ELSEVIER

Theoretical Computer Science 286 (2002) 139–149

Theoretical Computer Science

www.elsevier.com/locate/tcs

Optimal algorithms for complete linkage clustering in d dimensions

Drago Krznaric, Christos Levcopoulos *

Department of Computer Science, Lund University, Box 118, S-221 00 Lund, Sweden

Abstract

It is shown that the complete linkage clustering of n points in $\mathbb{R}^d$, where $d \geqslant 1$ is a constant, can be computed in optimal $O(n\log n)$ time and linear space, under the L_1 and L_∞-metrics. Furthermore, for every other fixed L_t-metric, it is shown that it can be approximated within an arbitrarily small constant factor in $O(n\log n)$ time and linear space.

Keywords: Complete linkage clustering; Hierarchical clustering; Multidimensional; Optimal algorithms; Approximation algorithms

1. Introduction

Given a set S of n objects, the complete linkage (c-link) method produces a hierarchy of clusters as follows. Initially, each object in S constitutes a cluster. Then, as long as there is more than one cluster, a closest pair of clusters is merged into a single cluster. The c-link distance of two clusters C and C', denoted $\delta(C, C')$, is defined as $\max\{|xy| : x \in C \text{ and } y \in C'\}$, where $|xy|$ stands for the distance (dissimilarity) between objects x and y.

Hierarchical clustering algorithms are of great importance for structuring and interpreting data in domains such as biology, medicine, and image processing. Among the different methods for producing a hierarchy of clusters, the c-link clustering is one of the most well known, and has thus been used for applications [1].

The obvious algorithm for computing the c-link clustering takes cubic time. Using priority queues, Day and Edelsbrunner [4] showed that it can be obtained in $O(n^2 \log n)$ time. A quadratic-time algorithm was described by Murtagh [12]. Later, Křivánek [9] developed a quadratic-time algorithm based on the (a, b)-tree data structure. (These

* Corresponding author. Fax: +46-46-131021.
E-mail address: christos@cs.lth.se (C. Levcopoulos).

PII: S0304-3975(01)00239-0

three algorithms also work for some other clustering methods.) A quadratic-time algorithm that uses linear space was proposed by Defays [5]; however, it only approximates the hierarchy since its output depends on a certain insertion order [2]. Parallel algorithms for the c-link clustering have also been developed [8, 11], but asymptotically the total work was still at least quadratic.

If the input does not need to explicitly contain the distance of each of the quadratic number of pairs of objects (for example, when the objects correspond to points in some metric space) then faster algorithms can be found. Recently, we showed that for n points in the Euclidean plane, the c-link clustering can be computed in $O(n \log^2 n)$ time and linear space [7]. In addition, we developed an $O(n \log n + n \log^2(1/\varepsilon))$ algorithm for constructing a c-link ε-approximation. By a *c-link ε-approximation* we mean a hierarchy that can be produced according to the c-link method, except that the following holds before each merging: if P is the pair of clusters next merged and P' is a closest pair of clusters then $\delta(P) \leqslant (1+\varepsilon)\ \delta(P')$. However, considering only the 2-dimensional case is too restrictive for real applications. Therefore, the results of this paper are especially important since, in addition to reducing the time complexity from $O(n \log^2 n)$ to $O(n \log n)$ for the L_1 and L_∞-metrics, they hold also for the multi-dimensional case. (In [7] we used some data structures that are not known to work efficiently in higher dimensions. In this paper, we dispense with those data structures by making a non-trivial use of an efficient data structure for dynamic closest point queries [3]. This gives, at the same time, a simplification of the methods in [7].)

We assume that the input set S corresponds to n distinct points in d-dimensional real space, where $d \geqslant 1$ is an integer constant, and that we are given a real constant $\varepsilon > 0$ and a fixed L_t-metric. We use $|xy|$ to denote the distance between points $x = (x_1, x_2, \ldots, x_d)$ and $y = (y_1, y_2, \ldots, y_d)$, that is,

$$|xy| = \left(\sum_{i=1}^{d} |x_i - y_i|^t \right)^{1/t}.$$

(Note that the L_∞-distance is given by $\max\{|x_i - y_i|:\ 1 \leqslant i \leqslant d\}$.) Under these assumptions, we show that a c-link ε-approximation of S can be obtained in $O(n \log n)$ time and $O(n)$ space. Moreover, for $t = 1$ and $t = \infty$, we show that the complete linkage clustering of S can be computed in optimal $O(n \log n)$ time and $O(n)$ space.

The rest of the paper is organized as follows. In the next section, we describe how c-link distances can be approximated in order to compute a c-link ε-approximation. In Section 3, we give a lemma that we use in order to keep track of relevant pairs of clusters during the merging process. Then, in Section 4 we present the algorithm, and in Section 5 we make a run time analysis. In Section 6, we note that the algorithm fits in the algebraic computation tree model, and we show that it can compute c-link distances exactly under the L_1 and L_∞-metrics, thus concluding that it is optimal up to a constant factor in these two cases. Finally, in Section 7, we make some remarks concerning other hierarchical clustering methods.

2. Approximating complete linkage distances

For each cluster we keep track of k points of the cluster where k is a constant $\geqslant 2d$. These so-called *k-extremes* are used in order to approximate c-link distances of pairs of clusters. The k-extremes of a cluster C are determined by a set $\mathscr{V}$ of k d-dimensional vectors, each vector having its tail at the origin ($\mathscr{V}$ will be defined in a moment). If $v_1, v_2, \ldots, v_k$ are the vectors in $\mathscr{V}$, then the k-extremes of C consist of points $p_1, p_2, \ldots, p_k$ such that each p_i is an extreme point of C in the direction given by vector v_i; that is, there is no point p in C such that $\langle p, v_i\rangle > \langle p_i, v_i\rangle$, where $\langle \cdot, \cdot \rangle$ denotes the inner product of two vectors. (Note that the k-extremes are not necessarily distinct and that they are not uniquely defined.) To define the set $\mathscr{V}$, let $w \geqslant 1$ be an integer and let

$$W = \{i/w\colon\ i \text{ is an integer and } -w < i < w\}.$$

Next, define V_i^+ as the set of all possible vectors $(x_1, x_2, \ldots, x_d)$ such that $x_i = 1$ and $x_j \in W$ for every $j \neq i$. The set V_i^- is defined in the same way except that it consists of vectors having -1 in position i. Then,

$$\mathscr{V} = \bigcup_{i=1}^{d} (V_i^+ \cup V_i^-).$$

So there are $k = 2d(2w-1)^{d-1}$ vectors in $\mathscr{V}$ and they are almost evenly spread around the origin.

To give a more intuitive picture of the k-extremes, let us examine the 3-dimensional case. Consider the axis-aligned cube centered at the origin such that each of its edges has length 2. Partition each side of this cube into $(2w)^2$ subsquares. Then the vectors in $\mathscr{V}$ correspond to those vertices of these subsquares that do not lie on an edge of the cube. Now, consider k planes that come from infinity and move toward a cluster, where each plane is orthogonal to a distinct vector in $\mathscr{V}$ and comes from the direction pointed by that vector. For each of these planes, select a point of the cluster such that the plane hits this point first (if more than one points are hit simultaneously, select one arbitrarily). Then, the set of all selected points, considering all k planes, constitutes the k-extremes of the cluster.

We define the *k-distance* of two clusters C and C', denoted $\delta_k(C, C')$, as the distance of two k-extremes, one from C and the other from C', that are farthest apart (according to the L_t-metric). It is not hard to realize that for any $\varepsilon > 0$ there exists a constant k, depending on ε and d, such that $\delta_k(P) \leqslant (1+\varepsilon)\delta(P)$ for any pair P of clusters. In the remainder of this paper, let k be such a constant. (For the L_1 and L_∞-metrics, it is shown in Section 6 that we can define the k-extremes so that $\delta_k(P) = \delta(P)$ for any pair P of clusters.)

When a new cluster C is created by merging a pair P of clusters, we compute the k-extremes of C by selecting points among the k-extremes of the clusters of P. More precisely, for each of the k directions, we compare the k-extreme for that direction of

one of the clusters of P with the k-extreme for that direction of the other cluster of P, and select one which is most extreme in that direction.

3. Representing clusters

For each cluster we have a what we call *leader*, which is simply a k-extreme of the cluster (we could actually choose any point of the cluster). All leaders are stored in a data structure for dynamic closest pair queries. Henceforth, that data structure will be referred to as the *DCP-structure*. The DCP-structure supports the following three operations: (i) find a closest pair of points, (ii) delete a point, and (iii) insert a new point. We assume that the DCP-structure uses linear space and makes it possible to carry out each of these three operations in logarithmic worst-case time (although logarithmic amortized time per operation would suffice), and that it fits in the algebraic decision tree model of computation. These requirements can be met by using the data structure of Bespamyatnikh [3].

Initially, each point of the input is inserted into the DCP-structure. Then, each time a new cluster C is created by merging a pair P of clusters, the leader for one of the clusters of P is deleted from the DCP-structure, and the leader for the other cluster of P becomes the leader for C.

As it is described in the next section, our algorithm works in phases where each phase uses the DCP-structure to find relevant pairs of clusters. The following lemma states an upper bound for how fast these pairs can be found.

Lemma 3.1. *Let S be a set of n points in $\mathbb{R}^d$, let l be a positive real, and let κ be an integer such that for each point of S there are at most $\kappa-1$ other points in S within distance less than l from it (distances are according to some arbitrary but fixed L_t-metric, and $d \geqslant 1$ is an integer constant). Then, if the points of S are stored in the DCP-structure, the set $\{(x,y)$: $|xy|<l$ and $x,y \in S\}$ can be found in $O(\kappa^d \eta \log n)$ time, where η denotes the cardinality of the set $\{(x,y)$: $|xy|<3l$ and $x,y \in S\}$.*

Proof. For each point x we have a set S_x where $S_x = \{x\}$ initially. First we repeat the following four steps until the condition at Step 2 holds:

1. Find a closest pair (x,y) of points in the DCP-structure.
2. If $|xy| > l/(\#S_x + \#S_y)$ then halt.
3. Let S_x equal $S_x \cup S_y$.
4. Delete y from the DCP-structure.

After this procedure, let l' be the distance of a closest pair of points in the DCP-structure, and consider an arbitrary point x in the DCP-structure. If p_1 and p_m are any two points in S_x, then there must exist points $p_1, p_2, \ldots, p_m$ in S_x such that $|p_i p_{i+1}| \leqslant l/\#S_x$ for each $i = 1,2,\ldots,m-1$. Hence, $|p_1 p_m| < l$. This means that there are at most κ points in S_x, and that $l' > l/(2\kappa)$. Clearly, the above procedure takes $O(\eta \log n)$ time.

Next, as long as the distance of a closest pair of points in the DCP-structure is $<3l$, we repeat the following five steps:

(i) Find a closest pair (x, y) of points in the DCP-structure.
(ii) Find every point z in the DCP-structure such that $|xz|<3l$.
(iii) For each z found at Step (ii), output every (p_x, p_z) such that $p_x \in S_x$, $p_z \in S_z$, and $|p_x p_z|<l$.
(iv) Output each pair of points in S_x.
(v) Delete x from the DCP-structure.

This procedure outputs the set $\{(x, y)$: $|xy|<l$ and $x, y \in S\}$. Therefore, it remains only to show that it takes $O(\kappa^d \eta \log n)$ time. To do this, we need the following observation.

Observation 3.2. *One iteration of Step* (ii) *takes* $O(\kappa^d \log n)$ *time.*

Proof. Let h be the d-dimensional axis-aligned cube centered at x such that each of its edges has length $6l$. Clearly, every point z that we are looking for lies in h. To find the points in h, partition h into $(12ld/l')^d$ subcubes with edges of length $l'/(2d)$. Note that the L_t-diameter of a subcube is at most equal to its L_1-diameter, which is equal to $l'/2$.

Now, let h_i be one of the subcubes, let c_i be the point at which h_i is centered, and suppose that there is a point z of the DCP-structure that lies in h_i. Then, if we insert c_i into the DCP-structure, (c_i, z) becomes the closest pair of points in the DCP-structure. This is because $|c_i z|<l'/2$ whereas $|zz'|\geqslant l'$ for every $z' \neq c_i$ in the DCP-structure. Hence, we can find all points in h by using one insertion, query, and deletion per subcube, which takes total $O((24d\kappa)^d \log n)$ time (recall that $l'>l/(2\kappa)$ so the number of subcubes is $<(24d\kappa)^d$). □

In Step (iii), we can compute the distance from each point in S_x to each point in S_z for every z found at Step (ii). But, if we in this way compute the distance from a point in S_x to a point in S_z, then that point in S_z must be within distance $<4l$ from x. By partitioning the cube with edges of length $8l$ and centered at x into $(8d)^d$ subcubes, each subcube having L_1-diameter l, we realize that for each point in S_x we compute at most $\kappa(8d)^d$ distances. Consequently, one iteration of Step (iii) takes $O(\kappa^2)$ time. We can now conclude that one iteration of Steps (i) through (v) takes $O(\kappa^d \log n)$ time, and since, we do at most η iterations, the total time used by the algorithm is $O(\kappa^d \eta \log n)$. □

4. The algorithm

Let S be a set of n points in $\mathbb{R}^d$. We compute a c-link ε-approximation of S in a sequence $p_1, p_2, \ldots$ of phases. The objective of phase p_i is to merge every pair P

of clusters such that $l_i \leqslant \delta_k(P) < 2l_i$, where the parameter l_i is defined as follows. At phase p_1, l_1 equals the distance of a closest pair of points in S. For $i > 1$, let l be the distance of a closest pair of leaders in the DCP-structure immediately after phase p_{i-1}. So $\delta_k(P) \geqslant l$ for every pair P of clusters. Therefore, if $l > 2l_{i-1}$ then we set l_i to l, otherwise, we set l_i to $2l_{i-1}$.

Using a priority queue and Lemma 3.1, phase p_i is rather straightforward to compute. First, we do the following two operations:

(a) Find each pair of clusters such that the distance of its corresponding pair of leaders is $< 2l_i$.
(b) Insert each such pair of clusters into a priority queue, initially empty, according to the k-distance of the pair.

Then, as long as the k-distance of a closest (according to the k-distance) pair of clusters in the priority queue is $< 2l_i$, repeat the following three steps:

1. Remove a closest pair P of clusters from the priority queue.
2. Merge P into a single cluster C_P.
3. For each pair (C, C') in the priority queue such that C' is a cluster of P do the following:
   - Remove (C, C') from the priority queue.
   - If $\delta_k(C, C_P) < 2l_i$, then insert (C, C_P) into the priority queue.

5. Run time analysis

As our algorithm works, at phases $p_1, p_2, \ldots, p_{i-1}$ we do not merge a pair P of clusters for which $\delta_k(P) \geqslant 2l_{i-1}$. Hence, at the beginning of p_i, there must be at least one pair of clusters of k-distance $\leqslant 3l_i$. This means that if no merging occurs during p_i, then at least one merging must occur at p_{i+1}. Thus, the total number of phases is no more than $2n - 2$ (the total number of mergings is $n - 1$).

To proceed with the analysis we need a couple of notations. Let $\mathscr{P}_i$ be the set consisting of every pair of clusters such that the pair exists at some time during phase p_i and the distance of its corresponding pair of leaders is $< 6l_i$. Further, let η_i denote the cardinality of $\mathscr{P}_i$. Finally, let κ be the smallest integer such that the following holds at any time during any phase p_i: for any leader there are at most $\kappa - 1$ other leaders within distance $< 6l_i$ from it. (In Lemma 5.1 below we show that κ is never greater than some constant.)

By Lemma 3.1, the operation (a) in Section 4 takes $O(\kappa^d \eta_i \log n)$ time. Clearly, (b) takes $O(\eta_i \log \eta_i)$ time. Observe that only a pair in $\mathscr{P}_i$ may be considered at Step 3, and that each pair in $\mathscr{P}_i$ is considered at most once at Step 3. So, the total time for Step 3 is $O(\eta_i \log \eta_i)$. Next, each time we iterate Steps 1 through 3, a pair of clusters is merged into a single cluster. Consequently, the total time for Steps 1 and 2 is $O(m \log \eta_i)$, where m denotes the total number of mergings

performed during p_i. We can thus conclude that the total time used by phase p_i is $O(\kappa^d \eta_i \log n + \eta_i \log \eta_i + m \log \eta_i)$.

Now, consider a pair P in $\mathscr{P}_i$. It holds that $\delta_k(P) < 10 l_i$. Therefore, one of the clusters of P must participate in a merging at one of the phases p_i, p_{i+1}, or p_{i+2}. Let us associate each pair in $\mathscr{P}_i$ with the first merging in which one of the clusters of the pair participate. If we do this for each set $\mathscr{P}_i$, we associate at most $3(\kappa - 1)$ pairs to each merging. Hence, the total sum of the η_i's is $O(\kappa n)$, which implies that the total time used by the algorithm is $O(\kappa^{d+1} n \log(\kappa n))$.

From the following lemma we can conclude that our algorithm actually runs in $O(n \log n)$ time. (A special case of this lemma handling only the Euclidean plane was stated but not proved in [7].)

Lemma 5.1. *There exists a constant* (*depending on d and k*) *greater than* κ.

Proof. Throughout the proof, by a *cube* we mean a d-dimensional axis-aligned cube. The proof is by induction on i. The induction hypothesis is as follows: at the beginning of phase p_i (before any clusters are merged) any cube having edges of length l_i contains at most $\lambda(d+1)^d$ leaders, where $\lambda \geqslant 1$ is a constant that we specify later. Recall, at the beginning of p_1, each leader corresponds to a single point in the input point set S, and l_1 equals the distance of a closest pair of points in S. It is not hard to show that any cube having edges of length l_1 contains at most $(d+1)^d$ points of S. Thus the statement is true for $i = 1$. Let h be an arbitrary cube with edges of length l_{i+1}. To complete the proof it suffices to show that h contains at most $\lambda(d+1)^d$ leaders at the beginning of phase p_{i+1}.

From the definition of the parameter l_i it follows that if $l_{i+1} \neq 2 l_i$ then, at the beginning of p_{i+1}, the distance of a closest pair of leaders equals l_{i+1}. So, similarly as for $i = 1$, h contains at most $(d+1)^d$ leaders in this case. Therefore, in the continuation we can (and will) assume that $l_{i+1} = 2 l_i$.

From the set of clusters that exist at the beginning of phase p_i we extract two subsets H and H' as follows. The set H consists of every cluster that has a k-extreme in h. The set H' consists of every cluster that, at the end of phase p_i, is included in a cluster containing a cluster of H. Note that $H \subseteq H'$ and that the clusters in H' are merged only with each other during phase p_i. We aim to show that sufficiently many mergings of clusters in H' occur during phase p_i. But first we need some more definitions.

Let h' be the cube concentric with h and having edges of length $6 l_i$. So each cluster in H' has all of its k-extremes in h'. Define the *triggers* as the r^d cubes that partition h' into subcubes with edges of length $6 l_i / r$, where $r = 6d(1 + (d+1)\log_2 6)$. Now, consider a cluster with all its k-extremes in h'. For each k-extreme of the cluster we select the trigger in which it is contained, thus selecting k triggers $t_1, t_2, \ldots, t_k$. We say that the cluster is of *type* τ, where τ is the unordered k-tuple $(t_1, t_2, \ldots, t_k)$. We are now in position to set the constant λ, namely, λ equals the maximum number of

distinct types of clusters. By standard combinatorics,

$$\lambda = \binom{r^d + k - 1}{k}.$$

Next, define the *k-diameter* of a cluster as the distance of its two k-extremes that are farthest apart. So, if C and C' are any two clusters produced by our algorithm, the k-diameter of $C \cup C'$ equals $\delta_k(C, C')$.

Observation 5.2. *Let C and C' be two clusters with all their k-extremes in h' such that each of them is of type τ and has k-diameter less than l. Then the k-diameter of $C \cup C'$ is less than $l + 6l_i d/r$.*

Proof. Since C and C' are of the same type, there are k triggers (not necessarily distinct) such that each of them contains two k-extremes, one from C and the other from C'. Let p and p' be any two k-extremes of $C \cup C'$ such that the distance between them is maximized. So the k-diameter of $C \cup C'$ equals $|pp'|$. We can assume that p is a k-extreme of C and p' is a k-extreme of C', because $|pp'|$ would otherwise be $< l$. Let t_p be the trigger containing p. As mentioned above, t_p also contains a k-extreme q' of C', and we know that $|p'q'| < l$. But $|pq'|$ is at most the L_1-diameter of t_p, which is equal to $6l_i d/r$. Hence, by triangle inequality, $|pp'| < l + 6l_i d/r$. □

Suppose that there is a subset T of H' consisting only of (at least two) clusters that are of the same type. Let C and C' be two clusters in T. First we observe that both C and C' have k-diameter $< l_i$, because they were created at some phase before p_i. By Observation 5.2, the k-diameter of $C \cup C'$ is $< l_i + 6l_i d/r$. Consequently, during p_i, at least one of C and C', let us say C, will be merged with a cluster C'' such that $C \cup C''$ has k-diameter $< l_i + 6l_i d/r$ (C'' and C' might be the same cluster). Thus, we realize that each cluster in T except at most one will participate in a merging during phase p_i, in such a way that the new clusters resulting from these mergings have k-diameters $< l_i + 6l_i d/r$.

In the remainder we only consider the clusters in H' and those clusters that are created during p_i by merging two or more clusters of H'. Let n' be the number of clusters in H'. As indicated in the previous paragraph, at least $n' - \lambda$ clusters will participate in a merging during p_i, and the new clusters resulting from these mergings have k-diameters $< l_i + 6l_i d/r$. The number of clusters that remain after these mergings is at most $(n' - \lambda)/2 + \lambda$. We can repeat the scenario for these clusters. After having done that we are left with at most $(n' - \lambda)/2^2 + \lambda$ clusters, each cluster having k-diameter $< l_i + 2 \times 6l_i d/r$. Indeed, we can repeat the scenario as long as we do not merge two clusters whose union has k-diameter $\geqslant 2l_i$ (we may assume that there are after each repetition sufficiently many clusters left for the next repetition to work).

Now, if we repeat the scenario j times, we are left with at most

$$\frac{n' - \lambda}{2^j} + \lambda < \frac{n'}{2^j} + \lambda$$

clusters, each cluster having k-diameter less than

$$l_i + j \cdot 6l_i d/r,$$

which is $<2l_i$ for $j = \lfloor 1+(d+1)\log_2 6 \rfloor > (d+1)\log_2 6$. But since h' can be partitioned into 6^d subcubes with edges of length l_i, we have by our induction hypothesis that $n' \leqslant 6^d \lambda(d+1)^d$. Hence, at the end of phase p_i, the number of clusters that have a k-extreme in h is at most

$$\frac{6^d \lambda(d+1)^d}{2^{(d+1)\log_2 6}} + \lambda = \frac{\lambda(d+1)^d}{6} + \lambda < \lambda(d+1)^d,$$

which completes the proof. □

We can summarize this section by the following theorem.

Theorem 5.3. *Let $\varepsilon > 0$ be a real constant, and let S be a set of n points in $\mathbb{R}^d$, where $d \geqslant 1$ is an integer constant. Then, under any fixed L_t-metric, a c-link ε-approximation of S can be computed in $O(n \log n)$ time and $O(n)$ space.*

6. Under the L_1 and L_∞-metrics

Under the L_∞-metric it is possible to define the k-extremes so that $\delta_k(P) = \delta(P)$ for any pair P of clusters. To see this, consider a cluster and a point x. All points within L_∞-distance $\leqslant l$ from x, for some $l > 0$, comprise an axis-aligned d-dimensional cube centered at x such that each of its edges has length $2l$. Hence, a point of the cluster that is farthest away from x must be an extreme point of the cluster in one of the $2d$ coordinate directions. Therefore, under the L_∞-metric, for every cluster we only need to keep track of an extreme point of the cluster in each of the $2d$ coordinate directions. (This corresponds to the definition of k-extremes given in Section 2 if the set $\mathcal{V}$ of vectors is defined for $w = 1$.)

A similar observation can be made for the L_1-metric. In this case, all points within L_1-distance $\leqslant l$ from x comprise a d-dimensional cross-polytope in which each edge has length $\sqrt{2}\,l$, that is, a regular polytope with d diagonals (a straight-line segment connecting two vertices of the polytope such that its interior does not intersect the boundary of the polytope) each diagonal being parallel to one of the coordinate axes. (In $\mathbb{R}^3$ it is a octahedron with diagonals parallel to the coordinate axes.) But this polytope is bounded by 2^d planes. Therefore, under the L_1-metric, for every cluster we only need to keep track of an extreme point of the cluster in each of these 2^d directions. The set $\mathcal{V}$ of vectors given in Section 2 has to be defined in a slightly different way in order to correspond to these 2^d directions. Namely, in this case we define $\mathcal{V}$ so that it consists of all possible vectors $(x_1, x_2, \ldots, x_d)$ such that each $x_i \in \{1, -1\}$. So the vectors in $\mathcal{V}$ correspond to the binary representation of $0, 1, \ldots, d$ where each 0 is replaced by -1.

It is not hard to see that our algorithm can be implemented using only operations allowed by the algebraic decision tree model. Moreover, by reduction to the static closest pair problem it is easy to conclude that $\Omega(n \log n)$ is a lower bound for the c-link clustering, even if we restrict ourselves to 1-dimensional space, in the algebraic decision tree model (see, for example, [13]). Thus, our algorithm is optimal in that model of computation.

This section is summarized in the following theorem.

Theorem 6.1. *Let S be a set of n points in $\mathbb{R}^d$, where $d \geqslant 1$ is an integer constant. Then, under the L_1 and L_∞-metrics, the complete linkage clustering of S can be computed in $O(n \log n)$ time and $O(n)$ space, which is optimal in the algebraic decision tree model.*

7. Final remarks

The c-link clustering belongs to a family of clustering methods known as SAHN methods (sequential, agglomerative, hierarchical, and nonoverlapping). Two other methods in this family are the centroid [4, 6, 14] and the median [4, 6, 10] method. Given n points in $\mathbb{R}^d$, these two methods work by repeatedly replacing a closest pair of points with a single (centroid respectively median) point. Hence, using the DCP-structure of Bespamyatnikh [3], these two clustering methods can be trivially computed in $O(n \log n)$ time, under any fixed L_t-metric.

References

[1] P. Arabie, L.J. Hubert, G. De Soete (Eds.), Clustering and Classification, World Scientific, Singapore, 1996.
[2] F. Aurenhammer, R. Klein, Voronoi diagrams, Tech. Report 198-5, Informatik, FernUniversität, Hagen, Germany, 1996.
[3] S.N. Bespamyatnikh, An optimal algorithm for closest pair maintenance, Proc. 11th ACM Symp. on Computational Geometry, 1995, pp. 152–161.
[4] W.H.E. Day, H. Edelsbrunner, Efficient algorithms for agglomerative hierarchical clustering methods, J. Classification 1 (1) (1984) 7–24.
[5] D. Defays, An efficient algorithm for a complete link method, Comput. J. 20 (1977) 364–366.
[6] J.C. Gower, A comparison of some methods of cluster analysis, Biometrics 23 (1967) 623–638.
[7] D. Krznaric, C. Levcopoulos, Fast algorithms for compete linkage clustering, Discrete Comput. Geom. 19 (1998) 131–145, A preliminary version of that paper appeared as: The first subquadratic algorithm for complete linkage clustering. Proc. 6th Internat. Symp. on Algorithms and Computation, Lecture notes in Computer Science, Vol. 1004, Springer, Berlin, 1995, pp. 392–401.
[8] T. Kurita, An efficient agglomerative clustering algorithm using a heap, Pattern Recognition 24 (3) (1991) 205–209.
[9] M. Křivánek, Connected admissible hierarchical clustering. Paper presented at the DIANA III Conf. Bechyne, Czechoslovakia, June 1990. Published in the KAM-Series as Tech. Report No. 90-189, 8 pages, School of Computer Science, Faculty of Mathematics and Physics, Charles University, Prague, 1990.
[10] G.N. Lance, W.T. Williams, A generalised sorting strategy for computer classifications, Nature 212 (1966) 218.

[11] X. Li, Parallel algorithms for hierarchical clustering and cluster validity, IEEE Trans. Pattern Anal. Mach. Intell. 12 (11) (1990) 1088–1092.
[12] F. Murtagh, Complexities of hierarchic clustering algorithms: state of the art, Comput. Statist. Quart. 1 (1984) 101–113.
[13] F.P. Preparata, M.I. Shamos, Computational Geometry: An Introduction, Springer, New York, 1985.
[14] R.R. Sokal, C.D. Michener, A statistical method for evaluating systematic relationships, Univ. Kans. Sci. Bull. 38 (1958) 1409–1438.