Exact Algorithms for Size Constrained 2-Clustering in the Plane

Jianyi Lin[*], Alberto Bertoni, Massimiliano Goldwurm

Dipartimento di Informatica, Università degli Studi di Milano
Via Comelico 39/41, 20135 Milano, Italy

Abstract

We study the problem of determining an optimal bipartition $\{A, B\}$ of a set X of n points in $\mathbb{R}^2$, under the size constraints $|A| = k$ and $|B| = n - k$, that minimizes the dispersion of points around their centroid in A and B, both in the case of Euclidean and Manhattan norm. Under the Euclidean norm, we show that the problem can be solved in $O(n\sqrt[3]{k}\log^2 n)$ time by using known properties on k-sets and convex hulls; moreover, the solutions for all $k = 1, 2, \ldots, \lfloor n/2 \rfloor$ can be computed in $O(n^2 \log n)$ time. In the case of Manhattan norm, we present an algorithm working in $O(n^2 \log n)$ time, which uses an extended version of red-black trees to maintain a bipartition of a planar point set. Also in this case we provide a full version of the algorithm yielding the solutions for all size constraints k. All these procedures work in $O(n)$ space and rely on separation results of the clusters of optimal solutions.

Keywords: algorithms for clustering, cluster size constraints, convex hull, k-set, Euclidean norm, Manhattan norm

1. Introduction

The Clustering Problem we consider in this work consists in finding a partition of a set X of n points into m subsets (called clusters) that minimizes the total dispersion of points around the centroid in every subset. This is a fundamental problem in many research areas like data mining, image analysis, pattern recognition and bioinformatics [24]. Clustering is a classical method in unsupervised machine learning, frequently used in statistical data analysis [7, 15].

[*] Corresponding author

Email address: jianyi.lin@unimi.it (Jianyi Lin)

Preliminary results of this research were presented at the conferences *SOFSEM 2015* [4] and *ICTCS 2014* [5].
This work has been supported by project PRIN #H41J12000190001 “Automata and formal languages: mathematical and applicative aspects”.

The computational complexity of the problem depends on a variety of parameters: the dimension d of the point space (usually $\mathbb{R}^d$), the distance or semi-distance used to measure the dispersion of points, the number m of clusters (which may be arbitrary, as part of the instance, or fixed in advance), the size of the clusters and possibly others constraints [3, 27, 28]. In most cases the problem is difficult. For instance, assuming the squared Euclidean semi-distance, when the dimension d is arbitrary the general Clustering Problem is NP-hard even if the number m of clusters is fixed to 2 [1, 10]. The same occurs if m is arbitrary and the dimension is $d = 2$ [20]. The problem is solvable in polynomial time when fixing both m and d [16]. Moreover there exists a well-known, usually fast, heuristic for finding an approximate solution, called k-Means [19], which however requires exponential time in the worst case [25].

Thus, a natural goal is the analysis of the problem under input hypotheses that allow us to design polynomial time algorithms. In this work we consider the Clustering Problem in $\mathbb{R}^2$ when the number of clusters is $m = 2$ and their size is given by the instance as useful background information [3], both assuming the Euclidean and Manhattan norm. We call this problem *Size Constrained 2-Clustering* in $\mathbb{R}^2$ (2-SCC-2 for short).

The relevance of the 2-clustering problems is due to the wide spread of hierarchical clustering techniques that repeatedly apply the 2-clustering as the key step. The 2-clustering problem with cluster size constraints has been already studied in [18, 6], where it is shown that in dimension 1 the problem is solvable in polynomial time for every norm ℓ_p with integer $p \geq 1$, while there is some evidence that the same result does not hold for non-integer p. It is also known that for arbitrary dimension d the same problem is NP-hard even assuming equal sizes of the two clusters [6].

Under the Euclidean norm, an instance of 2-SCC-2 is given by a set $X \subset \mathbb{R}^2$ of n points in general position together with an integer k such that $1 \leq k \leq n/2$, while the solution is a bipartition $\{A, B\}$ of X such that $|A| = k$ minimizing the total weight $W(A) + W(B)$. Here the weight $W(A)$ (respectively, $W(B)$) is the sum of the squares of the ℓ_2-distances of all points $a \in A$ (resp. $b \in B$) from the centroid of A (resp. B). Recall that the unconstrained version of the same problem, with an arbitrary number of clusters, is NP-hard [20].

In this case we show two results. First, we describe an algorithm that solves 2-SCC-2 in $O(n\sqrt[3]{k}\log^2 n)$ time. This is obtained by using known results of computational geometry concerning dynamic data structures for convex hulls [23, 22] and the enumeration of k-sets in $\mathbb{R}^2$ [13, 11] ([1]). Then, we present an algorithm for the full version of the problem, i.e. a procedure yielding a solution for all $k = 1, 2, \ldots, \lfloor n/2 \rfloor$, that works in $O(n^2 \log n)$ time. This procedure is based on a similar approach used in [14] to solve the unconstrained version of the problem.

[1] We notice that such a time complexity can be reduced of a logarithmic factor if one uses the dynamic data structure for convex hull proposed in [8], whose journal version however, to our knowledge, has not appeared yet.

We also study the 2-SCC-2 problem under the Manhattan distance (ℓ_1 norm). In this case we first present a full algorithm computing a solution for all size constraints in $O(n^3 \log n)$ time. Then, we describe another procedure yielding a solution for a single (arbitrary) size constraint k, that works in $O(n^2 \log n)$ time.

All these algorithms work in $O(n)$ space and are based on separation results of the clusters of optimal solutions of the problem, under ℓ_1 and ℓ_2 norms, which intuitively extend to the bidimensional case the so-called String Property on the real line [21, 26].

We note that there exists a wide literature on clustering algorithms based on different optimality criteria to choose the best partition. For instance, in [2] efficient procedures are given that find solutions in the plane that either minimize the cluster diameters or maximize the minimum intercluster distance.

2. Problem definition

In this section we define the 2-clustering problem in the plane and fix our notation. For any point $a \in \mathbb{R}^2$, we denote by a_x and a_y the abscissa and the ordinate of a, respectively. Moreover, for every real $p \geq 1$, let $\|a\|_p$ be the usual ℓ_p norm of point a, i.e. $\|a\|_p = (|a_x|^p + |a_y|^p)^{1/p}$. Clearly, $\|a\|_2$ and $\|a\|_1$ are the Euclidean and the Manhattan norm of a, respectively.

To define our problem formally, let us consider a finite set $X \subset \mathbb{R}^2$: we say that X is in *general position* if it does not contain three collinear points. A *cluster* of $X \subset \mathbb{R}^2$ is a non-empty subset $A \subset X$, while the pair $\{A, \bar{A}\}$ is a *2-clustering* of X, where $\bar{A} = X \smallsetminus A$. Assuming the ℓ_p norm, the *centroid* and the *weight* of A are the values $C_A \in \mathbb{R}^2$ and $W(A) \in \mathbb{R}_+$ defined, respectively, by

$$C_A = \operatorname*{argmin}_{\mu \in \mathbb{R}^2} \sum_{a \in A} \|a - \mu\|_p^p, \tag{1}$$

$$W(A) = \sum_{a \in A} \|a - C_A\|_p^p \tag{2}$$

Then, the Size Constrained 2-Clustering Problem in $\mathbb{R}^2$ (2-SCC-2, for short) is defined as follows:

> **2-SCC-2 problem under ℓ_p norm**
> Given a set $X \subset \mathbb{R}^2$ of cardinality n in general position and an integer k, $1 \leq k \leq n/2$, find a 2-clustering $\{A, \bar{A}\}$ of X, with $|A| = k$, that minimizes the weight $W(A, \bar{A}) = W(A) + W(\bar{A})$.

In the case $p > 1$ the solutions of the problem above satisfy the following property, proved in [6] in a more general dimension.

Theorem 1 (Separation Result in ℓ_p, for $p > 1$). *For any fixed $p > 1$, let $\{A, B\}$ be an optimal solution of the 2-SCC-2 problem under ℓ_p norm for an* *instance $X \subset \mathbb{R}^2$ with size constraint $|A| = k$. Then, we have $C_A \neq C_B$ and there exists $c \in \mathbb{R}$ such that for every $u \in X$*

$$u \in A \textit{ implies } \|u - C_A\|_p^p - \|u - C_B\|_p^p < c$$
$$u \in B \textit{ implies } \|u - C_A\|_p^p - \|u - C_B\|_p^p > c$$

The previous theorem states that (for variable z ranging over $\mathbb{R}^2$) equation

$$\|z - C_A\|_p^p - \|z - C_B\|_p^p = c \tag{3}$$

defines a curve in $\mathbb{R}^2$ that strictly separates the clusters of any optimal solution $\{A, B\}$.

In section 7 a similar result is presented for case $p = 1$, where however strict bounds $<$ and $>$ are replaced by $\leq$ and $\geq$, respectively. In both cases $p = 1$ and $p = 2$, we use such a property to design algorithms for the 2-SCC-2 problem. More general results already appeared in the literature for arbitrary dimension also for the unconstrained 2-clustering problem [14, 6].

3. The 2-SCC-2 problem under Euclidean norm

In this section we study the 2-SCC-2 problem assuming the Euclidean norm ℓ_2. For simplicity, here we omit the index 2 to denote the norm of a point. Thus, the centroid and the weight of a cluster A, defined in (1) and (2), are now given by

$$C_A = \operatorname*{argmin}_{\mu \in \mathbb{R}^2} \sum_{a \in A} \|a - \mu\|^2 = \frac{\sum_{a \in A} a}{|A|},$$
$$W(A) = \sum_{a \in A} \|a - C_A\|^2$$

This means that C_A is the average value of points in A, i.e. the sample mean of A, while the value $\frac{W(A)}{|A|-1}$ becomes the traditional sample variance of A, once we interpret its elements as sample points picked up from a probability distribution in $\mathbb{R}^2$. Thus, $W(A)$ here represents a typical measure of the dispersion of points in A around their mean value and the 2-SCC-2 problem is now equivalent to finding the bipartition $\{A, B\}$ of given sizes that minimizes the sum of the sample variances of A and B.

Moreover, it is clear that the weight of any bipartition $\{A, B\}$ of a set $X \subset \mathbb{R}^2$,

$$W(A, B) = \sum_{a \in A} \|a - C_A\|^2 + \sum_{b \in B} \|b - C_B\|^2,$$

does not change under simultaneous translation of all elements of X. Thus, we may compute the weight of a 2-clustering by using the following proposition.

Proposition 1. *Given a set $X \subset \mathbb{R}^2$ of n points with centroid $C_X = (0,0)$, let $\{A, B\}$ be a 2-clustering of X such that $|A| = k$ for some $k \in \{1, 2, \dots, \lfloor n/2 \rfloor\}$. Then*

$$W(A,B) = \sum_{p \in X} \|p\|^2 - \frac{n}{k(n-k)} \|S_A\|^2$$

where $S_A = \sum_{a \in A} a$.

Proof. Since $C_X = (0,0)$, the relation $(0,0) = nC_X = S_A + S_B = kC_A + (n-k)C_B$ yields $C_B = -\frac{k}{n-k} C_A$. Thus, from the definition of W we get

$$\begin{aligned} W(A,B) &= \sum_{a \in A} \|a\|^2 - |A| \|C_A\|^2 + \sum_{b \in B} \|b\|^2 - |B| \|C_B\|^2 \\ &= \sum_{p \in X} \|p\|^2 - \frac{nk}{n-k} \|C_A\|^2 \end{aligned}$$

and the result follows by observing that $C_A = S_A/k$. □

As a consequence, under ℓ_2 norm solving the 2-SCC-2 problem for an instance (X, k) with $C_X = (0,0)$ is equivalent to determining a subset $A \subseteq X$ of size k that maximizes the value $\|S_A\|$. Since translating the input set (as well the solution) of a given quantity only requires $O(n)$ time, in our algorithms we always assume that the origin $(0,0)$ is the centroid of input set.

Note that for $k = 1$ the 2-SCC-2 problem is solved in $O(n)$ time just applying Proposition 1: after translating the input set X so that $C_X = (0,0)$, it is sufficient to find a point with maximum ℓ_2 norm. For this reason, hereafter we assume $n \geq 4$.

Another key property in the case of Euclidean norm is the following separation result that immediately follows from Theorem 1.

Corollary 1 (Separation Result in ℓ_2). *Let $\{A, B\}$ be an optimal solution of the 2-SCC-2 problem under ℓ_2 norm for an instance $X \subset \mathbb{R}^2$ with constraint $|A| = k$. Then, there exists a constant $c \in \mathbb{R}$ such that, for every $u \in X$,*

$$\begin{aligned} u \in A \quad &\Rightarrow \quad 2u_x(C_{Bx} - C_{Ax}) + 2u_y(C_{By} - C_{Ay}) \;<\; c + \|C_B\|^2 - \|C_A\|^2, \\ u \in B \quad &\Rightarrow \quad 2u_x(C_{Bx} - C_{Ax}) + 2u_y(C_{By} - C_{Ay}) \;>\; c + \|C_B\|^2 - \|C_A\|^2. \end{aligned}$$

As a consequence, both clusters of any optimal solution $\{A, B\}$ of the 2-SCC-2 problem under ℓ_2 norm are separated by the straight line of equation

$$2x(C_{Bx} - C_{Ax}) + 2y(C_{By} - C_{Ay}) = c + \|C_B\|^2 - \|C_A\|^2$$

for some constant $c \in \mathbb{R}$. This implies that A is a k-set and B is a $(n-k)$-set.

This result can be interpreted as a natural extension of the well-known String Property, stating that any optimal clustering on the real line consists of subsets of the input set whose convex closures are pairwise disjoint [26, 21]. Moreover, by the previous discussion, it implies that the problem is essentially solved by computing a k-set $A \subseteq X$ with maximum norm of sum $\|S_A\|$.

4. Some notions of Computational Geometry

Our procedures for solving the 2-SCC-2 problem under Euclidean norm, require some tools and results of computational geometry we recall in this section [12, 23].

First of all we fix a total order on points in $\mathbb{R}^2$: for every $a, b \in \mathbb{R}^2$, we set $a <_y b$ if either $a_y < b_y$ or $a_y = b_y \wedge a_x < b_x$.

Given two points $a, b \in \mathbb{R}^2$, the oriented line segment from a to b is called *oriented edge* and is identified by the pair (a, b). By a little abuse of language we also denote by (a, b) the *straight line* through the two points oriented from a to b. We define the (positive) *phase* of (a, b) as the angle between the oriented edges $(a, (a_x + 1, a_y))$ and (a, b), measured counter-clockwise. We denote it by $\mathit{phase}(a, b)$. Clearly, $0 \le \mathit{phase}(a, b) < 2\pi$. Note that two oriented edges have the same phase if and only if they are parallel and have the same orientation.

We also define the *slope* of (a, b) as the remainder of the division $\mathit{phase}(a, b)/\pi$ and we denote it by $\mathit{slope}(a, b)$. Observe that (a, b) and (b, a) have the same slope and hence two oriented edges have the same slope if and only if they are parallel (with either equal or opposite orientation).

Moreover, for every pair of oriented edges (a, b), (c, d), we say that (a, b) is *on the right* of (c, d) and write $(a, b)\mathrm{Right}(c, d)$ if they have different slope and the straight line (c, d) is obtained by a counter-clockwise rotation of the straight line (a, b) (around their intersection point) smaller than π. On the contrary, if such a rotation is greater than π we say that (a, b) is *on the left* of (c, d) and write $(a, b)\mathrm{Left}(c, d)$. Note that relations *Right* and *Left* correspond respectively to the positive and negative sign of the cross product $(a, b) \times (c, d)$, which is determined by the well-known right-hand rule. Relations *Right* and *Left* also extend in obvious way to pairs of oriented straight lines in the plane.

Clearly, once the coordinates of points are known, one can establish in constant time whether relations *Right* or *Left* hold between two oriented edges. Also note that for any 4-tuple of distinct points $a, b, c, d \in \mathbb{R}^2$, the following implications hold:

$$(a, b)\mathrm{Right}(c, d) \iff (c, d)\mathrm{Left}(a, b) \iff (d, c)\mathrm{Right}(a, b) \iff (d, c)\mathrm{Left}(b, a)$$

Moreover, if a set $X \subset \mathbb{R}^2$ is in general position and $a, b, c \in X$ are distinct then either $(a, b)\mathrm{Right}(b, c)$ or $(a, b)\mathrm{Left}(b, c)$ holds.

Other notions we need in this work concern convex sets in $\mathbb{R}^2$ and k-set. Recall that the intersection of an arbitrary collection of convex sets is convex. Then for any set $A \subseteq \mathbb{R}^2$, the *convex hull* or *convex closure* of A, denoted by $\mathrm{Conv}(A)$, is defined as the smallest convex subset of $\mathbb{R}^2$ containing A, i.e.

$$\mathrm{Conv}(A) = \bigcap \{Y \subseteq \mathbb{R}^2 : A \subseteq Y, Y \text{ is convex}\}$$

It is well-known that the convex closure of a finite set A of points in $\mathbb{R}^2$ is a convex polygon. It is possible to identify a polygon by giving its vertices, and hence the determination of the convex closure $\mathrm{Conv}(A)$ of a given set $A \subset \mathbb{R}^2$ consists in finding the vertices of the associated polygon. We recall that the convex hull of a set X of n points in $\mathbb{R}^2$ can be computed in time $O(n \log n)$ [23]. Moreover, a dynamic data structure for convex hulls can be designed which allows one to perform Insert and Delete operations in $O(\log^2 n)$ time [22].

Now, given a finite set $A \subset \mathbb{R}^2$ in general position, let $a, b, c \in \mathbb{R}^2$ be three counter-clockwise consecutive vertex points on the perimeter of $\mathrm{Conv}(A)$. Then,

$$(a, b)\mathrm{Right}(b, c) \tag{4}$$

holds.

Now, turn our attention to k-sets. For any $X \subset \mathbb{R}^2$ of cardinality n and any integer k such that $1 \leq k \leq n$, a *k-set* of X is a subset $A \subseteq X$ of k points such that $A = X \cap H$ for a suitable half-plane $H \subset \mathbb{R}^2$. This means that A is separable from $\bar{A} = X \setminus A$ by a straight line ℓ, where $A \cap \ell = \emptyset = \ell \cap \bar{A}$. We also say that ℓ is a *separating line* of A. Determining the maximum number of k-sets of a family of n points in $\mathbb{R}^2$ is a central problem in combinatorial geometry, first posed in [13]. An upper bound to this quantity is given by the following result, obtained in [11].

Theorem 2. *For any set X of n points in $\mathbb{R}^2$ and any $k \in \mathbb{N}$, $1 \leq k \leq \lfloor \frac{n}{2} \rfloor$, the number of k-sets of X is less than $6.48n\sqrt[3]{k}$.*

5. Solving the plain problem under Euclidean norm

In this section we present an efficient technique, based on Proposition 1, to solve the 2-SCC-2 problem under ℓ_2 norm. The input is given by a set $X \subset \mathbb{R}^2$ of $n \geq 4$ points in general position with $C_X = (0, 0)$, and an integer k such that $1 < k \leq n/2$. Our purpose is to show that the algorithm works in $O(n\sqrt[3]{k} \log^2 n)$ time.

We first introduce some preliminary notions. Given two disjoint polygons it is easy to see that there exist 4 straight lines tangent to both polygons: they are called *bitangents*. Two bitangents keep one polygon on one side and the other polygon on the other side, while the other two bitangents keep both polygons on the same side. Bitangents as well as straight lines can be oriented.

Given two points $a, b \in X$ we define

$$X^r(a, b) = \{p \in X \mid (a, p)\mathrm{Right}(a, b)\},$$
$$X^l(a, b) = \{p \in X \mid (a, p)\mathrm{Left}(a, b)\}.$$

In other words, $X^r(a, b)$ is the set of points in X lying on the right-hand side of the oriented edge (a, b), while $X^l(a, b)$ is the set of points in X lying on the left-hand side of (a, b).

Definition 1. *For any $a, b \in X$, $a \neq b$, we say that the oriented edge (a, b) is a* $(k-1)$-set edge *if $|X^r(a, b)| = k - 1$ (and hence $|X^l(a, b)| = n - k - 1$).*

Moreover, we define $\mathcal{S}(a, b) = X^r(a, b) \cup \{a\}$ and $\overline{\mathcal{S}}(a, b) = X^l(a, b) \cup \{b\}$. Clearly, if (a, b) is a $(k-1)$-set edge then $\mathcal{S}(a, b)$ is a k-set and the straight line (a,b) is the unique bitangent between $\mathrm{Conv}(\mathcal{S}(a,b))$ and $\mathrm{Conv}(\overline{\mathcal{S}}(a,b))$ that keeps $X^r(a,b)$ on its right-hand side. Indeed, as illustrated in Figure 1a, two of the other three bitangents do not separate the two sets, and the remaining one does not keep $X^r(a,b)$ on its right-hand side. This proves the following proposition for any set $X \subset \mathbb{R}^2$ of $n \geq 4$ points in general position and any $k = 2, \ldots, \lfloor n/2 \rfloor$.

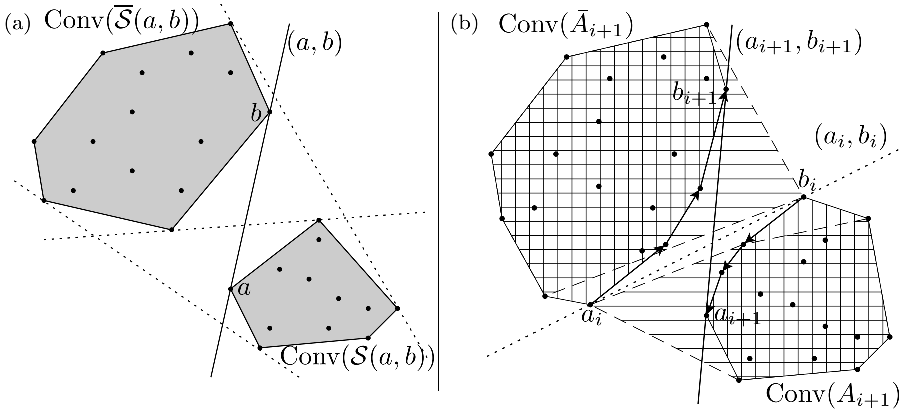

Figure 1: (a) The 4 bitangents between $\mathrm{Conv}(\mathcal{S}(a,b))$ and $\mathrm{Conv}(\overline{\mathcal{S}}(a,b))$. (b) Given the k-set $A_i = \mathcal{S}(a_i,b_i)$ (polygon with horizontal lines) we can compute the subsequent k-set $A_{i+1} = \mathcal{S}(a_{i+1},b_{i+1})$ (polygon with vertical lines) by removing a_i and inserting b_i in the convex hull. Segments with arrow represent oriented edges scanned by procedure NextBitangent on the perimeter of the convex hulls.

Proposition 2. *There exists a bijection between the $(k-1)$-set edges of X and the k-sets of X: each $(k-1)$-set edge (a,b) can be associated with the k-set $\mathcal{S}(a,b)$, while any k-set A corresponds to the unique oriented edge (a,b), bitangent to Conv(A) and Conv$(\bar{A})$, such that $A = \mathcal{S}(a,b)$.*

Now, let us define the finite sequence of distinct $(k-1)$-set edges

$$E_{k-1} = \{(a_1,b_1), ..., (a_h,b_h)\}$$

such that (a_1,b_1) is the $(k-1)$-set edge of X of smallest phase and, for each (a_i,b_i), the subsequent pair (a_{i+1},b_{i+1}) is the $(k-1)$-set edge associated with the k-set obtained by exchanging points a_i and b_i, i.e.

$$\mathcal{S}(a_{i+1},b_{i+1}) = \mathcal{S}(a_i,b_i) \cup \{b_i\} \smallsetminus \{a_i\}. \tag{5}$$

Figure 1b illustrates the construction.

It is clear that $\mathit{phase}(a_i,b_i) < \mathit{phase}(a_{i+1},b_{i+1})$ and hence E_{k-1} is totally ordered w.r.t. increasing phase. Actually, E_{k-1} includes all $(k-1)$-set edges of X. Indeed, assume on the contrary that there exists a $(k-1)$-set edge $\beta \notin E_{k-1}$; then there are two bitangents $\delta = (a_i,b_i)$, $\eta = (a_{1+\langle i \rangle_h}, b_{1+\langle i \rangle_h}) \in E_{k-1}$ such that $\beta \mathrm{Left} \delta$ and $\beta \mathrm{Right} \eta$. Let ℓ be the oriented straight line through point $\delta \cap \eta$ having the same phase of β. Since $\ell \neq \beta$ the set $\mathcal{L}$ of all points in X lying on the right hand side of ℓ, cannot have cardinality k. However, this is a contradiction since $\mathcal{L} = \mathcal{S}(\eta)$ and $\mathcal{S}(\eta)$ is a k-set. This proves the following

Lemma 1. *Sequence E_{k-1} includes all $(k-1)$-set edges of X.*

Now observe that, setting

$$S_i = \sum_{p \in \mathcal{S}(a_i, b_i)} p \tag{6}$$

by equality (5) we have $S_{i+1} = S_i - a_i + b_i$, for each $i = 1, \ldots, h-1$. This suggests to solve the 2-SCC-2 problem by first computing the oriented edge (a_1, b_1) and then determining (a_{i+1}, b_{i+1}) from (a_i, b_i), for each $i = 1, 2, ..., h-1$, updating the current value S_i at each iteration and keeping the maximum one. By Proposition 1 the pair (a_i, b_i) with maximum S_i yields to the optimal solution.

In order to compute E_{k-1}, let us start with the first element.

Proposition 3. *Given a set $X \subset \mathbb{R}^2$ of $n \geq 4$ points in general position, for any $k = 2, \ldots, \lfloor n/2 \rfloor$ the $(k-1)$-set edge of smallest phase can be computed in $O(n \log n)$ time.*

Proof. First, it is easy to determine the k-th smallest point a in X with respect to the total order $<_y$, together with set $A = \{q \in X \mid q \leq_y a\}$. Clearly, A is a k-set of X. Similarly, we can determine the $(k+1)$th element b in X with respect to $<_y$, i.e. the smallest point in $\bar{A} = X \smallsetminus A$. To determine the $(k-1)$-set edge associated with A, the algorithm first computes the convex hulls $\mathrm{Conv}(A)$ and $\mathrm{Conv}(\bar{A})$. Then, starting from a and b, it moves two points u and v counter-clockwise on the perimeter of $\mathrm{Conv}(A)$ and $\mathrm{Conv}(\bar{A})$, respectively, stopping at the first edge on $\mathrm{Conv}(A)$ (respectively, on $\mathrm{Conv}(\bar{A})$) that is on the right (respectively, on the left) of the oriented edge (u, v).

The procedure is formally described by Procedure FirstSetEdge given below, where for every vertex point p on the perimeter of a convex hull, $p' = \mathrm{Succ}(p)$ is the counter-clockwise successor of p on the same perimeter.

```
Procedure FirstSetEdge($a$,$b$)
begin
 $u := a$; $u' := \mathrm{Succ}(u)$
 $v := b$; $v' := \mathrm{Succ}(v)$
 while $(u,u')\mathrm{Left}(u,v) \ \vee \ (v,v')\mathrm{Right}(u,v)$ do
       if $(v,v')\mathrm{Left}(u,v)$
          then $u := u'$; $u' := \mathrm{Succ}(u)$
          else if $(u,u')\mathrm{Right}(u,v)$
                 then $v := v'$; $v' := \mathrm{Succ}(v)$
                 else if $(u,u')\mathrm{Right}(v,v')$
                        then $v := v'$; $v' := \mathrm{Succ}(v)$
                        else $u := u'$; $u' := \mathrm{Succ}(u)$
 return $(u,v)$
end
```

To enter the while loop, at each iteration the procedure checks whether the current edges (u, u') and (v, v') on the two perimeters verify the exit condition

$$(u, u')\mathrm{Right}(u, v) \ \wedge \ (v, v')\mathrm{Left}(u, v), \tag{7}$$

which guarantees $A \setminus \{u\} = X^r(u, v)$ and $\bar{A} \setminus \{v\} = X^l(u, v)$. In the affirmative case, (u, v) is the required $(k-1)$-set edge and hence the while loop is not entered and the procedure ends.

On the contrary if (7) is not satisfied the procedure enters the while loop and there are three possible cases. Either $(v, v')\mathrm{Left}(u, v)$ holds and then u is moved one step forward because (v, v') already satisfies the required condition (see Fig. 2a) or, symmetrically, $(u, u')\mathrm{Right}(u, v)$ and then v is moved one step forward (Fig. 2b), or both $(u, u')\mathrm{Left}(u, v)$ and $(v, v')\mathrm{Right}(u, v)$ hold. In such a third case one has to choose which point between u and v is to be moved. To this end, the procedure checks whether $(u, u')\mathrm{Right}(v, v')$ holds; in the affirmative case v is chosen since the half-line (u', u) could intersect $\mathrm{Conv}(\bar{A})$, in which case the current u is the required point and no further move on the perimeter of $\mathrm{Conv}(A)$ would be correct (Fig. 2c). For a symmetric reason, in the negative case (i.e. $(u, u')\mathrm{Right}(v, v')$ does not hold), point u is moved (Fig. 2d). Note that in the parallel case, when (u, u') and (v, v') have same slope (but opposite phase), both points could be moved (here the procedure chooses u first).

The most expensive operation in the overall procedure is the computation of the convex hulls $\mathrm{Conv}(A)$ and $\mathrm{Conv}(\bar{A})$, which can be done in $O(n \log n)$ time [23]. Also note that the procedure FirstSetEdge requires $O(n)$ time. □

To illustrate the execution of Procedure FirstSetEdge, consider the example of Figure 3, where X is a set of $n = 16$ points and $k = 7$. Points a and b are the 7-th and 8-th elements w.r.t. $\leq_y$, respectively. Therefore, at the beginning, $u = a, v = b, u' = a', v' = b'$. The procedure first moves v once, setting $v = b'$, and then moves u twice, reaching $u = a''$. At this point the while-loop condition is no more satisfied and hence the $(k-1)$-set edge (a'', b') is returned.

Once the first element of E_{k-1} is determined, one can compute all the other $(k-1)$-set edges of E_{k-1} in phase order. For each $(a_i, b_i) \in E_{k-1}$ we can also determine the squared norm of S_i, defined in (6), maintaining in a variable q the largest value. The details are given in Algorithm 1, where procedure NextBitangent, called at lines 10 and 16, yields the subsequent $(k-1)$-set edge in phase order.

Procedure NextBitangent is defined by Algorithm 2, which uses function *Succ* defined in the proof of Proposition 3. It first computes the convex hulls $\mathcal{A}$ and $\bar{\mathcal{A}}$ of the new k-set A and of its complement $\bar{A}$ by means of two Insert and Delete operations. Then, in the main loop, the procedure determines the $(k-1)$-set edge associated with the new k-set by following a path counter-clockwise on the perimeter of the two convex hulls as in Procedure FirstSetEdge. Note that the exit condition is the same as (7), and it guarantees that (u, v) is the required $(k-1)$-set edge. Moreover, here a repeat-until instruction is used, rather than a while-loop, since we are sure that at least one iteration is executed. The correctness of the main loop is proved reasoning as in Proposition 3.

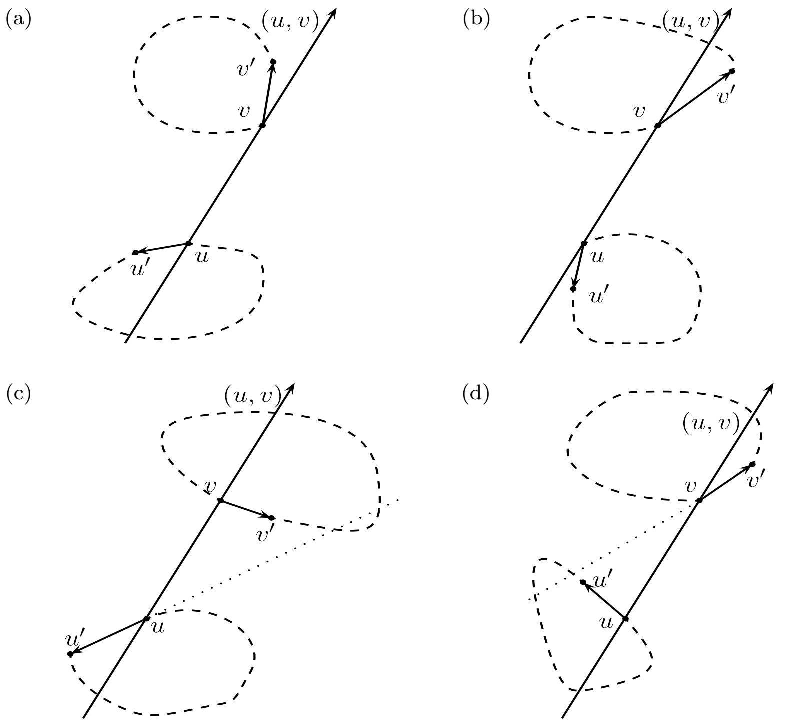

Figure 2

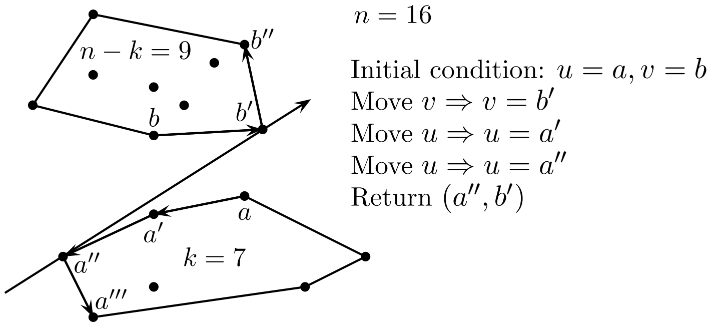
$n - k = 9$
b''
b
b'
a
a'
a''
a'''
$k = 7$
$n = 16$
Initial condition: $u = a, v = b$
Move $v \Rightarrow v = b'$
Move $u \Rightarrow u = a'$
Move $u \Rightarrow u = a''$
Return (a'', b')

Figure 3: Execution of FirstSetEdge(a, b) for a point set X of size $n = 16$ and $k = 7$.

Algorithm 1 Solving 2-SCC-2 under ℓ_2 norm

```
Input: a set $X \subset \mathbb{R}^2$ of $n \geq 4$ points in general position for which $C_X = (0, 0)$;
    an integer $k$ such that $1 < k \leq \lfloor n/2 \rfloor$.
Output: the solution $\pi = \{A, B\}$ of the 2-SCC-2 problem on instance $X$ with
    constraint $|A| = k$.
 1: Compute the $k$-th smallest point $a$ in $X$ with respect to $<_y$
 2: $A := \{p \in X : p \leq_y a\}$; $\bar{A} := X \smallsetminus A$
 3: Compute the smallest point $b$ in $\bar{A}$ with respect to $<_y$
 4: $\mathcal{A} :=$ Conv$(A)$
 5: $\bar{\mathcal{A}} :=$ Conv$(\bar{A})$
 6: $(a_1, b_1) :=$ FirstSetEdge$(a, b)$
 7: $S := \sum_{x \in A} x$
 8: $q := \|S\|^2$
 9: $(x, y) := (a_1, b_1)$
10: $(r, s) :=$ NextBitangent$(a_1, b_1)$
11: while $(r, s) \neq (a_1, b_1)$ do
12:     $S := S - r + s$
13:     if $q < \|S\|^2$ then
14:         $q := \|S\|^2$
15:         $(x, y) := (r, s)$
16:     $(r, s) :=$ NextBitangent$(r, s)$
17: $\pi := \{X^r(x, y) \cup \{x\}, X^l(x, y) \cup \{y\}\}$
18: return $\pi$
```

Thus, Algorithm 1 computes all oriented edges of E_{k-1} in phase order and hence, by Lemma 1 and Proposition 2, we obtain the following

Theorem 3. *Algorithm 1 solves the 2-SCC-2 problem under ℓ_2 norm.*

Now, let us study the complexity of the algorithm. As shown below this is mainly based on Theorem 2.

Theorem 4. *For an input (X, k) where $|X| = n$ and $1 < k \leq \lfloor \frac{n}{2} \rfloor$, Algorithm 1 works in $O(n\sqrt[3]{k} \cdot \log^2 n)$ time and $O(n)$ space.*

Proof. First recall that, by Proposition 3, the first part of the algorithm from line 1 to line 8, can be executed in $O(n \log n)$ time. The remaining computation time is dominated by calls to procedure NextBitangent. The algorithm maintains the convex hulls $\mathcal{A}$ and $\bar{\mathcal{A}}$, which are updated at lines 1–2 of each call of NextBitangent in $O(\log^2 n)$ time by using the dynamic data structure for convex hulls introduced in [22]. Since there is just one call to NextBitangent for each $(k-1)$-set edge, by Proposition 2 and Theorem 2 the total cost of all updates of $\mathcal{A}$ and $\bar{\mathcal{A}}$ is $O(n\sqrt[3]{k}\log^2 n)$.

The time cost of the other operations of NextBitangent is due to the repeat-until loop of the procedure. Here, the key observation is that each edge (u, u')

Algorithm 2 `Procedure` NextBitangent(r, s)

```
Input: a $(k-1)$-set edge $(r, s)$ of $X$ computed in Algorithm 1.
Output: the subsequent $(k-1)$-set edge of $X$ in phase order.
$\mathcal{A} :=$ Insert(Delete($\mathcal{A}, r$),$s$)
$\bar{\mathcal{A}} :=$ Insert(Delete($\bar{\mathcal{A}}, s$),$r$)
$u := s$; $u' :=$ Succ($u$)
$v := r$; $v' :=$ Succ($v$)
repeat
    if $(v, v')$Left$(u, v)$
        then $u := u'$; $u' :=$Succ($u$)
        else if $(u, u')$Right$(u, v)$
            then $v := v'$; $v' :=$Succ($v$)
            else if $(u, u')$Right$(v, v')$
                then $v := v'$; $v' :=$Succ($v$)
                else $u := u'$; $u' :=$Succ($u$)
until $(u, u')$Right$(u, v)$ $\wedge$ $(v, v')$Left$(u, v)$
return $(u, v)$
```

inside the repeat-until loop is a $(n-k)$-set edge, because the phase of the opposite edge (u', u) is included between the phases of two consecutive $(k-1)$-set edges. These edges (u, u') scan counter-clockwise the perimeter of $\mathcal{A}$ and each of them is considered just by one call of NextBitangent. The same occurs for the k-set edges (v, v') scanning counter-clockwise the perimeter of $\bar{\mathcal{A}}$. Therefore, the time required by the main loop in all calls to NextBitangent is at most proportional to $|E_k| + |E_{n-k}| = 2|E_k|$, which is again $O(n\sqrt[3]{k})$ by Theorem 2. Thus, the time cost of all calls to NextBitangent turns out to be $O(n\sqrt[3]{k}\log^2 n)$.

The space required by the computation is dominated by the data structure maintaining the two current convex hulls, which consists of augmented binary trees of overall size $O(n)$ (see [23, p.127]). □

6. Solving the full problem under Euclidean norm

In this section we present an algorithm to solve the problem for all sizes of clusters. The procedures receives an input set $X \subset \mathbb{R}^2$ of $n \geq 4$ points in general position with $C_X = 0$ and, for every $k = 1, 2, \ldots, \lfloor n/2 \rfloor$, returns an optimal 2-clustering $\{A_k, \bar{A}_k\}$ of X such that $|A_k| = k$. This algorithm works in $O(n^2 \log n)$ time and $O(n)$ space.

For our purpose we introduce a further relation. For every pair of distinct points $a, b \in X$, we say that the oriented edge (a, b) is *associated with* the 2-clustering $\{A, \bar{A}\}$ of X such that either A or $\bar{A}$ equals the set $\mathcal{R}(a, b)$ defined by

$$\mathcal{R}(a,b) = \begin{cases} X^r(a,b) \cup \{b\} & \text{if } a <_y b \\ X^l(a,b) \cup \{b\} & \text{otherwise.} \end{cases}$$

Note that here $\{A, \bar{A}\}$ is an unordered pair of sets. Moreover, A and $\bar{A}$ are, respectively, a k-set and a $(n-k)$-set for some $k \in \{1, 2, \dots, n-1\}$. Also observe that (a, b) and (b, a) are always associated different 2-clusterings.

Lemma 2. *Given a set $X \subset \mathbb{R}^2$ of n points in general position, let A be a k-set of X for some $k \in \{1, 2, \dots, n-1\}$. Then $\{A, \bar{A}\}$ is the 2-clustering of X associated with an oriented edge (a, b), for some $a, b \in X$.*

Proof. Since A is a k-set we can consider a bitangent (u, v) that separates A and $\bar{A}$ with $u \in \bar{A}$ and $v \in A$. Assume that $u <_y v$: if $A = X^r(u, v) \cup \{v\}$ then (u, v) is the required oriented edge because $A = \mathcal{R}(u, v)$; otherwise, A equals $X^l(u, v) \cup \{v\}$ and the same 2-clustering $\{A, \bar{A}\}$ is associated with (v, u) because $\bar{A} = \mathcal{R}(v, u)$. A symmetric reasoning holds in case $v <_y u$. □

Note that in the previous proof we can choose the bitangent (u, v) separating A and $\bar{A}$ in two different ways. This proves that every k-sets of X is associated with two oriented edges. Hence, such a correspondence is quite different from the bijection of Proposition 2 introduced in the previous section.

By the proposition above, one can design an algorithm that scans all k-sets by considering in some order all oriented edges outgoing from each point. In order to compute efficiently the weights of the clusters we introduce a special order among the oriented edges $E = \{(a, b) \mid a, b \in X, a \neq b\}$: for every $(u, v), (w, z) \in E$, we define $(u, v) <_e (w, z)$ if either $u <_y w$ or $u = w$ and $slope(u, v) < slope(w, z)$.

Theorem 5. *For any input set of n points in $\mathbb{R}^2$ (in general position), Algorithm 3 solves the 2-SCC-2 problem for all $k = 1, 2, \dots, \lfloor n/2 \rfloor$ in $O(n^2 \log n)$ time.*

Proof. The procedure computes point $S_{\mathcal{R}(a,b)}$ (as defined in Proposition 1) for every oriented edge $(a, b) \in E$, taken according to the total order $<_e$. For each $k = 1, 2, \dots, \lfloor n/2 \rfloor$, the procedure maintains the maximal value $q[k] = \|S_A\|^2/k(n-k)$, where A is a k-set of X and stores in $e[k]$ the associated oriented edge. Corollary 1 and Lemma 2 guarantee that the procedure considers all 2-clusterings of X that are feasible solutions.

The computation first considers all points in X in the order $<_y$ and, for each $a \in X$, it determines $R = \sum_{p \in X, p <_y a} p$. Then, it computes $S_{\mathcal{R}(a,b)}$ for every edge (a, b) such that $b \in X \setminus \{a\}$ in the order $<_e$: for any pair of consecutive edges (a, b), (a, c), the value $S_{\mathcal{R}(a,c)}$ is obtained from $S_{\mathcal{R}(a,b)}$ by adding or subtracting c according whether $a <_y c$ or $c <_y a$ (see instructions 10 and 14, respectively). Note that such a computation only requires constant time.

The time complexity of the algorithm is dominated by the operation of sorting the oriented edges with the same starting point a. This can be done in $O(n \log n)$ time for each $a \in X$. Note that the other operations, in the inner for-loop, require at most constant time and are executed $O(n^2)$ many times. Therefore, the overall time of the algorithm is $O(n^2 \log n)$. □

It is also clear that the previous algorithm works in $O(n)$ space.

Algorithm 3 Solving full 2-SCC-2 problem under ℓ_2 norm

```
Input: a set $X \subset \mathbb{R}^2$ of $n$ points in general position, such that $C_X = (0, 0)$
Output: the sequence $(e[1], \ldots, e[\lfloor n/2 \rfloor])$ of oriented edges, where each $e[k]$ is
    associated with the solution $\{A_k, \bar{A}_k\}$ of 2-SCC-2 for $X$ such that $|A_k| = k$
for $k = 1, 2, \ldots, \lfloor n/2 \rfloor$ do
    $q[k] := 0$
$T := \sum_{p \in X} p$
$(a_1, a_2, \ldots, a_n) :=$ Sort($X$) w.r.t. $<_y$
for $i = 1, 2, \ldots, n$ do
    $R := \sum_{j<i} a_j$
    $g := i - 1$
    $(b_1, b_2, \ldots, b_{n-1}) :=$ Sort($X \setminus \{a_i\}$) w.r.t. $slope(a_i, \cdot)$
    for $j = 1, 2, \ldots, n-1$ do
        if $a_i <_y b_j$ then $\begin{cases} R := R + b_j \\ g := g + 1 \end{cases}$
        if $g \leq n - g$ then $\begin{cases} m := g \\ S := R \end{cases}$
                        else $\begin{cases} m := n - g \\ S := T - R \end{cases}$
        if $q[m] < \frac{\|S\|^2}{g(n-g)}$ then $\begin{cases} q[m] := \frac{\|S\|^2}{g(n-g)} \\ e[m] := (a_i, b_j) \end{cases}$
        if $b_j <_y a_i$ then $\begin{cases} R := R - b_j \\ g := g - 1 \end{cases}$
return $(e[1], \ldots, e[\lfloor n/2 \rfloor])$
```

7. The 2-SCC-2 problem under Manhattan norm

In this section we study the 2-clustering problem on the plane assuming the Manhattan norm, also called ℓ_1-norm. This is defined by setting $\|a\|_1 = |a_x| + |a_y|$ for any $a \in \mathbb{R}^2$. In this case the definition of centroid of a set $A \subset \mathbb{R}^2$, given in (1), yields the value

$$C_A = \underset{\mu \in \mathbb{R}^2}{\operatorname{argmin}} \sum_{a \in A} \left(|a_x - \mu_x| + |a_y - \mu_y| \right)$$

Here the function to be minimized is not strictly convex, and hence C_A is not necessarily unique. Indeed, now C_A is the component-wise median of the elements of A; in other words, C_{Ax} and C_{Ay} are the medians, respectively, of the abscissae and of the ordinates of the points in A. Moreover, the weight of A is here given by

$$W(A) = \sum_{a \in A} \big(|a_x - C_{Ax}| + |a_y - C_{Ay}| \big)$$

This allow us to extend the separation result of Theorem 1 to the case $p = 1$. The new statement is weaker than the previous one ($<$ is replaced by $\leq$) and the separating curves of the optimal solutions now are piecewise straight lines of special forms.

Lemma 3 (Swapping Lemma). *Let $\{A, B\}$ be an optimal solution of the 2-SCC-2 problem under ℓ_1 norm. Then, for any $a \in A$ and $b \in B$, we have*

$$||a - C_A||_1 + ||b - C_B||_1 \leq ||a - C_B||_1 + ||b - C_A||_1 . \tag{8}$$

Proof. By contradiction, assume there exists $a \in A$ and $b \in B$ such that

$$||a - C_A||_1 + ||b - C_B||_1 > ||a - C_B||_1 + ||b - C_A||_1$$

and define $A' = A \smallsetminus \{a\} \cup \{b\}$ and $B' = B \smallsetminus \{b\} \cup \{a\}$. This implies

$$\begin{aligned} W(A, B) &= \sum_{u \in A} ||u - C_A||_1 + \sum_{v \in B} ||v - C_B||_1 \\ &> \sum_{u \in A'} ||u - C_A||_1 + \sum_{v \in B'} ||v - C_B||_1 \\ &\geq \sum_{u \in A'} ||u - C_{A'}||_1 + \sum_{v \in B'} ||v - C_{B'}||_1 = W(A', B') \end{aligned}$$

proving that $W(A, B) > W(A', B')$, which is a contradiction since $\{A, B\}$ is an optimal solution. □

Proposition 4 (Separation Result in ℓ_1). *Let $\{A, B\}$ be an optimal solution of the 2-SCC-2 problem under ℓ_1 norm for an instance $X \subset \mathbb{R}^2$ with constraint $|A| = k$. Then, there exists a constant $g \in \mathbb{R}$ such that, for every $u \in X$,*

$$\begin{aligned} u \in A &\quad \Rightarrow \quad ||u - C_A||_1 - ||u - C_B||_1 \leq g \\ u \in B &\quad \Rightarrow \quad ||u - C_A||_1 - ||u - C_B||_1 \geq g \end{aligned}$$

Proof. By Lemma 3 the inequality

$$||a - C_A||_1 - ||a - C_B||_1 \leq ||b - C_A||_1 - ||b - C_B||_1$$

is satisfied for all $a \in A$ and $b \in B$, and hence, taking the maximum over a's on the left and the minimum over b's on the right, we get:

$$c_A := \max_{a \in A}\{||a - C_A||_1 - ||a - C_B||_1\} \leq \min_{b \in B}\{||b - C_A||_1 - ||b - C_B||_1\} =: c_B$$

Therefore, choosing $g \in [c_A, c_B]$, all points $u \in A$ and $v \in B$ satisfy inequality $||u - C_A||_1 - ||u - C_B||_1 \leq g$ and $||v - C_A||_1 - ||v - C_B||_1 \geq g$, respectively. □

Thus, for variable z ranging over $\mathbb{R}^2$ equation

$$\|z - C_A\|_1 - \|z - C_B\|_1 = g \tag{9}$$

defines a curve separating the clusters of any optimal solution $\{A, B\}$ of the 2-SCC-2 problem.

Note that, if relation (8) holds with $\leq$ replaced by $=$, then both a and b lie on the curve of equation (9) and $W(A,B) = W(A',B')$, where $A' = A \setminus \{a\} \cup \{b\}$ and $B' = B \setminus \{b\} \cup \{a\}$, proving that also $\{A',B'\}$ is an optimal solution. Thus, if several points lie on the separating curve of an optimal solution then exchanging an equal number of such points between the two clusters yields again an optimal solution.

Proposition 5. *Let $\{A,B\}$ be defined as in Proposition 4 and let $\mathcal{C}$ be the corresponding separation curve of equation (9). Also define $\bar{A} = \{a \in A \mid a$ lie on $\mathcal{C}\}$ and $\bar{B} = \{b \in B \mid b$ lie on $\mathcal{C}\}$. Then, for every $\alpha \subseteq \bar{A}$ and every $\beta \subseteq \bar{B}$ such that $|\alpha| = |\beta|$, the bipartition $\{A',B'\}$ given by $A' = A \setminus \alpha \cup \beta$ and $B' = B \setminus \beta \cup \alpha$ is an optimal solution too.*

Setting the variable $z = (x,y)$, and the constants $C_A = (c,d)$, $C_B = (e,f)$, equation (9) becomes

$$|x-c| - |x-e| + |y-d| - |y-f| = g \tag{10}$$

Clearly, this equation always represents a connected curve on the plane, consisting of piecewise straight lines. A rather standard study of this equation, considering all possible values of c,d,e,f,g, leads to derive 8 types of separating curves, we denote by symbols C_1, C_2, C_3, C_4 and S_1, S_2, S_3, S_4, according to their shape. They are depicted in Figures 4 and 5. In particular cases these curves degenerate into rectangular regions bounded by two perpendicular half-lines with common vertex and parallel to the Cartesian axes. These curves are known in the literature as two-foci Taxicab hyperbolas [17] and are studied in Geometry as extension of traditional conics defined by means of Manhattan distance.

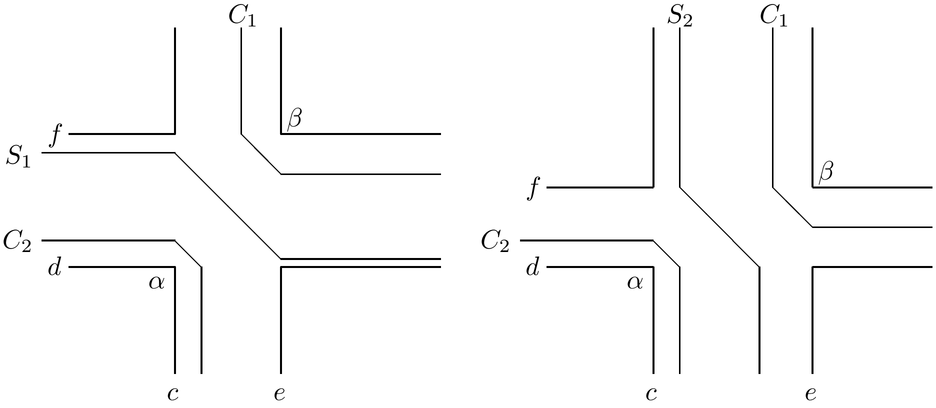

Figure 4: Curves C_1, C_2, S_1, S_2

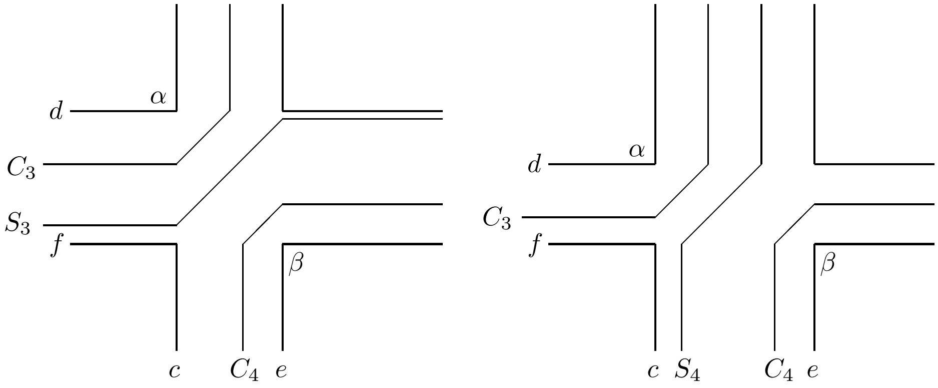

Figure 5: Curves C_3, C_4, S_3, S_4

8. Bipartition red-black trees

In this section we describe a data structure to maintain a bipartition of an arbitrary totally ordered finite set of elements. In addition to the traditional operations of membership, insertion and deletion, such a data structure allows us to execute efficiently two further operations: selecting the j-th smallest element in any cluster of the bipartition and moving an element from one cluster to the other.

Let U be a set of n distinct elements and let $\preceq$ be a total order relation over U. We understand that $u \prec v$ means $u \preceq v$ and $u \neq v$, for every $u, v \in U$. Consider an (ordered) bipartition (A, B) of U, i.e. a pair of subsets $A,\ B \subseteq U$, such that $A \cap B = \emptyset$ and $A \cup B = U$, and set $n_A = |A|$, $n_B = |B|$.

We define functions $\text{Select}_A^{\preceq}$ and $\text{Select}_B^{\preceq}$ by setting, for any $j = 1, 2, \ldots, n_A$, $\text{Select}_A^{\preceq}(j)$ equals to the j-th smallest element of A and, for any $i = 1, 2, \ldots, n_B$, $\text{Select}_B^{\preceq}(i)$ equals to the i-th smallest element of B.

Analogously, for every $u \in U$ we define $\text{Move}_{A \to B}^{\preceq}(u) = (A \backslash \{u\}, B \cup \{u\})$ if $u \in A$, while $\text{Move}_{A \to B}^{\preceq}(u) = \perp$ otherwise. Function $\text{Move}_{B \to A}^{\preceq}(u)$ is defined symmetrically.

A natural data structure, able to execute these operations in $O(\log n)$ time, is an augmented version of the well-known red-black trees [9]. Formally, a *Bipartition Red-Black Tree* (BRB-tree for short) for a bipartition (A, B) of U, with respect to a total order relation $\preceq$, is defined as a red-black tree T for $(U, \preceq)$ where the set of internal nodes coincides with U (while the leaves carry no information and can be considered as pointers to "nil") and every node $u \in U$ carries the values $\chi(u)$, $c_A(u)$ and $c_B(u)$, such that $\chi(u) = 1$ if $u \in A$, while $\chi(u) = 0$ if $u \in B$, and $c_A(u)$ (resp., $c_B(u)$) is the number of internal nodes in A (resp., in B) lying in the subtree rooted at u. Note that $c_A(r) = n_A$ and $c_B(r) = n_B$, where r is the root of T.

It is clear that a BRB-tree for a bipartition of a set of n elements can be constructed in time $O(n \log n)$ by adapting the traditional procedures on red-black trees [9]. The standard Membership, Insert and Delete procedures can be arranged for the new structure and still work in $O(\log n)$ time.

The new operations Select and Move are processed as follows. For each $j = 1, 2, \ldots, n_A$, $\mathrm{Select}_A^{\prec}(j)$ can be computed by a binary search for the j-th smallest element in A (say a), based on the values $c_A(left(u))$ for all nodes u along the path from r to vertex a. Clearly, for $j = 1$ and $j = n_A$, we obtain the minimum and the maximum element of A with respect to $\prec$, which we denote by $\min_{\prec}(A)$ and $\max_{\prec}(A)$, respectively. $\mathrm{Select}_B^{\prec}(j)$ can be computed similarly. To carry out the Move operation, one has to modify the BRB-tree representing the current bipartition. For every $u \in A$, to execute $\mathrm{Move}_{A \to B}^{\prec}(u)$ one first searches for node u in T, updating the coefficients $c_A(v)$ and $c_B(v)$ for every vertex v along the path from r to u; then $\chi(u)$ is set to 0. The execution of $\mathrm{Move}_{B \to A}^{\prec}$ is analogous. Since the tree is balanced, these procedures only require $O(\log n)$ time.

8.1. Updating the weight of clusters on the real line

As an auxiliary tool for our subsequent discussion, here we describe a method for updating the weight of a cluster of (possible equal) real numbers upon insertion and deletion.

Let α be a multi-set in $\mathbb{R}$, i.e. a finite set of real numbers where any value may occur several times. It can be represented as a sequence $\alpha = \{a_1, a_2, \ldots, a_k\}$, where $a_1 \leq a_2 \leq \cdots \leq a_k$. Under ℓ_1-norm the centroid of α is the median of its values, given by

$$M(\alpha) = \begin{cases} a_{\frac{k+1}{2}} & \text{if } k \text{ is odd} \\ \frac{1}{2}\left(a_{\frac{k}{2}} + a_{\frac{k}{2}+1}\right) & \text{if } k \text{ is even} \end{cases}$$

Note that $M(\alpha)$ can be computed by determining the j-th smallest element in α for one or two suitable j's. Then, given $m = \frac{k+1}{2}$, we define the left and the right sum of α as

$$L(\alpha) = \sum_{1 \leq i < m} a_i \ , \qquad R(\alpha) = \sum_{m < i \leq k} a_i$$

It is easy to see that the weight $W(\alpha) = \sum_{i=1}^{k} |a_i - M(\alpha)|$ can be computed from $R(\alpha)$ and $L(\alpha)$. Indeed, if k is even then m is not integer and hence

$$W(\alpha) = \sum_{1 \leq i < m} (M(\alpha) - a_i) + \sum_{m < i \leq k} (a_i - M(\alpha)) = -L(\alpha) + R(\alpha)$$

If k is odd an analogous reasoning holds. This leads to the following

Proposition 6. *For every sequence of real numbers $\alpha = \{a_1, a_2, \ldots, a_k\}$, where $a_1 \leq a_2 \leq \cdots \leq a_k$, we have $W(\alpha) = R(\alpha) - L(\alpha)$.*

Clearly, once $M(\alpha)$ is known, $W(\alpha)$ can be computed in $O(k)$ time. However, knowing $M(\alpha)$, $R(\alpha)$ and $L(\alpha)$, upon insertion of a real value x in α, the weight of the new multi-set can be computed in constant time simply by comparing x with $M(\alpha)$ and updating the left and right sums of the current multi-set. A similar reasoning holds upon deletion of an element x from α. These computations define two procedures, called UpdateW-ins(α, x) and UpdateW-del(α, x), which update the weight of a multi-set α of real numbers, using $M(\alpha)$, respectively upon insertion and deletion of a value $x \in \mathbb{R}$; since $M(\alpha)$ is known, both procedures work in time $O(1)$.

8.2. Updating the weight of a bipartition in the plane

Now, let us describe how to maintain the weight of a bipartition of a finite set of points in the plain, upon the operation of moving the elements from one cluster to the other.

Consider a set of n distinct points $X = \{p_1, p_2, \ldots, p_n\} \subset \mathbb{R}^2$, where $p_i = (x_i, y_i)$ for every i, and define the total order relations $\leq_x$ and $\leq_y$ over the set of indices $\{1, 2, \ldots, n\}$ (and hence over X), given by

$$\begin{array}{ll} 1) & i \leq_x j \ \text{ if } \ x_i < x_j \vee (x_i = x_j \wedge y_i \leq y_j) \\ 2) & i \leq_y j \ \text{ if } \ y_i < y_j \vee (y_i = y_j \wedge x_i \leq x_j) \end{array}$$

for every $i, j \in \{1, 2, \ldots, n\}$.

Let (A, B) be a bipartition of X and consider the corresponding multi-sets of coordinates $A_x = \{x_i \mid p_i \in A\}$, $B_x = \{x_j \mid p_j \in B\}$, $A_y = \{y_i \mid p_i \in A\}$, $B_y = \{y_j \mid p_j \in B\}$. Such a bipartition (A, B) can be maintained by a data structure including two BRB-trees, one w.r.t. $\leq_x$ and the other w.r.t. $\leq_y$, and a suitable array to keep the values $M(\alpha), L(\alpha), R(\alpha)$ for every $\alpha \in \{A_x, B_x, A_y, B_y\}$. Note that $\leq_x$ is a refinement of $\leq$ in A_x, in the sense that for any pair of distinct indices i, j, $x_i < x_j$ implies $i <_x j$ while if $x_i = x_j$ then either $i <_x j$ or $j <_x i$. Thus, for any $j \in \{1, 2, \ldots, n_A\}$, the j-th smallest element of A_x is computable in $O(\log n)$ time by applying $\text{Select}_A^{<_x}(j)$ and hence the same holds true for the computation of $M(A_x)$. This allows us to update the value $W(A_x)$, upon insertion (resp., deletion) of a point p_i in A (resp., from A), as described at the end of Section 8.1. An analogous reasoning can be carried out for the other multi-sets B_x, A_y, B_y.

Therefore, a complete subroutine $\text{Update}_{B \to A}(p_i)$ can be defined, which moves p_i from B to A (assuming $p_i \in B$) updating the entire data structures (both the BRB-trees and the array) and returns the weight of the new bipartition.

```
Procedure Update$_{B\to A}(p_i = (x_i, y_i))$
begin
    $W(A_x) :=$ UpdateW-ins$(A_x, x_i)$;  $W(B_x) :=$ UpdateW-del$(B_x, x_i)$
    $W(A_y) :=$ UpdateW-ins$(A_y, y_i)$;  $W(B_y) :=$ UpdateW-del$(B_y, y_i)$
    Move$^{<_x}_{B\to A}(p_i)$; Move$^{<_y}_{B\to A}(p_i)$
    for every $\alpha \in \{A_x, B_x, A_y, B_y\}$ do compute $M(\alpha)$
```

```
        return $(W(A_x) + W(A_y) + W(B_x) + W(B_y))$
end
```

Analogously, a symmetric subroutine $\text{Update}_{A\to B}(p_i)$ can be defined which moves a point p_i from A to B, updating the entire data structure and returning the weight of the new bipartition. By the discussion above, both these procedures work in $O(\log n)$ time.

In the following we use a procedure $\text{Start}(A, B)$ to initialize our data structure to a given bipartition (A, B). Such a procedure is based on standard subroutines managing red-black trees and works in $O(n \log n)$ time.

9. Solving the full problem under Manhattan norm

In this section we describe an algorithm for the 2-SCC-2 problem in the full version (i.e. for every size of cluster) assuming the Manhattan norm. Therefore the algorithm yields a solution also for the general (unconstrained) 2-Clustering problem in the plane. The computation is based on the data structure introduced in Section 8.2 and the separation result of Proposition 4, stating that any optimal solution of the 2-SCC-2 problem consists of two clusters separated by a curve of type C_i or S_i, for some $i = 1, 2, 3, 4$ (see Figures 4 and 5).

First we focus on the case of a separating curve of type C_1 and treat the other cases in Section 9.2. Recall that, by Proposition 5, the separating curve may pass through some points of the input set and, as described below, these points can be used to represent the curve itself. Also observe that here the input set is not assumed to be in general position.

9.1. Curves of type C_1

To fix notation, let the input of the procedure be a set of distinct points $X = \{p_1, p_2, \ldots, p_n\} \subset \mathbb{R}^2$, where $p_i = (x_i, y_i)$ for any $i = 1, 2 \ldots, n$. Without loss of generality we assume that all p_i's lie in the first quadrant, i.e. $x_i > 0$ and $y_i > 0$ for all i's. For technical reasons we add a further fictitious point p_0 of coordinates $x_0 = 0$ and $y_0 = 0$; we use relations $\leq_x$ and $\leq_y$ defined in section 8.2, extended to index 0 in obvious way, together with another total order over the set of indices, denoted by $\leq_{x+y}$, defined by

$$3)\quad i \leq_{x+y} j \ \text{ if } \ x_i + y_i < x_j + y_j \vee (x_i + y_i = x_j + y_j \wedge x_i \leq x_j)$$

for every $i, j \in \{0, 1, \ldots, n\}$.

The algorithm returns an array of pairs $(W[k], \Pi[k])$, for $k \in \{1, 2, \ldots, \lfloor n/2 \rfloor\}$, where $W[k]$ is the weight of an optimal bipartition (A, B) such that $k = \min\{|A|, |B|\}$, while $\Pi[k]$ is a triple of indices (i, j, t) representing a curve $\mathcal{C}$ of type C_1 that separates A and B. This means that $\mathcal{C}$ consists of a vertical half-line of abscissa x_i, a horizontal half-line of ordinate y_j and a segment on the line of equation $y + x = y_t + x_t$ connecting the two half-lines; moreover, A is the subset of X containing all points on the left-hand side of $\mathcal{C}$, including some points over $\mathcal{C}$, while B is its complement. Formally, we only consider triples (i, j, t), where $i, j, t \in \{0, 1, \ldots, n\}$, such that $i = 0 \vee j \leq_y i$, $j = 0 \vee i \leq_x j$, and $t = 0 \vee (i <_x t \wedge j <_y t)$; the bipartition (A, B) associated with such a triple (i, j, t) is given by

$$A = \{p_\ell \in X \mid \ell \leq_x i \vee \ell \leq_y j \vee \ell \leq_{x+y} t\},\ B = X \smallsetminus A \tag{11}$$

The procedure first sorts the index set with respect to $<_{x+y}$. Then the elements of X are processed by three for-loops, one nested into the other, corresponding to the values x_i, y_i and $x_i + y_i$, $i = 0, 1, \ldots, n$. For each abscissa x_i and each ordinate y_j it first computes the data structure of the bipartition (A, B) such that $A = \{p_u \in X \mid u \leq_x i \vee u \leq_y j\}$; then, it moves each $p_t \in B$ taken in order w.r.t. $<_{x+y}$, from B to A and, at each moving, it checks whether the new bipartition is better than those obtained previously having the same cluster sizes. For each admissible cluster size k, the procedure maintains the triple $\Pi[k] = (i, j, t)$ corresponding to the bipartition of minimum weight.

This procedure uses the data structure and the subroutines described in Section 8 to maintain the current bipartition and update its weight. The overall algorithm is described in detail by the scheme given below, where instruction Sort_{x+y} yields the list of the indices $\{1, 2, \ldots, n\}$ sorted according to $<_{x+y}$.

```
Procedure Full-2-clustering$_{C_1}(p_1, p_2, \ldots, p_n)$
begin
  $x_0 := 0$ ; $y_0 := 0$; $(t_1, t_2, \ldots, t_n) := \mathrm{Sort}_{x+y}(1, 2, \ldots, n)$
  for $k = 0, 1, \ldots, \lfloor n/2 \rfloor$ do
      $W[k] := +\infty$
      $\Pi[k] := \perp$
  for $i = 0, 1, \ldots, n$ do
      for $j = 0, 1, \ldots, n$ do
          if $(i = 0 \vee j \leq_y i) \wedge (j = 0 \vee i \leq_x j)$ then
             $h := 0$; $A := \emptyset$; $B := \{p_1, \ldots, p_n\}$; Start$(A, B)$; $Z := +\infty$
             for $\ell = 1, 2, \ldots, n$ do
                 if $\ell \leq_x i \vee \ell \leq_y j$ then
                     $Z := \mathrm{Update}_{B \to A}(p_\ell)$
                     $h := h + 1$
             $k := \min\{h, n - h\}$
             if $Z < W[k]$ then
                 $W[k] := Z$
                 $\Pi[k] := (i, j, 0)$
             for $u = 1, 2, \ldots, n$ do
                 if $i <_x t_u \wedge j <_y t_u$ then
                     $Z := \mathrm{Update}_{B \to A}(p_{t_u})$
                     $h := h + 1$; $k := \min\{h, n - h\}$
                     if $Z < W[k]$ then
                         $W[k] := Z$
                         $\Pi[k] := (i, j, t_u)$
  return $(W[k], \Pi[k],$ for $k = 1, 2, \ldots, \lfloor n/2 \rfloor)$
end
```

Taking into account the results of Section 8.2, one can see that the algorithm works in $O(n^3 \log n)$ time; moreover, it has a space complexity $O(n)$ since each BRB-tree only requires linear space.

9.2. Curves of other types

As far as the other types of separating curves are concerned, note that any curve of type C_2, C_3 and C_4 can be obtained from a curve of type C_1 by a rotation of the plane. Therefore, Procedure Full-2-clustering$_{C_1}$ can be easily modified to solve the problem when the separating curve is one of those types.

The case of separating curves of type S_i, $i = 1, .., 4$, can be treated in a similar way. An algorithm for type S_1, rather similar to Procedure Full-2-clustering$_{C_1}$, can be designed which mainly consists of three nested loops, where the corresponding loop variable ranges over the sorted values of the ordinates y_i (for the first two loops) and over the sum values $x_i + y_i$ (for the third one). The difference between the two procedures is purely technical, simply due to the shapes of types C_1 and S_1. Time and space evaluations remain the same and hence we omit the detailed description of the algorithm.

At last, note that any curve of type S_2, S_3 and S_4 can be obtained from a curve of type S_1 by a rotation or a reflection through a line parallel to a coordinate axis. Thus, also in those cases, the problem can be solved by similar procedures with the same complexity bounds.

10. Solving the plain problem under Manhattan norm

Now, still assuming ℓ_1 norm, we solve the same problem only for a given cluster size. Here the input is a pair (X, k) where $X = \{p_1, p_2, \ldots, p_n\} \subset \mathbb{R}^2$, with $p_i = (x_i, y_i)$, $x_i > 0$ and $y_i > 0$ for any $i = 1, 2 \ldots, n$, and $k \in \{1, 2, \ldots, \lfloor n/2 \rfloor\}$. The algorithm computes a bipartition (A, B) of X of minimum weight $W(A) + W(B)$ such that $k = \min\{|A|, |B|\}$. As in the previous section we study in detail only the case when the clusters of the solution are separated by a curve of type C_1. The algorithm is based on the program scheme given below which returns a pair (W, Π), where W is the weight of the solution (A, B) and Π is a triple of indices (i, j, t) representing (A, B) in the sense of equations (11). Thus, $\Pi = (i, j, t)$ denotes the separating curve as described in Section 9.1, A is the cluster lying bottom-left of the curve, while B is the cluster on its top-right side. The procedure finds the best solution (A, B) with $|A| = k$ and $|B| = n - k$; the same program scheme with k replaced by $n-k$, yields the other possible solution (A, B) such that $|A| = n - k$ and $|B| = k$.

```
Procedure Plain-2-clustering$_{C_1}$($\{p_1, \ldots, p_n\}, k$)
begin
  $x_0 := 0$ ; $y_0 := 0$; $(t_0, t_1, \ldots, t_n) := \mathrm{Sort}_{x+y}(0, 1, \ldots, n)$
  $W := +\infty$;  $\Pi := \perp$;  $A := \emptyset$;  $B := \{p_1, \ldots, p_n\}$;  Start$(A, B)$
  for $\ell = 0, 1, \ldots, k$ do
       if $\ell \neq 0$ then $Z := \mathrm{Update}_{B \to A}(p_{t_\ell})$
       $Q := \Lambda$
       repeat $k - \ell$ times $\left\{ \begin{array}{l} i := \min_{<_x}(B);\ Q := \mathrm{Push}(Q, i) \\ Z := \mathrm{Update}_{B \to A}(p_i) \end{array} \right.$
```

```
        if $Z < W$ then
            $W := Z$
            if $\ell = k$ then $i := 0$
            $\Pi := (i, 0, t_\ell)$
        $j := \min_{<_y}(B)$;  $T := \mathrm{Push}(\Lambda, j)$
        $Z := \mathrm{Update}_{B\to A}(p_j)$
        while $Q \neq \Lambda$ do
                $i := \mathrm{Top}(Q)$;  $Q := \mathrm{Pop}(Q)$
                if $j <_y i$ then
                        $Z := \mathrm{Update}_{A\to B}(p_i)$
                        if $Z < W$ then
                            $W := Z$
                            if $Q \neq \Lambda$ then $u := \mathrm{Top}(Q)$ else $u := 0$
                            $\Pi := (u, j, t_\ell)$
                        $j := \min_{<_y}(B)$;  $T := \mathrm{Push}(T, j)$
                        $Z := \mathrm{Update}_{B\to A}(p_j)$
                else $T := \mathrm{Push}(T, i)$
        while $T \neq \Lambda$ do
            $j := \mathrm{Top}(T)$;  $T := \mathrm{Pop}(T)$
            $Z := \mathrm{Update}_{A\to B}(p_j)$
    return $(W, \Pi)$
end
```

The procedure scans all bipartitions (A, B) of X, with clusters separated by a curve of type C_1, such that cluster A, lying bottom left of the curve, has cardinality k; the current bipartition (A, B) is maintained by the data structure described in Section 8 and its weight is updated by calling procedures $\mathrm{Update}_{B\to A}$ and $\mathrm{Update}_{A\to B}$. Such a current bipartition is represented by triple (i, j, t) in the sense of equations (11).

After sorting the indices of the input set X w.r.t. $\leq_{x+y}$, the procedure enters a main loop running over the first $k + 1$ elements of the sorted array $(t_0, t_1, \ldots, t_n)$. At the ℓ-th iteration (for $\ell = 0, 1, \ldots, k$), A first includes all points of indices $(t_0, \ldots, t_\ell)$, which corresponds to assigning value t_ℓ to the third term of the current triple (i, j, t). Then, other $k - \ell$ points are added to A in all feasible ways. This is done by first moving from B to A the $k - \ell$ points with smallest abscissa x_i and then exchanging (between A and B) each of them with the element in B of minimum ordinate y_j. Two auxiliary stacks (Q and T) are used to pick points in A and B to be exchanged and to reset cluster A to the appropriate value at the end of each iteration in the main loop (here Λ denotes the empty stack).

Inside the main loop, the procedure executes other three loops, each of which is repeated $k - \ell$ times and every iteration requires $O(\log n)$ time, due to operations $\mathrm{Update}_{B\to A}$ and $\mathrm{Update}_{A\to B}$, for a total computation time of the order $O(n^2 \log n)$. Running the same procedure with k replaced by $n - k$ yields the same time evaluation. The space complexity remains of the order $O(n)$.

As regards the other types of separating curves one can reason as in Section 9.2, obtaining the same evaluations for time and space complexity. In particular, in case of a separating curve of type S_1, a procedure similar to $\text{Plain-2-clustering}_{C_1}$ can be designed which solves the problem by a main loop over the indices of points sorted w.r.t. $\leq_{x+y}$ and other appropriate inner loops based on relation $\leq_y$. The analysis of this procedure is similar to the discussion above.

11. Conclusions

In this paper we have presented efficient exact algorithms for solving the 2-clustering problem in the plane, assuming cluster size constraints, under Euclidean and Manhattan norm. In the Euclidean case, for an input set of n points, an optimal bipartition formed by two subsets of k and $n-k$ elements, respectively, can be computed in $O(n\sqrt[3]{k}\log^2 n)$ time. Under Manhattan norm, we have shown that the same problem is solvable in $O(n^2 \log n)$ time. The gap between these two bounds is mainly due to the different separating curves of optimal solutions obtained in the two cases. In the Euclidean case the separating curve is a straight line and hence we can use known enumeration results of the so-called k-sets [13, 11]. On the other hand, in the Manhattan case, the curves are Taxicab hyperbolas formed by piecewise straight lines and we have implicitly used a rather obvious $O(n^2)$ estimation of the number of k-combinations of a set of n points in the plain separated from their complement by a such a curve. It is clear that a tight evaluation of such a number could improve the time complexity bounds of our procedures.

Acknowledgement. We thank Antonio Lanteri for useful discussions on Taxicab conics and for pointing us reference [17]. Moreover, we would like to warmly thank Linda Pini for her support during the initial stages of this research.

References

[1] D. Aloise, A. Deshpande, P. Hansen, and P. Popat. NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning*, 75:245–249, 2009.

[2] T. Asano, B. Bhattacharya, M. Keil, and F. Yao. Clustering algorithms based on minimum and maximum spanning trees. In *Proceedings of the Fourth Annual Symposium on Computational Geometry, Urbana-Champaign, IL, USA, June 6-8, 1988*, pages 252–257, 1988.

[3] S. Basu, I. Davidson, and K. Wagstaff. *Constrained Clustering: Advances in Algorithms, Theory, and Applications.* Chapman and Hall/CRC, 2008.

[4] A. Bertoni, M. Goldwurm, and J. Lin. Exact algorithms for 2-Clustering with size constraints in the Euclidean plane. In *Proc. 41st SOFSEM: Theory and Practice of Computer Science*, volume 8939 of *Lecture Notes in Computer Science*, pages 128–139. Springer, 2015.

[5] A. Bertoni, M. Goldwurm, J. Lin, and L. Pini. Size-constrained 2-Clustering in the Plane with Manhattan Distance. In *Proc. 15th Italian Conference on Theoretical Computer Science*, CEUR Workshop Proceedings, CEUR-WS.org/Vol-1231, pages 33–44, 2014.

[6] A. Bertoni, M. Goldwurm, J. Lin, and F. Saccà. Size Constrained Distance Clustering: Separation Properties and Some Complexity Results. *Fundamenta Informaticae*, 115(1):125–139, 2012.

[7] C. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[8] G. Brodal and R. Jacob. Dynamic planar convex hull. In *Proceedings 43rd Annual Symposium on Foundations of Computer Science*, pages 617–626. IEEE, 2002.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 2nd edition, 2001.

[10] S. Dasgupta. The hardness of k-means clustering. Technical Report CS2007-0890, Department of Computer Science and Engineering, University of California, San Diego, 2007.

[11] T. Dey. Improved Bounds for Planar k-Sets and Related Problems. *Discrete & Computational Geometry*, 19(3):373–382, 1998.

[12] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* EATCS monographs on theoretical computer science. Springer, 1987.

[13] P. Erdős, L. Lovász, A. Simmons, and E. G. Straus. Dissection graphs of planar point sets. In *A survey of combinatorial theory (Proc. Internat. Sympos., Colorado State Univ., Fort Collins, Colo., 1971)*, pages 139–149. North-Holland, Amsterdam, 1973.

[14] P. Hansen, B. Jaumard, and N. Mladenovic. Minimum sum of squares clustering in a low dimensional space. *Journal of Classification*, 15(1):37–55, 1998.

[15] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer-Verlag, 2nd edition, 2009.

[16] M. Inaba, N. Katoh, and H. Imai. Applications of Weighted Voronoi Diagrams and Randomization to Variance-based K-clustering: (Extended Abstract). In *Proceedings of the Tenth Annual Symposium on Computational Geometry*, SCG '94, pages 332–339, USA, 1994. ACM.

[17] R. Kaya, Z. Akça, I. Günaltili, and M. Özcan. General equation for taxicab conics and their classification. *Mitt. Math. Ges. Hamburg*, 19:135–148, 2000.

[18] J. Lin. *Exact algorithms for size constrained clustering.* PhD Thesis, Università degli Studi di Milano. Ledizioni Publishing, 2013.

[19] J. B. MacQueen. Some method for the classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Structures*, pages 281–297, 1967.

[20] M. Mahajan, P. Nimbhorkar, and K. Varadarajan. The planar k-means problem is NP-hard. *Theoretical Computer Science*, 442:13–21, 2012.

[21] B. Novick. Norm statistics and the complexity of clustering problems. *Discrete Applied Mathematics*, 157:1831–1839, 2009.

[22] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23(2):166 – 204, 1981.

[23] F. Preparata and M. Shamos. *Computational geometry: an introduction.* Springer-Verlag, 1985.

[24] S. Theodoridis and K. Koutroumbas. *Pattern Recognition.* Academic Press, Elsevier, 4th edition, 2008.

[25] A. Vattani. K-means requires exponentially many iterations even in the plane. *Discrete & Computational Geometry*, 45(4):596–616, 2011.

[26] H. Vinod. Integer programming and the theory of grouping. *Journal of the American Statistical Association*, 64(326):506–519, 1969.

[27] K. Wagstaff and C. Cardie. Clustering with instance-level constraints. In *Proc. of the 17th Intl. Conf. on Machine Learning*, pages 1103–1110, 2000.

[28] S. Zhu, D. Wang, and T. Li. Data clustering with size constraints. *Knowledge-Based Systems*, 23(8):883–889, 2010.