Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

A sweep-line algorithm for spatial clustering

Krista Rizman Žalik [a,b], Borut Žalik [b,*]

[a] Faculty of Natural Sciences and Mathematics, University of Maribor, Slovenia
[b] Faculty of Electrical Engineering and Computer Science, University of Maribor, Slovenia

ARTICLE INFO

Article history:
Received 4 January 2008
Received in revised form 16 May 2008
Accepted 6 June 2008
Available online 23 July 2008

Keywords:
Algorithm
Spatial dataset
Sweep-line paradigm
Clustering

ABSTRACT

This paper presents an agglomerative hierarchical clustering algorithm for spatial data. It discovers clusters of arbitrary shapes which may be nested. The algorithm uses a sweeping approach consisting of three phases: sorting is done during the preprocessing phase, determination of clusters is performed during the sweeping phase, and clusters are adjusted during the post processing phase. The properties of the algorithm are demonstrated by examples. The algorithm is also adapted to the streaming algorithm for clustering large spatial datasets.

1. Introduction

The main goal of data mining is to discover implicit information and hidden patterns in data, by grouping data objects. A given dataset of n objects $o_1, o_2, \ldots, o_n$ is clustered in k, $1 \leqslant k \leqslant n$ groups of similar objects. These groups are called clusters. Clustering algorithms can be divided into two main categories: hierarchical and partitional. Hierarchical algorithms find successive clusters using previously established clusters. Two approaches exist: agglomerative ("bottom-up") or divisive ("top-down"). The first one starts with each object as an individual cluster and merges them, if possible, into larger clusters. In contrast, divisive algorithms start with all objects in one cluster and split it/them into smaller clusters. When partitional clustering, each object is associated with the nearest cluster centre (called a centroid). The oldest, and still the most popular partitional clustering method, is k-means [1]. It is based on a so-called "winner-takes-all" principle. Many variants of the k-means approach have been developed to date [2–6]. The main problem of k-means algorithm is that the correct number of clusters k should be pre-determined by the user in advance. k-means clustering produces strange results for wrong values of k. For example, if the input parameter k is 1, the algorithm finds only one cluster, although other well-separated clusters exist. If k is greater than the correct number of clusters, some clusters are unnecessarily split. k-means also assumes that the clusters are spherical, which is not always the case, especially regarding spatial data. On the other hand, the number of clusters need not be specified when hierarchically clustering. Unfortunately, the hierarchical methods work locally, and cannot separate overlapping clusters. Many new approaches have been proposed recently for overcoming the drawbacks of existing clustering methods. Neural networks have been used to identify clusters of arbitrary shapes [7,8]. They are multi-stage, and therefore adequate training time is also necessary. A combined k-means and genetic k-means algorithm using an ant system was proposed by Tsai et al. [9]. Random sampling has been suggested for clustering large datasets [10].

Spatial clustering is a special subset of clustering. Tobler [11] stated the famous first law of geography: *Everything is related to everything else, but near things are more related than distant things*. Spatial clustering is frequently associated with graph-based methods. For example, from a maximally-connected planar graph, a minimum spanning tree is firstly determined and then some certain edges, typically the longest ones, are removed from the tree. Zahn suggests comparing the length of each arc against the average length and removing those with lengths of more than double the average [12]. Page proposed an algorithm, which removes from the graph any arc with a length greater than d [13]. A minimal spanning tree can be constructed within $O(n^2)$ complexity, while discovering the clusters is then done in $O(n)$ time. Narendra suggested the use of Voronoi diagrams for clustering [14]. A Voronoi diagram can be constructed in $O(n \log n)$, but the algorithms are difficult to implement. Kang et al. used Delaunay triangulation, a dual of the Voronoi diagram [15]. After constructing triangulation, which can be done in $O(n \log n)$, they remove those edges whose lengths are greater than d. Yujian presented a clustering algorithm based on maximal Θ-distant subtrees [16].

In this paper, we propose an efficient hierarchical spatial clustering algorithm, which determines arbitrary shaped, possibly-nested, well-defined clusters. The proposed algorithm generates

* Corresponding author.
E-mail address: zalik@uni-mb.si (B. Žalik).

0965-9978/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.advengsoft.2008.06.003

clusters in one pass instead of modifying the initial model (for example the minimal spanning tree, Voronoi diagram, or Delaunay triangulation). It can also be easily adapted to a streaming algorithm for clustering large datasets.

This paper is structured as follows. Section 2 introduces the main idea of the new algorithm. Section 3 presents the experimental results, Section 4 provides an extension to the streaming algorithm for large data sets, and Section 5 concludes the paper.

2. The sweep-line clustering algorithm

Let us observe a set of points $\boldsymbol{p}_i$, $0 \leqslant i < n$, in Fig. 1a. According to Tobler [11] the points are considered to be similar, if the distance between two points $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ is small enough. There are different distance measurements (Euclidean, Manhattan, Hausdorf, Minkowski, cost-of-transport). In our case, Euclidian distance between two points has been applied. In Fig. 1b, there are three well-separated clusters set-up from the points in Fig. 1a. For clarity, those points whose distances are small enough, are connected using line segments. As seen, the similarity is considered locally. Points $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ in Fig. 1b are assigned to one cluster. In this way, points $\boldsymbol{p}_1$ and $\boldsymbol{p}_3$ are dissimilar although they are nearer than $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$.

The proposed algorithm is based on a sweep-line paradigm. This paradigm is widely applied in computational geometry and computer graphics [17]. The basic idea is very simple. Let us imagine a line, which glides through the plane. When it hits a geometric element, it stops for a while and updates a data structure, usually considered as a sweep-line status. The problem is completely solved after the sweep-line, yet unknown before it. Characteristically, the algorithm works over three phases: initialization, sweeping and finalization. During initialization the input points are sorted according to the direction of the sweep-line's movement. The sweeping and finalization phases are described below.

2.1. The sweeping phase

The clusters are constructed during the sweeping phase. Two horizontal sweep-lines s_1 and s_2 are introduced; s_1 is in front of s_2 over distance d. Let us suppose, the sweep-line s_1 has already passed the first $\boldsymbol{p}_{i-1}$ points (see Fig. 2a). The points between both sweep-lines are linked according to their x coordinates, in order to form a polyline named *advancing front* – AF. All points, which have been passed by sweep-line s_1 are already arranged in clusters C_i according to the proximity parameter d. The membership of points to clusters is marked by the numbers in Fig. 2. In the next step, sweep-line s_1 moves to the next point ($\boldsymbol{p}_i$) and sweep-line s_2 follows it at distance d. The points, which have been swept by s_2 are removed from the AF (compare AF in Fig. 2a and b). Vertical projection of point $\boldsymbol{p}_i$ to the AF is then determined (the projection point is marked by a squared marker in Fig. 2b).

Two possibilities are then possible:

– the projection hits the AF or
– the projection misses the advancing front.

In the most common cases, the projection of $\boldsymbol{p}_i$ hits the AF between points $\boldsymbol{p}_l$ and $\boldsymbol{p}_r$. The distances $d_l = \|\boldsymbol{p}_i - \boldsymbol{p}_l\|$ and $d_r = \|\boldsymbol{p}_i - \boldsymbol{p}_r\|$ are calculated. Four possibilities can appear:

- $d_l > d$ and $d_r > d$: $\boldsymbol{p}_i$ is marked as the first element in the new cluster,
- $d_l \leqslant d$ and $d_r > d$: $\boldsymbol{p}_i$ is marked as a member of the left cluster,
- $d_l > d$ and $d_r \leqslant d$: $\boldsymbol{p}_i$ is marked as a member of the right cluster,
- $d_l \leqslant d$ and $d_r \leqslant d$: if $\boldsymbol{p}_r$ and $\boldsymbol{p}_l$ are the members of the same cluster, $\boldsymbol{p}_i$ is also assigned to this cluster, otherwise the left and right clusters are merged via $\boldsymbol{p}_i$.

The example in Fig. 2a indicates the *merging* case (clusters 2 and 3 are going to be merged). An auxiliary structure stores information about those clusters whose indices need to be unified. This information is used over the last phase of the algorithm (see next section).

If the projection misses the AF (it could also be empty), the situation is even simpler. The corresponding end-point of AF is tested as to whether it is near enough to the point $\boldsymbol{p}_i$. If it is, point $\boldsymbol{p}_i$ is marked as belonging to the cluster; otherwise it forms a new cluster.

2.2. Finalization phase

The indices of those clusters, which have to be merged, are adjusted during the finalization phase. The data structure, which stores cluster indices, consists of an array of lists. An example is shown in Fig. 3. Clusters 2, 3, and 5 have to be merged (see Fig. 3a). In each list, the record having the smallest index value is preserved, whilst others are removed. As can be seen in Fig. 3b, the cluster indices 3 and 5 are replaced by 2. In this way, all points, which had, for example, the cluster index 5, are considered as points of cluster 2.

2.3. Time complexity analysis

As described, this algorithm works in three steps: initialization, sweeping, and finalization. During initialization, the n input points are sorted in $O(n \log n)$ time. During sweeping, each point is projected onto the advancing front. The hit advancing front interval

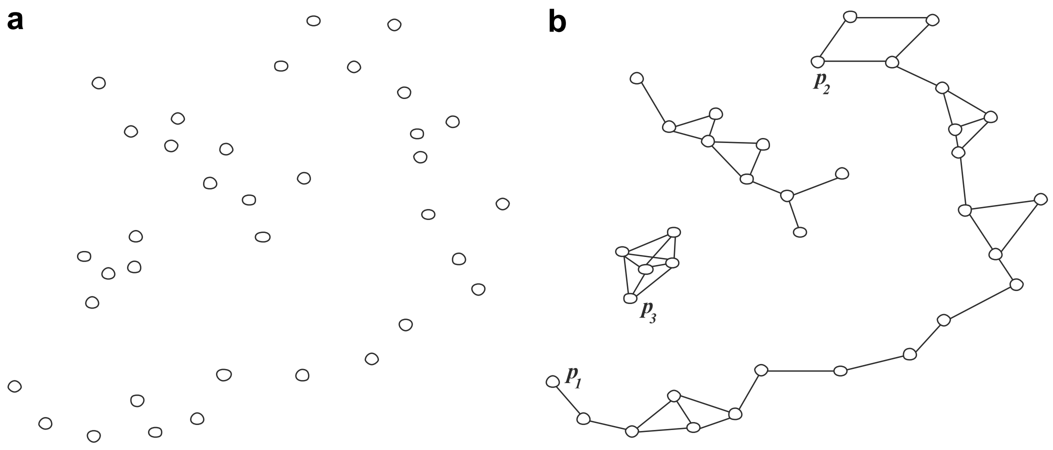

Fig. 1. Set of points (a), similarity of points in clusters (b).

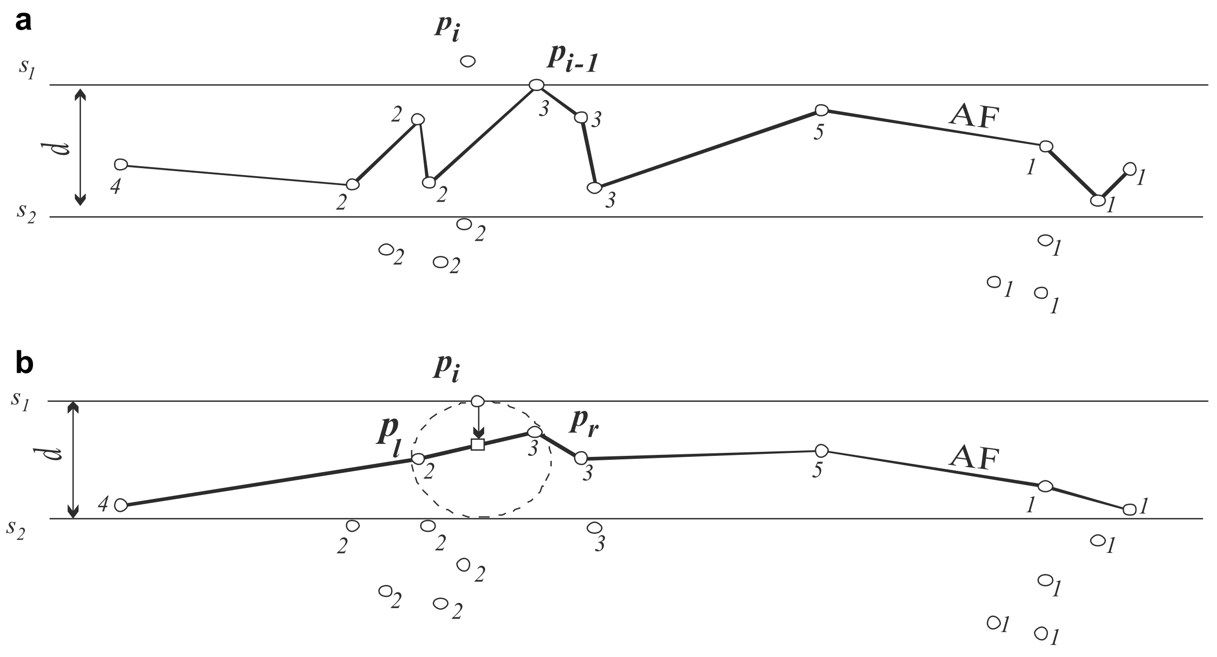

Fig. 2. Sweeping the points.

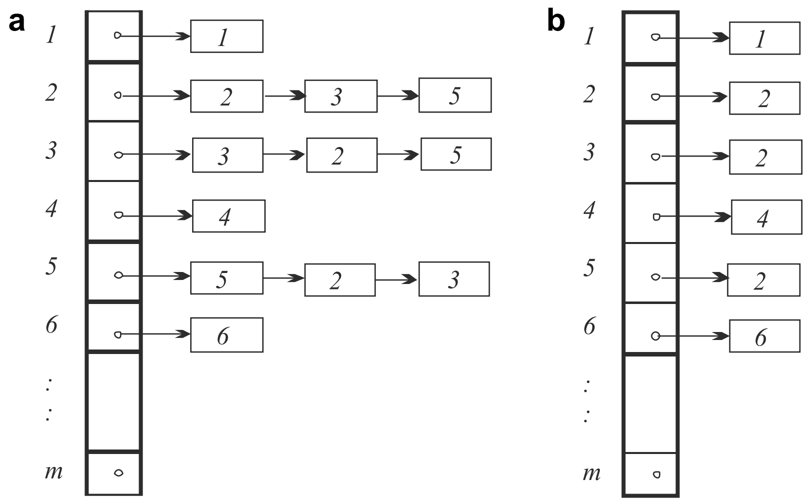

Fig. 3. Data structure before adjusting (a) and after it (b).

is determined in time O(log$m$), $m < n$, where m is the number of points in the advancing front. This step is, therefore, realized in O($n$log$m$) = O($n$log$n$), time. Adjustment of merged cluster indices is carried-out during the finalization step. The expected number of clusters for which indices should be adjusted is $c \ll n$, this step is thus terminated over a time better than O(n). The expected total time complexity of the proposed algorithm is, therefore, O($n$log$n$).

3. Experiments

The clustering performance of the proposed algorithm is demonstrated on three different datasets. The distance d and the cardinality C, defining the smallest cluster, need to be specified.

3.1. Experiment 1

The dataset used in this experiment is illustrated in Fig. 4. It can be seen that the proposed algorithm could discover nested clusters of arbitrary shape. This is an artificial dataset consisting of 753 data points obtained by plotting the image by MS Paint, after that the points were extracted from the raster image. In this case d was set in regard to the bounding rectangle of the data points. A suitable value, in this case, appears to be 1% of the bounding box's diagonal. Cardinality was set to 1. Therefore, the isolated point in the right picture remains in the result.

3.2. Experiment 2

Ten thousand uniformly distributed points were used in the second experiment. The cardinality of the clusters was set at 50. Parameter d was varied, as shown in Fig. 5. Obviously, the result largely depends on parameter d. In a real application dealing with spatial data, d has specific meaning as, for example, the walking distance to the nearest bus station or the distance to the nearest place of interest. Because of this, it can be supposed that the user provides a suitable value for d and that there is no need to set it

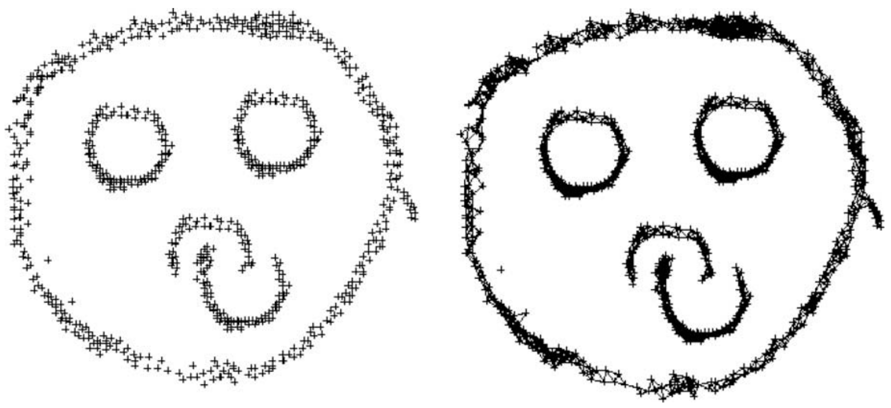

Fig. 4. Input dataset (on the left) and results (on the right).

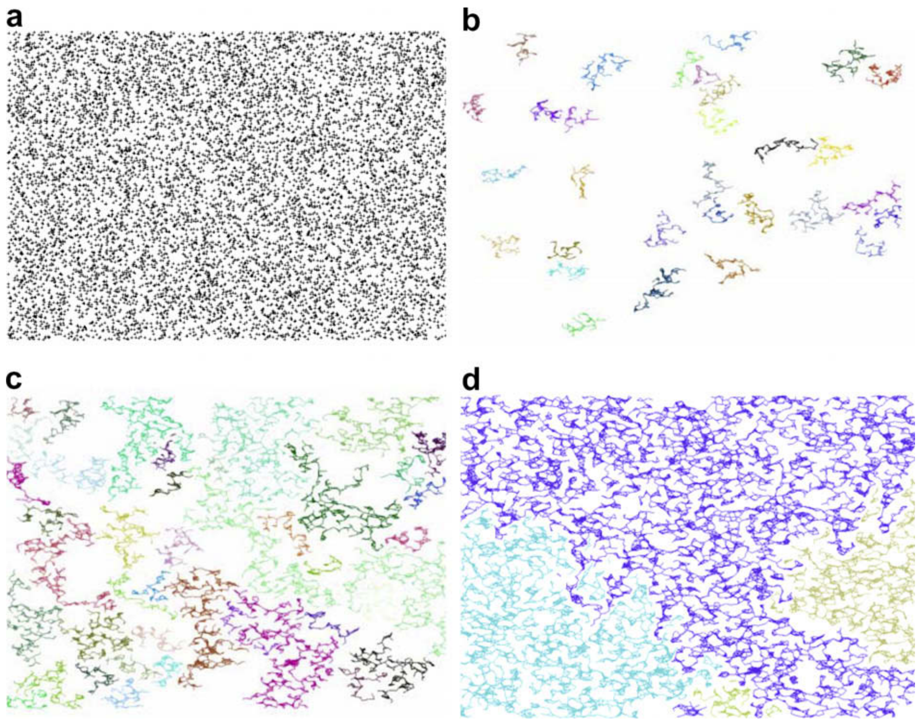

Fig. 5. Clustering of 10,000 points (a), 30 clusters obtained when $d = 0.007$ (b), 41 clusters obtained when $d = 0.0085$ (c) and 6 clusters when $d = 0.01$ (d).

using some heuristic approach (although this could also be done) [15].

3.3. Experiment 3

Fig. 6a shows 20,014 boundary stones from a GIS database. The obtained clusters with cardinality greater than 50 are shown in Fig. 6b. Parameter d was set to 25 m.

3.4. Spent CPU time

Table 1 collates the spent CPU time while clustering datasets from Figs. 5 and 6. Although the spent CPU time depends on many factors, such as the hardware used, the programming language, the compiler, the skill of the programmer, the operating system, and the input data sets, it does provide an initial impression of an algorithm's efficiency. In our case, the algorithm was implemented in C++ under Windows XP. Using Standard Template Library (STL), implementation of the algorithm consists of less than 200 lines of code. An STL map container is used for the advancing front, and according to [18] it is realized as a red–black searching tree. A personal computer with an AMD Athlon 2.2 GHz processor and 1 Gbytes of RAM was used for experiments.

It can be seen from Table 1 that the actual efficiency depends on the number of clusters. More CPU time is needed for clusters which contain larger numbers of points. Such clusters are obtained by merging many small clusters, which require unification of the used data structures.

We compared our algorithm against the algorithm using Delaunay triangulation, as described in [15]. An example of Delaunay triangulation constructed on points from Fig. 6a is shown in Fig. 7a, and the obtained clusters are shown on Fig. 7b. There exist a lot of different algorithms for constructing Delaunay triangulation, majority of them work in $O(n \log n)$ time [19]. We have used the fastest among them developed recently [20]. There are $3n - 3 - k$ edges in each triangulation, where k is the number of points in the convex hull of the input data set. In order to form the clusters, each has to be visited, which is done in linear time

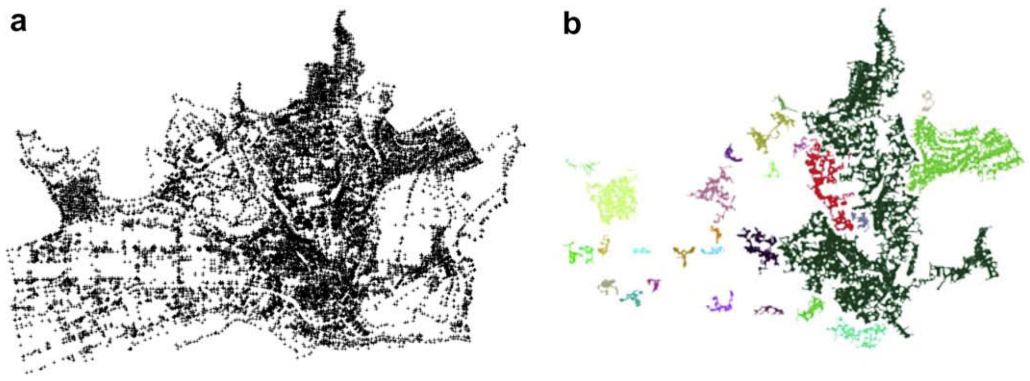

Fig. 6. Twenty thousand and fourteen points at the input (left) and 28 clusters at the output (right).

Table 1
Spent CPU time for clustering

Dataset	Spent CPU time (s)
Fig. 5b	0.062
Fig. 5c	0.102
Fig. 5d	0.262
Fig. 6	0.472

Table 2
Spent CPU time for clustering with algorithm using Delaunay triangulation

Dataset	DT time (s)	Clustering time (s)	Total (s)
Fig. 5b	0.031	0.281	0.312
Fig. 5c	0.031	0.422	0.453
Fig. 5d	0.031	0.563	0.594
Fig. 6	0.047	3.016	3.063

O(n). However, forming the clusters from the set of remaining edges is the same problem as constructing a minimal spanning tree, which can be solved in O(n^2) time. In Table 2 can be seen the experimental results obtained by the Delaunay-based clustering method. We can see, that the triangulation time does not contribute considerably to the common clustering time. Comparing the spent CPU times in Tables 1 and 2, it can be concluded that the proposed sweep-line clustering algorithm is considerably more efficient. It should be noticed that the result of the Delaunay triangulation-based method could also be much worse if a less efficient algorithm would be used (see for example the comparisons in [20]).

4. Large datasets

Spatial data grows extremely rapidly because new methods and new instruments for capturing spatial information (satellite date, LIDAR) are available today. LIDAR sensors, for example, are normally carried by aircraft. They produce small footprints (5–30 cm), commonly used for detailed local mapping of surface elevations, and large footprint (e.g., 10–25 m), collecting an average value for a greater surface area (e.g., forest mapping) [21].

These data are typically in the form of discrete points associated with additional scalar values and provide new possibilities for various operations, spatial analyses, and for research. Unfortunately, the sizes of such data drastically exceed the amount of available computer memory, which makes their processing very difficult. Such data sets are considered as large data sets and different strategies have been developed for their processing:

- The divide-and-conquer approach splits the data into small manageable units, which are solved independently and then merged into a final solution. The main problem is how to split the data to ensure that the merging of the results is done locally and that the result is not of worse quality across the cuts [22]. The divide-and-conquer approach is attractive for distributed and parallel processing.
- External memory algorithms use disks to store data that does not fit into the working memory. The application program directly controls which portion of data should be loaded or saved, with the main aim of reducing the number of I/O operations [23].

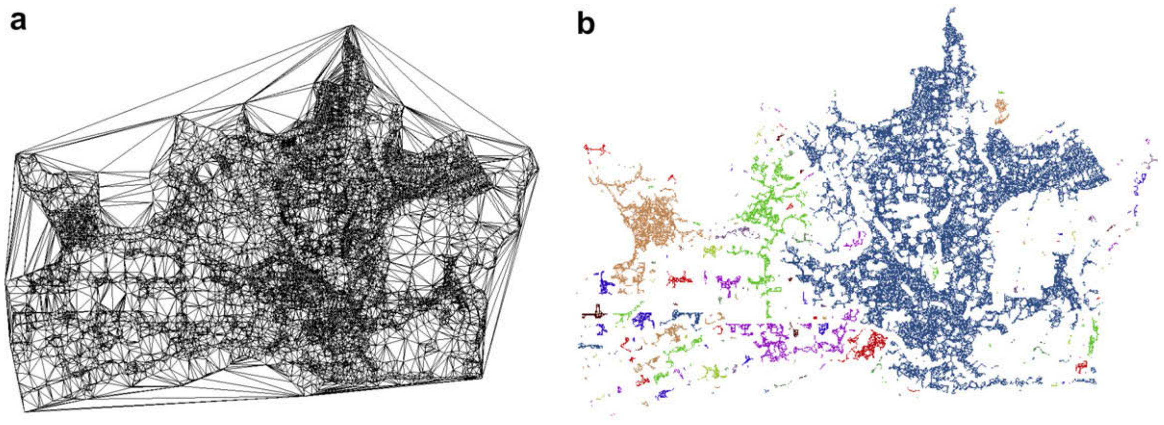

Fig. 7. A Delaunay triangulation constructed on points from Fig. 6a (left) and discovered clusters (right).

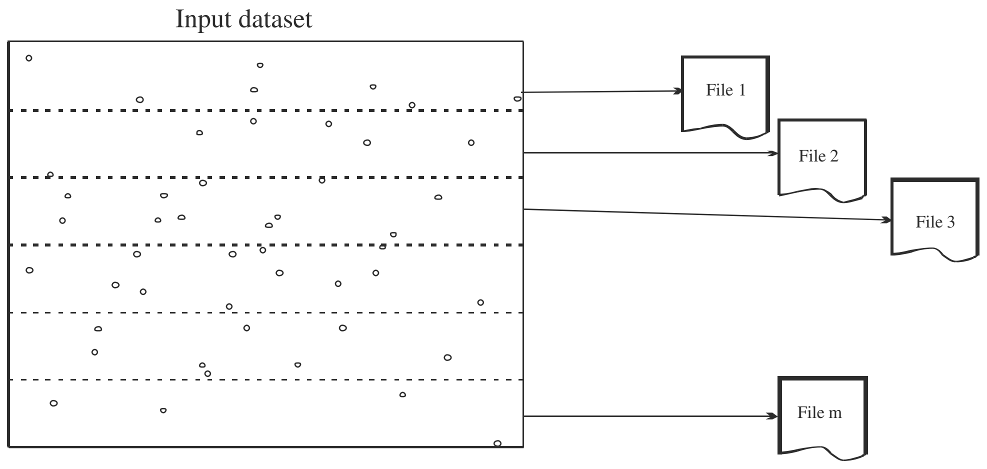

Fig. 8. Points are arranged in strips.

Table 3
Spent time of streaming clustering algorithm

Phases of the algorithm	Spent time (s)
Analysing	34.0
Splitting	94.9
Sorting & sweeping	393.7
Adjusting	122.3
Total	644.9

- Streaming algorithms use disks only for input and output as, for example, digital video [24], or data compression [25]. This approach loads a manageable amount of data into the memory, processes them, and writes the result to the output. The algorithms do not need any backward access to already-processed data.

The proposed algorithm can be easily adapted to the streaming algorithm using two preprocessing stages:

- analysing phase and
- splitting phase.

The analysing phase is executed as a single pass through the input dataset to obtain the actual number of elements n, and information about extreme values of the stored objects (typically $x_{\min}, y_{\min}, x_{\max}, y_{\max}$). If these data are already known, then the analysing phase is omitted. In the second phase, the input points are split into strips, according to the movements of the sweep-lines. The strips are divided until each of them can fit into the computer memory. Points from strips are stored in temporal files (see Fig. 8).

The data from each strip are sequentially loaded into the computer's memory, and sorted. When the dataset from the strip is ready, the sweep-line clustering algorithm is activated until all data objects in the strip are processed. In this way both sweep-lines scan the entire dataset over one streaming pass, and detect the clusters. As an experiment, we performed clustering on 100,000,000 points stored in a binary file of size 1,562,500,000 bytes. A very moderate Maxtor 6L300S0 disk was used, rotating at 7200 rpm with an IDE ATA/ATAPI NVIDIA CK804 disk driver. Table 3 shows the spent times of individual parts of the streaming algorithm.

5. Conclusions

This paper introduces a new agglomerative hierarchical clustering algorithm for spatial data. This algorithm detects well-separated, possibly-nested clusters of arbitrary shapes. The algorithm is very efficient concerning time and space complexity, which is very important during spatial data processing. The algorithm is easy to understand and implement (implementation is done in less than 200 lines of code using Standard Template Library). It generates clusters in one pass which enables it to be adapted without any modification into a streaming algorithm for clustering large spatial datasets. The algorithm itself works in E^2, but it could be extended into higher dimensions, where the sweep-line would be replaced by the sweep-hyperplane.

Acknowledgments

This work has been supported by the Slovenian Research Agency under Projects P2-0041 and J1-7218. The authors would like also to thank the reviewers for their valuable comments.

References

[1] MacQueen JB. Some methods for classification and analysis of multivariate observations. In: Proceedings of the fifth berkeley symposium on mathematical statistics and probability; 1967. p. 281–97.
[2] Jain AK, Dubes RC. Algorithms for clustering data. Englewood Cliffs, New Jersey: Prentice/Hall; 1988.
[3] Chinrungrueng C, Sequin CH. Optimal adaptive K-means algorithm with dynamic adjustment of learning rate. IEEE Trans Neural Networks 1995;6(1):157–69.
[4] Krishna K, Narasimha M. Genetic k-means algorithm. IEEE Trans Syst Man Cybernet B 1998;29(3):433–9.
[5] Jain AK, Murty MN, Flynn PJ. Data clustering: a review. ACM Comput Surv 1999;31(3):264–323.
[6] Han J, Kamber M. Data mining, concepts and techniques. Morgan Kaufmann/Elsevier; 2001.
[7] Su MC, Liu YC. A new approach to clustering data with arbitrary shapes. Pattern Recognit 2005;38(11):1887–901.
[8] Bishop CM. Neural networks for pattern recognition. Clarendon: Oxford University Press; 1997.
[9] Tsai C-F, Tsai C-W, Wu H-C, Yang T. ACODF: a novel data clustering approach for data mining in large databases. J Syst Softw 2000;73(1):133–45.
[10] Han RNJ. CLARANS: a method for clustering objects for spatial data mining. IEEE Trans Knowledge Data Eng 2000;14(5):1003–16.
[11] Tobler WR. A computer model simulation of urban growth in the Detroit region. Econ Geog 1970;46(2):234–40.
[12] Zahn CT. Graph–theoretical methods for detecting and describing gestalt clusters. IEEE Trans Comput 1971;C-20:68–86.
[13] Page RL. A minimal spanning tree clustering method. Commun ACM 1974;17(6):321–3.
[14] Narendra A. Dot pattern processing using Voronoi neighborhoods. IEEE Trans Pattern Anal Mach Intell 1982;4(3):336–43.
[15] Kang I-S, Kim TW, Li K-J. A spatial data mining method by Delaunay triangulation. In: GIS '97: proceedings of the fifth ACM international workshop on advances in geographic information systems. New York, NY, USA: ACM Press; 1997. p. 35–9.

[16] Yujian L. A clustering algorithm based on minimal Θ-distant subtrees. Pattern Recognit 2007;40(5):1425–31.
[17] O'Rourke J. Computational geometry in C. Cambridge: Cambridge University Press; 1998.
[18] Josuttis NM. The C++ standard library. Indianapolis: Addison Wesley; 1999.
[19] Su P, Drysdale RLS. A comparison of sequential Delaunay triangulation algorithms. In: Proceedings of SCG'95 (ACM symposium on computational geometry); 1995. p. 61–70.
[20] Žalik B. An efficient sweep-line Delaunay triangulation algorithm. Comput Aided Des 2005;37(10):1027–38.
[21] Weitkamp C. Lidar: range-resolved optical remote sensing of the atmosphere (Springer series in optical sciences). Berlin: Springer; 2005.
[22] Blelloch GE, Hardwick JC, Miller GL, Talmor D. Design and implementation of a practical parallel Delaunay algorithm. Algorithmica 1999;24(3–4):243–69.
[23] Vitter JS. External memory algorithms and data structures: dealing with massive data. ACM Comput Surv 2001;33(2):209–71.
[24] Li M, Claypool M, Kinicki R, Nichols J. Characteristics of streaming media stored on the web. ACM Trans Internet Technol 2005;5(4):601–26.
[25] Salomon D. Data compression: the complete reference. New York: Springer-Verlag; 2005.