IEICE TRANS. INF. & SYST., VOL.E83–D, NO.3 MARCH 2000

INVITED SURVEY PAPER *Special Issue on Algorithm Engineering: Surveys*

Effective Use of Geometric Information for Clustering and Related Topics*

Tetsuo ASANO†, ***Member***

SUMMARY This paper surveys how geometric information can be effectively used for efficient algorithms with focus on clustering problems. Given a complete weighted graph G of n vertices, is there a partition of the vertex set into k disjoint subsets so that the maximum weight of an innercluster edge (whose two endpoints both belong to the same subset) is minimized? This problem is known to be NP-complete even for $k = 3$. The case of $k = 2$, that is, bipartition problem is solvable in polynomial time. On the other hand, in geometric setting where vertices are points in the plane and weights of edges equal the distances between corresponding points, the same problem is solvable in polynomial time even for $k \geq 3$ as far as k is a fixed constant. For the case $k = 2$, effective use of geometric property of an optimal solution leads to considerable improvement on the computational complexity. Other related topics are also discussed.

key words: *bipartite graph, coloring, computational geometry, diameter, duality transform, geometric clustering, intercluster distance, maximum spanning tree, separability, Voronoi diagram*

1. Introduction

Clustering is one of the central problems in pattern recognition. A great number of algorithms have been proposed so far, for information retrieval [10], spatial data bases [30], facility location [12], [35] and computer vision [24], [34]. This paper surveys how geometric information can be effectively used for designing efficient algorithms with focus on clustering problems. A simple, but general form of a clustering problem is to find a partition of a set of given objects into k parts so that the maximum dissimilarity between objects in the same part does not exceed some predefined threshold [19]. Dissimilarity relations among objects are usually represented by a weighted graph. Then, the problem is to partition a vertex set into k disjoint subsets (called clusters) so that the maximum weight of an innercluster edge (whose two endpoints both belong to the same subset) is minimized. This problem is known to be NP-complete even for $k = 3$ [15], [37]. The case of $k = 2$, that is, bipartition problem is solvable in polynomial time [6]. On the other hand, its geometric version in which vertices are points in the plane and weights of edges equal the distances between corresponding points can be solved in polynomial time even for any fixed $k \geq 3$ [9]. Section 2 first describes variants of the clustering problem in two different settings: graph and geometry models. Then, inherent difference of their computational complexities is revealed. Section 3 suggests a scheme for transforming input data as dissimilarity measures between objects into a set of points in the plane based on the principal coordinate analysis. Section 4 deals with the case $k = 2$ in the geometric setting and describes how geometric property is used to improve the computational complexity of algorithms. Section 5 deals with the extension of the result in Sect. 3 to the general k-way partition problem in the geometric setting. Section 6 includes other related topics.

Manuscript received June 22, 1999.
Manuscript revised September 11, 1999.

†The author is with the School of Information Science, Japan Advanced Institute of Science and Technology, Ishikawa-ken, 923–1292 Japan.

*A part of this paper was presented at Japan Conference on Discrete and Computational Geometry, Tokyo, Japan, 1998.

2. Variants of Clustering Problems

Clustering is of course one of the most fundamental problems in pattern recognition. Although there are a number of variations of the problem definition depending on applications, a natural notion of clustering is the grouping of similar objects. Mathematically it means a partition of objects into disjoint subsets (clusters) to optimize some mathematical measure of similarity among objects so that objects in each cluster are similar to each other and no two objects belonging to different clusters are similar to each other. Similarity or dissimilarity relations among objects are usually represented by a weighted graph. Then, the problem is to partition a vertex set of the graph into k disjoint subsets (clusters) so that the maximum weight of an innercluster edge (whose two endpoints both belong to the same subset) is minimized. We call this problem *k-way graph partition problem.*

The same problem can be considered in geometric setting in which vertices are points in the plane and weights of edges equal the distances between corresponding points. Then, the maximum weight of an innercluster edge is the maximum distance between two points in a cluster, which equals the diameter of the cluster (as a point set). We call this modified problem *k-way point set partition problem.*

3. Graph to Geometry Model

In the graph model we are given a graph which rep-

resents dissimilarity between objects, where arbitrary values are allowed for dissimilarity measures as far as they are positive. On the other hand, in the geometry model given objects are located in a space so that the distance between two points coincides with the dissimilarity measure between their corresponding objects. So, if one insists on using the geometry model when input is given as a dissimilarity graph, then it is required to map vertices of the dissimilarity graph into points in a space of appropriate dimensions, say in the plane. One way to do this is to rely on the principal coordinate analysis (see also [5], [31]). The principal coordinate analysis is a method for computing the coordinates of points to be mapped from a matrix representing the squared distances among points. Starting with a dissimilarity graph, we extend it to a transitive dissimilarity graph by defining an edge weight by the total weight of the minimum-weight (dissimilarity as weight) between pairs of vertices (objects). The resulting graph is represented in a matrix form: $W = (w_{ij})$. Then, we define a matrix $A = (a_{ij})$ by

$$a_{ij} = -\frac{1}{2}(w_{ij} - w_{i*} - w_{*j} + w_{**}),$$

where $w_{*j} = \sum_{i=1}^{n} w_{ij}$, $w_{i*} = \sum_{j=1}^{n} w_{ij}$, and $w_{**} = \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}$. Since A is a symmetric matrix, it can be diagonalized:

$$A = V\Lambda V',$$

where Λ is a diagonal matrix with eigenvalues on the diagonal and $V = (v_1, v_2, \ldots, v_n)$ is a matrix whose columns are the corresponding eigenvectors. Choose two largest eigenvalues λ_1 and λ_2 and let v_1 and v_2 be their corresponding eigenvectors, respectively. Then, the vector X representing x- and y-coordinates of the corresponding points in the plane are obtained by

$$X = \left[\sqrt{\lambda_1}v_1, \sqrt{\lambda_2}v_2\right].$$

4. Geometric Bipartition Problem

4.1 Definition of Problem

The first problem we consider is to partition a set of n points in the plane into k subsets (clusters) so that their maximum diameter is minimized [8]. Here the diameter of a point set S, denoted by $D(S)$, is defined by the maximum distance between any points in the set. For the case $k = 2$ there is a polynomial-time solution [A86]. In this section, starting with a brute-force algorithm for the problem, we describe how algorithms are improved using graph properties and further geometric properties of optimal solutions.

4.2 Brute-Force Algorithm

Let S be a set of n given points in the plane. A brute-force algorithm listed below can find an optimal bipartition of the set:

Algorithm 1: Brute-Force Algorithm
$d = \infty$;
for each subset S_1 of S
  Compute $S_2 = S - S_1$;
  Compute the diameters $D(S_1)$ and $D(S_2)$;
  if $\max\{D(S_1), D(S_2)\} < d$ then
    $d = \max\{D(S_1), D(S_2)\}$;
    Keep (S_1, S_2) as the current optimum (S_1^*, S_2^*);
Output the bipartition (S_1^*, S_2^*) as a solution;
end of Algorithm

This algorithm takes time proportional to the number of different subsets of S, which grows exponentially in n. Thus, the running time is also exponential.

4.3 Heuristic Algorithm

One way to improve the above brute-force algorithm is to rely on a general heuristic strategy called iterative improvement, as follows:

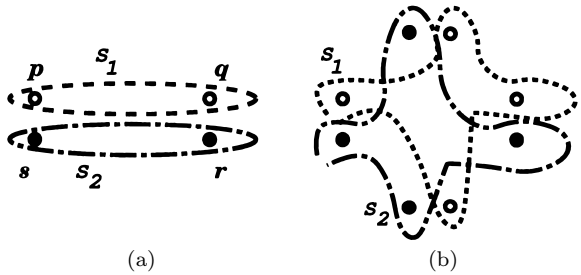
Algorithm 2: Iterative-Improvement Heuristic
Let (S_1, S_2) be any bipartition of a point set S;
do{
  Compute the diameters $D(S_1)$ and $D(S_2)$;
  Let $d = \max\{D(S_1), D(S_2)\}$;
  if $D(S_1) > D(S_2)$ then
    Find a pair (p, q) of points in S_1 which defines the diameter of S_1;
    if moving p or q from S_1 to S_2 improves the current bipartition
    then update the current optimal bipartition by the move;
  if $D(S_2) > D(S_1)$ then
    Find a pair (r, s) of points in S_2 which defines the diameter of S_2;
    if moving r or s from S_2 to S_1 improves the current bipartition
    then update the current optimal bipartition by the move;
} while(any improvement is obtained);
Output the current optimal bipartition (S_1, S_2);
end of Algorithm

The operation of moving a point from one cluster to another is usually referred to as a flip. Then, is it possible to have infinite number of flips? The answer is NO, since any flip properly decreases the larger diameter and we have only $O(n^2)$ different diameters in total. This implies $O(n^2)$ iterations of the loop.

Another question is whether this heuristic always leads to an optimal solution? The answer is again NO. Consider four points p, q, r and s such that they are vertices of a rectangle in this order and that p, q and r, s are equally distant while p, s and q, r are equally close to each other. See Fig. 1 (a). If $S_1 = \{p, q\}$ and $S_2 = \{r, s\}$, then any single flip cannot improves the bipartition. To have an optimal solution, we need a double flip which moves two points simultaneously between two clusters.

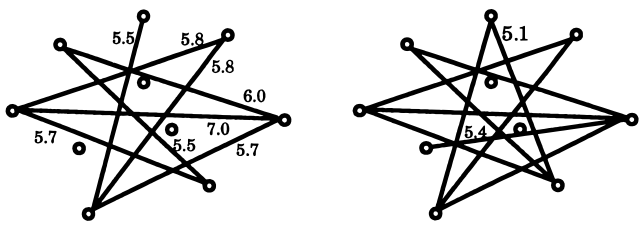
It is also easy to construct a simple example in which no double flip leads to any optimal solution in a similar manner. See Fig. 1 (b). This means that such iterative improvement heuristic does not always lead to an optimal solution.

In the following subsections we will describe two different approaches toward a polynomial-time algorithm for the bipartition problem. One is an approach based on graph model, and the other is based on geometric properties of optimal solutions.



(a) (b)

Fig. 1 Simple examples: (a) no single flip leads to improvement, and (b) no double flip gives any improvement.



(a) (b)

Fig. 2 Graphs associated with mimimum distances. (a) $G_{5.5}$: bipartite, and (b) $G_{5.1}$: non-bipartite.

4.4 Graph-Theoretic Approach

Given a set S of n points in the plane, we can build a graph G with those points as vertices and edges for all pairs of points. Each edge is associated with a weight that equals the Euclidean distance between two corresponding points.

Two different strategies may be possible: Delete edges iteratively in the increasing order of their weights starting from a complete graph or add edges iteratively in the decreasing order of their weights starting from a graph with n isolated vertices. The problem is when we stop the iteration. For the deletion-based strategy edges are deleted while a graph is non-bipartite. On the other hand, for the addition-based strategy the iteration terminates when a graph becomes non-bipartite. Figure 2 shows a simple example. The graph on the left side joins every pair of points with distance ≥ 5.5, which is bipartite or colorable by two colors. The graph on the right defined by edges with distance ≥ 5.1, however, is not bipartite any more. More concretely, the algorithm is described as follows.

Algorithm 3:
for each pair (p, q) of points, compute their distance;
Sort those point pairs in the decreasing order;
Let G be the graph with no edge;
do{
  Add en edge associated with a point pair (p, q)
  in the above order to G;
} while (graph G is bipartite);
Remove the last edge from G;
Output the bipartition of vertices of G;
end of Algorithm

Lemma 1: Algorithm 3 computes an optimal bipartition in $O(n^4)$ time and $O(n^2)$ space.

Proof: Correctness of the algorithm is rather easy to be verified. Let $(d_0, d_1, \ldots, d_m), m = n(n-1)/2 - 1$ be the sorted sequence of edge weights in the decreasing order, i.e., $d_0 \geq d_1 \geq \ldots \geq d_m$. In the algorithm we add an edge one by one in the sorted order to a graph starting from an empty graph. By G_i we denote the graph after adding the edge of weight d_i. Suppose that the iteration terminated when we added an edge of weight d_i. Then, due to the loop condition, G_i is not bipartite while G_{i-1} is. Since G_i is not bipartite, there must be an odd cycle. Therefore, whatever we partition the vertex set into two clusters, at least one of the edges in the odd cycle must be included in one of the subgraphs associated with the bipartition. This means that the larger diameter must be greater than or equal to d_i. Or for any bipartition, the larger diameter must be greater than or equal to d_i.

On the other hand, since the graph G_{i-1} is bipartite, the corresponding sets of vertices form bipartition of the point set so that no edge of weight greater than d_{i-1} is included in either of the resulting subgraphs. This means that there is a bipartition whose larger diameter is less than or equal to d_{i-1}. Therefore, the larger diameter of an optimal bipartition is d_i and its corresponding bipartition is given by a point set of G_{i-1}.

Next, we consider the computational complexities. The space required is obvious, since we need $O(n^2)$ storage is required for sorting a given edge set. The loop is iterated $O(n^2)$ times and each iteration is done in $O(n^2)$ time for the check of bipartiteness. Thus, it takes $O(n^4)$

time in total. □

The running time could be drastically reduced. The basic idea is binary search on the sorted list of edges. Once the distances are sorted, then in $O(\log n)$ iterations we can find largest d_i such that G_i is not bipartite but G_{i-1} is. The basic idea is presented in [6].

Lemma 2: Using binary search on the sorted list of point pairs in the order or their distances, an optimal bipartition can be computed in $O(n^2 \log n)$ time and $O(n^2)$ space.

Now the highest barrier against practical use is the space complexity. It is easy to see that $O(n^2)$ space is required as far as we rely on the sorted list of edge weights. In the next subsection we will see linear space is sufficient without any sacrifice of the order of the running time.

4.5 Improvement Using Geometric Properties

A key property for the geometric approach is the following theorem.

Theorem 1: Given two sets S_1 and S_2 of points in the plane, one can find two separable sets S_1' and S_2' such that $S_1' \cup S_2' = S_1 \cup S_2$ and

$$\max\{D(S_1'), D(S_2')\} \leq \max\{D(S_1), D(S_2)\}.$$

Here, two sets of points are said to be *separable* if they can be separated by a line. It is well known that two sets are separable if and only if their convex hulls are disjoint. A bipartition (S_1, S_2) is called a *separable partition* if S_1 and S_2 are separable. In this case, we also say that the bipartition (S_1, S_2) is *induced by line* ℓ if ℓ is a separating line for S_1 and S_2 [33].

The above theorem shows that the separability restriction can indeed be imposed without affecting optimality. In other words, we have only to check all of separable bipartitions. Then, how many separable bipartitions are there in total? A simple observation is enough. Suppose that we have a separable bipartition (S_1, S_2). Then, by the definition there is a line separating their convex hulls. Then, by appropriate rotation and translation the separating line touches one point from S_1 and another point from S_2. In this way, any separating line can be characterized by a pair of points from different sets. On the other hand, given such a pair, the bipartition is uniquely determined except for the points in the pair. Thus, we can conclude that there are $O(n^2)$ separable bipartitions. Furthermore, there is an algorithm for enumerating all separable bipartitions in $O(n^2 \log n)$ time while maintaining the diameters of two convex hulls. This suggests an $O(n^2 \log n)$-time and $O(n)$-space algorithm, which is superior in the space complexity to the graph-theoretic approach.

The theorem above looks straightforward, but the proof is in fact rather complicated. First imagine a situation depicted in Fig. 3 in which two convex hulls for S_1 and S_2 intersect twice. Then, moving all these points of S_1 to S_2 that are properly included in the convex hull of S_2 and doing the same thing from S_2 to S_1, each of the diameters never increases since all the new points lie inside each of the convex hulls. The case in which one convex hull is totally included in the other is easier. Partition the whole set roughly in the middle, and the larger diameter never increases since the convex hulls of the resulting sets are included in the original larger convex hull.

There still remains one case in which two convex hulls intersect many times. The proof for the case is rather complicated. So, we shall only sketch it.

Let us start with some key facts which often play important roles in similar proofs based on geometric properties. Before giving facts, we need some notations: For two points a and b, $\overline{ab}$ denote the line segment joining a and b, and ab denotes the Euclidean distance between a and b. For two point sets A and B, let AB denote the maximum distance between A and B; that is, $AB = \max\{ab | a \in A, b \in B\}$.

Fact 1: If a, b, c and d are the cyclically ordered vertices of a convex quadrilateral, then $ac + bd \geq \max\{ab + cd, ad + bc\}$.

Fact 2: In a triangle with an obtuse angle, the side lying opposite the obtuse angle is the longest side in the triangle.

Let us denote by $CH(S)$ the convex hull of a point set S. We assume that $CH(S_1) \cap CH(S_2) \neq \emptyset$, $CH(S_1) \not\subset CH(S_2)$ and $CH(S_2) \not\subset CH(S_1)$. Let $< u_1, u_2, \ldots, u_{2k} >$ be the sequence of points in clockwise order where $CH(S_1)$ and $CH(S_2)$ intersect. Write $CH(S_1) - CH(S_2)$ and $CH(S_2) - CH(S_1)$ as two interlacing sequences of polygons $< A_1, A_2, \ldots, A_k >$ and $< B_1, B_2, \ldots, B_k >$, respectively, where A_i intersects B_i at u_{2i}, and A_i intersects B_{i-1} at u_{2i-1}. See Fig. 4. The boundaries of $CH(S_1)$ and $CH(S_2)$ may touch without crossing each other, or they may even coincide on a small piece. However, those details are neglected here for simplicity of arguments.

We will separate the set $S_1 \cup S_2$ into S_1' and S_2' by a line L through two points u_i and u_j, whose choice

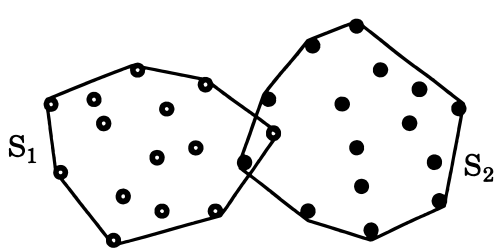


Fig. 3 A simple case in which two convex hulls share one connected regions.

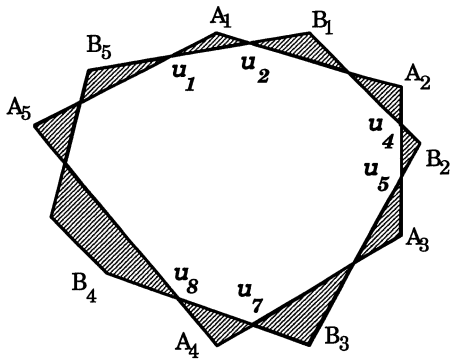


Fig. 4 Regions created by two intersecting convex polygons.

will be desired.

Without loss of generality, we may assume that $D(S_1) \geq D(S_2)$. We call a pair (A_i, B_j) *bad*, if $D(A_i \cup B_j) > D(S_1)$. The bad pairs are those pairs of polygons which must be separated by the line L in order to make both diameters $\leq D(S_1)$. Call a polygon A_i or B_j *bad*, if it appears in some bad pair. The following lemma holds for the relative positions of two bad pairs. We say that two pairs (A_i, B_j) and $(A_{i'}, B_{j'})$ with $A_i \neq A_{i'}$ and $B_j \neq B_{j'}$ *cross* if their cyclic sequence is $A_i, A_{i'}, B_j, B_{j'}$ or $A_i, B_{j'}, B_j, A_{i'}$. In other words, they cross if and only if the two segments connecting a point in A_i to a point in B_j and a point in $A_{i'}$ to a point in $B_{j'}$ intersect, independently of the choice of these points. Such segments are called *bad* segments.

Lemma 3: Any two disjoint bad pairs cross.

To prove the lemma, let us assume that there are two bad pairs (A_i, B_j) and $(A_{i'}, B_{j'})$ with $A_i \neq A_{i'}$ and $B_j \neq B_{j'}$ that do not cross. For each bad pair, we choose a bad line segment connecting two points at distance $> D(S_1)$ that lie in the polygons belonging to the pair. Let us call these points $a_i, b_j, a_{i'}$ and $b_{j'}$, respectively. The two possibilities for the relative positions of these points (disregarding symmetric variations) are depicted in Fig. 5. The bad segments are represented by double lines. Their endpoints are shown as black circles (points in S_1) and white circles (points in S_2).

(a) The case shown on the left side of Fig. 5 immediately leads to a contradiction: By Fact 1, the sum of the diagonals in the convex quadrilateral $a_i b_j a_{i'} b_{j'}$ is larger than the sum of two opposite sides. Hence,

$$D(S_1) + D(S_2) \geq a_i a_{i'} + b_j b_{j'} > a_i b_j + b_{j'} a_{i'} > 2D(S_1)$$

must hold, a contradiction to the assumption $D(S_1) \geq D(S_2)$.

(b) In the case shown on the right side of Fig. 5, we observe that the convex quadrilateral $a_i b_j b_{j'} a_{i'}$ must have an angle larger or equal to $\pi/2$. Without loss of generality, let this be the angle at b_j. Between the half-moons B_j and $B_{j'}$, there lies at least one (not necessarily bad) polygon A_m. Select an arbitrary point $a_m \in A_m$. Then the angle $\angle a_i b_j a_m$ is obtuse, and hence, by Fact 2,

$$a_i a_m > a_i b_j > D(S_1)$$

This is again a contradiction. □

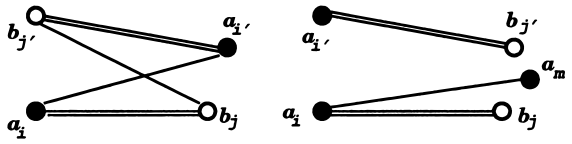


Fig. 5 Two impossible configurations of bad pairs.

In this manner we can show that the bad pairs give rise to a complete matching. Based on the fact that bad pairs must cross, we can guarantee the existence of a line which separates all bad pairs. Clearly, the resulting subsets S_1' and S_2' are separable and have diameter bounded by $\max\{D(S_1), D(S_2)\}$ as required. This completes the rough sketch for the proof of Theorem 1. The detail of the proof is found in [9]. The proof in [4] is incomplete, which was completed in [9].

4.6 Efficient Algorithm Based on Geometric Properties

We have known that there are $O(n^2)$ different bipartitions of a set of n points in the plane. This fact leads to a polynomial-time algorithm. In fact we can enumerate all possible bipartitions as follows:

Naive Enumeration of Bipartitions:
Let S be a given set of points;
for each pair (u, v) of points of S
  Let A be the set of points lying above the line through u and v;
  $S_1 = A \cup \{u\}$, $S_2 = S - S_1$;
  Compute diameters of S_1 and S_2;
  Compare the bipartition with the optimal solution found so far;
end of Algorithm

The computational complexities are easily analyzed. For each of $O(n^2)$ different pairs of points, the corresponding bipartition is computed in linear time. Once a point set is given the diameter of the set can be computed as the distance between the farthest points, and thus there is no need for pairwise distances. Given a point set, its diameter can be computed in $O(n \log n)$ time. Note that if points are sorted the running time is further reduced to $O(n)$. Thus, the running time of the algorithm above is $O(n \log n + n \times n^2) = O(n^3)$. The space complexity is obviously $O(n)$, which is an improvement from $O(n^2)$ in the graph-theoretic approach.

The inefficiency comes from the fact that convex hulls are computed each time from the scratch. If we could enumerate bipartitions so that next bipartitions

are obtained by constant number of changes, i.e., by exchanging at most two points each time, we would improve the time complexity. An idea is to fix one point u and rotate the separating line counterclockwisely around u like a circular plane sweep. The following is the algorithm based on this idea.

Efficient Enumeration of Bipartitions:
Let S be a given set of points;
Let $(S, \emptyset, d=\infty)$ be a candidate of an optimal solution;
for each point u in S do
  Sort the remaining points v_i by the angle of the line passing through u and v_i;
  Let $< v_1, v_2, \ldots, v_{n-1} >$ be the sorted list;
  Let A be the set of points above the line $\overline{uv_1}$;
  Define $S_1 = A \cup \{u\},\ S_2 = S - S_1$;
  Compute $D(S_1)$ and $D(S_2)$;
  $d' = \max\{D(S_1), D(S_2)\}$;
  if $d' < d$ then
    update the current optimum to $(S_1, S_2, d = d')$;
  for $i = 2$ to $n-1$ do
    Rotate the sweep line from $\overline{uv_{i-1}}$ to $\overline{uv_i}$;
    if v_i lies above $\overline{uv_{i-1}}$ (see Fig. 6 (a)) then
      move v_i from S_1 to S_2;
    if v_{i-1} lies above $\overline{uv_i}$ (see Fig. 6 (b)) then
      move v_{i-1} from S_2 to S_1;
    Update $D(S_1)$ and $D(S_2)$ if any move;
    if $d > d' = \max\{D(S_1), D(S_2)\}$ then
      update the current optimum to $(S_1, S_2, d = d')$;
Output (S_1, S_2, d) as an optimal bipartition together with the larger diameter;
end of Algorithm

Let us analyze the running time of the above algorithm. In each iteration we sort $n-1$ points in $O(n \log n)$ time, compute diameters of the initial sets in $O(n \log n)$ time. Then, in the inner loop which is iterated $n-2$ times, we update the sets S_1 and S_2 by moving at most two points between S_1 and S_2. More precisely, at most one point moves from S_1 to S_2 and at most one point moves from S_2 to S_1. Thus, we need an appropriate data structure for efficient implementation of dynamic set operations "insertion" and "deletion" while maintaining the diameter of a point set. Such a data structure is known. In fact, if we rely on the dynamic data structure proposed by Overmars and van Leeuwen [32], operations of insertion and deletion are implemented in $O(\log^2 n)$ time. If we could also answer the diameter of a convex hull, or the farthest point pair in $O(\log^2 n)$ time, the overall running time would be $O(n^2 \log^2 n)$. Fortunately, there is such a data structure [11] although $O(\log^2 n)$ time is achieved by amortized analysis and it holds for a semi-online model. Refer to [11] for details.

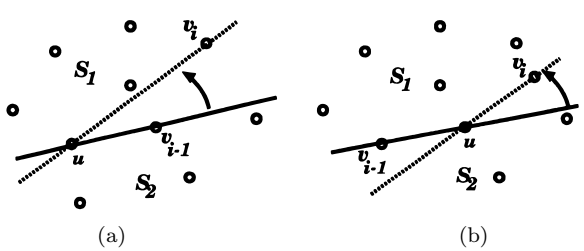


Fig. 6 Enumeration of all possible bipartitions by plane sweep around each point u.

4.7 Further Improvement Based on Geometric Properties

In view of Theorem 1, a set S has a bipartition with diameter less than t if and only if some line ℓ intersects every line segment in the set $\{(a,b) \in S \times S | ab \geq t\}$. In Edelsbrunner et al. [13], a line which intersects each segment (in its interior) of a collection of line segments L is called a *stabbing line* for L.

Definition 1: Let L be a set of line segments. Order the line segments of L in non-increasing order by length as $|e_1| \geq |e_2| \geq \cdots \geq |e_p|$. For any $i \leq p$, let $L(i)$ denote the list $< e_1, e_2, \ldots, e_i >$. The maximum index i such that $L(i)$ admits a stabbing line while $L(i+1)$ does not is called the *threshold index* for L, and a stabbing line for $L(i)$ is called a *threshold stabbing line* for L.

The algorithm in [6] finds a min-diameter separable bipartition for S in $O(n^2 \log^2 n)$ time and $O(n^2)$ space by computing a threshold stabbing line for $E = S \times S$. Theorem 1 implies that the bipartition so obtained is a true optimum. Furthermore, we can significantly reduce the space and time requirements of the algorithm by replacing E with a subset of size only $O(n)$, namely, the edge set of a *maximum spanning tree* on S (A maximum spanning tree is a spanning tree whose total edge length is as large as possible). An example is shown in Fig. 7.

Theorem 2: A threshold stabbing line for a maximum spanning tree of S induces a min-diameter bipartition of S.

This theorem leads to the following algorithm:

Efficient Min-Diameter Bipartition:
Input: a set S of n points in the plane.
Output: a min-diameter bipartition (S_1, S_2) of S.

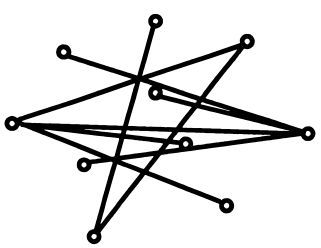


Fig. 7 Maximum spanning tree for a point set.

Algorithm:
1. Compute a maximum spanning tree M for S.
2. Find a threshold stabbing line ℓ for M.
3. Output the partition (S_1, S_2) induced by ℓ.

end of Algorithm

The correctness of this algorithm follows from Theorem 2. We now analyze its complexity. In step 1, a maximum spanning tree can be constructed in $O(n \log n)$ time and $O(n)$ space by employing the algorithm in [28]. The partition (S_1, S_2) in step 3 can be obtained in $O(n)$ time easily once ℓ is known. It thus remains to show that step 2 can be carried out in $O(n \log n)$ time and $O(n)$ space.

To find a threshold stabbing line for a set L of n line segments, we can first sort the line segments in L by length, and then perform a binary search using the algorithm from [13] as a subroutine, which has the following performance.

Fact 3: Given a set L of n line segments, there is an algorithm which decides in $O(n \log n)$ time and $O(n)$ space whether there exists a stabbing line for L, and finds one if there does.

However, the binary search can introduce an additional $\log n$ factor in the running time if care is not taken. We thus take a closer look at the algorithm mentioned the fact above. The algorithm is based on the observations that the point-line duality transforms line segments into 'double wedges' (see, for example, [3], [7]), and that L has a stabbing line if and only if the corresponding set of double wedges has non-empty intersection. We will make use of the following two facts established in [13], where the *stabbing region* for a set of line segments refers to the intersection of the corresponding double wedges.

Fact 4: The stabbing regions for n line segments in the plane can be computed in $O(n \log n)$ time.

Fact 5: Given two stabbing regions for n_1 and n_2 line segments respectively, their intersection can be computed in $O(n_1 + n_2)$ time.

Now, in the binary search for a threshold stabbing line, we assume at the beginning of the j-th iteration that the first m segments of L are known to have a nonempty stabbing region R. (Initially, $m = 0$ and R is the entire plane.) We then use Fact 4 to compute the stabbing region R' for the subset consisting of the $m + 1$-st through the $m + n/2^j$-th segments of L. If R' is non-empty, we let $m := m + n/2^j$ and compute $R := R \cap R'$ by Fact 5. The loop is then repeated with $j := j + 1$.

We analyze the complexity of the above procedure. The j-th iteration of the loop uses $O(n/2^j \log(n/2^j))$ time for computing the stabbing region R', and $O(n)$ time for finding the intersection $R \cap R'$. Summing over $j = 1, \ldots, \log n$, we obtain an $O(n \log n)$ bound for the total time. The space required is linear. This completes the analysis of our algorithm for min-diameter bipartition.

Theorem 3: A min-diameter bipartition for a set of n points in the plane can be computed in time $O(n \log n)$ and space $O(n)$.

5. Generalizations of the Result

The result stated in Theorem 3 was extended to more than two clusters by Capoyleas, Rote and Woeginger [9] in the following manner.

Theorem 4: Consider the optimal k-clustering problem for the diameter with monotone increasing function $\mathcal{F}$. For every point set S in the plane, there is an optimal k-clustering such that each pair of clusters is linearly separable.

Proof: [9] Consider the optimal k-clustering for which the sum of perimeters of all clusters becomes minimal. Assume that there are two clusters which are not linearly separable. Applying Theorem 1 to the above two clusters, we get a k-clustering with smaller sum of perimeters. As both affected diameters do not increase, the value of $\mathcal{F}$ does not increase, too. □

So far, we have only dealt with the diameter as the quality measure of a cluster. For the radius, an analog of Theorem 4 can be shown directly [9].

Theorem 5: Consider the optimal k-clustering problem for the radius with a monotone increasing function $\mathcal{F}$. For every point set S in the plane, there is an optimal k-clustering such that each pair of clusters is linearly separable.

Based on the discussions so far, we reach the following generalized theorem [9].

Theorem 6: For any fixed k, the geometric k-clustering problem for the diameter or for the radius with respect to some monotone increasing function $\mathcal{F}$ is solvable in $O(n^{6k})$ time.

For the case $k = 3$ the current best known algorithm runs in $O(n^2 \log^2 n)$ time (see Hagauer and Rote [17]).

Another generalization is established by Hershberger and Suri [21]. The problem they considered is the following: given a planar set of points S, a measure μ acting on S, and a pair of values μ_1 and μ_2, does there exist a bipartition (S_1, S_2) satisfying $\mu(S_i) \leq \mu_i$ for $i = 1, 2$? $O(n \log n)$-time algorithms are presented for several natural measures, including the diameter, the area, perimeter or diagonal of the smallest enclosing axes-parallel rectangles, and so on. Even for geometric setting, the k-way partition problem is known to be NP-complete for many of these measures.

6. Related Topics

6.1 Minimum-Diameter Balanced Bipartition

So far we have been interested in the criterion of minimizing the largest diameter. There are a number of studies under similar but slightly different criteria. Avis [6] dealt with balanced bipartition. That is, given a set of points in the plane, the problem is to find the smallest t such that the set can be partitioned into two equal-sized subsets each of which has diameter at most t. With this additional constraint, the separability condition of an optimal solution does not hold. The graph-theoretic approach described in the previous section is the basic idea in [6]. We consider the following predicate $Q_0(t)$:

> *S can be partitioned into subsets S_1 and S_2 so that* $\max(D(S_1), D(S_2)) \leq t$.

By the observation in Sect. 2, $Q_0(t)$ is true if and only if the graph G_t defined by edges interconnecting points with mutual distances greater than t is bipartite. The problem is to find the smallest value of t such that $Q_0(t)$ is true. Difficulty is how to take the size constraint into accounts. If G_t is connected then there is a unique bipartition. But if it is not connected, it seems that there may be exponentially many different bipartitions. In this case, fortunately, the total size is bounded by the number of points and so it can be checked in $O(n^2)$ time by a so-called pseudo-polynomial time algorithm whether there is a balanced bipartition, i.e., a representation of a disconnected bipartite graph with two vertex sets of equal size in the two sides. Based on the results an $O(n^2 \log n)$ time algorithm is derived for the problem.

6.2 Maximizing the Intercluster Distances

As an analogue to the minimum-diameter clustering described in the previous sections, we can define a farthest k-partition: Given a set S of points in the plane, a k-partition of S is a decomposition of S into k disjoint subsets (clusters) $\mathcal{P} = \{C_1, C_2, \ldots, C_k\}$. We call $\mathcal{P}$ a *farthest k-partition* if the minimum intercluster distance $\min_{i,j} \min\{ab | a \in C_i, b \in C_j\}$ is maximized aong all k-partitions of S. Compared with a minimum-diameter partition, it is easy to construct a farthest k-partition of a set by observing the following property of a minimum spanning tree.

Let the edges of $S \times S$ be sorted in non-decreasing order by length as $|e_1| \leq |e_2| \leq \cdots \leq |e_m|$, where $m = n(n-1)/2$. Let MST be a minimum spanning tree on S, with edges $|e_{M(1)}| \leq |e_{M(2)}| \leq \cdots \leq |e_{M(n-1)}|$.

Theorem 7: The collection Q of connected components formed by the edges $\{e_{M(1)}, e_{M(2)}, \ldots, e_{M(n-k)}\}$ gives a farthest k-partition of S.

This theorem [4] immediately leads to an $O(n \log n)$-time and $O(n)$-space algorithm for computing a farthest k-partition for a set of n points in the plane. The algorithm first computes a minimum spanning tree, sorts its edges, and then constructs the k components by a Union-Find algorithm [2].

6.3 Minimizing the Sum of Diameters

Another natural optimization criterion is to minimize the sum of diameters. Monma and Suri discussed the problem in [29]. They discussed the problem in two different settings. For a set of n points in the plane, their algorithm runs in $O(n^2)$ time and $O(n)$ space. For a weighted graph with n vertices and m edges they improved the previously known time bound $O(n^3 \log n)$ [18] into $O(mn \log n)$ time. Basic ideas behind their algorithms are a maximum spanning tree and a notion of (r_1, r_2)-partition, which is a bipartition such that two subsets are bounded by r_1 and r_2, respectively. One important subroutine is the $O(m)$ algorithm for deciding whether a (r_1, r_2)-partition exists. The result for the geometric setting was further improved by Hershberger [20] into $O(n \log n / \log\log n)$ time.

6.4 Variance-Based k-Clustering

Inaba, Katoh, and Imai [23] studied the problem of variance-based k-clustering, i.e., one of partitioning n points into k clusters so as to minimize the sum of variances of clusters, and compared it with popular algorithm based on iterative improvement called 'k-means algorithm' [36] by computer experiments. The basic tool is a general parametric technique by Katoh and Ibaraki [26] for minimizing quasiconcave functions, which leads to characterization of an optimal clustering by means of higher-order Varonoi diagram. They showed that optimal solutions can be characterized by weighted Voronoi diagram generated by k points, and evaluated the primary shutter function of the k-Voronoi space.

6.5 Euclidean p-Center Problem

The Euclidean p-center problem is to cover a set of points by p congruent balls of the smallest possible radius. The simplest version of this problem, i.e., planar p-center problem is known to be NP-hard [14]. The latest result in this area is found in [1], where an $n^{O(k^{1-1/d})}$ time algorithm for solving the k-center problem in the d-dimensional space, under L_∞ and L_2 metrics. This is an improvement of the previous result by Hwang et al. [22] for the planar case. A simple $(1+\epsilon)$-approximation algorithm for the k-center problem is also presented. The running time of the algorithm is

$O(n \log k) + (k/\epsilon)^{O(k^{1-1/d})}$. Some extended version of the problem is also considered in [1]. It is named L-capacitated p-center problem for some integer L. In this version we are given an integer L and asked to solve the p-center problem with an additional constraint that each cluster has at most L points.

6.6 Circuit Partition

A number of methods have been proposed for determining placement of modules in VLSI chips. One of the most popular methods is "Mincut Placement Algorithm" proposed by Lauther in 1980 [27]. The basic strategy of this algorithm is so-called divide-and-conquer. That is, a set of modules is divided into two parts so that the number of interconnections between different sides is minimized under the constraint that the total areas are compatible in the two resulting sides. Then, an optimal placement of modules is determined in each side in a recursive manner. Finally, the interconnections between the two sides are completed.

This Mincut Algorithm seems to be very promising if we find an optimal partition of modules. Unfortunately the problem seems to be intractable [16]. This is mainly because of exponentially many different partitions. Two different approaches may be considered. One of them is to rely on heuristic search toward an approximated goal, such as maximizing the minimum connectivity among modules in the same part. The other approach is to find a partition that is best in a restricted search space. In the latter approach we map modules into points in the plane so that the distance between any two points is anti-proportional to the connectivity between the corresponding modules as much as possible. Thus, two tightly connected modules should be placed close to each other. Furthermore, we put a restriction on partitions, that is, we only consider partitions by straight lines (called linear partitions). A combinatorial observation tells us the fact that there are only $O(n^2)$ different linear partitions, which allows us to examine all possible linear partitions in polynomial time. An efficient algorithm for this purpose is presented in [5], which is based on duality transform and Topological Walk.

7. Concluding Remarks

I believe that the greatest contribution of the Algorithm theory is discovery of efficient or polynomial-time algorithms for those problems which look intractable. Most of the problems considered in this survey paper are such ones. Dynamic programming has been a common algorithmic paradigm for polynomial-time solvability. This paper suggests several other approaches for the same goal, I hope.

Acknowledgment

This work was partially supported by Grant in Aid for Scientific Research of the Ministry of Education, Science and Cultures of Japan. The author would like to express his sincere thanks to Dr. Takeshi Tokuyama of IBM Tokyo Research Laboratory for his valuable comments based on careful reading of the draft.

References

[1] P.K. Agarwal and C.M. Procopiuc, "Exact and approximation algorithms for clustering," Proc. 9th ACM-SIAM Sympos. Discrete Algorithms, pp.658–667, 1998.
[2] A. Aho, J. Hopcroft, and J. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass., 1974.
[3] T. Asano, Computational Geometry, Asakura-Shoten, 1990.
[4] T. Asano, B. Bhattacharya, J.M. Keil, and F.F. Yao, "Clustering algorithms based on minimum and maximum spanning trees," Proc. 4th Annual ACM Symp. on Computational Geometry, pp.252–257, 1988.
[5] T. Asano and T. Tokuyama, "Circuit partitioning algorithms: Graph model versus geometry model," Proc. 2nd International Symposium on Algorithms, pp.94–103, Taipei, 1991.
[6] D. Avis, "Diameter partitioning," Discrete and Computational Geometry, vol.1, pp.265–276, 1986.
[7] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, Computational Geometry: Algorithms and Applications, Springer, 1997.
[8] P. Brucker, "On the complexity of clustering problems," in Optimization and Operations Research, Lecture Notes in Economics and Mathematical Systems, eds. R. Henn, B. Korte, and W. Oletti, pp.45–54, Springer, Berlin, 1978.
[9] V. Capoyleas, G. Rote, and G. Woeginger, "Geometric clusterings," J. Algorithms, vol.12, pp.341–356, 1991.
[10] M. Charikar, C. Chekuri, T. Feder, and R. Motwani, "Incremental clustering and dynamic information retrieval," Proc. 29th Annu. ACM Sympos. Theory of Computing, pp.626–634, 1997.
[11] D. Dobkin and S. Suri, "Maintenance of geometric extrema," J. ACM, vol.38, no.2, pp.275–298, 1991.
[12] Z. Drezner, ed., Facility Location, Springer-Verlag, New York, 1995.
[13] H. Edelsbrunner, H.A. Mauer, F.P. Preparata, A.L. Rosenberg, E. Welzl, and D. Wood, "Stabbing line segments," BIT, vol.22, pp.274–281, 1982.
[14] R.J. Fowler, M.S. Paterson, and S.L. Tanimoto, "Optimal packing and covering in the plane are NP-complete," Inform. Process. Lett., vol.12, pp.133–137, 1981.
[15] M.R. Garey and D.S. Johnson, Computers and Intractability, Freeman, New York, 1979.
[16] M.R. Garey, D.S. Johnson, and L. Stockmeyer, "Some simplified NP-complete graph problems," Theor. Comput. Sci., vol.1, pp.237–267, 1976.
[17] J. Hagauer and G. Rote, "Three-clustering of points in the plane," Proc. 1st Annual European Symp. on Algorithms (ESA'93), Lecture Notes in Computer Science, vol.726, pp.192–199, 1993.
[18] P. Hansen and B. Jaumard, "Minimum sum of diameters clustering," J. Classification, pp.215–226, 1987.
[19] J.A. Hartigan, Clustering Algorithms, John-Wiley, New

York, 1975.
[20] J. Hershberger, "Minimizing the sum of diameters efficiently," Computational Geometry: Theory and Applications, vol.12, pp.111–118, 1992.
[21] J. Hershberger and S. Suri, "Finding tailored partitions," Proc. 5th Annual ACM Symp. Computational Geometry, pp.255–265, 1989.
[22] R.Z. Hwang, R.C.T. Lee, and R.C. Chang, "The slab dividing approach to solve the euclidean p-center problem," Algorithmica, vol.9, pp.1–22, 1993.
[23] M. Inaba, N. Katoh, and H. Imai, "Applications of weighted voronoi diagrams and randomization to variance-based k-clustering," Proc. 10th ACM Symp. Computational Geometry, pp.332–339, 1994.
[24] J. Jolion, P. Meer, and S. Batauche, "Robust clustering with applications in computer vision," IEEE Trans. Pattern Anal. & Mach. Intell., vol.13, pp.791–802, 1991.
[25] D.S. Johnson, "The NP-completeness column: Ongoing guide," J. Algorithms, vol.3, pp.182–195, 1982.
[26] N. Katoh and T. Ibaraki, "A parametric characterization and an ϵ-approximation scheme for the minimization of a quasiconcave program," Discrete Applied Mathematics, vol.17, pp.39–66, 1987.
[27] U. Lauther, "A min-cut placement algorithm for general cells assemblies based on a graph representation," J. Digital Systems, vol.4, pp.21–34, 1980.
[28] C. Monma, M. Paterson, S. Suri, and F. Yao, "Computing Euclidean maximum spanning trees," Proc. 4th Annual ACM Symp. Computational Geometry, pp.241–251, 1988.
[29] C. Monma and S. Suri, "Partitioning points and graphs to minimize the maximize or the sum of diameters," Proc. 6th International Conf. on Theory and Applications of Graphs, 1988.
[30] R.T. Ng and J. Han, "Efficient and effective clustering methods for spatial data mining," Proc. 20th International Conf. on Very Large Databases, pp.144–155, 1994.
[31] R.H.J.M. Otten, "Automatic floorplan design," Proc. 19th Design Automation Conf., pp.261–267, 1982.
[32] M. Overmars and J. van Leeuwen, "Maintenance of configurations in the plane," J. Comput. Syst. Sci., vol.23, pp.166–204, 1981.
[33] F.P. Preparata and M.I. Shamos, Computational Geometry — An Introduction, Springer Verlag, New York, 1985.
[34] P. Schroeter and J. Bigün, "Hierachical image segmentation by multi-dimensional clustering and orientation-adaptive boundary refinement," Pattern Recognition, vol.28, pp.695–709, 1995.
[35] D. Shmoys, E. Tardos, and K. Aardal, "Approximation algorithms for facility location problems," Proc. 29th ACM Sympos. Theory Comput., pp.265–274, 1997.
[36] S.Z. Selim and M.A. Ismail, "K-means-type algorithms: A generalized convergence theorem and characterization of local optimality," IEEE Trans. Pattern Anal. & Mach. Intell., pp.81–87, 1984.
[37] K.J. Supowit, "Topics in computational geometry," Ph.D. thesis, Dept. of Computer Science, University of Illinois at Urbana-Champaign, Report UIUCDCS-R-81-1062, 1981.

Tetsuo Asano received the B.E., and M.E., and Ph.D. degrees in Engineering from Osaka University in 1972, 1974, and 1977, respectively. He is currently a professor of JAIST (Japan Advanced Institute of Science and Technology). His research interest includes Computational Geometry, Discrete Algorithms, Combinatorial Optimization and their applications. Dr. Asano is a member of IEEE, ACM, SIAM, IPSJ, and ORS. He is a member of the editorial boards of Discrete and Computational Geometry, International Journal of Computational Geometry and Applications, Computational Geometry: Theory and Applications, etc.