Fast Fencing

Mikkel Abrahamsen
Department of Computer Science
University of Copenhagen
Denmark
miab@di.ku.dk

Anna Adamaszek
Department of Computer Science
University of Copenhagen
Denmark
anad@di.ku.dk

Karl Bringmann
Saarland Informatics Campus
Max Planck Institute for Informatics
Germany
kbringma@mpi-inf.mpg.de

Vincent Cohen-Addad
Sorbonne Université, CNRS,
Laboratoire d'informatique de Paris 6,
LIP6, F-75252 Paris, France.
France
vcohenad@gmail.com

Mehran Mehr
Department of Computer Science
TU Eindhoven
Netherlands
m.mehr@tue.nl

Eva Rotenberg
Department of Applied Mathematics
and Computer Science
Technical University of Denmark
Denmark
erot@dtu.dk

Alan Roytman
Department of Computer Science
University of Copenhagen
Denmark
alanr@di.ku.dk

Mikkel Thorup
Department of Computer Science
University of Copenhagen
Denmark
mikkel2thorup@gmail.com

ABSTRACT

We consider very natural "fence enclosure" problems studied by Capoyleas, Rote, and Woeginger and Arkin, Khuller, and Mitchell in the early 90s. Given a set S of n points in the plane, we aim at finding a set of closed curves such that (1) each point is enclosed by a curve and (2) the total length of the curves is minimized. We consider two main variants. In the first variant, we pay a unit cost per curve in addition to the total length of the curves. An equivalent formulation of this version is that we have to enclose n unit disks, paying only the total length of the enclosing curves. In the other variant, we are allowed to use at most k closed curves and pay no cost per curve.

For the variant with at most k closed curves, we present an algorithm that is polynomial in both n and k. For the variant with unit cost per curve, or unit disks, we present a near-linear time algorithm.

Capoyleas, Rote, and Woeginger solved the problem with at most k curves in $n^{O(k)}$ time. Arkin, Khuller, and Mitchell used this to solve the unit cost per curve version in exponential time. At the time, they conjectured that the problem with k curves is NP-hard for general k. Our polynomial time algorithm refutes this unless P equals NP.

STOC'18, June 25–29, 2018, Los Angeles, CA, USA

ACM ISBN 978-1-4503-5559-9/18/06…$15.00
https://doi.org/10.1145/3188745.3188878

CCS CONCEPTS

• **Theory of computation → Computational geometry**; *Discrete optimization*;

KEYWORDS

Geometric Clustering, Minimum Perimeter Sum

ACM Reference Format:
Mikkel Abrahamsen, Anna Adamaszek, Karl Bringmann, Vincent Cohen-Addad, Mehran Mehr, Eva Rotenberg, Alan Roytman, and Mikkel Thorup. 2018. Fast Fencing. In *Proceedings of 50th Annual ACM SIGACT Symposium on the Theory of Computing (STOC'18).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3188745.3188878

1 INTRODUCTION

We consider some very natural "fence enclosure" problems studied by Capoyleas, Rote, and Woeginger [6] and Arkin, Khuller, and Mitchell [4] in the early 90s. Given a set S of n points in the plane, we aim at finding a set of closed curves such that (1) each point is enclosed by a curve and (2) the total length of the curves is minimized. We consider two main variants. In the first variant, we pay an opening cost $\eta > 0$ per curve, which is part of the input. An equivalent formulation is that a circle with radius $\eta/2\pi$ is centered at each point, and we need to enclose these circles with curves of minimal total length, paying no opening cost. The equivalence is illustrated and explained in Figure 1. By a suitable scaling, we may assume that the circles are unit circles. We thus refer to this variant as the *unit disk fencing problem*. In the other variant, we are allowed to use at most k closed curves and pay no cost per curve. We can think of this as dividing the points into k clusters and then viewing the closed curves as perimeters of the convex hulls of the clusters. For this reason, we refer to the variant as the *k-cluster fencing problem* (also referred to as the *minimum perimeter sum problem* in the literature).

Capoyleas, Rote, and Woeginger [6] presented an algorithm for the k-cluster fencing problem that runs in time $\rho(k)n^{O(k)}$, where $\rho(k)$ is the number of nonisomorphic planar graphs on k nodes. This yields a polynomial running time when k is fixed. Arkin et al. [4] conjectured the problem to be NP-hard when k is part of the input and neither an NP-hardness proof nor a polynomial time algorithm has been found so far. To solve the unit disk version, which is equivalent to having a unit cost per cluster, Arkin et al. suggested running their algorithm for the k-cluster fencing problem for all $k \leq n$, adding k to the total perimeter length. These were the best known bounds except in the special case of the k-cluster fencing problem for $k \in \{1, 2\}$ (see the discussion on related work for more details).

1.1 Our Results

We present polynomial time algorithms for both problems. More specifically, for the unit disk fencing problem, we present an efficient near-linear time algorithm (Theorem 2.11). For the k-cluster fencing problem, we present an algorithm that is polynomial in both n and k (Theorem 3.1). In particular, this refutes the conjectured hardness unless P = NP.

Our algorithm for the unit disk fencing problem can be generalized to the case where the input consists of objects that are allowed to be disks with different diameters or polygonal objects that have to be fenced. For this variant, our running time increases by a factor that is logarithmic in the ratio between the maximum and the minimum object diameter.

In order to achieve near-linear bounds for the unit disk fencing problem, we introduce new techniques that we believe can have other applications in computational geometry. We give a detailed overview of the techniques below.

Throughout our paper, we assume that it is possible to compare the costs of two different clusterings efficiently. Note that this is a standard assumption in computational geometry.

1.2 Applications

The problem of fencing in disks or objects appears very commonly in the real world. A good example is the protection of trees, either at construction sites to protect the roots, or in the wild to protect rare trees from deer and other animals. When trees are planted by nature, we have no control over their location. In this context, each disk should have a sufficient diameter to protect rare trees from wildlife (see Figure 1).

There are many standards that specify how far fences should be from trees, and even discussions on different advantages of grouping trees beyond the fence cost (e.g., see [8]).

1.3 Our Techniques

Approach for the unit disk fencing problem. Our approach is as follows. We first partition the plane and recursively subdivide it into four quadrants, using a quadtree dissection-type approach. This divides the plane into *cells* of geometrically decreasing sizes. Each cell of width w consists of four cells of width $w/2$. We find optimal partitions for the smallest cells first (i.e., the lowest level of the quadtree). We then work with increasing cell sizes, solving the problem layer by layer in the quadtree. To obtain a solution at a given cell size, we rely on solutions to smaller cell sizes. For this to work, we need to have precomputed the solutions for various *polyominoes* made of some constant number of cells of smaller sizes.

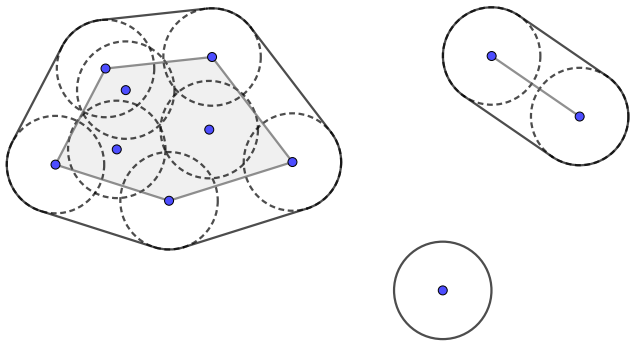

Figure 1: A set of 11 points and the set of enclosing curves minimizing the total length plus the opening cost η per curve. There are three curves, enclosing 1, 2, and 8 points, drawn in gray (one cannot be seen, as it consists of just one input point). A dashed circle centered at each point with radius $\eta/2\pi$ is drawn. For each cluster, the curve enclosing the respective circles is drawn in black. We note that the perimeter of each black curve is exactly η larger than the perimeter of the gray curve, as the linear pieces sum to the perimeter of the gray curve and the circular arcs sum to η. Hence, the problem of enclosing points with curves, minimizing the total length plus an opening cost η per curve, is equivalent to that of enclosing circles with curves of minimal total length.

For a given polyomino, we show how to obtain an optimal clustering of the points within the polyomino by merging the solutions for smaller polyominoes. In some cases, merging is not enough as there can be one large cluster intersecting all the cells of the polyomino, which we do not find by merging solutions of smaller polyominoes.

Hence, we need an efficient algorithm for finding the best cluster C intersecting all the cells of the polyomino. In order to do this, we give a subroutine running in time $O(n \log^3 n)$ that finds the best cluster that intersects all cells. The subroutine works in two steps: it first finds a point x_0 that belongs to the cluster we are looking for together with a point p on the boundary of the cluster. Once we have x_0 and p, the idea is to make an angular sweep of a ray from x_0, and consider the points in the order the ray sweeps over them. For each point v, we calculate the "best" path from p to v, in terms of both its length and how many clusters' opening costs it may save. We prove that the "best" path consists of line segments between points, and save for each v information about the last line segment on the path to v (see the red lines in Figure 2). This information allows us to finally retrieve the boundary of the convex hull of C recursively.

Approach for the k-cluster fencing problem. Our algorithm for the k-cluster fencing problem shares some similarities with the work of Gibson et al. [9], in which a dynamic programming approach was used to solve the minimum radius sum problem. One main

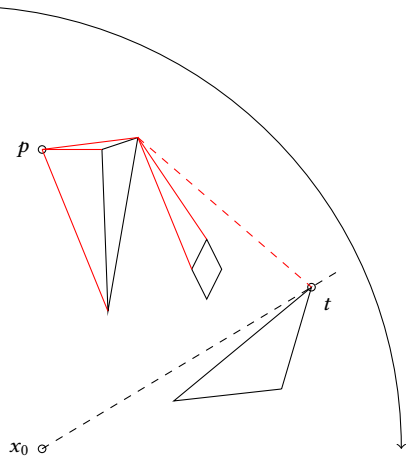

Figure 2: Given the advice that p lies on the perimeter of the cluster containing x_0 in an optimal clustering, we can compute the cluster containing x_0 with an angular sweep. Here, the angular sweep has reached the point t.

difference is that the running time complexity they obtain is $O(n^{881})$, whereas our approach works in $O(n^{27})$ time. Their technique is to divide a given instance (S, k) of the minimum radius sum problem into subproblems, where S is the set of points to be clustered and k is a constraint on the number of clusters we can use. The problem (S, k) is solvable if we have a solution to all subproblems. However, the number of subproblems is exponential. To get a polynomial time algorithm, they showed that a solution can be found after considering only a polynomial number of subproblems.

We also use a dynamic programming approach for our problem, although we need new techniques since the number of candidate clusters for the minimum radius sum problem is only $O(n^2)$ (as each disk is determined by two points in S, one determining the center and the other determining the radius). In contrast, our problem has an exponential number of candidate clusters (dictated by all subsets of S). We define subproblems based on *boxes*, which are rectangles that cover some portion of the plane and some number of input points from S. Our key observation is that there is some separator of each box (i.e., a vertical line segment or a horizontal line segment) that splits the box into two strictly smaller boxes such that an optimal solution only has a constant number of line segments that intersect this separator (in fact, we give a bound of two on the number of such intersecting line segments).

A dynamic programming approach naturally follows: we simply guess the position of such a separator (for which we argue there are $O(n)$ choices), and then guess which segments crossing this separator belong to an optimal solution. We first obtain solutions for smaller boxes, and then glue together solutions for smaller boxes to obtain solutions for larger boxes.

We note that there is an unpublished solution [11] (i.e., polynomial time algorithm) for the rectilinear version of the problem, where we must enclose n points using k axis-parallel rectangles rather than convex hulls (as in our setting). The solution to the axis-parallel version uses similar ideas. In particular, it is possible to argue the existence of separators that do not cut through any clusters of an optimal solution in this setting. This is not possible for our problem. For the k-cluster fencing problem, it is possible that any such vertical or horizontal separator cuts at least one cluster, and allowing skew separators would result in subproblems of high complexity. In particular, consider an example with n sufficiently large and assume $k < n/3$. We have $n - k + 1$ points spread evenly on a circle as the corners of a regular $(n - k + 1)$-gon, and $k - 1$ points spread evenly on a surrounding circle with the same center. The surrounding points are fairly close yet sufficiently far enough away that the optimal solution is to cluster the inner $n - k + 1$ points together, and open a cluster for each of the $k - 1$ points on the surrounding circle. Cutting away the $k - 1$ points on the outside (with not necessarily axis-aligned separators) creates subproblems defined on polygonal regions with $k - 1$ sides, resulting in high complexity subproblems.

1.4 Related Work

The literature on geometric clustering is vast [3], and thus we focus on the most relevant prior works. Arkin, Khuller, and Mitchell [4] considered many clustering variants related to the problems studied in the present paper. For the variant where points have a value associated with them, they showed that the problem of maximizing profit (i.e., sum of values of points enclosed minus the total perimeter) is NP-hard when values are unrestricted in sign. When values are strictly positive, they gave an $O(n^3)$ time algorithm. For the version in which there is a budget on the total perimeter we can use, the problem of maximizing profit is also NP-hard, even when values are strictly positive (they provided a pseudo-polynomial algorithm when the values are integers).

The k-cluster fencing problem for $k = 1$ is the very well-known problem of computing the convex hull of a set of points in the plane [7]. There has also been some work for the special case of $k = 2$ clusters. The work of Mitchell and Wynters [12] studied four flavors of the problem: minimizing the sum of perimeters, the maximum of the perimeters, the sum of the areas enclosed by the fences, and the maximum of the areas. They gave polynomial time solutions for all four flavors, running in time $O(n^3)$ (for some of them, they gave improved running time bounds of $O(n^2)$). More recently, the work of Abrahamsen et al. [2] gave an algorithm running in time $O(n \log^4(n))$ that solves the case of $k = 2$ clusters, yielding the first subquadratic time algorithm for this setting.

There have been many other papers studying related geometric clustering problems. Capoyleas, Rote, and Woeginger [6] studied a general geometric k-clustering framework in which the cost of a solution is determined by some weight function that assigns real weights to any subset of points in the plane (i.e., each cluster), after which a symmetric k-ary function over k-tuples is applied (e.g., the sum function). For the case when the weight function is the diameter, radius, or perimeter and the symmetric k-ary function is an arbitrary monotone increasing function (such as the sum or the maximum), they gave an algorithm running in time $\rho(k)n^{O(k)}$, where $\rho(k)$ is the number of nonisomorphic planar graphs on k nodes. This is polynomial if k is fixed and not given as input.

In addition, the work of Behsaz and Salavatipour [5] studied objectives such as minimizing the sum of radii and minimizing

the sum of diameters subject to the constraint of having at most k clusters. For minimizing the sum of radii, they gave a polynomial time algorithm for clustering points in metric spaces that are induced by unweighted graphs, assuming no singleton clusters are allowed. They also showed that finding the best single cluster for each connected component of the graph yields a $\frac{3}{2}$-approximation algorithm, assuming no singleton clusters are allowed. For the problem of minimizing the sum of diameters, they gave a polynomial time approximation scheme when points lie in the plane with Euclidean distances, along with a polynomial time exact algorithm when k is constant (for the metric setting).

Many classic clustering problems are NP-hard when k is given as part of the input, though there are some notable exceptions. In 2012, Gibson et al. [9] devised a polynomial time algorithm for finding k disks, each centered at a point in S, such that the sum of the radii of the disks is minimized subject to the constraint that their union must cover S. In their paper, they used a dynamic programming approach to get a running time of $O(n^{881}T(n))$, where $T(n)$ is the time needed to compare two candidate solutions.

2 THE UNIT DISK FENCING PROBLEM

Given a set of points A in the plane, we denote by $H(A)$ the convex hull of A, and by $h(A)$ the perimeter of $H(A)$.

Let A be a finite set of points in $\mathbb{R}^2$ and let $\eta > 0$ be the *opening cost*. Consider a partition $C := \{C_1, \ldots, C_\ell\}$ of A. We refer to each set C_i as a *cluster*. The *cost* of C with respect to η is $\eta \cdot \ell + \sum_{i=1}^{\ell} h(C_i)$. The partition C is *optimal* for (A, η) if no partition of A has a lower cost. We denote the cost of an optimal partition for (A, η) as $\textsc{Opt}(A, \eta)$. We sometimes omit the opening cost when it is clear from context. For two points $p_1, p_2 \in \mathbb{R}^2$, we sometimes use the notation p_1p_2 to represent the line segment connecting the points, and $\|p_1p_2\|$ to represent their distance. In the *unit disk fencing problem*, we are given a set of points A and an opening cost η, and the goal is to find an optimal partition for (A, η).

Observation 1. *In an optimal partition, the clusters have pairwise disjoint convex hulls.*

We say that A is *indivisible* if $\{A\}$ is an optimal partition for (A, η).

Observation 2. *Each cluster of an optimal partition is indivisible.*

An optimal partition $\{C_1, \ldots, C_\ell\}$ for (A, η) is *maximal* if there is no optimal partition $\{C'_1, \ldots, C'_{\ell'}\}$ of (A, η), where $\ell' < \ell$, such that for each $i \in \{1, \ldots, \ell\}$, there is some $j \in \{1, \ldots, \ell'\}$ such that $C_i \subseteq C'_j$.

2.1 Structural Properties

Lemma 2.1. *Let A, B be two indivisible sets of points in $\mathbb{R}^2$ under the opening cost η. If $H(A)$ and $H(B)$ intersect, then the set $A \cup B$ is indivisible.*

Proof. Let C be a maximal optimal partition of $(A \cup B, \eta)$. Let $\{C_1, \ldots, C_m\}$ be the clusters from C such that $A \cap C_i \neq \emptyset$ and $B \cap C_i \neq \emptyset$ for $1 \leq i \leq m$. Assume for contradiction that $m = 0$. This means that $A \cap B = \emptyset$ and each cluster of C is a subset of either A or B, and hence that $|C| \geq 2$. If $|C| = 2$, then $C = \{A, B\}$, and C cannot be optimal as $H(A)$ and $H(B)$ intersect. If $|C| > 2$, then C cannot be maximal as A and B are indivisible. Hence $m \geq 1$.

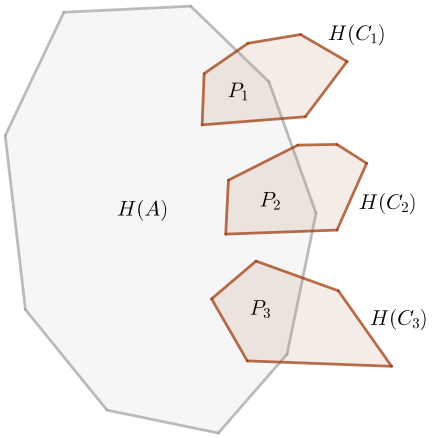

Figure 3: The polygon Q from the proof of Lemma 2.1 is the set $H(A) \cup \bigcup_{i=1}^{3} H(C_i)$. Note that the perimeter is $h(Q) = h(A) + \sum_{i=1}^{3}(h(C_i) - h(P_i))$.

We want to show that the set $A' := A \cup \bigcup_{i=1}^{m} C_i$ is indivisible. Let $\{A_1, \ldots, A_j\}$, where possibly $j = 0$, be the clusters of C that are contained in A. Then the set $\mathcal{A} := \{A_1, \ldots, A_j, C_1, \ldots, C_m\}$ is a partition of A'. Note that $\mathcal{A}$ must be an optimal partition of A', as otherwise C would not be optimal for $A \cup B$.

The cost of $\mathcal{A}$ is thus

$$\textsc{Opt}(A') = (j + m)\eta + \sum_{i=1}^{j} h(A_i) + \sum_{i=1}^{m} h(C_i).$$

Now, for each cluster C_i, we define a polygon $P_i = H(A) \cap H(C_i)$. As $H(A)$ and $H(C_i)$ are both convex, so is P_i.

Note that all points in A' are contained in the polygon $Q := H(A) \cup \bigcup_{i=1}^{m} H(C_i)$, see Figure 3. It hence follows that

$$h(A') \leq h(Q) = h(A) + \sum_{i=1}^{m}(h(C_i) - h(P_i)).$$

Note that $\{A_1, \ldots, A_j, P_1 \cap A, \ldots, P_m \cap A\}$ is a partition of A. Hence, by the indivisibility of A, we have

$$\eta + h(A) \leq (j + m)\eta + \sum_{i=1}^{j} h(A_j) + \sum_{i=1}^{m} h(P_i).$$

Combining these two inequalities yields

$$h(A') + \eta \leq (j + m)\eta + \sum_{i=1}^{j} h(A_j) + \sum_{i=1}^{m} h(C_i) = \textsc{Opt}(A'),$$

and so A' is indivisible.

As A' is indivisible, A' is the union of clusters of C, and C is maximal, it follows that A' is itself a cluster of C, i.e., $m = 1$ and $j = 0$. Recall furthermore that A' contains A and intersects B. In a similar way, it can be shown that $B' := B \cup \bigcup_{i=1}^{m} C_i$ is a cluster of C

that contains B and intersects A. Since $A' \cap B' \neq \emptyset$ and C is optimal and maximal, we must have $A' = B' = A \cup B$, so $C = \{A \cup B\}$ and $A \cup B$ is indivisible. □

Lemma 2.2. *Let $A \subseteq B$ be sets of points in $\mathbb{R}^2$, and let A be indivisible. Let $C = \{C_1, \ldots, C_\ell\}$ be a maximal optimal partition of B. Then $A \subseteq C_i$ for some $i \in \{1, \ldots, \ell\}$.*

Proof. Let $\mathcal{S} \subseteq C$ be the set of clusters of C that intersect A. By Lemma 2.1, each of the sets $S \cup A$, where $S \in \mathcal{S}$, is indivisible. It thus follows that $A \cup \bigcup_{S \in \mathcal{S}} S = \bigcup_{S \in \mathcal{S}} S$ is also indivisible. Since C is maximal, it must then be the case that $\mathcal{S}$ consists of a single cluster C_i that contains A. □

Lemma 2.3. *Each instance (A, η) of the unit disk fencing problem has a unique maximal optimal partition.*

Proof. Let $C = \{C_1, \ldots, C_\ell\}$ and $C' = \{C'_1, \ldots, C'_{\ell'}\}$ be two maximal optimal partitions of A. Lemma 2.2 gives that each cluster C_i of C is contained in some cluster C'_j of C'. Likewise, each cluster C'_j of C' is contained in some cluster C_i of C. Since the clusters of an optimal partition are disjoint, it now follows that there is a one-to-one correspondence between the partitions C and C', i.e., they are the same partition of A. □

Lemma 2.4. *Let $A = \bigcup_i A_i$ be a set of points such that $A = \bigcup_i A_i^{\complement}$, where $A_i^{\complement} := A \setminus A_i$ for all i. Let C_i be the maximal optimal partition of $A_i^{\complement}$, and C the maximal optimal partition of A. Any cluster of C_i that intersects a cluster $C \in C$ is also contained in C, and it follows in particular that each cluster $C \in C$ is the union of clusters of the partitions C_i. Furthermore, each cluster $C \in C$ is either a cluster of some partition C_i or has a non-empty intersection with each set A_i.*

Proof. In order to prove the first part, consider some cluster $C_i \in C_i$ that intersects $C \in C$. Since C_i is indivisible, Lemma 2.2 gives that there is a cluster of C containing C_i. This cluster must be C, as the clusters of C are disjoint. It hence also follows that each cluster $C \in C$ is the union of some clusters of the partitions C_i.

In order to prove the second part, consider a cluster $C \in C$ that does not intersect some set A_i. Then, $C \subseteq A_i^{\complement}$. Since C is indivisible, Lemma 2.2 gives that there is a cluster C_i of C_i such that $C \subseteq C_i$. By the above, we also have that $C_i \subseteq C$. Hence $C = C_i$, i.e., C is a cluster of C_i. □

2.2 Partitioning into Independent Instances

Consider an instance (A, η) of the unit disk fencing problem. Observe that any two points $p_1, p_2 \in A$ such that $\|p_1 p_2\| < \eta/2$ must be in the same cluster in any optimal partition of A. We will prove that we can efficiently decompose the problem instance (A, η) into a collection of independent subinstances (A_i, η), such that for each subinstance $diam(A_i) = \text{poly}(|A_i|) \cdot \eta$.

Lemma 2.5. *Any n-point instance (A, η) of the unit disk fencing problem can be reduced in $O(n \log^2 n)$ time into a disjoint collection of subinstances (A_i, η), where $A = \bigcup_i A_i$ and each subinstance is bounded by a box of side length at most $2|A_i|^2 \cdot \eta$. The subinstances are independent in the sense that an optimal partition for (A, η) is the union of the optimal partitions for (A_i, η).*

Proof. Clearly, the optimal partition for (A, η) has a cost of at most $n \cdot \eta$, as a partition of cost $n \cdot \eta$ can be obtained by opening n singleton clusters. Therefore, if any two points are at a distance greater than $n \cdot \eta/2$, they must be in separate clusters of any optimal partition (as the perimeter of any cluster containing two such points would be greater than $\textsc{Opt}(A)$).

We first sort all points from A with respect to their x-coordinate. Denote the sorted points by $p_1, \ldots, p_n$. Whenever two consecutive points p_i, p_{i+1} have a difference in their x-coordinate that is greater than $n \cdot \eta/2$, we know that the sets of points $\{p_1, \ldots, p_i\}$ and $\{p_{i+1}, \ldots, p_n\}$ can be treated separately (i.e., each cluster of any optimal partition will be contained in either $\{p_1, \ldots, p_i\}$ or $\{p_{i+1}, \ldots, p_n\}$). This gives us a partition of $\{p_1, \ldots, p_n\}$ into subinstances, where each subinstance is contained in a vertical slab of width at most $n^2 \cdot \eta/2$. Now, for each subinstance we sort the points according to their y-coordinate, and perform a similar operation. Therefore, in time $O(n \log n)$ we can partition (A, η) into subinstances (A_i, η), where $A = \bigcup_i A_i$, such that each subinstance has a bounding box of size at most $n^2 \cdot \eta/2 \times n^2 \cdot \eta/2$, and an optimal partition for (A, η) is the union of the optimal partitions for (A_i, η).

Note that if $|A_i| \geq |A|/2$, then $n^2 \cdot \eta/2 = |A|^2 \cdot \eta/2 \leq 2|A_i|^2 \cdot \eta$, and the size of the bounding box is as required. It therefore remains to consider subinstances (A_i, η) for which $|A_i| < |A|/2$ and the smallest bounding box has size at least $2|A_i|^2 \cdot \eta$. In such an instance, there must be two consecutive points in the order of x- or y-coordinates where the respective coordinates differ by at least $\frac{2|A_i|^2 \cdot \eta}{|A_i|-1} > |A_i| \cdot \eta/2$. Thus, the instance (A_i, η) can be recursively partitioned into yet smaller subinstances. Since at each recursive level, at most half of the points from the previous level remain, the depth of the recursive tree is at most $O(\log n)$. The total running time is therefore $O(n \log^2 n)$. □

2.3 Cells and Polyominoes

Consider an instance (A, η) of the unit disk fencing problem with bounding box S (which is an axis-parallel square of side length $\text{poly}(|A|) \cdot \eta$). We will recursively subdivide S into *cells*, starting with a single cell S, and recursively partitioning each cell into four smaller squares, ending when the side length of the cells is at most $\eta/8$. This happens after some $L = O(\log n)$ recursive operations due to the partitioning described in Section 2.2. We call a cell a *level i cell* if it has been obtained after $i - 1$ levels of subdivision. The square S is thus a level 1 cell.

We consider level i cells and define *polyominoes* to be simple polygons consisting of some level i cells. Two cells are *neighboring* if their boundaries share an edge or a corner. We will be particularly interested in *monominoes*, which consist of a single cell, *dominoes*, which are the union of two cells sharing an edge, *L-trominoes*, which are the union of three pairwise neighboring cells (i.e., which are in the shape of the letter L), and *square-tetrominoes*, which are a 2×2-square of cells. A *basic polyomino* is a monomino, a domino, an L-tromino, or a square-tetromino. See Figure 4 for all the basic polyominoes. Note that any non-basic polyomino contains two non-neighboring cells. Polyominoes consisting of level i cells are called level i polyominoes. We say that a polyomino π is *convex* if the intersection of π with any horizontal or vertical line has at most one connected component.

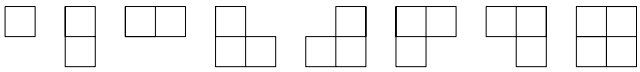

Figure 4: Basic polyominoes.

Note that each level i monomino, domino, tromino, or tetromino, for $i < L$, contains 4, 8, 12, or 16 cells at level $i+1$, respectively. We define a *subpolyomino* at level $i+1$ to be a convex polyomino at level $i+1$ that is contained in a basic polyomino at level i. Note that each basic polyomino at level $i+1$ and at level i is also a subpolyomino at level $i+1$. For all subpolyominoes, see Figure 5.

As we want each input point to belong to exactly one cell of a given level, we define a cell to include its right and bottom edge and the bottom-right corner. The other edges and corners then belong to the neighboring cells. For any collection of cells π (not necessarily a polyomino), we define $A_\pi := A \cap \pi$ to be the input points in π. We say that a polyomino π is *empty* if $A_\pi = \emptyset$. We will consider subproblems of the original unit disk fencing problem instance (A, η), each subproblem corresponding to the input points A_π lying in a non-empty basic polyomino π of some level. Note that the number of non-empty basic polyominoes of a given level i is $O(n)$, as each point belongs to a constant number of such polyominoes. Therefore, the total number of non-empty basic polyominoes is at most $O(n \log n)$.

We compute an optimal partition for each subproblem (A_π, η), where π is a non-empty basic polyomino. We start with the polyominoes at level L. At level L, any two points in the same basic polyomino π have distance less than $\eta/2$ and therefore A_π is indivisible. Suppose now that we have already computed the optimal partition for each non-empty basic polyomino at some level $i \leq L$. As we will see, this makes it possible to compute the optimal partitions for all non-empty subpolyominoes at level i. Since each basic polyomino at level $i-1$ is a subpolyomino at level i, we thus also know the optimal partitions of each basic polyomino at level $i-1$. It follows that the process can be repeated until we reach level 1, i.e., we have computed an optimal partition of A.

To compute and process the polyominoes efficiently, we will use a quadtree construction as described in the next section.

2.4 Quadtree Construction

A *quadtree* is a geometric data structure for objects in the Euclidean plane. The root of the quadtree corresponds to a square containing all the input objects, while the children of each node correspond to the four subsquares of the given square. The leaves correspond to subsquares that have small enough side length, or where the input objects have small complexity, e.g., subsquares containing at most some constant number of input objects. See [10, Chapter 2] for more information on quadtrees, and for their applications.

In our case, the root of the quadtree corresponds to the level 1 cell S. Each node corresponding to a level i cell C has at most 4 children, which correspond to level $i+1$ cells contained in C. We do not create nodes corresponding to empty cells, i.e., with no points from A. The leaves correspond to the highest level cells, i.e., the side length of the leaf cells is at most $\eta/8$. As there are at most n non-empty cells at each level, the number of nodes of the quadtree is at most $n \cdot L = O(n \log n)$. The quadtree can be constructed in time $O(n \log n)$, as at each of the L levels, we have to compute the subsquares for the n points.

While constructing the quadtree, for each node corresponding to each level i cell, we can compute the nodes corresponding to the eight *neighboring cells* (if such nodes exist, i.e., the corresponding cells are non-empty). Remembering this information will allow us to construct the polyominoes more easily.

From the quadtree, we can construct the set of all non-empty basic polyominoes in time $O(n \log n)$, assigning each basic polyomino π to the nodes of the quadtree corresponding to the cells of π. We do this by considering all nodes of the tree, considering the corresponding cell c for each node of the tree and the 21 basic polyominoes containing cell c, and either constructing a new polyomino, or assigning an existing polyomino to the currently considered node.

2.5 Finding an Optimal Partition for Each Basic Polyomino

We now describe an algorithm for finding an optimal partition for each basic polyomino.

Leaf polyominoes. Consider a basic polyomino π at level L. As π consists of level L cells, and the side length of such cells is at most $\eta/8$, the distance between any two points in π is smaller than $\eta/2$. Therefore, an optimal partition for (A_π, η) consists of one indivisible set of points A_π. Therefore, optimal partitions for leaf polyominoes can be computed efficiently.

At most one big cluster. Suppose that we have already computed the optimal partitions for all basic polyominoes at level i. In order to compute the optimal partitions for the basic polyominoes at level $i-1$, we first compute the optimal partitions for all subpolyominoes at level i. This suffices as the basic polyominoes at level $i-1$ are also subpolyominoes at level i. To find an optimal partition for each subpolyomino π efficiently, we make use of the following property.

Lemma 2.6. *Let π be a non-basic polyomino at some level i. Let $\mathcal{C} = \{C_1, \ldots, C_\ell\}$ be a maximal optimal partition of A_π. For any pair Γ_1, Γ_2 of non-neighboring cells of π, there is at most one cluster $C \in \mathcal{C}$ such that $H(C)$ intersects both Γ_1 and Γ_2. In particular, $\mathcal{C}$ has at most one cluster C such that $H(C)$ intersects all cells of π.*

Proof. Assume that there are two clusters $C_1, C_2 \in \mathcal{C}$ such that $H(C_1)$ and $H(C_2)$ both intersect Γ_1 and Γ_2. The situation is depicted in Figure 6. Since $\mathcal{C}$ is optimal, $H(C_1)$ and $H(C_2)$ are disjoint, and it follows that both boundaries $\partial\Gamma_1$ and $\partial\Gamma_2$ intersect both boundaries $\partial H(C_1)$ and $\partial H(C_2)$. We may then define the point p_j^i, for each choice $i, j \in \{1, 2\}$, to be the intersection point of $\partial\Gamma_i$ with $\partial H(C_j)$ such that (i) $p_1^i p_2^i \cap (H(C_1) \cup H(C_2)) = \{p_1^i, p_2^i\}$, and (ii) $p_j^1 p_j^2 \cap (\Gamma_1 \cup \Gamma_2) = \{p_j^1, p_j^2\}$. Let S be the quadrilateral with vertices at p_j^i for $i, j \in \{1, 2\}$, and consider the polygon $C_{12} = H(C_1) \cup H(C_2) \cup S$. We will show that $h(C_{12})$ is not larger than $h(C_1) + h(C_2)$. As C_{12} contains all points of $C_1 \cup C_2$, this shows that $\mathcal{C}$ is not optimal or maximal.

In the following, we show the inequality

$$\|p_1^1 p_2^1\| + \|p_1^2 p_2^2\| \leq \|p_1^1 p_1^2\| + \|p_2^1 p_2^2\|. \tag{1}$$

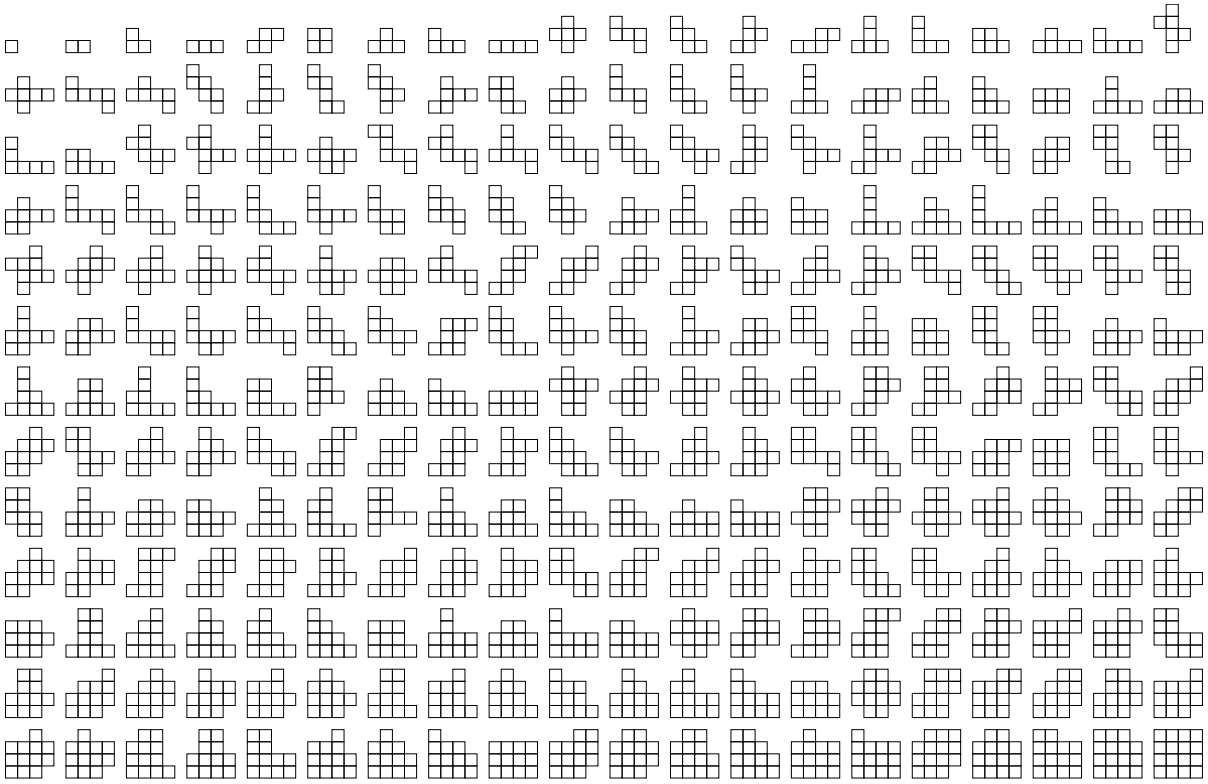

Figure 5: The 260 different subpolyominoes (up to rotations and reflections).

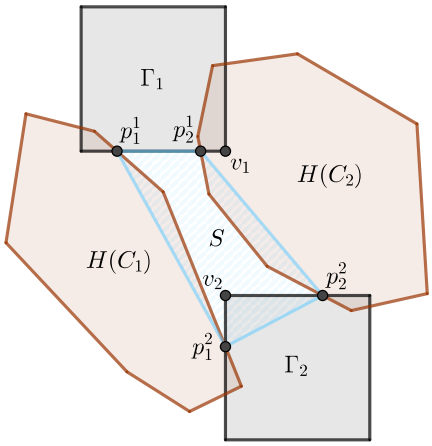

Figure 6: A situation from the proof of Lemma 2.6.

The desired result then follows as the latter sum is at most the length of the perimeter of $H(C_1)$ and $H(C_2)$ contained in S. Let α be the side length of the cells. If Γ_1 and Γ_2 are in the same row or column of cells, then clearly $\|p_1^i p_2^i\| \leq \alpha \leq \|p_j^1 p_j^2\|$, and Inequality (1) holds. Otherwise, let v_1, v_2 be the corners of Γ_1, Γ_2 minimizing the distance between the cells. By considering cases where the points p_j^i are on $\partial\Gamma_i$, one can observe that it always holds that

$$\|p_1^i p_2^i\| = \sqrt{\|p_1^i v_i\|^2 + \|p_2^i v_i\|^2} \leq \sqrt{\alpha^2 + \|p_j^i v_i\|^2}$$
$$\leq \sqrt{\alpha^2 + \|p_j^1 v_1\|^2 + \|p_j^2 v_2\|^2} \leq \|p_j^1 p_j^2\|,$$

and Inequality (1) follows. □

Cluster unions. Consider a given subpolyomino π at some level i for which we want to compute the maximal optimal partition of the input points A_π. The overall approach is the following. We use maximal optimal partitions C_j for A_{π_j} for various smaller collections $\pi_j \subsetneq \pi$ of level i cells. We then construct the *merger* of the partitions C_j, which is the partition of A_π we get when we unite the partitions C_j and keep merging clusters with overlapping convex hulls. The merger C thus consists of clusters with pairwise disjoint convex hulls, but is in general not optimal. As we shall see, a maximal optimal partition of A_π can be obtained by uniting one subset of the clusters of C into one big cluster. This is the motivation for the following definitions.

Let $C = \{C_1, \ldots, C_\ell\}$ be a partition of a set A of points such that $H(C_i) \cap H(C_j) = \emptyset$ for any $i \neq j$. For any subset $\mathcal{S} \subseteq C$ consider the set $U_\mathcal{S} = \bigcup_{C_i \in \mathcal{S}} C_i$. Let $C[\mathcal{S}]$ be the partition consisting of the cluster $U_\mathcal{S}$ and every $C_i \notin \mathcal{S}$. Consider the set $\mathcal{S}^* \subseteq C$ minimizing the cost of the partition $C[\mathcal{S}^*]$. If there is more than one such partition, we are interested in a maximal one. We say that $\mathcal{S}^*$ is an *optimal cluster union* for the clustering C.

Lemma 2.7. *Let $A = \bigcup_i A_i$ be a set of points such that $A = \bigcup_i A_i^{\complement}$, where $A_i^{\complement} := A \setminus A_i$ for all i. Let C^* be the maximal optimal partition of A and suppose that there is at most one cluster in C^* that intersects each set A_i. Let C_i be the maximal optimal partition of $A_i^{\complement}$ and let C be the merger of the partitions C_i. Let $\mathcal{S}^*$ be an optimal cluster union for C. Then $C[\mathcal{S}^*]$ and C^* are the same partition.*

Proof. By Lemma 2.4, each cluster of C^* is either a cluster of some C_i, or has non-empty intersection with each set A_i. Since there is at most one cluster in C^* of the latter kind, it follows that C^* has the form $\{C_1, \ldots, C_k\}$, where each C_i, $i \geq 2$, is contained in a cluster of the partition C. Each cluster C of C is indivisible by Lemma 2.1, so it is contained in some cluster $C_i \in C^*$ by Lemma 2.2. For each cluster C_i where $i \geq 2$, there must be a cluster $C \in C$ contained in C_i, and it follows that $C = C_i$. Hence, C has the form $\{D_1, \ldots, D_\ell, C_2, \ldots, C_k\}$, where $C_1 = \bigcup_{i=1}^{\ell} D_i$. Therefore, the optimal cluster union for C is $\mathcal{S}^* := \{D_1, \ldots, D_\ell\}$, and $C[\mathcal{S}^*]$ is the partition C^*. □

Lemma 2.8. *Let π be a non-basic convex polyomino. There are two cells Γ_1, Γ_2 of π with the following properties:*

(i) *Γ_1, Γ_2 are non-neighboring,*
(ii) *$\pi \setminus \Gamma_1$ and $\pi \setminus \Gamma_2$ are convex, and either*
(iii.a) *the horizontal distance between Γ_1 and Γ_2 is at least as large as the vertical distance, Γ_1 is leftmost and Γ_2 is rightmost in π, or*
(iii.b) *the vertical distance between Γ_1 and Γ_2 is at least as large as the horizontal distance, Γ_1 is topmost and Γ_2 is bottommost in π.*

Proof. Since π is non-basic, it has either a width of at least 3 cells or a height of at least 3 cells. Assume without loss of generality that the width of π is at least as large as the height of π. We will choose Γ_1 to be one of the leftmost cells of π, and Γ_2 to be one of the rightmost cells of π. As we want $\pi \setminus \Gamma_1$ and $\pi \setminus \Gamma_2$ to be convex, Γ_1 and Γ_2 have to be topmost or bottommost in their columns. If there is only one cell in the leftmost (rightmost) column, we take it as Γ_1 (respectively, Γ_2), as then clearly $\pi \setminus \Gamma_1$ (respectively, $\pi \setminus \Gamma_2$) remains a convex polyomino. If there are at least two cells in the column, then, by convexity of π, at least one of them can be removed without disconnecting the polyomino. □

The following lemma states that we can find optimal cluster unions efficiently. Please see the full version of this paper for a proof of the lemma [1].

Lemma 2.9. *Let π be a collection of cells and Γ_1, Γ_2 be two non-neighboring cells of π such that either*

(i) *the horizontal distance between Γ_1 and Γ_2 is at least as large as the vertical distance, Γ_1 is leftmost and Γ_2 is rightmost in π, or*
(ii) *the vertical distance between Γ_1 and Γ_2 is at least as large as the horizontal distance, Γ_1 is topmost and Γ_2 is bottommost in π.*

Let C be a partition of the points A_π such that for $i \in \{1, 2\}$, C restricted to the points of $A_{\pi \setminus \Gamma_i}$ is the maximal optimal partition of $A_{\pi \setminus \Gamma_i}$. An optimal cluster union for C can be found in time $O(n \cdot \text{polylog}\, n)$, where n is the number of points in A_π.

Solving non-basic subpolyominoes. We can now describe how to find maximal optimal partitions of the points in non-basic subpolyominoes.

Lemma 2.10. *Let π be a non-basic subpolyomino at level i. Given maximal optimal partitions for basic polyominoes at level i, we can compute an optimal partition of A_π in time $O(n \cdot \text{polylog}\, n)$, where n is the number of points in A_π.*

Proof. We first find cells Γ_1, Γ_2 of π as in Lemma 2.8. Let $\pi_1 := \pi \setminus \Gamma_1$, $\pi_2 := \pi \setminus \Gamma_2$, and $\pi_3 := \Gamma_1 \cup \Gamma_2$. As each monomino Γ_i is a basic polyomino at level i, we know the maximal optimal partition of A_{Γ_i} by assumption. Let C_3 be the merger of the maximal optimal partitions of A_{Γ_1} and A_{Γ_2} (which is in fact just the union of the partitions, as the cells are disjoint). Then, $\Gamma_1 \cup \Gamma_2$ together with the partition C_3 satisfy the conditions of Lemma 2.9, and we can find an optimal cluster union $\mathcal{S}_3$ for C_3 in time $O(n \cdot \text{polylog}\, n)$. Define $C_3^* := C_3[\mathcal{S}_3]$. By Lemma 2.6, the maximal optimal partition of $A_{\Gamma_1 \cup \Gamma_2}$ contains at most one cluster that intersects Γ_1 and Γ_2. Hence, by Lemma 2.7, C_3^* is the maximal optimal partition of $A_{\Gamma_1 \cup \Gamma_2}$. See Figure 7.

Denote the maximal optimal partition of A_π as C^*. Consider the sets $A_{\pi_1}, A_{\pi_2}, A_{\pi_3}$, and suppose for now that we know their optimal partitions C_1^*, C_2^*, C_3^*. By Lemma 2.4 (taking $A := A_\pi$, $A_1 := A_{\Gamma_1}$, $A_2 := A_{\Gamma_2}$, and $A_3 := A_{\pi \setminus (\Gamma_1 \cup \Gamma_2)}$), we get that a cluster of C^* that is not a cluster of any C_i^* must intersect both Γ_1 and Γ_2. Due to Lemma 2.6, there is at most one such cluster in C^*. Let C be the merger of the partitions C_1^*, C_2^*, C_3^*. Applying Lemma 2.9 for π, Γ_1, Γ_2, and C, we obtain an optimal cluster union $\mathcal{S}$ for C. By Lemma 2.7, we get that $C[\mathcal{S}] = C^*$.

As the polyomino π has at most 16 cells, we need to find optimal partitions for a constant number of subpolyominoes π_i before we get down to the basic polyominoes. This gives the total running time of $O(n \cdot \text{polylog}\, n)$. □

Summing up. Consider the subpolyominoes Π_i at some level i. By Lemma 2.10, the total computation for level i takes time

$$\sum_{\pi \in \Pi_i} O(|A_\pi| \cdot \text{polylog}\, |A_\pi|) \leq \text{polylog}\, n \cdot \sum_{\pi \in \Pi_i} O(|A_\pi|) = O(n \cdot \text{polylog}\, n),$$

where the equality follows since each point belongs to $O(1)$ level i subpolyominoes. Due to the preprocessing described in Lemma 2.5, the number of levels is $O(\log n)$, so the total running time of the algorithm is $O(n \cdot \text{polylog}\, n)$. We have thus proven the following theorem:

Theorem 2.11 (The unit disk fencing problem). *There is an algorithm running in time $O(n\,\text{polylog}(n))$ that, given any set A of n points in the plane and an opening cost η, finds a set of ℓ closed curves such that each point in S is enclosed by a curve and the total length of the curves plus $\eta \cdot \ell$ is minimized.*

2.6 Finding Optimal Cluster Unions

In order to find optimal cluster unions, we first solve a more specialized problem where we require a special point x_0 to be contained in the cluster union. To be more precise, the optimal cluster union $\mathcal{S}^*$ for the pair (C, x_0), where C is a partition, is a maximal subset of

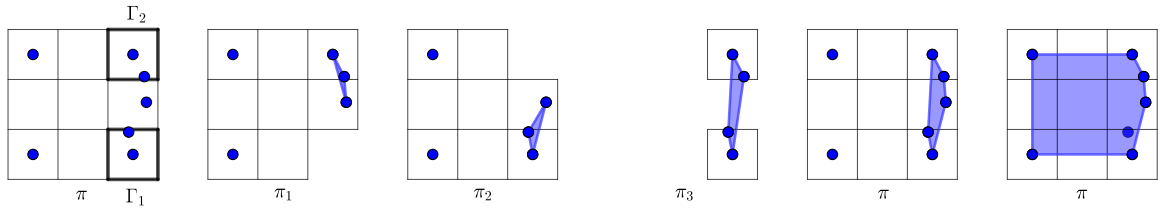

Figure 7: A demonstration of how the optimal partition C^* of the set A_π of points in the non-basic 3×3-cell polyomino π is obtained from optimal partitions C_1^*, C_2^*, C_3^* of points $A_{\pi_1}, A_{\pi_2}, A_{\pi_3}$ in smaller collections of cells π_1, π_2, π_3, as described in the proof of Lemma 2.10. The cells have width 1 and the opening cost is $\eta := 5/4$. The second, third, and fourth figure show the optimal clusterings C_1^*, C_2^*, C_3^*. The fifth figure shows the merger C of these. The final figure shows C^* (the optimal cluster union consists of all the clusters of C).

the clusters of the partition $C_{x_0} := C \cup \{\{x_0\}\}$ such that $\{x_0\} \in \mathcal{S}^*$ and the cost of $C_{x_0}[\mathcal{S}^*]$ is minimal.

In this section, we prove the following result. Please see the full version of this paper for more details [1].

Lemma 2.12. *Let $C = \{C_1, \ldots, C_\ell\}$ be a partition of a given set A of points such that $H(C_i) \cap H(C_j) = \emptyset$ for any $i \neq j$, and let x_0 be an arbitrary point. A maximal optimal cluster union for (C, x_0) can be found in time $O(n \cdot \text{polylog}\, n)$, where n is the number of points in A.*

We give intuition regarding how we achieve the above result. Our first goal is to solve the following more specialized problem. Given an "internal" point x_0 and a "perimeter" point $p \neq x_0$, and given a set of input points and pre-clustered input points, the goal is to find the optimal cluster containing x_0 with an angle-monotone perimeter seen from x_0 and with p on its perimeter. We present an algorithm to solve this problem. The algorithm can take into account that the cluster must be contained within some delimiting outer boundary.

The idea is to make an angular sweep of a ray from x_0, and consider the points in the order the ray sweeps over them. For each point v, we calculate the *best* path from p to v, in terms of both its length and how many clusters' opening costs it may save. In this process, we only store for each point its *parent* $\text{par}(v)$, which is an input point with the property that a best path to v ends with the line segment from $\text{par}(v)$ to v. Finally, we calculate the parent of p, and have thus specified the entire cluster and may generate its convex hull by recursively outputting the parent until we end back at p.

Then, given a point x_0 that is contained in an optimal cluster union, we show how to efficiently find such an optimal cluster union containing x_0. In particular, we guess a perimeter point $p \neq x_0$ that lies on the boundary of such an optimal cluster union. We show how to do this efficiently by using our algorithm for finding the optimal cluster containing x_0 and a point p on its boundary as a subroutine and essentially applying a binary search procedure on top of it to guess the perimeter point. Finally, we also give an efficient procedure for finding a point x_0 belonging to an optimal cluster union, which the proof of Lemma 2.9 makes use of (in combination with Lemma 2.12).

3 THE k-CLUSTER FENCING PROBLEM

In the k-cluster fencing problem, we are given a set of n points S in the plane and an integer k as an input instance. The following theorem states our main result for this problem. Please see the full version of this paper for more details [1].

Theorem 3.1 (The k-cluster fencing problem). *There is a polynomial time algorithm that, given any set S of n points in the plane and an integer k, finds a set of at most k closed curves such that each point in S is enclosed by a curve and the total length of the curves is minimized.*

Our main approach is to use dynamic programming to solve this problem. Our subproblems are based on the notion of boxes (i.e., rectangles in the plane), which contain some subset of the input points that we would like to cluster optimally. Great care is needed in order to ensure that there are only polynomially many subproblems we need to consider. To this end, we prove some structural properties regarding an optimal solution that enable us to reduce the complexity of the subproblems that we solve.

3.1 Preliminaries

We first describe some relevant notation and preprocessing that we do to the input. An *edge* e is defined by its distinct head and tail $p, q \in \mathbb{R}^2$ and is denoted by $e := pq$. For a convex polygon P, we define $h(P)$ to be the perimeter of P and orient the boundary of P in counterclockwise order (hence inducing edges). We define a *k-clustering* of S as a partition $C := \{S_1, \ldots, S_k\}$ of S into k subsets or *clusters* $S_1, \ldots, S_k \subseteq S$. Let $\Phi(C) := \sum_{i=1}^{k} h(S_i)$ be the cost of the clustering C. An *optimal k-clustering* is a k-clustering $C_k^{\text{OPT}}(S)$ such that $\Phi(C_k^{\text{OPT}}(S)) \leq \Phi(C)$ for any k-clustering C. Here, $C_k^{\text{OPT}}(S)$ denotes an arbitrary optimal k-clustering of S. We slightly abuse notation and refer to the edges of $C_k^{\text{OPT}}(S)$ as the edges induced by the convex hulls of the clusters in $C_k^{\text{OPT}}(S)$.

Let $p_1, \ldots, p_n$ be the points in S sorted by their x-coordinates $x_1 < \cdots < x_n$ (we assume here for simplicity that the x-coordinates are all distinct). We construct two vertical lines v_i^- and v_i^+ with each x-coordinate x_i, such that v_i^- formally is to the left of p_i and v_i^+ formally is to the right of p_i. Thus, an edge $p_j p_i$ for $j < i$ intersects v_i^- at p_i but does not intersect v_i^+, whereas $p_i p_j$ for $i < j$ intersects v_i^+ at p_i but does not intersect v_i^-. The set of all lines v_i^-, v_i^+ for all $i \in \{1, \ldots, n\}$ are the *vertical main lines*.

Between any two consecutive vertical main lines v_i^+, v_{i+1}^-, we define 19999 *vertical help lines* with x-coordinates $x_i + \frac{x_{i+1}-x_i}{20000} \cdot j$ for $j \in \{1, \ldots, 19999\}$. That is, these 19999 vertical help lines induce 20000 intervals between x_i and x_{i+1}, each of which has the same length given by $\frac{x_{i+1}-x_i}{20000}$. In a similar way, we define two horizontal main lines with the y-coordinate of each input point p, one formally below p and the other formally above p. Let $h_1^-, h_1^+, \ldots, h_n^-, h_n^+$ be the horizontal main lines sorted by ascending y-coordinate. We also define 19999 equidistant horizontal help lines between any two consecutive horizontal main lines h_i^+ and h_{i+1}^-.

Boxes. Let $\mathcal{B}(S)$ be the set of all rectangles (i.e., boxes) with edges contained in main or help lines. Note that the size of $\mathcal{B}(S)$ is $O(n^4)$. For any box $B \in \mathcal{B}(S)$, we denote by $l(B)$ and $r(B)$ the left and right vertical edges of B, respectively, and by $b(B)$ and $t(B)$ the bottom and top edges of B, respectively.

For a box B, let ℓ be a vertical line defined by an x-coordinate lying strictly between the x-coordinates of $l(B)$ and $r(B)$. We say that the vertical line segment $s := \ell \cap B$ is a *vertical separator* of B. A vertical separator of a box B is *good* if it intersects at most two edges in $C_k^{\text{OPT}}(S)$. We also analogously define a *horizontal separator*, along with the notion of a *good horizontal separator*. We call a box *elementary* if there are no vertical help or main lines that lie strictly in between $l(B)$ and $r(B)$, and no horizontal help or main lines that lie strictly in between $b(B)$ and $t(B)$.

3.2 Structural Properties

The main structural property we aim to show is that in a subproblem for a box $B \in \mathcal{B}(S)$, there is a good vertical or horizontal separator contained in a main or help line that divides B into two smaller boxes. In particular, in order to ensure subproblems of low complexity, we aim to maintain the following invariant.

Definition 3.2. A box $B \in \mathcal{B}(S)$ satisfies the *box invariant* if each of its edges is intersected by at most two edges of $C_k^{\text{OPT}}(S)$.

The following lemma is a key ingredient in our algorithm, and it is the main structural property we show in this section. Please see the full version of this paper for a proof of the lemma [1].

Lemma 3.3. *If $B \in \mathcal{B}(S)$ is an elementary box, then there are at most two edges of $C_k^{OPT}(S)$ intersecting B (in particular, B satisfies the box invariant). Otherwise, let $B \in \mathcal{B}(S)$ be any box satisfying the box invariant. Then B has a good vertical or horizontal separator s contained in a main or help line such that s divides B into two strictly smaller boxes B_l and B_r, both of which satisfy the box invariant.*

With this structural lemma at hand, we can naturally use a dynamic programming approach: we simply guess the position of such a separator (which lies on a main or help line, either vertical or horizontal, and hence there are only $O(n)$ such choices), and then guess which segments crossing this separator belong to an optimal solution. We first obtain solutions for smaller boxes, and then glue together solutions for smaller boxes to obtain solutions for larger boxes until we have a solution for our original problem.

ACKNOWLEDGMENTS

The authors are very grateful to Joseph S. B. Mitchell for valuable discussions and bringing some related works to our attention. The authors also thank the reviewers for their insightful comments and for improving the quality of the paper.

This research was initiated when the fifth author, Mehran Mehr, visited the Department of Computer Science at the University of Copenhagen in March 2017. He wishes to express his gratitude to the other authors and the department for their hospitality.

The work of M. Abrahamsen, A. Roytman, and M. Thorup is partially supported by Thorup's Advanced Grant from the Danish Council for Independent Research under Grant No. DFF-0602-02499B, and partly by Basic Algorithms Research Copenhagen (BARC), supported by Thorup's Investigator Grant from the Villum Foundation under Grant No. 16582. The work of M. Abrahamsen is furthermore supported by the Innovation Fund Denmark through the DABAI project. The work of A. Adamaszek is supported by the Danish Council for Independent Research DFF-MOBILEX mobility grant. The work of V. Cohen-Addad was done as a Postdoctoral Fellow at the University of Copenhagen and supported by the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie Grant Agreement No. 748094. V. Cohen-Addad was also part of BARC. M. Mehr is supported by the Netherlands Organisation for Scientific Research (NWO) under Project No. 022.005.025.

REFERENCES

[1] Mikkel Abrahamsen, Anna Adamaszek, Karl Bringmann, Vincent Cohen-Addad, Mehran Mehr, Eva Rotenberg, Alan Roytman, and Mikkel Thorup. 2018. Fast Fencing. *CoRR* abs/1804.00101 (2018).
[2] Mikkel Abrahamsen, Mark de Berg, Kevin Buchin, Mehran Mehr, and Ali D. Mehrabi. 2017. Minimum perimeter-sum partitions in the plane. In *Proceedings of the 33rd International Symposium on Computational Geometry*. 4:1–4:15.
[3] Pankaj K. Agarwal and Micha Sharir. 1998. Efficient algorithms for geometric optimization. *Comput. Surveys* 30, 4 (1998), 412–458.
[4] Esther M. Arkin, Samir Khuller, and Joseph S. B. Mitchell. 1993. Geometric knapsack problems. *Algorithmica* 10, 5 (1993), 399–427.
[5] Babak Behsaz and Mohammad R. Salavatipour. 2015. On minimum sum of radii and diameters clustering. *Algorithmica* 73, 1 (2015), 143–165.
[6] Vasilis Capoyleas, Günter Rote, and Gerhard Woeginger. 1991. Geometric clusterings. *Journal of Algorithms* 12, 2 (1991), 341–356.
[7] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Cheong. 2008. *Computational geometry: algorithms and applications (3rd edition).* Springer-Verlag.
[8] Stephen G. Dicke and Britt Hubbard. 2008. Tree protection standards in construction sites. (2008). The Forest and Wildlife Research Center, Mississippi State University Extension Service, http://fwrc.msstate.edu/pubs/treeprotection.pdf.
[9] Matt Gibson, Gaurav Kanade, Erik Krohn, Imran A. Pirwani, and Kasturi Varadarajan. 2012. On clustering to minimize the sum of radii. *SIAM J. Comput.* 41, 1 (2012), 47–60.
[10] Sariel Har-Peled. 2011. *Geometric approximation algorithms.* Vol. 173. American Mathematical Society.
[11] Joseph S. B. Mitchell. 2017. Private communication. (2017).
[12] Joseph S. B. Mitchell and Erik L. Wynters. 1991. Finding optimal bipartitions of points and polygons. In *Proceedings of the 2nd Workshop on Algorithms and Data Structures*. 202–213.