Nearest neighbour query* — find the K closest items from a fixed query item x_i.

This is different from our case because we are interested in finding all pairs that are within some distance ε. Nevertheless, it would be possible to use the procedure solving range queries for our task. We could just run range queries with distance ε for every possible item x_i. But an algorithm designed specially for our task would probably run faster. Therefore, we hold on to Algorithm 3 and try to give a more efficient version of step 2.

4.2 Finding L_∞-Similar Pairs

First note that in step 2 of Algorithm 3 the only information we use for every data item is q real numbers — the distances to each pivot. Hence, in step 2 we can view each item x_i as a point $\overline{x_i}$ in space $\mathbb{R}^q$. Let us denote $\overline{x_i}(t) = d(x_i, p_t)$ for all $i \in \{1, \ldots, n\}$ and $t \in \{1, \ldots, q\}$. The condition to check in step 2 can now be written as

$$\max_{t=1}^{q} |\overline{x_i}(t) - \overline{x_j}(t)| \geqslant \varepsilon.$$

By the definition of the L_∞ distance in space $\mathbb{R}^q$ (see definition 4 on page 12), this is exactly the same as $d_{L_\infty}(\overline{x_i}, \overline{x_j}) \geqslant \varepsilon$. All the pairs satisfying this condition are thrown away. Thus, we actually have to find all pairs $(\overline{x_i}, \overline{x_j})$ which are ε-similar in the sense of the L_∞-distance! In the following we call these pairs L_∞-ε-similar. The ε-similar pairs in the sense of the original distance d will be referred to as d-ε-similar pairs. To sum up, we have found that step 2 essentially involves the whole problem in a specialized form. Next, all our energy goes to finding L_∞-ε-similar pairs.

With one pivot ($q = 1$) this is easy. In this case $\overline{x_1}, \ldots, \overline{x_n}$ are just real numbers, among which we have to find those that differ by less than ε. We can do it by first sorting the numbers. Let the ordered list be $\overline{x_{i_1}}, \ldots, \overline{x_{i_n}}$. Next, starting from each number $\overline{x_{i_k}}$ we move forward in the sorted order, while the numbers differ from $\overline{x_{i_k}}$ by less than ε. On the way we output each item $\overline{x_{i_{k+c}}}$ together

with $\overline{x_{i_k}}$ because they are L_∞-ε-similar. The time complexity of this algorithm is $\mathcal{O}(n \log n + s)$ where $\mathcal{O}(n \log n)$ goes for sorting and s is the size of the output.

Unfortunately, the same trick does not work in the general case of many pivots. The best we can do in the same fashion is the following. First, find all the pairs of items which are L_∞-ε-similar with respect to the first pivot. And after that check manually, if they also differ by less than ε in all the other coordinates. But this may become an $\Omega(n^2 q)$ algorithm in the case when distances to the first pivot differ by less than ε most of the time.

Let us try to come up with a better algorithm for the general case. The key idea is discretization. We divide each dimension in the space $\mathbb{R}^q$ into the following ranges:

$$\begin{array}{rcl} \text{range } 0 & — & [0, \varepsilon), \\ \text{range } 1 & — & [\varepsilon, 2\varepsilon), \\ \text{range } 2 & — & [2\varepsilon, 3\varepsilon), \\ \ldots & — & \ldots \end{array}$$

where $[x, y)$ denotes the set $\{r \in \mathbb{R} \,|\, x \leqslant r < y\}$. Recall that the numbers are non-negative as we are working with distance values. With these ranges we have divided the space $\mathbb{R}^q$ into hypercubic buckets of size ε in each dimension. Each bucket is of the form $[i_1\varepsilon, i_1\varepsilon + \varepsilon) \times \ldots \times [i_n\varepsilon, i_n\varepsilon + \varepsilon)$ for some $i_1, \ldots, i_n \in \mathbb{N}$. We now go through all q-tuples $\overline{x_1}, \ldots, \overline{x_n}$ and throw them into the buckets they belong to. Suppose now some two items $\overline{x_i}$ and $\overline{x_j}$ are in the same bucket. Then they must be L_∞-ε-similar because in each coordinate they differ by less than ε (because they are in the same bucket). Thus we can output all the pairs of items that fall into same buckets. The time complexity is $\mathcal{O}(nq + s)$, where s is again the size of the output.

However, the serious drawback is that we may miss some (or even most) of the pairs. They may be divided into neighbouring buckets, close to the border between the buckets. Therefore, we must also check all the pairs between neighbouring buckets. But since we are working in the q-dimensional space, there are $3^q - 1$ neighbours for each bucket. Now we have time complexity $\mathcal{O}(nq + s + n \cdot 3^q + s') = \mathcal{O}(s + n \cdot 3^q + s')$, where s' is the number of pairs located in neighbouring buckets but not being L_∞-ε-similar.

The downside is that it is difficult to predict the order of s' because it depends on the distribution of data. In the worst case, s' may be $\Omega(n^2)$. For example, if half

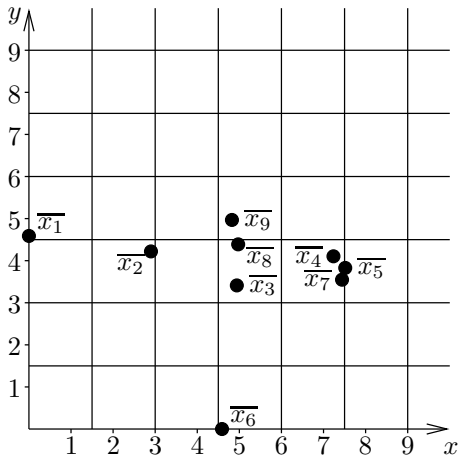


Figure 4.2: The items of the sample dataset plotted in the plane defined by the distances to the pivots x_1 and x_6.

of the items $\overline{x_1}, \ldots, \overline{x_n}$ fall into one bucket and the other half to the neighbouring one, while no two from different halves are L_∞-ε-similar.

The other drawback is that there may be a huge amount of buckets, which requires a lot of space. This can be overcome by using a hash table for keeping the buckets because no more than n buckets are used.

Example 3 Let us take our sample dataset with items x_1 and x_6 as pivots. Figure 4.2 is the plot of items $\overline{x_1}, \ldots, \overline{x_9}$ in the plane $\mathbb{R}^2$. Let the task be to find all d-ε-similar pairs for $\varepsilon = 1.5$. We generate the buckets, which are hypercubes of size ε in each dimension, in our case squares. We calculate the vectors $\overline{x_1}, \ldots, \overline{x_9}$ for the items $x_1, \ldots, x_9$, and throw them into the buckets. Any pair of items that fall into the same bucket are L_∞-ε-similar. Therefore we get the following L_∞-ε-similar pairs: $(\overline{x_3}, \overline{x_8})$ and $(\overline{x_4}, \overline{x_7})$. But we also have to check the pairs situated in neighbouring buckets: $(\overline{x_1}, \overline{x_2})$, $(\overline{x_3}, \overline{x_4})$, $(\overline{x_3}, \overline{x_7})$, $(\overline{x_3}, \overline{x_9})$, $(\overline{x_4}, \overline{x_5})$, $(\overline{x_4}, \overline{x_8})$, $(\overline{x_4}, \overline{x_9})$, $(\overline{x_5}, \overline{x_7})$, $(\overline{x_7}, \overline{x_8})$, $(\overline{x_7}, \overline{x_9})$, and $(\overline{x_8}, \overline{x_9})$. Among them $(\overline{x_4}, \overline{x_5})$, $(\overline{x_5}, \overline{x_7})$, and $(\overline{x_8}, \overline{x_9})$ are ε-similar. Finally, we check if the L_∞-ε-similar pairs are d-ε-similar. As in examples 1 and 2, we find that (x_4, x_5) and (x_8, x_9) are d-ε-similar.

We can get the term $n \cdot 3^q$ in the time complexity down to $n \cdot 2^q$ by using

intersecting buckets. Let us now divide each coordinate into the following larger intersecting ranges:

$$\begin{array}{rcl} \text{range } 0 & — & [0\varepsilon, 2\varepsilon), \\ \text{range } 1 & — & [1\varepsilon, 3\varepsilon), \\ \text{range } 2 & — & [2\varepsilon, 4\varepsilon), \\ \ldots & — & \ldots \end{array}$$

This division gives us hypercubic buckets of size 2ε in each of the q dimensions. Each bucket is of the form $[i_1\varepsilon, i_1\varepsilon + 2\varepsilon) \times \ldots \times [i_n\varepsilon, i_n\varepsilon + 2\varepsilon)$ for some $i_1, \ldots, i_n \in \mathbb{N}$. Thus, each item $\overline{x_i}$ is contained in up to 2^q buckets, because each coordinate belongs to either one or two ranges (one if the value is less than ε). The algorithm for finding L_∞-similar pairs is now as follows.

Algorithm 4

1. Go through all the items $\overline{x_1}, \ldots, \overline{x_n}$ and throw them into all of the buckets they belong to.
2. For any bucket and any two items $\overline{x_i}$ and $\overline{x_j}$ in that bucket:
   (a) Check if they are L_∞-ε-similar.
   (b) If they are L_∞-ε-similar and if we have not output the pair $(\overline{x_i}, \overline{x_j})$ yet, then output it now.

Here, step 2a is needed because the buckets are large, and two items in the same bucket may be not L_∞-ε-similar. Step 2b is needed because every item belongs to many buckets and therefore, the same pair may be together in more than one bucket. But now we do not have to check the neighbouring buckets anymore. The reason is that if two items differ in a certain coordinate by less than ε, then by the definition of the regions, there exists a region containing both values. Therefore, there must exist a bucket which contains both of the items.

The worst case time complexity of this algorithm is $\mathcal{O}(n \cdot 2^q + s'')$, where s'' is the number of times steps 2a and 2b are executed. Let us analyze, when s'' is large. First, s'' is large when two items are together in many buckets. Second, s'' is large when many items fall into one bucket, and hence many pairs can be

formed. We will try to solve the first problem by making the buckets larger, but keeping the amount of overlap the same. This will generate a lot of space that is not overlapping. And if an item falls into the non-overlapping area, it will be put into only one bucket. While relieving the first problem, it makes the second one even worse. Now, the buckets are larger and therefore, contain more items. We will try to find a compromise by starting from huge buckets and splitting them until they contain less items than some fixed threshold.

We will now put these ideas in a more formal language. First, we give a generalized way for dividing $\mathbb{R}^q$ into intersecting buckets. The new ranges are the following:

$$\begin{array}{rcl} \text{range } 0 & — & [0\delta, 1\delta + \varepsilon), \\ \text{range } 1 & — & [1\delta, 2\delta + \varepsilon), \\ \text{range } 2 & — & [2\delta, 3\delta + \varepsilon), \\ \ldots & — & \ldots \end{array} \tag{4.5}$$

where $\delta > 0$ is the parameter that determines the size of buckets, which is $\delta + \varepsilon$. Every two consecutive ranges have intersection of size ε. Note that in the case of $\delta = \varepsilon$ we get exactly the same ranges as before. Due to the overlaps of size ε we can here also be sure that if two items are L_∞-ε-similar, then they are together in at least one bucket. Therefore, we can use the same Algorithm 4 for finding the similar pairs.

In Algorithm 4 we looked through all pairs in the same bucket and checked if they are ε-similar. We could do it here too. But if the bucket is large, there may be many items and therefore too many pairs. Thus, if the number of items in the bucket is greater than some threshold, we split this bucket into many small intersecting buckets by using some smaller δ. Then we can start working in these little buckets with possibly less elements. A natural choice of a new δ is half of the original, because then the original region $[i\delta, i\delta + \delta + \varepsilon)$ gets nicely split into two overlapping regions $[2i\frac{\delta}{2}, 2i\frac{\delta}{2} + \frac{\delta}{2} + \varepsilon)$ and $[2(i+1)\frac{\delta}{2}, 2(i+1)\frac{\delta}{2} + \frac{\delta}{2} + \varepsilon)$. Thus, the original bucket gets split into 2^q small buckets.

If the new small buckets are still too large, we can continue splitting. It seems to be a bad idea to continue splitting when $\delta < \varepsilon$ because then we have already more than two regions overlapping in some places. Hence a natural final split

would be $\delta = \varepsilon$. In order to reach that by halving δ, we should start with value $\delta = 2^r\varepsilon$ for some $r \in \mathbb{N}$.

No time complexity estimations for this procedure are obtained in this thesis. However, the efficiency is investigated experimentally in section 5.4. We now turn back to the general case and see how the L_∞-ε-similar pairs are used for finding d-ε-similar pairs.

4.3 Finding Similar Pairs in General Case

Now we are able to find L_∞-ε-similar pairs, and we can carry out step 2 in Algorithm 3. Algorithm 5 (on the next page) is our method for finding d-ε-similar pairs. Note that the method for finding the L_∞-ε-similar pairs is used in steps 3–7.

Example 4 Let us try to find the d-ε-similar items in the sample dataset for $\varepsilon = 1.5$. Let the threshold value in step 5a be 4, and the pivots chosen in step 1 be x_1 and x_6. In step 2 we calculate distances from each item to all pivots. Since all the obtained values are less than 8ε but some are greater than 4ε, we have $\delta = 8\varepsilon$ in step 3. The large bucket is depicted in Figure 4.3a on page 39, the right and upper sides do not fit in the figure. In step 4 we go down to $\delta = 4\varepsilon$ and since the condition in step 5a does not hold for the large bucket of Figure 4.3a, we split it into four small buckets in step 5b (Figure 4.3b). After step 6 the four small buckets are large and the old large bucket is thrown away. From step 7 we go back to start the next iteration from step 4.

In this iteration we have $\delta = 2\varepsilon$. The number of items in the upper buckets of Figure 4.3b is zero which is below the threshold in step 5a. But since there are no items, we have no pairwise distances to calculate. The lower right bucket has three items inside and the condition in step 5a also holds. We calculate all the pairwise L_∞-distances between those items $\overline{x_4}$, $\overline{x_5}$, and $\overline{x_7}$ and find out that all pairs are L_∞-ε-similar. We store these and go on to the lower left bucket. For this bucket the condition in step 5a does not hold since all items except $\overline{x_5}$ are in this bucket. Therefore we split this bucket into four small buckets in step 5b (Figure 4.3c). Again, in step 6 we announce the four small buckets to be large and from step 7 we go back to start a new iteration.

Algorithm 5: Finding d-ε-similar pairs

1. Choose pivots $p_1, \ldots, p_q$ among all items $x_1, \ldots, x_n$.
2. For every item x_i ($1 \leqslant i \leqslant n$) calculate distances to all pivots:
$$\overline{x_i} = (d(x_i, p_1), d(x_i, p_2), \ldots, d(x_i, p_q)).$$
3. Find minimal $a \in \mathbb{N}$ such that $d(x_i, p_t) < 2^a\varepsilon$ for all $i \in \{1, \ldots, n\}$ and $t \in \{1, \ldots, q\}$. Let $\delta = 2^a\varepsilon$. According to (4.5) and our definition of δ, all items $\overline{x_1}, \ldots, \overline{x_n}$ fit into the first bucket $[0, \delta + \varepsilon) \times \ldots \times [0, \delta + \varepsilon)$. Throw away all other buckets and keep this one. We call it a *large bucket*.
4. Let $\delta := \delta/2$.
5. For each large bucket B do the following (in the first iteration there is just one large bucket):
   (a) If the number of items in B is less than some threshold value or if $\delta < \varepsilon$ then calculate all pairwise L_∞-distances that have not been calculated yet. Store the pairs that are ε-similar.
   (b) Otherwise split the bucket B into 2^q small buckets by splitting in every coordinate into two overlapping parts using ranges (4.5).
6. Discard all the large buckets and announce the current small buckets to be large.
7. If there are any buckets left, start a new iteration from step 4.
8. Now we have found all the L_∞-ε-similar pairs. We check one by one and output those pairs that are d-ε-similar.

Now we have $\delta = \varepsilon$. The left buckets of Figure 4.3c meet the condition of step 5a and the pairwise L_∞-distances are calculated. In this case this is just the distance between $\overline{x_1}$ and $\overline{x_2}$ which turns out to be greater than ε. The right buckets have both more items than the threshold value of 4 in step 5a and they are split in step 5b into altogether 8 small buckets (Figure 4.3d). In step 6 we call them large and after that start a new iteration.

This iteration is the last one because in step 4, δ becomes smaller than ε, and therefore the condition in step 5a holds for all buckets. The two uppermost and the two lowermost buckets contain just one item each and therefore there are no pairwise distances to calculate. Calculations are performed for the four central buckets with lower left corners at coordinates $(3.0, 1.5)$, $(3.0, 3.0)$, $(4.5, 1.5)$, and $(4.5, 3.0)$. The first one contains items $\overline{x_3}$ and $\overline{x_8}$ which turn out to be ε-similar in the sense of L_∞-distance. The second bucket contains items $\overline{x_3}$, $\overline{x_8}$, and $\overline{x_9}$. Distance between $\overline{x_3}$ and $\overline{x_8}$ has already been calculated, now distances between pairs $(\overline{x_3}, \overline{x_9})$ and $\overline{x_8}, \overline{x_9})$ are additionally calculated. The last pair is L_∞-ε-similar. The third bucket contains items $\overline{x_3}$, $\overline{x_4}$, $\overline{x_7}$, and $\overline{x_8}$. Here, all L_∞-ε-similar pairs have been discovered already. The last bucket has an additional item $\overline{x_9}$ but no more L_∞-ε-similar pairs emerge.

Now the iterations are finished and we proceed to step 8. The L_∞-ε-similar pairs we have found are $(\overline{x_4}$,$\overline{x_5})$, $(\overline{x_4}$,$\overline{x_7})$, $(\overline{x_5}$,$\overline{x_7})$, $(\overline{x_3}$,$\overline{x_8})$, and $(\overline{x_8}$,$\overline{x_9})$. Now we check if they are d-ε-similar. As a result we obtain that (x_4, x_5) and (x_8, x_9) are d-ε-similar.

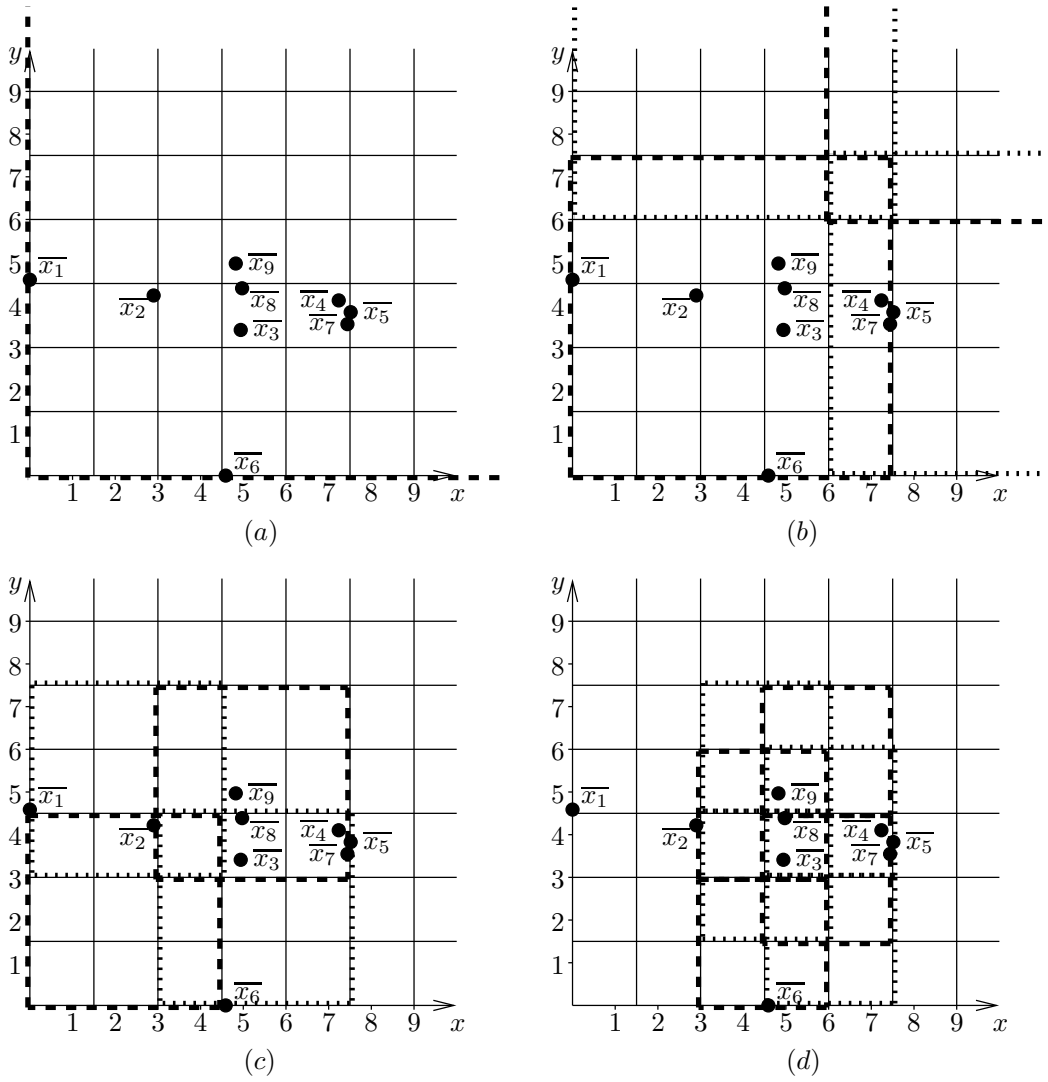


Figure 4.3: Algorithm 5 working on the sample dataset with $\varepsilon = 1.5$, pivots x_1 and x_6, and threshold value 4 in step 5a. The size of buckets in the figure is $\delta + \varepsilon$ where $\delta = 8\varepsilon$ (a), $\delta = 4\varepsilon$ (b), $\delta = 2\varepsilon$ (c), and $\delta = 1\varepsilon$ (d).

Chapter 5

Fast Approximate Hierarchical Clustering

Standard hierarchical clustering calculates all the pairwise distances and then performs the clustering, as explained in section 3.2. This is the reason why the time complexity cannot be lower than $\Omega(n^2)$. Our aim in this chapter is to get below this bound by not calculating all the distances. Instead, we try to perform hierarchical clustering by using only some of the distances. The drawback is that we may not know the exact values of distances between clusters anymore. Because for that we would need to know all pairwise distances between the two clusters for all three hierarchical clustering methods described in section 3.2. Hence, hierarchical clustering becomes approximate. We will call the new method *approximate hierarchical clustering*, whereas the old method will be referred to as *full hierarchical clustering*.

5.1 Approximate Hierarchical Clustering

In order to explain how it is possible to perform hierarchical clustering using partial information, we express hierarchical clustering in the language of graphs. First, we describe the situation we are facing after the calculation of distances.

Let G be an undirected graph with vertices $C_1, C_2, \ldots, C_n$. Vertex C_i corresponds to data item x_i and represents a singleton cluster at the beginning of hierarchical clustering. Let the edge (C_i, C_j) between two vertices C_i and C_j de-

note that we have calculated the distance $d(x_i, x_j)$. This distance value will be referred to as the *length* of the edge (C_i, C_j). The whole graph will be called the *distance graph*.

Full hierarchical clustering calculates all pairwise distances, making the distance graph complete. The clustering procedure itself can be thought of as manipulating this graph. Full hierarchical clustering joins two vertices having the shortest edge between them, calculates the distances (*i.e.* the lengths of edges) from the joined vertex, and keeps on doing it until there is only one vertex left.

Approximate hierarchical clustering will work the same way. The only difference is the calculation of distances from the new vertex. Suppose we are joining the clusters C_u and C_v. For any other cluster C_w there are three possibilities:

- There are no edges from C_w to C_u or C_v. In this case there will be no edge from C_w to the joined cluster $C_u \cup C_v$.

- The edge (C_w, C_u) is in the graph, but there is no edge (C_w, C_v) (or *vice versa*). Then the edge (C_w, C_u) will be transformed into an edge $(C_w, C_u \cup C_v)$ with exactly the same length.

- Both edges (C_w, C_u) and (C_w, C_v) exist in the graph. In this case there will be an edge $(C_w, C_u \cup C_v)$ after the joining of vertices C_u and C_v with length defined by the following formulas depending on the linkage method:
  (a) Single linkage: $d(C_w, C_u \cup C_v) = \min(d(C_w, C_u), d(C_w, C_v))$;
  (b) Complete linkage: $d(C_w, C_u \cup C_v) = \max(d(C_w, C_u), d(C_w, C_v))$;
  (c) Average linkage: $d(C_w, C_u \cup C_v) = \\ = \frac{m(C_w,C_u)\cdot d(C_w,C_u)+m(C_w,C_v)\cdot d(C_w,C_v)}{m(C_w,C_u)+m(C_w,C_v)}$,
  where $m(C, C')$ shows the number of original edges that the edge (C, C') is representing. In other words, this number of edges of the original distance graph have ended up as the edge (C, C'). This number is also stored together with the length of the edge and updated using the formula $m(C_w, C_u \cup C_v) = m(C_w, C_u) + m(C_w, C_v)$.

Intuitively, approximate hierarchical clustering calculates approximate distances between clusters. For example, in the case of average linkage it takes the average of the known distances between the clusters instead of all distances. We now give the algorithm of approximate hierarchical clustering. In the case of complete

distance graph (all pairwise distances calculated), the approximate clustering is exactly the same as the full clustering.

Algorithm 6: Approximate hierarchical clustering.

1. Find the shortest edge, let it be (C_u, C_v).
2. Join vertices C_u and C_v into one vertex $C_u \cup C_v$. Transform all edges of form (C_w, C_u) or (C_w, C_v) for any other vertex C_w into $(C_w, C_u \cup C_v)$. If there are now two edges between C_w and $C_u \cup C_v$, join them into one edge with length defined depending on the linkage method by the formulas above.
3. If there are any edges left, start a new iteration from step 1.

Example 5 Let us perform approximate hierarchical clustering with average linkage on our sample dataset. Suppose we have calculated the distances between pairs (x_1, x_3), (x_1, x_4), (x_1, x_5), (x_1, x_6), (x_1, x_7), (x_1, x_9), (x_2, x_3), (x_2, x_4), (x_2, x_5), (x_3, x_4), (x_3, x_6), (x_3, x_7), (x_3, x_8), (x_4, x_8), (x_4, x_9), (x_5, x_6), (x_7, x_8), and (x_8, x_9). We make the distance graph and start to cluster. As the first step we join the vertices C_8 and C_9, because (C_8, C_9) is the shortest edge. Let the new vertex be denoted as C_{8+9}. Vertices C_8 and C_9 have neighbours C_3, C_4, C_7, and C_1, C_4, respectively. Thus, the vertex C_{8+9} will have C_1, C_3, C_4, and C_7 as its neighbours. The lengths of the edges with C_1, C_3, and C_7 will be the same as the lengths of the edges (C_1, C_9), (C_3, C_8), and (C_7, C_8), respectively. The length of the edge (C_{8+9}, C_4) will be calculated from the lengths of (C_8, C_4) and (C_9, C_4) using the formula for average linkage given above. The result will be

$$d(C_{8+9}, C_4) = \frac{1 \cdot d(C_8, C_4) + 1 \cdot d(C_9, C_4)}{1 + 1}.$$

We mark the joining of C_8 and C_9 down on the dendrogram (see Figure 5.1a), and proceed to join the next vertices.

This time the shortest edge is (C_2, C_3). We join the two vertices and obtain a new vertex C_{2+3}. The neighbours of C_2 and C_3 are C_4, C_5, and C_1, C_4, C_6, C_7, C_{8+9}, respectively. So the neighbours of C_{2+3} will be C_1, C_4, C_5, C_6, C_7, and C_{8+9}. Again, we mark it down on the dendrogram and proceed.

Now the shortest edge is (C_{2+3}, C_6). The neighbours are C_1, C_4, C_5, C_7, C_{8+9}, and C_1, C_5, respectively. The new vertex C_{2+3+6} will have C_1, C_4, C_5, C_7, and C_{8+9} as neighbours. After five more joins there is only one vertex $C_{1+2+\ldots+9}$ left and we are done.

The dendrogram in Figure 5.1a has the joining levels (vertical lines) at locations that correspond to the approximated average linkage distances between the joined clusters. Figure 5.1d depicts the same dendrogram with joining levels shifted to the positions corresponding to the true average linkage distances between the joined clusters. The lines in this figure are crossing, indicating that we have joined clusters with longer distance first.

Figure 5.1b gives the dendrogram resulting from a different distance graph. In this case, no distances from data items x_1 or x_6 have been calculated. The result is that they remain isolated. Or, they can be joined in an arbitrary order. In Figure 5.1e the joining levels have been shifted to the true average linkage distance between the clusters based on all pairwise distances.

5.2 Fast Approximate Hierarchical Clustering

The main problem with Algorithm 6 for approximate hierarchical clustering is the efficiency. We definitely do not want to have $\Omega(n^2)$ time complexity, which would be the result if we implemented Algorithm 6 carelessly. Moreover, we would like the algorithm to run very quickly if the distance graph is sparse, *i.e.* contains a small number of edges. The complexity we will achieve is $\mathcal{O}(n \log n \log m + m)$, where m is the number of edges in the graph. For sparse graphs the prevailing term will be the first and for dense graphs the second. It is not possible to do better than $\Omega(n + m)$, because the size of input is $\Omega(m)$ (the m edges of the graph) and the size of output is $\Omega(n)$ (the dendrogram has $n - 1$ levels). Hence, we are not too far from the optimum.

We next describe the solution by listing the most important data structures, showing how to use them, and finally giving the efficient algorithm. The data structures are the following:

- A priority queue Q of all edges sorted by distance increasingly, implemented as a heap;

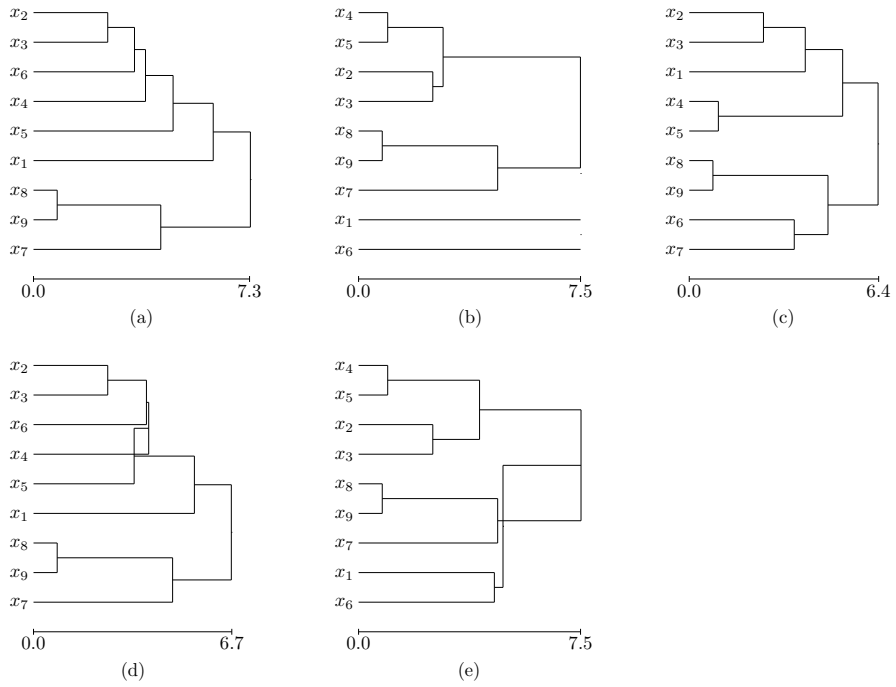


Figure 5.1: The results of two approximate (a,b) and the full (c) average linkage hierarchical clustering for the sample dataset. The dendrograms (d) and (e) have been obtained from the dendrograms (a) and (b) by shifting the join levels to the average linkage distance between clusters based on all pairwise distances.

- An index I of the queue Q, implemented as a hash table. Using I we can directly access any edge in Q by the vertices it is connecting. The index must change together with the queue;
- For each vertex C, a list N_C of neighbours of this vertex, implemented as a chain;
- For each vertex C, an index I_C for the list N_C, implemented as a hash table. Using I_C we can directly access any element in the list N_C by the number of the vertex. The index I_C is useful for checking if some element exists in the list N_C and for removing it. Of course, the index must change together with the list.

By using these structures we can perform the following operations:

- Access an edge (C_u, C_v) by the vertices C_u and C_v — we use index I to find the edge in the queue Q.
- Remove an edge (C_u, C_v) from the graph — we remove C_u from N_{C_v}, C_v from N_{C_u}, and finally the edge (C_u, C_v) from the queue Q. In order to perform the three removals we need the indices I_{C_v}, I_{C_u}, and I, respectively.
- Add an edge (C_u, C_v) to the graph — we add C_u to N_{C_v}, C_v to N_{C_u}, and finally edge (C_u, C_v) to queue Q.
- Change the length of an edge (C_u, C_v) — we update the priority queue Q.

Now we can give the efficient version of Algorithm 6 for approximate hierarchical clustering.

Algorithm 7: Fast approximate hierarchical clustering

1. Build the necessary data structures Q, I, and N_{C_i}, I_{C_i} for all $i \in \{1, \ldots, n\}$.
2. Take the edge (C_u, C_v) with smallest length from the priority queue Q.
3. Let the number of neighbours of C_u be smaller than the number of neighbours of C_v (if it is not the case, just swap C_u and C_v). In the next step we will join clusters C_u and C_v by adding the vertices of C_u to cluster C_v.
4. For each neighbour C_w of vertex C_u (using list N_{C_u}) do the following:
   (a) If C_w is a neighbour of C_v too, modify the length of edge (C_v, C_w) so that it would be the join of (old) edges (C_u, C_w) and (C_v, C_w) according to formulas on page 41. Then remove edge (C_u, C_w).
   (b) If C_w is not a neighbour of C_v, add an edge (C_v, C_w) with length equal to the length of the edge (C_u, C_w). Now remove the edge (C_u, C_w).
5. Now the vertex C_u has no neighbours left and we remove it from the graph.
6. If there is more than one vertex left, start a new iteration from step 2.

Theorem 1 *Algorithm 7 for fast approximate hierarchical clustering has time complexity $\mathcal{O}(n \log n \log m + m)$, where n is the number of vertices and m is the number of edges.*

Proof. The first step of the algorithm, *i.e.* building the data structures, can be performed in time $\mathcal{O}(n + m)$. This is because building a heap (priority queue Q) with m elements takes $\mathcal{O}(m)$ time, building n chains with a total of $2m$ elements takes $\mathcal{O}(n + m)$ time, and building $n + 1$ hash tables with a total of $3m$ elements takes also $\mathcal{O}(n + m)$ time.

The steps 2–6 are repeated for $n-1$ times. Step 2 takes $\mathcal{O}(\log m)$ time because we remove the first element of the heap. Step 3 can be performed in constant time as we can keep track of the lengths of the lists of neighbours. Steps 5 and 6 are also performed in constant time. Without taking into account the complexity of step 4,

the overall complexity is $\mathcal{O}(m+n)+(n-1)\cdot\mathcal{O}(\log m)$, which is $\mathcal{O}(n\log m+m)$. Hence, we just have to prove the limit for step 4.

Step 4a has time complexity $\mathcal{O}(\log m)$ because modifying the length of an edge means updating a value in the priority queue Q. Removing an edge also takes $\mathcal{O}(\log m)$ time. Analogically, step 4b has time complexity $\mathcal{O}(\log m)$. Now it is enough to prove that steps 4a and 4b are executed $\mathcal{O}(n\log n)$ number of times.

In the following we prove that every vertex C_w will be in the part of the neighbour in the condition of step 4 at most $\log n$ times. Since there are n vertices, it implies that steps 4a and 4b are executed $\mathcal{O}(n\log n)$ times. First note that every time the vertex C_w is in the part of the neighbour in step 4, it is in a smaller cluster that is going to be joined with a larger one. Therefore, the cluster containing C_w becomes at least twice as large. In the beginning C_w is a singleton cluster and after being in the part of the neighbour for t times, the cluster has size at least 2^t. Since the size cannot get larger than the number of vertices, we have $2^t \leqslant n$. From here it follows that $t \leqslant \log n$, which concludes the proof of the theorem.

5.3 Choosing the Distances to Calculate

Suppose we want to hierarchically cluster a dataset, but we do not have enough time to run full hierarchical clustering. We consider using approximate clustering. Of course, we would like to calculate as many distances as possible, because intuitively, more distances means a better result. At the same time we have to stay within the time limits. A suitable number of distances to calculate can be estimated using the bound given by theorem 1 and some experimental information about the program implementing approximate clustering.

Once we have decided the number of distances, we still have to choose which distances to calculate. In example 5 of the previous section we saw the results of approximate hierarchical clustering for two different distance graphs. Both of these graphs had the same number of edges. Yet, the results in Figure 5.1a and in Figure 5.1b are completely different. In some sense, the second one is better, because it has the three first joins exactly the same as the full clustering. On the other hand, the first one has the final join almost correct, except C_6 is in the wrong subtree. Thus, the same number of edges in the distance graph may yield

dendrograms of different quality. Next, we discuss three different strategies for choosing the distances to calculate. We denote the desired number of edges by m.

Strategy 1 The trivial strategy is to choose m edges randomly. Although it seems naïve, it still has some good properties. First, provided that m is not too small, all vertices have probably more neighbours than some threshold. This is good, because then we have somewhat enough information for each vertex. Second, for any two subsets of vertices the percent of calculated distances between the subsets is probably more than some threshold. This is also good, because we have enough information to estimate distances between any two clusters. Finally, this method is fast. We do not have to do any extra work when choosing the pairs.

Unfortunately, this strategy has a major drawback. It probably starts joining the clusters wrong right from the beginning. The reason is that the probability of randomly finding the shortest edge is equal to the percent of edges we have decided to choose. And we would like the method to work even with less than 1% of edges chosen.

Strategy 2 In order to overcome the downside of strategy 1, we propose the following one. Here we use our method for finding all d-ε-similar pairs. In Algorithm 5, given in the previous chapter, we found all L_∞-ε-similar pairs, but output only those of them that turned out to be d-ε-similar. Here, we do not have any reason for throwing away any calculated distances. Since we have calculated d-distances for all L_∞-ε-similar pairs, we keep them all. In other words, we modify Algorithm 5 to output all the calculated distances.

The big question here is how to choose ε, if we just know the number of distances we want to be calculated. Here random sampling can be used. The solution would be the following:

1. Calculate distances from all the data items to all the pivots.
2. Choose randomly r pairs of items, and calculate the L_∞-distances between them.
3. Sort the distances and take the value at position $\lceil m \cdot \frac{r}{n \cdot (n-1)/2} \rceil$ to be ε.

The first step has to be done anyway by the Algorithm 5 for finding L_∞-ε-similar pairs. The second step takes $\mathcal{O}(r \cdot q)$, where q is the number of pivots. Finally, the third step takes time $r \log r$, yielding total time complexity $\mathcal{O}(r \cdot q + r \log r))$.

The good thing about this strategy is that we at least start the clustering in the right way. For any two clusters that have all item-wise distances between them less than ε, we can calculate the correct distance value. Therefore, while we are joining such clusters, we are doing the same as the full clustering.

It appears that we probably do well even for some time after that. Suppose that two data items have distance between them just slightly greater than ε. Then there is still a good chance that these items are L_∞-ε-similar, because the L_∞-distance is less or equal than the d-distance. Therefore, we might still have calculated this original distance.

The downside of this strategy is that not enough distances between remote items are calculated. This problem arises in the last part of clustering where the distances between clusters are already large. Then we have poor information to correctly estimate the distances.

The other problem is that we have introduced a bias by choosing some specific pivots. For example, from any pivot distances for only d-ε-similar pairs are calculated, because for any pivot d-distances and L_∞-distances are equal.

Strategy 3 Of course, we can mix the strategies 1 and 2. We will choose half of the pairs using one strategy and the other half using the other strategy. The result should be better than one or the other strategy by itself because the strategies are compensating for the downsides of each other.

In the next section we will compare the three strategies experimentally.

5.4 Experiments

In order to evaluate approximate hierarchical clustering and to compare different strategies, we introduce a measure of quality of dendrograms. The following is by no means the only way to do it. But we will give some justification, why we use such a measure.

At each step, hierarchical clustering minimizes the distance between the clusters to be joined. It is quite natural to hope that it is also good at minimizing the sum of all steps. Since the number of steps is always the same, it just as well minimizes the average of all steps. Hence, we take the average of all joining distances to be the score of the dendrogram — lower score means better quality. Full

hierarchical clustering is a greedy algorithm for minimizing the score. Note that in the case of approximate clustering we calculate the score based on the actual distances between the clusters based on all pairs, and do not use the approximated values found by the approximate algorithm.

In the case of single linkage full hierarchical clustering, the obtained score is actually minimal. The reason is that the single linkage hierarchical clustering is actually nothing else than the Kruskal's algorithm for finding the minimal spanning tree of the graph, but with a different kind of output. Kruskal's algorithm greedily chooses the shortest edge at every step and provably finds the minimal spanning tree (Cormen *et al.* 2001). This is some kind of justification for using this quality measure.

Next we proceed to the experiments. All the experiments were performed on a computer with a 2.8 GHz Pentium 4 processor, 1 GB of RAM, and the Linux kernel 2.4. The implementation of the algorithms for finding similar pairs and performing the approximate hierarchical clustering was written in GNU C++.

The experiments were carried out on datasets 2a (the yeast's gene expression data with the Euclidean distance), 2b (the same with the Manhattan distance), and 3 (fragments of the yeast's genome). For the details about the datasets, see page 15. For each dataset, all the three linkage methods were tested. For every linkage method, the full clustering and many approximate clusterings with all three strategies were performed. When finding the similar pairs in datasets 2a and 2b, the number of pivots was 20, and the threshold of splitting was 1000. The same numbers for dataset 3 were 10 and 100, respectively. For each test case, the program was run for 10, 10 and 5 times for datasets 2a, 2b, and 3, respectively. The results of the runs were averaged and are listed in appendix A. The running time involves the distance calculations too.

Let us have a closer look at the results of the average linkage approximate hierarchical clustering on dataset 2a. The results are given in Table A.2 on page 58 and they are illustrated with three plots in Figure A.1 on page 59. First note that in Table A.2 the numbers of pairs for strategies 1, 2, and 3 in the same row are not exactly the same, but quite close. This is due to the fact that strategies 2 and 3 predict the value of ε, for which the wanted number of pairs are obtained. As the results show, the prediction is quite good.

Figure A.1a shows the relationship between the number of calculated dis-

tances and the running time of the program. As can be seen in the figure, the running time with strategy 1 is almost linear in the number of pairs. This was already predicted by theorem 1, which stated that the time complexity of approximate hierarchical clustering is $\mathcal{O}(n \log n \log m + m)$. The reason is that $\mathcal{O}(m)$ is the prevailing term for large m.

For strategy 2 the running time is up to 6 times higher, which is due to the procedure of finding similar pairs. However, the relationship is still almost linear. This is a sign of efficiency of the algorithm for finding similar pairs. Strategy 3 has the running times higher than strategy 1 and lower than strategy 2. This is as expected, because it is a mixture of the two.

Figure A.1b shows the relationship between the number of calculated distances and the obtained quality of the clustering. First, the score is decreasing when calculating more distances. This is natural, because more pairs enable to obtain a better clustering. When comparing the scores of the three strategies, the second and third strategies are clearly better than the first. For example, strategy 3 achieves the quality score 4.6 with 100000 pairs, whereas strategy 1 with about 1300000 pairs, which is 13 times more. However, the difference in the running times of the program is not so large. The times are 5 and 15 seconds, respectively. This is due to the overhead when finding similar pairs.

As can be seen in Figure A.1c, if the running time is more than 2 seconds, then the third strategy is the best. When looking at all the other figures in appendix A, about the same holds for all cases. Although, for single linkage the difference between the first and the third strategy is tiny.

Table A.1 lists the results of full hierarchical clustering. In the case of datasets 2a and 2b the clustering tool from Expression Profiler was used (Kapushesky *et al.* 2004 accepted for publication). Since this tool cannot handle textual data, the results with dataset 3 are obtained with the approximate algorithm with all pairwise distances as input. The running time is not given as this is a very inefficient way to perform full hierarchical clustering.

To conclude the results of the experiments, it makes sense to use similar pairs for approximate hierarchical clustering. We experimented on the mixture with half of the pairs similar. However, it is possible to take the similar and random pairs with some other proportions. This remains a subject of further investigation.

Chapter 6

Conclusion

There are several algorithms that can be used for clustering in metric spaces. However, we have seen that the two most commonly used clustering algorithms spend quadratic time with respect to the number of data items. This is not acceptable for some applications.

In this thesis we have developed a subquadratic clustering algorithm. It is an approximate version of the conventional agglomerative hierarchical clustering. The subquadratic time complexity is possible due to the fact that only a fraction of all pairwise distances between data items are calculated. As a result, the quality of the clustering drops.

It appeared, that it is possible to raise the quality by carefully choosing the pairs between which the distance is calculated. For that we have developed a method for finding pairs of similar items. This method can be used to find all pairs that are more similar to each other than some threshold value. The method uses the technique of pivots, which is commonly employed for nearest neighbour search.

In order to assess the quality of approximate hierarchical clustering, a quality measure of dendrograms has been proposed. The experiments on two datasets show that the quality strongly depends on the choice of pairs for which the distances are calculated. Three strategies for choosing the pairs have been tested. The best turned out to be the strategy that chooses half of the pairs randomly, and the other half using the algorithm for finding similar pairs. Here, not only the similar but all the calculated distances are taken into account.

Often some kind of time limit is set on the clustering procedure. It is desirable to obtain as good clustering as possible within this limit. If the full hierarchical clustering cannot be done within this limit, the approximate clustering with the best possible quality should be performed. For that reason we must be able to predict how long the algorithm would run for some fixed dataset and number of pairs.

The experiments showed that the approximate hierarchical algorithm together with the best strategy for choosing pairs works in almost linear time in the number of calculated distances. This gives hope that it is possible to predict the running time by considering the size of the data, the distance metric, and the number of pairs to be used. In order to get a good prediction, it might also be necessary to run the algorithm with a smaller number of pairs. Further research should be carried out in this area.

The algorithm for finding similar pairs leaves a few open questions. Firstly, several values for the number of pivots and the threshold of splitting should be tested on several datasets with several distance measures. Secondly, some kind of time complexity bound should be proved. Also, alternative ways of finding all the similar pairs with respect to the L_∞-distance should be looked for.

It may be possible to use the algorithm for finding similar pairs together with clustering algorithms other than the hierarchical. It should be considered with the k-medoids clustering as well as the data summarization methods.

Kiire klasterdamine meetrilistes ruumides

Magistritöö

Resümee

Andmete analüüsimisel on üks olulisi tehnikaid klasterdamine. Klasterdamise eesmärk on jagada andmed gruppidesse ehk *klastritesse* nii, et samasse klastrisse kuuluvad objektid on omavahel sarnasemad kui erinevatesse klastritesse kuuluvad objektid. Käesolevas töös on vaatluse all need klasterdusalgoritmid, mis on rakendatavad suvalistes meetrilistes ruumides. Sellised algoritmid eeldavad, et eelnevalt on objektidel fikseeritud sarnasusmõõt, mis on refleksiivne, sümmeetriline ja rahuldab kolmnurgavõrratust.

Töös on esiteks kirjeldatud levinud klasterdusalgoritme k-*means* ja k-*medoids* ning hierarhilist klasterdamist. Need algoritmid töötavad ruutkeerukusega ajas ning ei ole seepärast suurte andmehulkade puhul rakendatavad. Selle magistritöö eesmärk on leida meetod kiireks klasterdamiseks meetrilistes ruumides.

Käesolevaga on välja töötatud uus meetod — kiire ligikaudne hierarhiline klasterdamine. Seda on võimalik rakendada siis, kui täielik hierarhiline klasterdamine ei ole ajapiirangute tõttu teostatav. Meetodi idee seisneb selles, et erinevalt täielikust hierarhilisest klasterdamisest arvutatakse kaugused vaid osade objektipaaride vahel. Selle tulemusena klasterduse kvaliteet langeb.

Osutub, et kvaliteeti on võimalik tõsta, kui arvutada kaugusi pigem sarnaste kui erinevate objektide vahel. Selle teostamiseks on välja töötatud meetod sarnaste objektide paaride leidmiseks ilma, et peaks kõiki paarikaupa kaugusi mõõtma. Seda võimaldab kolmnurgavõrratus, mille abil saab osade objektide kohta tõestada, et nad ei saa sarnased olla.

Et kontrollida, kas sarnaste objektipaaride kasutamine ligikaudsel hierarhilisel klasterdamisel annab efekti, on defineeritud hierarhilise klasterduse kvaliteedi mõõt. Töös tehtud eksperimendid näitasid, et efekt on märgatav, vähendades kohati sama kvaliteedi saavutamiseks vajaminevat aega kuni kolm korda.

Bibliography

Breunig, M. M.; Kriegel, H.-P.; Kröger, P.; and Sander, J. 2001. Data bubbles: quality preserving performance boosting for hierarchical clustering. *SIGMOD Record (ACM Special Interest Group on Management of Data)* 30(2):79–90.

Chavez, E.; Navarro, G.; Baeza-Yates, R. A.; and Marroquin, J. L. 2001. Searching in metric spaces. *ACM Computing Surveys* 33(3):273–321.

Cormen, T. H.; Stein, C.; Rivest, R. L.; and Leiserson, C. E. 2001. *Introduction to Algorithms*. McGraw-Hill Higher Education.

Eisen, M.; Spellman, P.; Botstein, D.; and Brown, P. 1998. Cluster analysis and display of genome-wide expression patterns. In *Proceedings of National Academy of Science USA 95*, 14863–14867.

Eisen, M. 2004. Eisen lab, `http://rana.lbl.gov/index.htm`.

Elkan, C. 2003. Using the triangle inequality to accelerate k-means. In *Proceedings of the 20th International Conference on Machine Learning (ICML'03)*, 147–153.

Estivill-Castro, V. 2002. Why so many clustering algorithms: a position paper. *SIGKDD Explorations* 4(1):65–75.

Faloutsos, C., and Lin, K.-I. 1995. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In Carey, M. J., and Schneider, D. A., eds., *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, 163–174.

Ganti, V.; Ramakrishnan, R.; Gehrke, J.; Powell, A. L.; and French, J. C. 1999. Clustering large datasets in arbitrary metric spaces. In *Proc. 15th International Conference on Data Engineering (ICDE'99)*, 502–511.

Jain, A. K.; Murty, M. N.; and Flynn, P. J. 1999. Data clustering: a review. *ACM Computing Surveys* 31(3):264–323.

Kapushesky, M.; Kemmeren, P.; Culhane, A. C.; Durinck, S.; Ihmels, J.; Körner, C.; Kull, M.; Torrente, A.; Sarkans, U.; Vilo, J.; and Brazma, A. 2004 (accepted for publication). Expression Profiler: Next Generation - an online platform for analysis of microarray data. *Nucleic Acids Research*.

Legendre, L., and Legendre, P. 1998. *Numerical Ecology*. Elsevier, 2nd english edition.

Olson, C. F. 1995. Parallel algorithms for hierarchical clustering. *Parallel Computing* 21(8):1313–1325.

Vilo, J.; Kapushesky, M.; Kemmeren, P.; Sarkans, U.; and Brazma, A. 2003. Expression Profiler. In Parmigiani, G.; Garrett, E.; Irizarry, R.; and Zeger, S., eds., *The Analysis of Gene Expression Data: Methods and Software*. Springer Verlag.

Yosida, K. 1995. *Functional Analysis. Reprint of the 1980 Edition.* Springer-Verlag.

Zhou, J., and Sander, J. 2003. Data bubbles for non-vector data: speeding-up hierarchical clustering in arbitrary metric spaces. In *Proceedings of the 29th International Conference on Very Large Databases (VLDB'03).*

Appendix A

Results of the Experiments

dataset	linkage	pairs	score	time (sec)
2a	average	19347310	3.851	22.55
2a	single	19347310	3.207	22.32
2a	complete	19347310	4.209	23.12
2b	average	19347310	25.539	22.52
2b	single	19347310	21.117	22.05
2b	complete	19347310	28.036	23.08
3	average	499500	15.797	—
3	single	499500	13.318	—
3	complete	499500	17.015	—

Table A.1: Results of the full hierarchical clustering. The running times for the datasets 2a and 2b are obtained using the tool Expression Profiler.

strat-egy	pairs	score	time (sec)
1	100	6.702	0.33
1	130	6.704	0.37
1	180	6.703	0.37
1	240	6.697	0.41
1	320	6.700	0.40
1	420	6.700	0.38
1	560	6.696	0.36
1	750	6.691	0.37
1	1000	6.685	0.34
1	1300	6.663	0.37
1	1800	6.628	0.36
1	2400	6.578	0.39
1	3200	6.489	0.40
1	4200	6.346	0.41
1	5600	6.173	0.41
1	7500	6.010	0.44
1	10000	5.875	0.49
1	13000	5.778	0.52
1	18000	5.698	0.57
1	24000	5.645	0.66
1	32000	5.595	0.77
1	42000	5.553	0.88
1	56000	5.512	1.03
1	75000	5.457	1.24
1	100000	5.403	1.54
1	130000	5.342	1.88
1	180000	5.254	2.44
1	240000	5.173	3.11
1	320000	5.070	4.07
1	420000	4.991	5.19
1	560000	4.902	6.77
1	750000	4.810	8.94
1	1000000	4.722	11.87
1	1300000	4.634	15.25
1	1800000	4.537	21.05
1	2400000	4.448	28.04
1	3200000	4.364	37.28

strat-egy	pairs	score	time (sec)
2	160	6.625	0.77
2	181	6.608	0.80
2	151	6.625	0.81
2	196	6.605	0.84
2	168	6.629	0.83
2	425	6.525	0.88
2	310	6.569	0.84
2	616	6.459	0.89
2	1131	6.287	0.97
2	1273	6.243	0.96
2	1680	6.178	1.02
2	2375	5.964	1.07
2	3525	5.792	1.24
2	4119	5.760	1.23
2	5588	5.616	1.27
2	6730	5.563	1.36
2	11103	5.316	1.69
2	13747	5.208	1.80
2	18052	5.138	1.93
2	25015	5.043	2.75
2	31646	4.980	2.57
2	43961	4.953	4.04
2	55452	4.844	5.79
2	74089	4.802	7.69
2	98589	4.770	8.53
2	129531	4.712	11.81
2	178823	4.674	17.07
2	242034	4.614	22.42

strat-egy	pairs	score	time (sec)
3	256	6.610	0.76
3	243	6.613	0.80
3	286	6.604	0.81
3	321	6.595	0.85
3	428	6.574	0.87
3	375	6.621	0.83
3	533	6.592	0.83
3	517	6.640	0.82
3	846	6.553	0.87
3	1264	6.443	1.03
3	1526	6.438	0.92
3	2402	6.261	0.99
3	3108	6.118	1.04
3	4600	5.879	1.16
3	5365	5.813	1.11
3	7493	5.549	1.31
3	9484	5.396	1.36
3	12492	5.252	1.48
3	17115	5.101	1.63
3	23428	4.999	1.79
3	31519	4.892	2.04
3	41338	4.810	2.61
3	57183	4.726	3.13
3	74901	4.679	4.70
3	99682	4.614	5.62
3	129437	4.575	9.00
3	178749	4.535	8.86
3	237942	4.503	13.96
3	314275	4.463	15.89
3	417977	4.428	27.67

Table A.2: Results of the experiments with the average linkage approximate hierarchical clustering for dataset 2a.

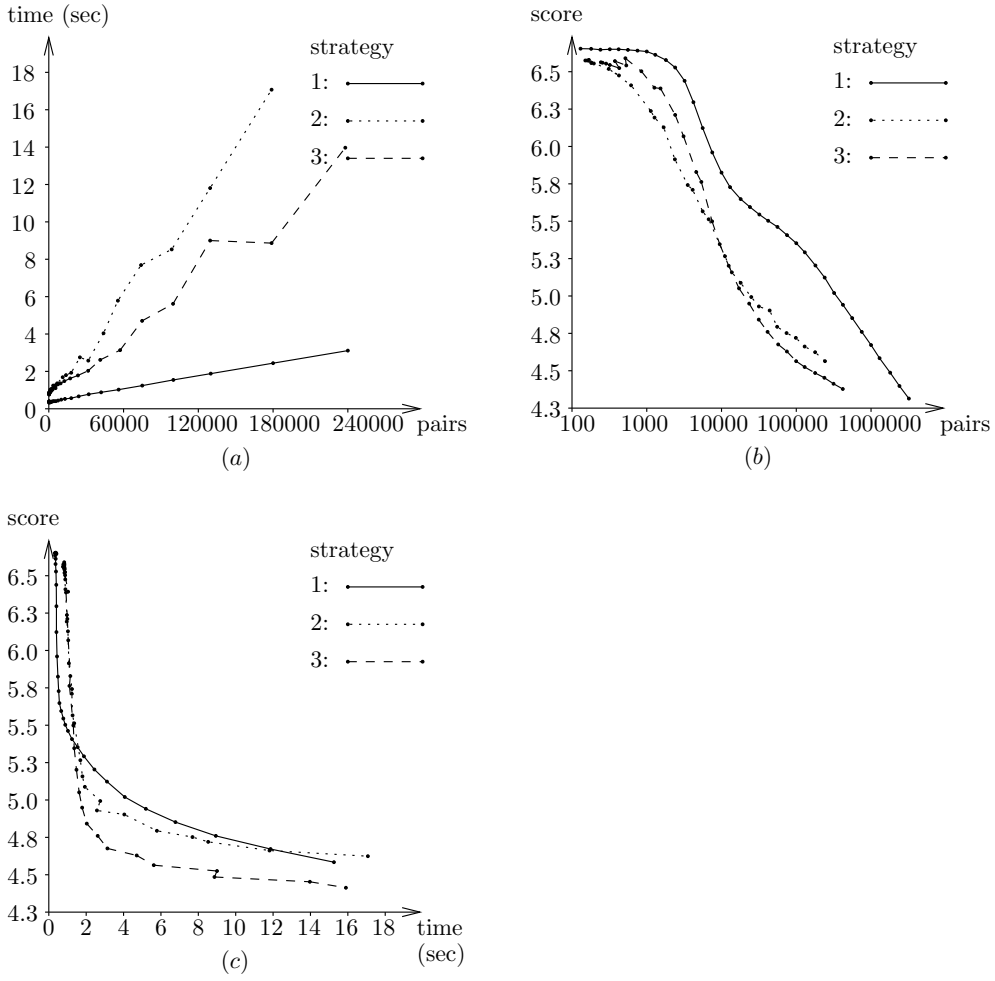


Figure A.1: The relations of the program running time, the quality score, and the number of calculated distances in the case of dataset 2a with average linkage.

strat-egy	pairs	score	time (sec)
1	100	5.377	0.34
1	130	5.370	0.33
1	180	5.375	0.33
1	240	5.376	0.33
1	320	5.376	0.33
1	420	5.369	0.36
1	560	5.373	0.33
1	750	5.364	0.36
1	1000	5.354	0.34
1	1300	5.338	0.33
1	1800	5.310	0.34
1	2400	5.260	0.37
1	3200	5.163	0.35
1	4200	4.963	0.36
1	5600	4.611	0.38
1	7500	4.237	0.42
1	10000	3.953	0.43
1	13000	3.770	0.46
1	18000	3.629	0.50
1	24000	3.547	0.59
1	32000	3.487	0.64
1	42000	3.442	0.71
1	56000	3.410	0.84
1	75000	3.382	1.00
1	100000	3.356	1.16
1	130000	3.337	1.40
1	180000	3.320	1.80
1	240000	3.306	2.32
1	320000	3.292	2.91
1	420000	3.280	3.73
1	560000	3.273	4.90
1	750000	3.263	6.56
1	1000000	3.256	8.61
1	1300000	3.248	11.58
1	1800000	3.241	15.86
1	2400000	3.235	21.92
1	3200000	3.229	29.55

strat-egy	pairs	score	time (sec)
2	154	5.357	0.72
2	228	5.354	0.78
2	294	5.342	0.79
2	156	5.354	0.78
2	169	5.358	0.73
2	258	5.341	0.78
2	301	5.337	0.77
2	503	5.303	0.84
2	1104	5.199	0.87
2	1263	5.203	0.91
2	1874	5.106	0.98
2	2542	5.035	1.09
2	3102	5.011	1.00
2	3696	4.949	1.13
2	5206	4.762	1.33
2	7505	4.649	1.36
2	9443	4.510	1.54
2	12916	4.379	1.54
2	18003	4.238	1.79
2	23012	4.139	2.13
2	31167	4.014	2.79
2	43344	3.917	3.18
2	56631	3.813	4.28
2	75614	3.699	6.75
2	97444	3.628	7.40
2	130751	3.588	10.46
2	179870	3.517	15.34
2	240252	3.452	26.99
2	320150	3.402	46.32

strat-egy	pairs	score	time (sec)
3	282	5.349	0.76
3	180	5.364	0.76
3	291	5.349	0.79
3	258	5.362	0.79
3	340	5.357	0.73
3	379	5.354	0.76
3	445	5.347	0.75
3	651	5.333	0.81
3	1048	5.291	0.80
3	1339	5.263	0.84
3	1554	5.256	0.84
3	2451	5.125	0.94
3	3367	4.965	0.94
3	3918	4.861	1.02
3	5526	4.504	1.10
3	7504	4.135	1.20
3	9836	3.817	1.25
3	12376	3.614	1.31
3	18496	3.409	1.63
3	23640	3.341	1.62
3	30708	3.306	1.80
3	41636	3.284	2.21
3	56164	3.272	2.65
3	74379	3.262	3.44
3	100663	3.251	4.95
3	128769	3.245	5.94
3	180886	3.236	7.33
3	238773	3.231	9.67
3	322201	3.225	16.83
3	419410	3.221	20.55

Table A.3: Results of the experiments with the single linkage approximate hierarchical clustering for dataset 2a.

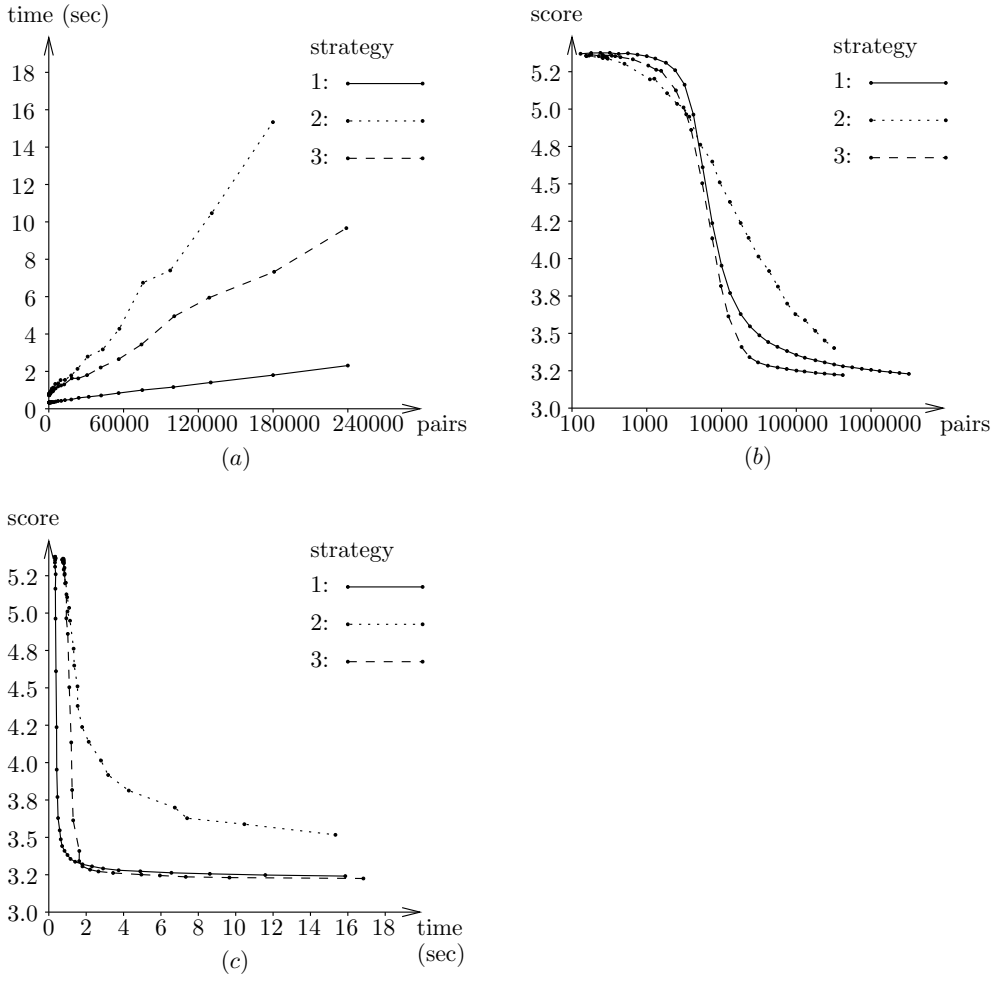


Figure A.2: The relations of the program running time, the quality score, and the number of calculated distances in the case of dataset 2a with single linkage.

strat-egy	pairs	score	time (sec)
1	100	8.897	0.35
1	130	8.905	0.34
1	180	8.916	0.34
1	240	8.909	0.36
1	320	8.901	0.36
1	420	8.899	0.36
1	560	8.901	0.35
1	750	8.889	0.34
1	1000	8.869	0.36
1	1300	8.875	0.34
1	1800	8.837	0.34
1	2400	8.764	0.37
1	3200	8.640	0.37
1	4200	8.506	0.39
1	5600	8.487	0.38
1	7500	8.440	0.41
1	10000	8.373	0.46
1	13000	8.306	0.47
1	18000	8.175	0.54
1	24000	8.058	0.62
1	32000	7.962	0.72
1	42000	7.889	0.84
1	56000	7.775	0.97
1	75000	7.620	1.19
1	100000	7.420	1.46
1	130000	7.225	1.79
1	180000	7.003	2.31
1	240000	6.823	2.98
1	320000	6.602	3.82
1	420000	6.447	4.91
1	560000	6.244	6.43
1	750000	6.087	8.49
1	1000000	5.911	11.15
1	1300000	5.764	14.43
1	1800000	5.587	19.77
1	2400000	5.442	26.32
1	3200000	5.299	35.34

strat-egy	pairs	score	time (sec)
2	185	8.811	0.80
2	187	8.809	0.76
2	184	8.811	0.77
2	195	8.802	0.78
2	124	8.830	0.77
2	363	8.730	0.81
2	241	8.770	0.79
2	568	8.620	0.84
2	863	8.496	0.90
2	1060	8.438	0.92
2	1517	8.252	0.96
2	2475	7.909	1.07
2	3023	7.847	1.15
2	4120	7.656	1.18
2	5379	7.511	1.24
2	7618	7.223	1.33
2	9816	7.057	1.47
2	12987	6.906	1.72
2	16740	6.784	1.81
2	24584	6.549	2.45
2	32769	6.423	3.10
2	42441	6.354	3.69
2	54821	6.294	5.36
2	74920	6.231	5.93
2	99196	6.159	8.85
2	128485	6.138	9.82
2	183410	6.064	17.92
2	241464	6.024	24.38
2	321192	6.041	36.95

strat-egy	pairs	score	time (sec)
3	200	8.820	0.79
3	217	8.822	0.77
3	242	8.830	0.78
3	252	8.828	0.79
3	279	8.838	0.80
3	321	8.846	0.78
3	596	8.722	0.83
3	528	8.819	0.77
3	979	8.646	0.86
3	1329	8.582	0.87
3	1658	8.499	0.88
3	2439	8.275	0.95
3	3138	8.188	0.98
3	4175	8.007	1.04
3	5604	7.890	1.09
3	7807	7.578	1.25
3	9897	7.444	1.36
3	13293	7.258	1.44
3	16965	7.045	1.52
3	24223	6.751	1.72
3	32087	6.552	2.00
3	41220	6.395	2.32
3	56511	6.164	3.50
3	74181	6.035	4.41
3	100133	5.943	4.78
3	130048	5.839	6.48
3	178775	5.763	7.64
3	239361	5.683	12.45
3	319461	5.631	14.71

Table A.4: Results of the experiments with the complete linkage approximate hierarchical clustering for dataset 2a.

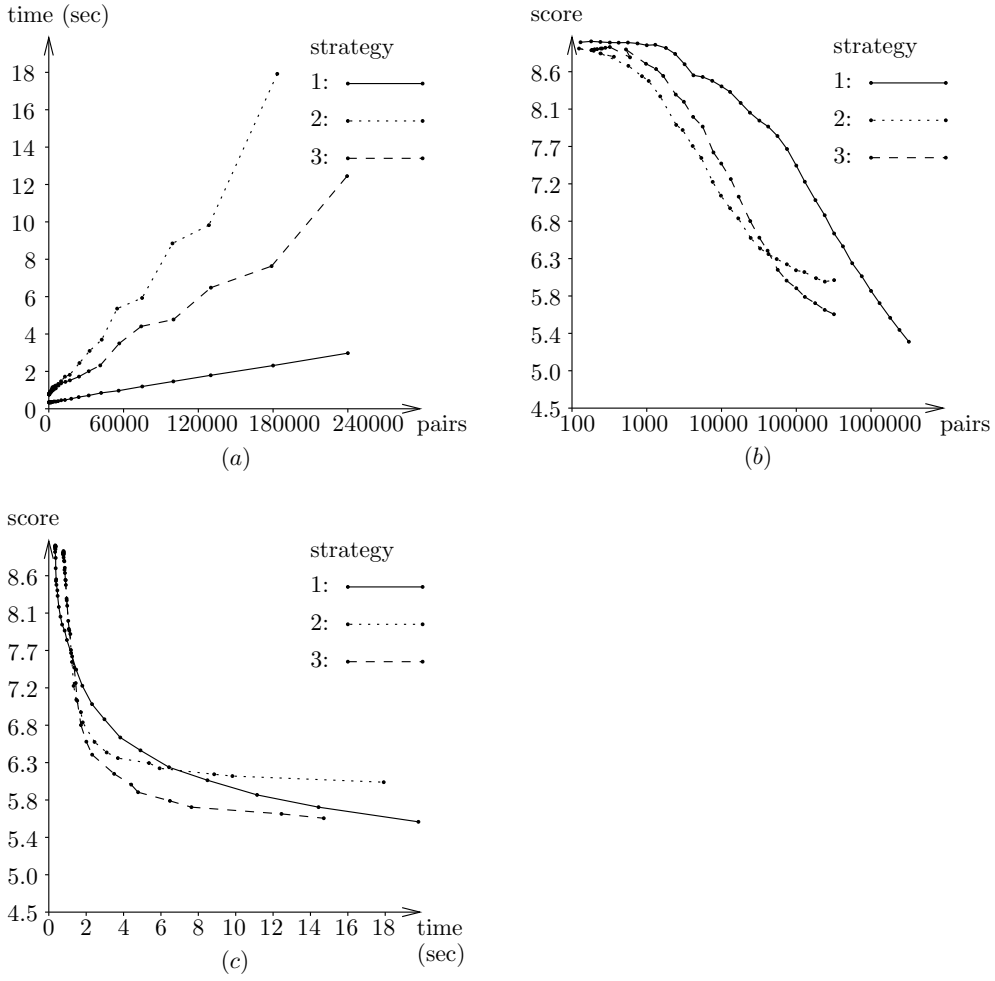


Figure A.3: The relations of the program running time, the quality score, and the number of calculated distances in the case of dataset 2a with complete linkage.

strat-egy	pairs	score	time (sec)
1	100	43.505	0.36
1	130	43.542	0.35
1	180	43.525	0.34
1	240	43.525	0.34
1	320	43.543	0.38
1	420	43.512	0.33
1	560	43.504	0.34
1	750	43.473	0.34
1	1000	43.400	0.37
1	1300	43.312	0.34
1	1800	43.132	0.35
1	2400	42.825	0.35
1	3200	42.230	0.37
1	4200	41.455	0.37
1	5600	40.383	0.39
1	7500	39.386	0.41
1	10000	38.590	0.48
1	13000	38.035	0.48
1	18000	37.513	0.60
1	24000	37.162	0.61
1	32000	36.914	0.74
1	42000	36.687	0.81
1	56000	36.411	0.97
1	75000	36.175	1.17
1	100000	35.866	1.48
1	130000	35.554	1.78
1	180000	35.055	2.31
1	240000	34.484	2.95
1	320000	33.890	3.91
1	420000	33.374	4.93
1	560000	32.730	6.46
1	750000	32.171	8.57
1	1000000	31.558	11.45
1	1300000	31.030	14.53
1	1800000	30.324	19.98
1	2400000	29.695	26.59
1	3200000	29.117	35.43

strat-egy	pairs	score	time (sec)
2	138	43.364	0.81
2	110	43.422	0.77
2	195	43.297	0.81
2	225	43.249	0.78
2	104	43.415	0.79
2	409	43.010	0.83
2	290	43.171	0.81
2	541	42.872	0.86
2	821	42.573	0.96
2	1107	42.183	0.98
2	1619	41.733	1.05
2	2280	41.065	1.14
2	3046	40.480	1.27
2	3923	40.002	1.31
2	5087	39.398	1.31
2	7934	38.251	1.66
2	9631	37.525	1.71
2	12949	36.832	2.00
2	18227	36.064	2.15
2	24011	35.163	3.82
2	32179	34.746	3.57
2	42370	34.181	5.07
2	55894	33.638	7.31
2	76126	33.014	10.92
2	99352	32.612	11.73
2	129174	32.193	14.67
2	180261	31.797	21.16

strat-egy	pairs	score	time (sec)
3	368	43.095	0.84
3	360	43.171	0.82
3	251	43.340	0.78
3	298	43.316	0.78
3	296	43.349	0.79
3	445	43.231	0.79
3	580	43.170	0.81
3	575	43.258	0.81
3	891	43.016	0.87
3	1437	42.524	0.93
3	1568	42.505	0.93
3	2462	41.624	1.01
3	3309	40.864	1.10
3	3975	40.378	1.09
3	5799	38.654	1.26
3	7557	37.537	1.34
3	10478	36.041	1.57
3	12828	35.207	1.49
3	17023	34.475	1.69
3	24248	33.496	1.95
3	31674	32.956	2.22
3	41880	32.527	3.06
3	56238	32.059	3.75
3	75146	31.666	5.50
3	99145	31.222	8.10
3	129539	30.936	7.85
3	180249	30.528	12.81
3	238976	30.267	13.38

Table A.5: Results of the experiments with the average linkage approximate hierarchical clustering for dataset 2b.

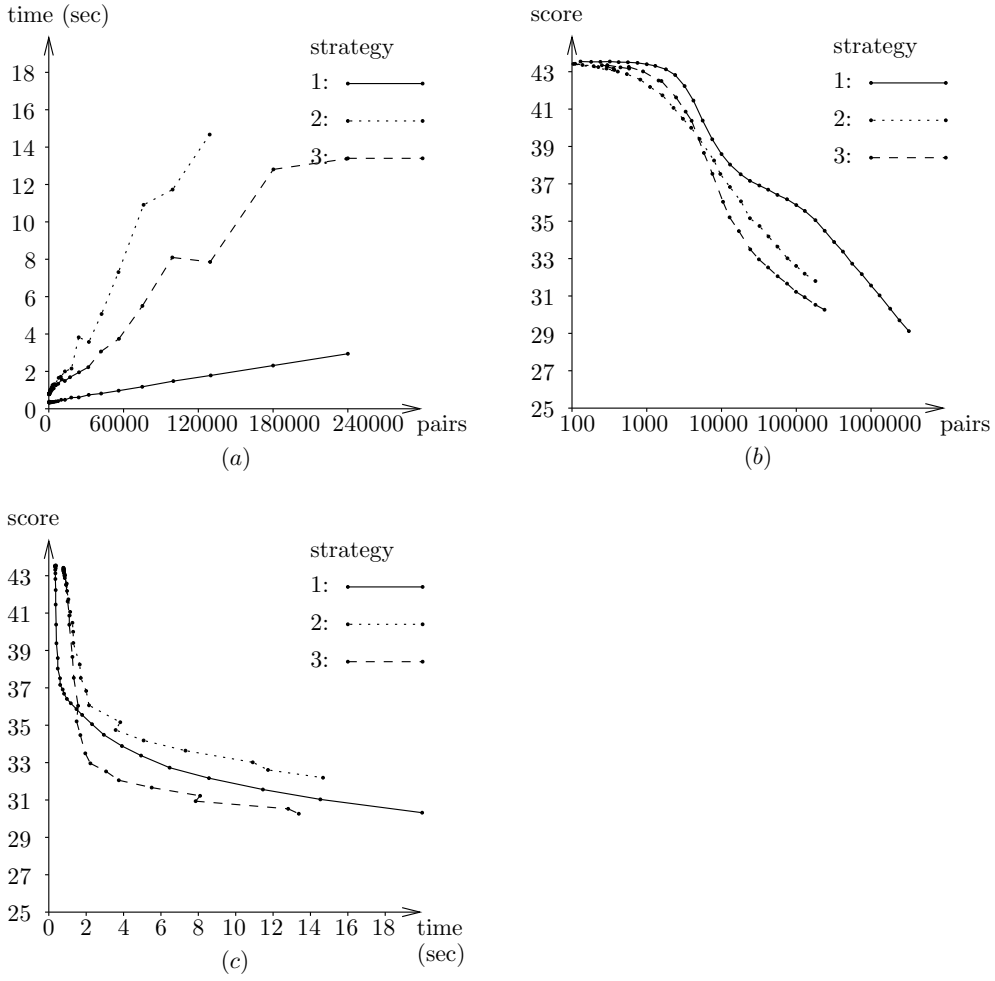


Figure A.4: The relations of the program running time, the quality score, and the number of calculated distances in the case of dataset 2b with average linkage.

strat-egy	pairs	score	time (sec)
1	100	35.180	0.34
1	130	35.184	0.36
1	180	35.197	0.35
1	240	35.185	0.35
1	320	35.129	0.34
1	420	35.109	0.36
1	560	35.142	0.36
1	750	35.083	0.34
1	1000	35.060	0.35
1	1300	34.972	0.40
1	1800	34.807	0.36
1	2400	34.479	0.36
1	3200	33.935	0.36
1	4200	32.681	0.39
1	5600	30.422	0.40
1	7500	27.954	0.40
1	10000	26.170	0.44
1	13000	24.978	0.49
1	18000	23.978	0.53
1	24000	23.420	0.59
1	32000	22.987	0.63
1	42000	22.708	0.73
1	56000	22.414	0.85
1	75000	22.236	0.99
1	100000	22.052	1.18
1	130000	21.924	1.44
1	180000	21.791	1.83
1	240000	21.700	2.29
1	320000	21.627	2.90
1	420000	21.573	3.78
1	560000	21.511	5.06
1	750000	21.470	6.57
1	1000000	21.427	9.09
1	1300000	21.391	11.53
1	1800000	21.335	16.20
1	2400000	21.300	22.19
1	3200000	21.261	30.46

strat-egy	pairs	score	time (sec)
2	133	35.205	0.77
2	415	35.073	0.85
2	209	35.173	0.82
2	243	35.137	0.80
2	128	35.207	0.76
2	415	35.130	0.85
2	476	35.079	0.83
2	462	35.090	0.86
2	929	34.811	0.94
2	1241	34.603	1.06
2	1664	34.340	1.14
2	1994	34.055	1.35
2	3326	33.267	1.26
2	4314	32.595	1.27
2	5238	32.098	1.49
2	7202	31.382	1.52
2	10362	30.269	1.74
2	12749	29.700	1.82
2	18232	28.611	2.61
2	23289	27.895	3.17
2	31953	27.027	3.52
2	42853	26.141	5.07
2	57128	25.527	7.48
2	74513	24.974	7.76
2	100728	24.425	10.82
2	131041	23.975	13.02
2	177310	23.541	16.89
2	242051	22.904	38.86

strat-egy	pairs	score	time (sec)
3	340	35.116	0.80
3	215	35.194	0.78
3	229	35.192	0.83
3	235	35.178	0.77
3	295	35.177	0.76
3	492	35.123	0.85
3	470	35.182	0.81
3	541	35.137	0.79
3	969	34.989	0.85
3	1218	34.911	0.90
3	1585	34.772	0.94
3	2496	34.114	1.03
3	3285	33.227	1.21
3	4288	32.194	1.14
3	5242	30.617	1.17
3	7382	27.841	1.27
3	9631	25.655	1.40
3	12775	23.976	1.55
3	18080	22.686	1.95
3	23712	22.104	1.96
3	32291	21.817	3.42
3	42961	21.672	3.29
3	56426	21.602	3.62
3	75311	21.522	4.69
3	101306	21.459	7.27
3	130345	21.419	7.78
3	179577	21.358	9.75
3	237087	21.302	12.40
3	320421	21.273	19.75

Table A.6: Results of the experiments with the single linkage approximate hierarchical clustering for dataset 2b.

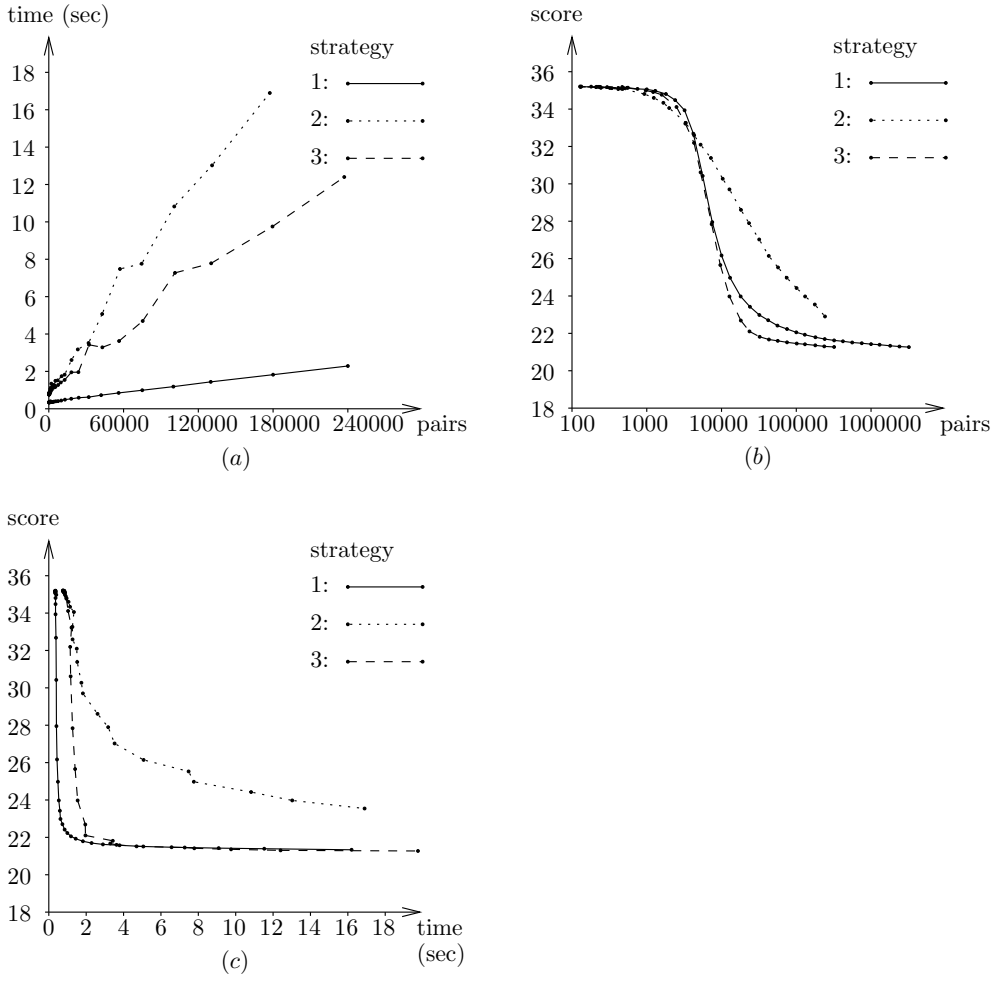


Figure A.5: The relations of the program running time, the quality score, and the number of calculated distances in the case of dataset 2b with single linkage.

strat-egy	pairs	score	time (sec)
1	100	56.194	0.36
1	130	56.131	0.37
1	180	56.141	0.36
1	240	56.055	0.36
1	320	56.149	0.37
1	420	56.170	0.45
1	560	56.067	0.35
1	750	56.133	0.34
1	1000	56.018	0.34
1	1300	56.017	0.39
1	1800	55.761	0.37
1	2400	55.390	0.37
1	3200	54.780	0.38
1	4200	54.093	0.39
1	5600	53.811	0.40
1	7500	53.733	0.41
1	10000	53.370	0.51
1	13000	52.933	0.51
1	18000	52.321	0.56
1	24000	51.750	0.64
1	32000	51.147	0.73
1	42000	50.784	0.84
1	56000	50.195	0.99
1	75000	49.537	1.21
1	100000	48.774	1.47
1	130000	47.696	1.84
1	180000	46.533	2.33
1	240000	45.233	3.01
1	320000	44.159	3.97
1	420000	43.211	4.98
1	560000	42.025	6.60
1	750000	40.897	8.62
1	1000000	39.835	11.47
1	1300000	38.787	14.64
1	1800000	37.631	20.04
1	2400000	36.677	26.62
1	3200000	35.655	36.37

strat-egy	pairs	score	time (sec)
2	130	55.830	0.75
2	121	55.947	0.79
2	153	55.797	0.79
2	203	55.781	0.80
2	134	55.789	0.80
2	387	55.416	0.87
2	413	55.390	0.85
2	546	55.040	0.86
2	1040	54.203	0.93
2	1249	53.858	1.01
2	1999	52.604	1.11
2	2160	52.333	1.16
2	3432	50.536	1.27
2	4050	50.367	1.31
2	5328	48.924	1.45
2	7150	47.858	1.57
2	9891	46.720	1.63
2	12680	45.878	1.97
2	18503	44.478	2.44
2	23760	44.013	3.35
2	31771	43.019	4.96
2	42346	42.493	5.20
2	57977	42.057	6.60
2	75583	41.518	8.71
2	99809	41.286	10.80
2	127673	40.533	14.83
2	176403	40.274	20.98

strat-egy	pairs	score	time (sec)
3	212	55.789	0.78
3	302	55.703	0.84
3	263	55.762	0.79
3	287	55.843	0.79
3	436	55.568	0.83
3	424	55.706	0.81
3	512	55.619	0.82
3	612	55.625	0.79
3	799	55.458	0.80
3	1264	54.786	0.92
3	1746	54.400	0.95
3	2367	53.750	1.03
3	3425	52.542	1.09
3	4222	52.061	1.22
3	5637	51.294	1.32
3	7113	51.030	1.33
3	10085	49.377	1.48
3	12585	48.901	1.59
3	17363	47.409	1.73
3	23856	45.594	2.06
3	31752	44.613	2.19
3	43415	42.856	2.68
3	54976	42.199	3.68
3	74733	41.030	5.74
3	100527	40.216	7.61
3	128145	39.779	8.56
3	179796	39.110	11.05
3	240352	38.506	15.32

Table A.7: Results of the experiments with the complete linkage approximate hierarchical clustering for dataset 2b.

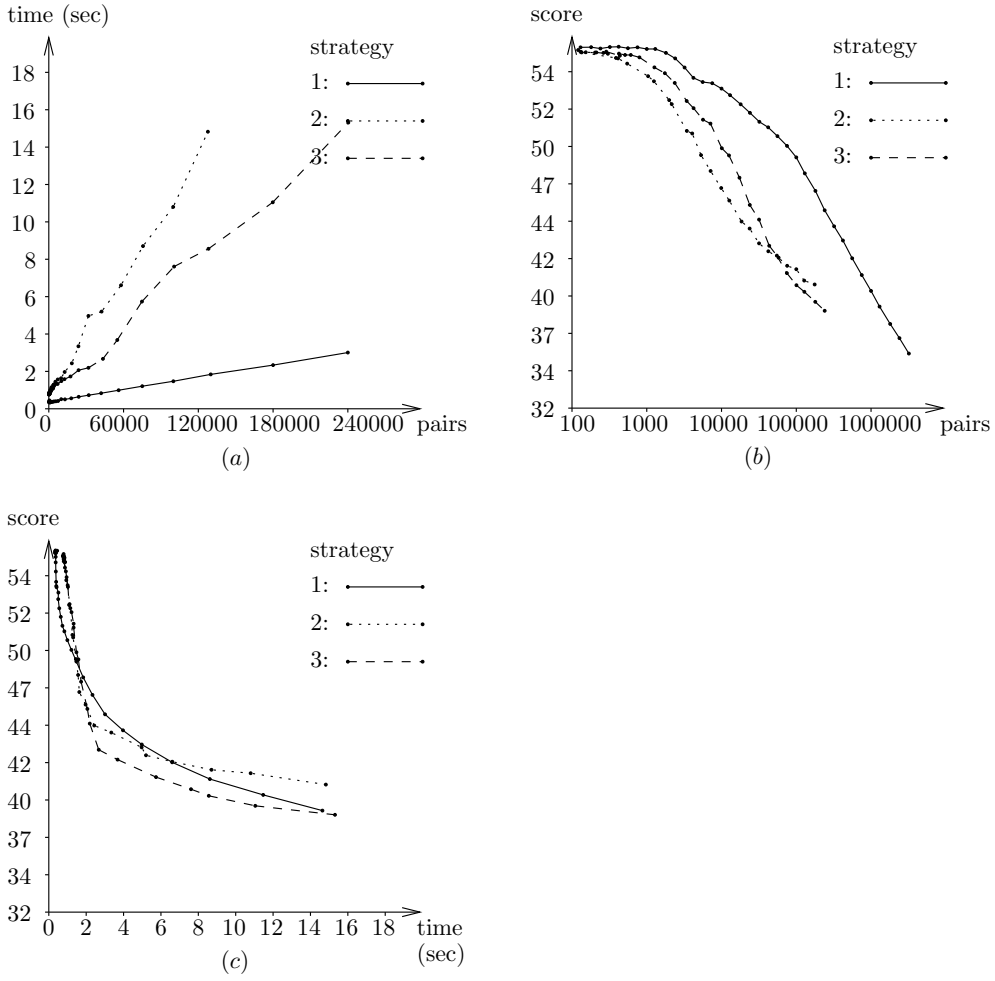


Figure A.6: The relations of the program running time, the quality score, and the number of calculated distances in the case of dataset 2b with complete linkage.

strat-egy	pairs	score	time (sec)
1	100	21.636	0.02
1	200	21.625	0.04
1	300	21.537	0.05
1	500	21.221	0.07
1	800	20.671	0.11
1	1300	20.137	0.17
1	2000	19.613	0.24
1	3200	19.267	0.38
1	5000	19.063	0.59
1	7800	18.877	0.91
1	12000	18.716	1.41
1	19000	18.496	2.22
1	30000	18.274	3.48
1	50000	17.917	5.81
1	80000	17.605	9.28
1	110000	17.376	12.71
1	160000	17.034	18.45
1	240000	16.647	27.62

strat-egy	pairs	score	time (sec)
2	236	21.082	1.29
2	8326	18.185	2.27
2	49740	17.148	7.92
2	115634	16.696	17.84
2	190501	16.356	30.32
2	262841	16.163	43.80

strat-egy	pairs	score	time (sec)
3	320	20.959	1.29
3	11092	18.050	2.60
3	65482	16.977	9.85
3	89757	16.931	12.68
3	176779	16.420	24.95
3	235996	16.288	33.66

Table A.8: Results of the experiments with the average linkage approximate hierarchical clustering for dataset 3.

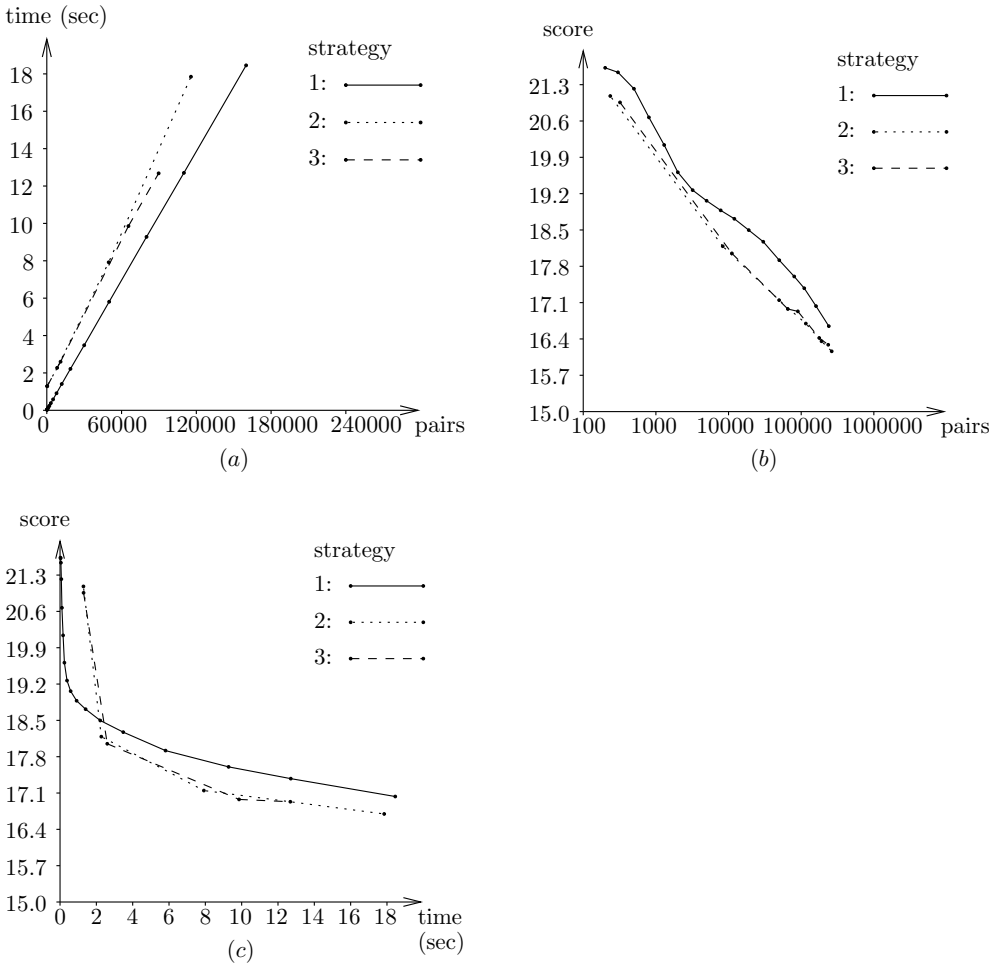


Figure A.7: The relations of the program running time, the quality score, and the number of calculated distances in the case of dataset 3 with average linkage.

strat-egy	pairs	score	time (sec)
1	100	18.461	0.02
1	200	18.382	0.04
1	300	18.321	0.05
1	500	18.073	0.07
1	800	17.175	0.11
1	1300	15.768	0.16
1	2000	14.835	0.24
1	3200	14.394	0.38
1	5000	14.122	0.58
1	7800	13.962	0.90
1	12000	13.850	1.39
1	19000	13.772	2.18
1	30000	13.646	3.44
1	50000	13.570	5.73
1	80000	13.530	9.18
1	110000	13.455	12.60
1	160000	13.392	18.36
1	240000	13.356	27.58

strat-egy	pairs	score	time (sec)
2	270	18.107	1.29
2	8592	14.555	2.31
2	53603	13.549	8.43
2	115169	13.411	17.68
2	195941	13.367	32.76
2	279027	13.356	46.79

strat-egy	pairs	score	time (sec)
3	272	18.090	1.28
3	10674	13.745	2.55
3	64191	13.407	9.60
3	94663	13.375	13.27
3	193066	13.332	27.28
3	229815	13.330	32.62

Table A.9: Results of the experiments with the single linkage approximate hierarchical clustering for dataset 3.

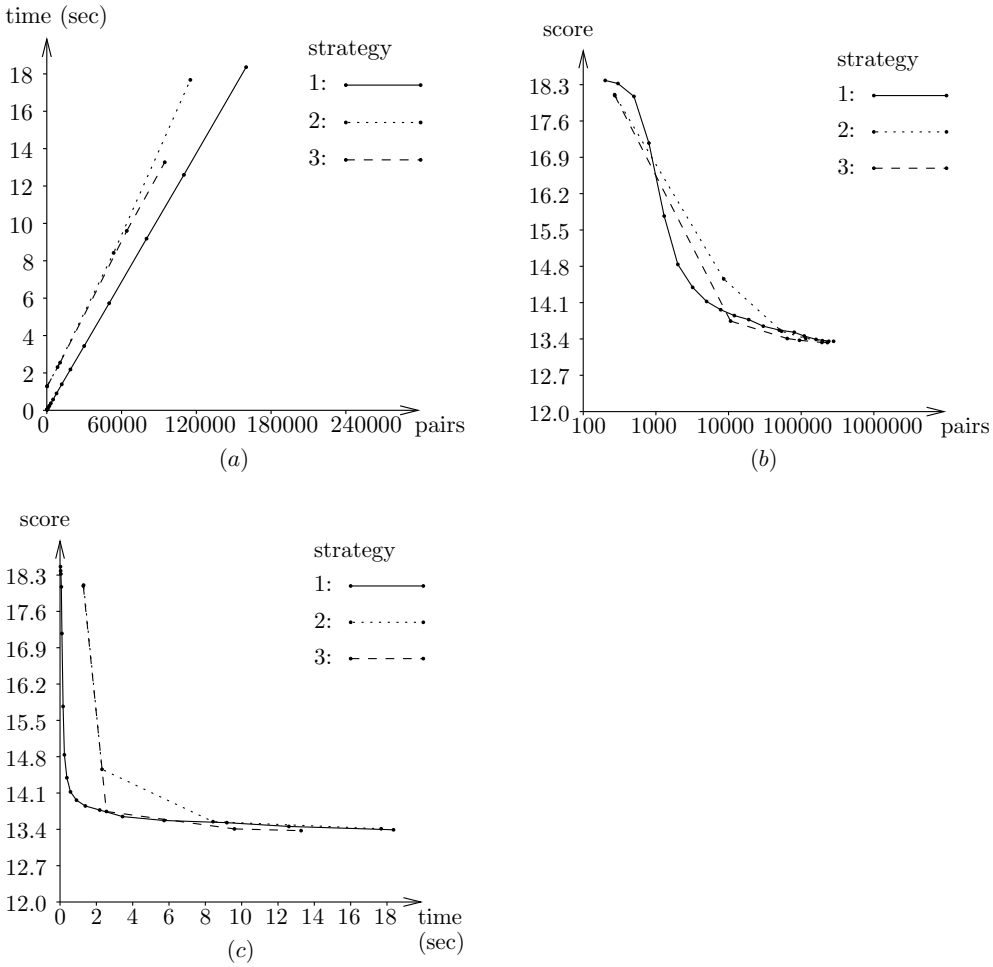


Figure A.8: The relations of the program running time, the quality score, and the number of calculated distances in the case of dataset 3 with single linkage.

strat-egy	pairs	score	time (sec)
1	100	25.206	0.02
1	200	25.186	0.04
1	300	24.985	0.05
1	500	24.709	0.07
1	800	24.168	0.10
1	1300	23.660	0.16
1	2000	23.312	0.24
1	3200	22.968	0.38
1	5000	22.674	0.61
1	7800	22.350	0.91
1	12000	21.982	1.40
1	19000	21.633	2.22
1	30000	21.192	3.49
1	50000	20.694	5.81
1	80000	20.239	9.26
1	110000	19.911	12.71
1	160000	19.484	18.47
1	240000	18.953	27.70

strat-egy	pairs	score	time (sec)
2	236	24.207	1.29
2	7823	20.582	2.21
2	51677	19.766	8.25
2	117597	19.560	18.19
2	192341	19.293	31.38
2	260020	18.779	45.25

strat-egy	pairs	score	time (sec)
3	303	24.041	1.29
3	12172	20.722	2.76
3	62656	19.769	9.44
3	93618	19.553	13.29
3	177407	19.064	24.92
3	225197	18.842	31.44

Table A.10: Results of the experiments with the complete linkage approximate hierarchical clustering for dataset 3.

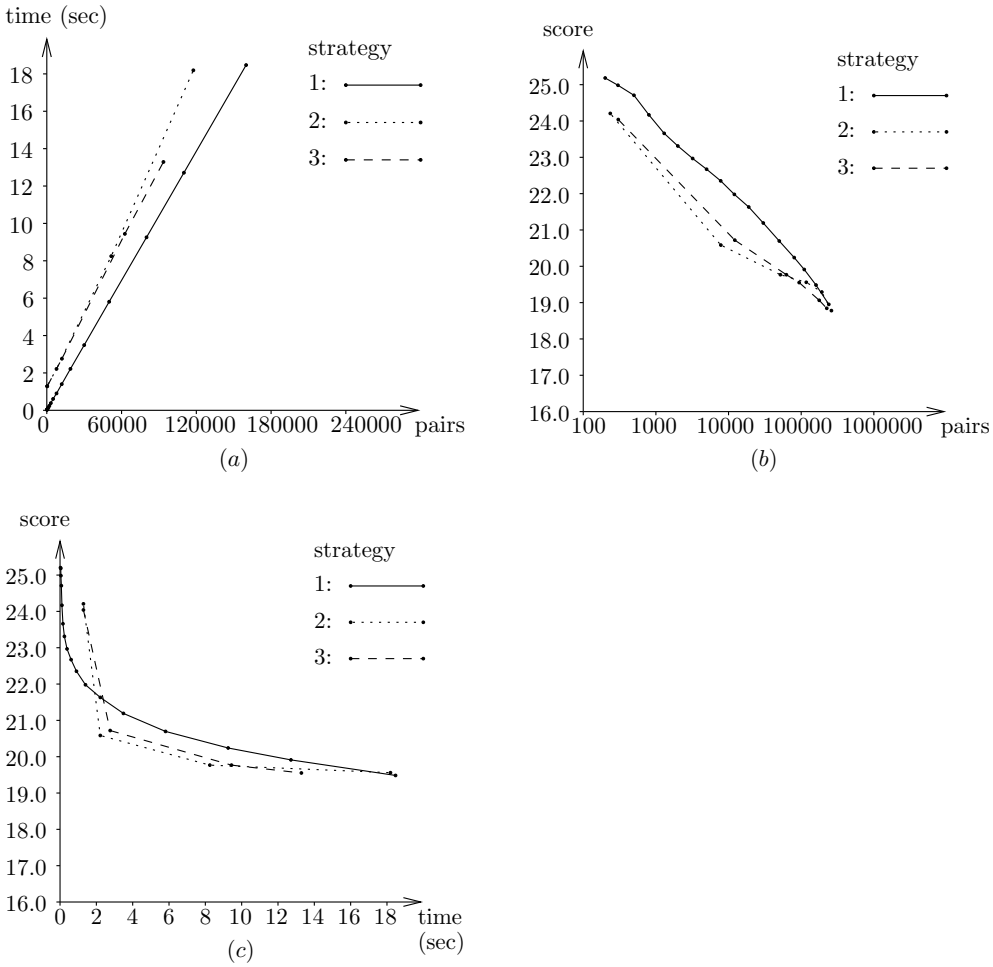


Figure A.9: The relations of the program running time, the quality score, and the number of calculated distances in the case of dataset 3 with complete linkage.