Universiteit Utrecht

MASTER THESIS

Finding dense and well-shaped convex clusters in 2-dimensional point sets

Student:
Gianmarco Picarella
g.picarella@students.uu.nl

Supervisors:
Marc J. van Kreveld
Frank Staals
Sjoerd de Vries

Department of Information and Computing Sciences
May 13, 2025

Abstract

We address the problem of computing a convex region with bounded area and diameter, enclosing the maximum number of points from a planar point set P.

This problem originated from previous work on an automatic pipeline for computing the mitotic count (MC) from histological images. The MC counts mitotic cells within a specific region and serves as a key indicator for determining cancer patient risk levels and treatment regimes. In this context, mitotic cells appear as points in a plane, and we need to identify the optimal mitotic hotspot region—a convex region that is not too elongated (bounded diameter), has a bounded area and contains the maximum number of points.

We developed a novel dynamic programming algorithm, the Area-Diameter (AD) algorithm, which solves this problem in $O(n^6k)$ time and $O(n^3k)$ space, where n is the size of P and k is the maximum number of enclosed points. To the best of our knowledge, this is the first polynomial time and exact algorithm to solve this problem.

We implemented the AD algorithm and an existing Area-only (A) algorithm that performs the same task without the diameter constraint in $O(n^3k)$ time and $O(n^2k)$ space. We experimentally compared their performance and solutions for three types of point sets: uniformly distributed sets with increasing size, normally distributed sets with increasing standard deviation and real-world medical data. For the AD algorithm, we test different diameters.

Our results show that despite the higher worst-case complexity of the AD algorithm, the bound on the diameter enables significant pruning that often makes our algorithm practically faster than the A algorithm for moderately dense point sets. The performance of the A algorithm does not change with different point set distributions, but the solutions tend to be too elongated and beyond acceptable limits for medical applications. Finally, we show that our algorithm can process real-world medical datasets in a reasonable time, delivering exact solutions that are better—in terms of the number of enclosed points—than those found by existing methods.

Contents

1 Introduction

1.1 Motivation

Breast cancer is among the most prevalent cancer types in the world and the primary cause of mortality due to cancer in women. According to the World Health Organization (WHO), breast cancer was responsible for $670,000$ deaths worldwide in 2022, and it is expected to reach approximately 1.5 million deaths by 2050 [19].

To improve early detection and survival rates, medical images are examined by experts to detect anomalies and provide effective prognostic. Pathologists use histological grading to assess a tumour by examining how abnormal the cancer cells and tissue appear under the microscope. This procedure, which is considered one of the most powerful prognostic tools for early stage invasive breast cancer, allows to estimate the likely growth and spread rate of the cancer cells. After observing specific features, pathologists use a standardized system such as the Nottingham grading system [15] to assign a grade to the tumour. Among many tumour features, the mitotic count (MC) has been shown to provide the strongest prognostic value [5].

The MC is a density measurement usually expressed as the number of mitoses per mm^2. Up until now, pathologists have been measuring this value by hand, using the following procedure. First, the pathologist selects the Whole-Slide Image (WSI; digitalized histological slide) containing a region that visually shows the highest tumour proliferation. Then, within this region, the pathologist locates a sub-region of area $2mm^2$ containing the highest number of mitotic cells. The MC is simply the number of mitotic cells within the sub-region located in the second step.

Several recent studies [17, 4] showed that both the selection of areas and the identification of mitotic cells are prone to variability between observers and even within the same observer over time, resulting in limited accuracy and reproducibility hampering the clinical utility of the MC. Consequently, there is a growing need for computer-assisted procedures to support pathologists in achieving more consistent and accurate results.

1.2 Current challenge

The work from Stathonikos et al. [16] presents a fully automated pipeline for the computation of the MC from an input WSI. This pipeline employs

the classification model from [18] based on a convolutional neural network capable of detecting mitoses with high accuracy. The model was trained with the MIDOG21 dataset [3], a collection of WSIs with annotated mitotic figures by expert pathologists.

The classification model produces a planar point set representing the locations of the detected mitotic figures. Then, the pipeline from [16] uses an area selection (AS) algorithm that automatically locates the mitotic hotspot region. According to medical specialists, a mitotic hotspot region is convex, has an area up to 2mm^2 [11], a maximum aspect ratio 1:8, and encloses the maximum number of points.

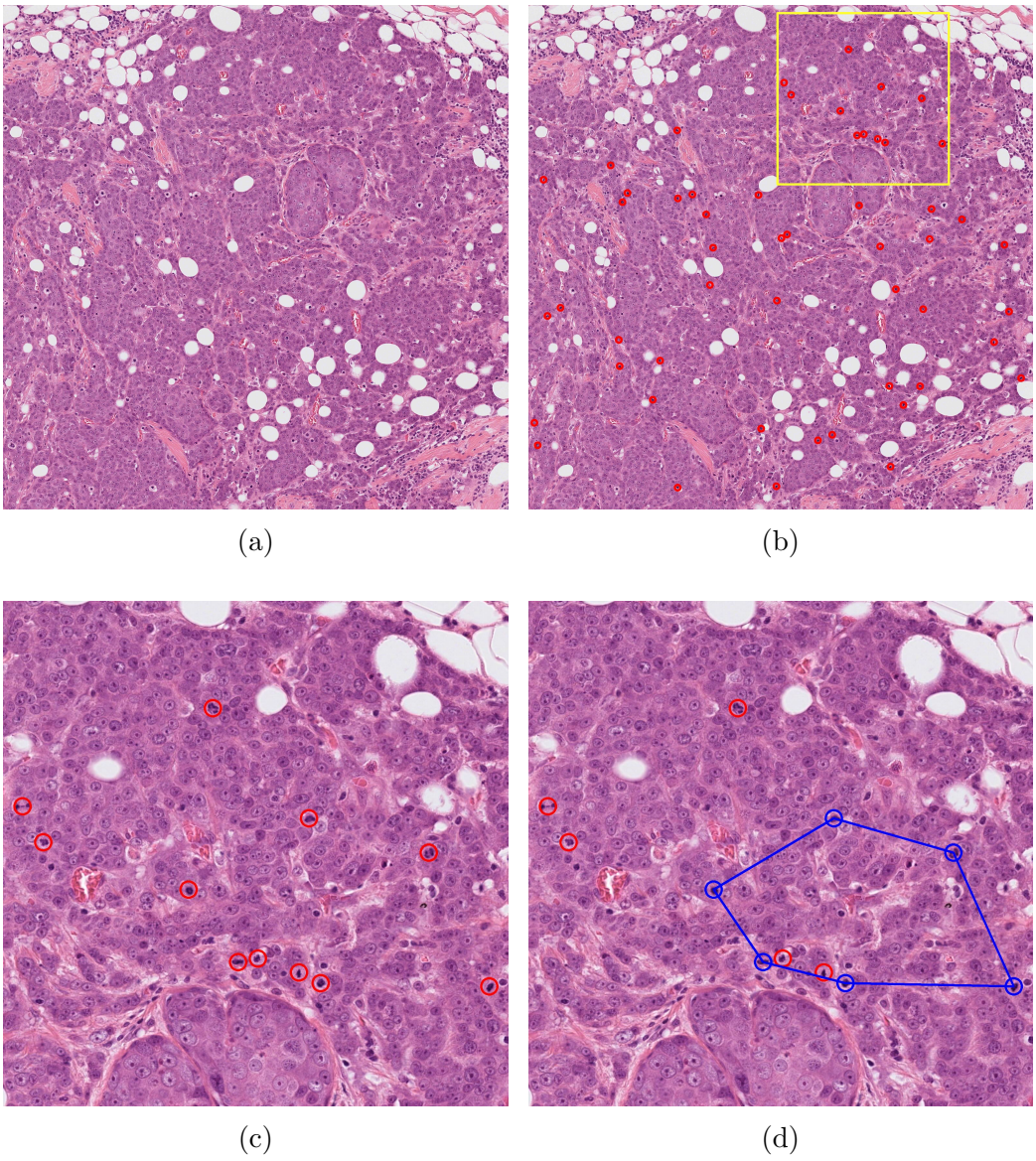
The AS algorithm first subdivides the point set into square patches. Then, each patch is processed with one of the two following algorithms. The first is a brute-force algorithm, which is exact but takes exponential time with respect to the number of points. The second is a heuristic algorithm running in polynomial time, which uses a greedy approach to shrink the convex hull of the input set until the mitotic hotspot requirements are satisfied. We note that the heuristic approach does not come with any approximation guarantee regarding the optimality of the output.

Therefore, the main challenge is to find an exact algorithm running in polynomial time that can automate the process of finding mitotic hotspot regions. In Section 1.4, we provide a formal definition of the problem we seek to solve.

1.3 Measures of compactness

The compactness constraint was originally stated in terms of aspect ratio. We focus on discrete convex regions, which are regions defined by a finite sequence of vertices. Initially, we considered using the width and diameter of the region to compute the aspect ratio and thus bound the shape. The width and diameter are determined by the length of the segment connecting the pair of closest and furthest vertices, respectively.

Given that there is already a clear constraint on the area of the region, we also explored other approaches to reuse this information in conjunction with one additional constraint defined on another measure, in an attempt to obtain full control over the range of allowed shapes as with the aspect ratio. We considered integrating the perimeter and diameter. Using a combination of area and perimeter looked like a promising approach. However, it is unclear whether finding the region maximizing the number of enclosed points with



(a) (b)

(c) (d)

Figure 1: (a) Input breast cancer tissue. (b) Overview on low magnification of the deep learning model output. Every detected mitotic figure is marked with a red dot. The detected hotspot area is marked in yellow. (c) High magnification of the hotspot area shown in (b). (d) Identical area as in (c) in which the automatic area selection located the highest proliferation (HP) area. The mitotic figures enclosing the HP area are marked in blue.

bounded area and perimeter is possible in polynomial time.

In the end, we opted to introduce an additional constraint on the diameter of the shape. While bounding the aspect ratio is not equivalent to bounding the diameter, combining it with a bounded area effectively achieves the same goal of identifying regions that are not overly elongated. We note that a bound on the aspect ratio also ensures a bound on the diameter, but the reverse is not true. Let us consider the case in which all points are (nearly) collinear. In this case, the convex hull of any subset of these points cannot attain an upper bound on the aspect ratio, whereas a bounded diameter region always exists.

We use the maximum allowed diameter $D_{\max}$ to set an upper bound on the major axis length of the shape. We also get an upper bound on the minor axis length d by setting the maximum allowed area $A_{\max}$ as

$$A_{\max} = \frac{\pi d^2}{4} \quad \text{with } 0 < d \leq D_{\max} \tag{1}$$

The diameter is conceptually simpler than the aspect ratio and turned out to integrate seamlessly into our dynamic programming algorithm. We provide more information regarding this in Section 4.

1.4 Problem formulation

Given a set of points, we seek a convex region that encloses the maximum possible number of points. This region must have bounded diameter and area, and among all shapes that contain the maximum number of points, we seek the one with the minimum possible area. This optimization problem is a variation of the geometric knapsack problem, a class of problems that have been studied for various objective functions and constraints [2].

In this work, we focus on an optimization problem that, to the best of our knowledge, has not been studied in existing research. The problem is defined as follows.

Problem 1. *Given a set of 2-dimensional points P, a maximum area $A_{\max}$, and a maximum diameter $D_{\max}$, find a subset $S \subseteq P$ such that the convex hull of S satisfies the following properties:*

1. *The diameter of S does not exceed $D_{\max}$.*
2. *The area of the convex hull of S is minimal among all subsets of P with exactly $|S|$ points and does not exceed $A_{\max}$.*

3. S contains the maximum possible number of points.

1.5 Objective

The goal of this thesis is to address Problem 1 both algorithmically and experimentally. We consider this goal of valuable interest, considering the scarcity of previous research addressing our specific problem and the positive impact such a technique could have on the development of new automatic procedures for breast cancer prognosis. Therefore, this thesis has the following sub-goals. (1) The identification of existing algorithms applicable to our problem and their limitations. (2) The definition of a new algorithm, dealing with the limitations of existing methods, which is exact and able to solve realistic instances of Problem 1 efficiently. (3) The implementation of these algorithms with careful attention to correctness and performance. (4) The empirical evaluation of the performance (runtime, memory and number of entries used) and solutions (number of enclosed points, area and diameter of solutions) obtained with these algorithms using synthetic and real-world data.

1.6 Contributions

The three main contributions of this thesis are listed below.

- The Area-Diameter (AD) algorithm, a novel dynamic programming algorithm finding the convex region with area bounded by $A_{\max}$ and diameter bounded by $D_{\max}$, enclosing the maximum number of points in $O(kn^6)$ time and $O(kn^3)$ space, where n is the size of the point set and k is the number of points enclosed by the optimal solution. Among all possible solutions containing k points, we select the one with the minimum area.
- The open source C++ implementation of the Area-only (A) algorithm presented in Eppstein et al.'s work [8] and the AD algorithm. These implementations are designed to process arbitrary point set configurations, including degenerate cases where points are not in general position or have the same x or y coordinate.

- The evaluation of the performance (runtime, memory and number of entries used) and solutions (number of enclosed points, area and diameter) obtained with the A and AD algorithms using synthetic data, examining how these are affected by the point set density, spread and maximum allowed diameter $D_{\max}$. Finally, we compare the solutions obtained with these two algorithms and the AS algorithm using the real-world data from the medical application [16] that motivated our research.

1.7 Organization

The relevant related work is discussed in Section 2. A brief introduction to the geometric knapsack problem and the related challenges is first provided. In Section 3, we describe the AS algorithm [16], which motivated our research, and the A algorithm from Eppstein et al.'s work [8]. In Section 4, we introduce the AD algorithm for finding exact solutions to instances of Problem 1. In Section 5, we describe the experimental setup and data used. In Section 6, we present and discuss the experimental results. We consider the performance (runtime, memory and number of entries used) and output features (i.e. the number of enclosed points, area and diameter of the solutions) obtained by the A, AS and AD algorithms for maximum diameters $D_{\max} = 2, 3, 4, 5$, and 6 mm. The conclusions in Section 7 will state that our experimental results show the AD algorithm performs efficiently in real-world scenarios (for specific point sets and values of $D_{\max}$), making it a valuable contribution to the existing literature. Also, we briefly discuss a way to immediately enhance the capabilities of the AS algorithm by integrating the A algorithm. Finally, we discuss the limitations of the AD algorithm and propose several research directions for future work.

2 Related work

The work of this thesis is motivated by the work of Stathonikos et al. [16], which introduced the problem of computing a convex region with bounded area and aspect ratio, enclosing the maximum number of points. Their work describes a fully automated mitosis detection pipeline, which is then extended with an Area Selection (AS) algorithm used to automatically locate the mitotic hotspot region. This method initially subdivides the region with points into partially overlapping square patches—the overlap is needed to consider the convex regions displaced over two patches. It then finds the best convex region within each patch and selects the overall best one.

Two algorithms are available for processing these patches: a brute-force algorithm and a heuristic algorithm. The brute-force algorithm produces exact results by testing all the possible $O(2^n)$ convex regions until it finds the optimal solution. Therefore, it is clear this algorithm can only be used for patches containing just a few points. The heuristic algorithm uses a greedy approach that shrinks the convex hull of the points contained inside the patch until the aspect ratio and area constraints are satisfied. It runs in $O(n^3)$ time and uses $O(n)$ space but does not guarantee to find the optimal solution nor provide any approximation guarantee.

The inefficiency and limited usage of the exact algorithm, the lack of approximation guarantees for the heuristic algorithm, and the need to find exact solutions for big point sets are the main reasons driving the work of this thesis.

2.1 Geometric knapsack problem

The knapsack problem is a combinatorial optimization problem which requires to fill a given knapsack of limited capacity with items from a given collection such that the sum of the values of the included items is maximized. The geometric knapsack problem is an extension of the classic knapsack problem to a geometric setting.

These problems asks us to find a maximum-value subset of a given set of points that are to be placed within a knapsack (convex polygon) while obeying a given capacity (area) constraint. More specifically, these problems are called fence enclosure problems. The challenge arising in the geometric version of the knapsack problem is that the cost of adding a new item to the knapsack dynamically changes based on the previously chosen items.

Arkin et al. [2] presented a complete analysis of several variations of the geometric knapsack problem. They discuss the fence enclosure problem for limited area or perimeter, and unlimited area and perimeter. They derive polynomial and pseudo-polynomial time algorithms for these scenarios, including an algorithm satisfying requirements 2 and 3 of Problem 1 in $O(kn^3)$ time and $O(kn^2)$ space.

2.2 Finding convex polygons with optimal area and k points

The work from Eppstein et al. [8] presented an $O(kn^3)$ time and $O(kn^2)$ space dynamic programming algorithm for finding minimum area convex k-gons. We refer to this as the area-only (A) algorithm. With minimal adjustments, their approach works for three different problems, namely: find a convex k-gon with minimum area; find an empty convex k-gon with minimum area; find a subset $S \subseteq P$ such that the area of the convex hull of this subset is minimal. We focus on this last problem, as it models requirements 2 and 3 of Problem 1. This approach is further generalized to work with any monotone decomposable weight function (MDF) defined on any convex polygon $\mathcal{C}$.

Definition 1. *A weight function W is decomposable iff for any polygon $\mathcal{C} = \langle p_1, ..., p_m \rangle$ and any index $2 < i < m$*

$$W(\mathcal{C}) = \diamond(W(\langle p_1, ..., p_i \rangle), W(\langle p_1, p_i, p_{i+1}, ..., p_m \rangle), p_1, p_i)$$

where $\diamond$ takes constant time to compute. W is called monotone decomposable iff $\diamond$ is monotone in its first (and, hence, in its second) argument.

When W is decomposable we can cut the polygon $\mathcal{C}$ into two subpolygons $\mathcal{C}_1$ and $\mathcal{C}_2$ along the segment p_1p_i and derive $W(\mathcal{C})$ from $W(\mathcal{C}_1)$, $W(\mathcal{C}_2)$ and some information about the cut segment. Some examples of MDF include the area, perimeter, sum of internal angles and the number of points enclosed by $\mathcal{C}$. We note that the diameter is not an MDF. Therefore, it is not possible to apply this algorithm to enforce it.

Several improvements on top of these results have been presented in later work by Eppstein in [7]. Given a point set P, Eppstein shows that the minimum area convex hull of k points is found in a small subset of P, namely the nearest vertical neighbours of a (non-vertical) segment determined by two of the input points. Eppstein provides a detailed description of a data

structure that allows to query the k nearest vertical neighbours from any pair of points in $O(kn^2 + n^2 \log n)$ time and $O(n \log n)$ space. The minimum convex hull is then found by applying the A algorithm to these small sets in $O(n^2 \log n + k^3 n^2)$ time and $O(n \log n + k^3)$ space, offering an asymptotic time improvement over [8] whenever $k = O(\sqrt{n})$. The space used is always an improvement over the previous $O(kn^2)$. Using the same data structure, the problem of finding the minimum area convex k-gon is treated with an algorithm running in $O(n^2 \log n + 2^{6k} n^2)$ time, offering an asymptotic improvement over [8] whenever $k < \log n/6$. For both problems, the vertical nearest neighbours approach is limited to area minimization tasks and seems to be not extendable to other MDFs.

A similar approach to the one presented in Eppstein's work [7] was previously used by Aggarwal et al. in [1] to find small sets for minimum diameter problems using higher-order Voronoi diagrams. More recently, Keikha [12] presented an asymptotic improvement for the A algorithm [8] leading to $O(n^3 \log k)$ time for point sets in convex position by using a divide-and-conquer technique which merges two convex polygons of size 2^t at each iteration, for $t = 0, ..., \log k$.

The work from Mitchell et al. [13] uses similar algorithms to that of Eppstein et al. [8] for counting convex k-gons, all convex polygons, all empty convex polygons and all convex polygons containing a given point.

Other related works are the one from Fisher [10], which proposed an $O(n^3 \log n)$ time algorithm for finding maximum area convex hull of k points, defined by points labelled as "positive" and without containing any point labelled as "negative". If the set of "negative" points is empty, then the result is the convex polygon having maximum area.

2.3 Finding convex polygons with minimum diameter and k points

Aggarwal et al. [1] proposes an $O(k^{2.5} n \log k + n \log n)$ time and $O(kn)$ space algorithm for finding a set of k points with minimum diameter. They show that the minimum diameter convex hull with k points is within one of the $O(kn)$ Voronoi sets in the $(3k-3)$th order Voronoi diagram. We have presented several optimization algorithms satisfying some of the requirements for Problem 1 but not all. The discussed approach is now satisfying requirements 1 and 3 of Problem 1. To the best of our knowledge, no polynomial

time algorithm exists that finds the convex region enclosing the maximum number of points with bounded diameter and area.

2.4 Statistical clusters analysis

Statistical clustering methods, such as DBSCAN [9], rely on clustering points based solely on pairwise distances, without considering the constraints in terms of area and diameter of the convex hull formed by the clustered points. Additionally, these techniques usually require careful parameter selection which may lead to significant differences in the outcomes even for the same input set.

3 Selected algorithms

In this section, we describe two existing algorithms that are related to our problem. In Section 3.1, we describe the Area Selection (AS) algorithm introduced in [16]. This work addresses Problem 1 but comes with several limitations. In Section 3.2, we describe the Area-only (A) algorithm [8]. This algorithm does not enforce a constraint on the diameter of the solution. Still, it may be possible that for specific point set distributions and densities, the found region satisfies the diameter constraint even if that is not enforced explicitly.

3.1 Area selection algorithm

Given an input set P, the AS algorithm comprises two phases. First, P is partitioned by square patches that partially overlap. Second, the points in each patch are processed independently using the brute-force algorithm (for patches with less than 26 points) or the heuristic algorithm as a fall-back. Below, we first explain how the partitioning works, then describe both algorithms.

3.1.1 Partitioning

Partitioning the input set has two purposes: first, it bounds the diameter of the solutions found by the brute-force and heuristic algorithms to the length of the diagonal of the square patch. Second, it limits the processing to (possibly) smaller point sets, thus reducing the amount of computational resources needed.

Let l be the side length of a patch, and let s be the horizontal and vertical step separating two adjacent patches. These parameters are carefully selected to satisfy the diameter constraint while considering the target accuracy and the available computational resources. Suppose $l = 2D_{\max}$ and $s \leq 0.5l$, and each patch is processed using the exact algorithm. Then, the final solution will likely be the global optimum with bounded area and diameter among all valid convex regions in P. This is because we also consider the convex regions defined by points in two consecutive patches.

It is important to note that this partitioning technique does not guarantee to find the optimal solution for a specific diameter bound. In fact, because the diagonal length of a patch defines the maximum allowed diameter, we

may still miss valid regions having diameter within $2D_{\max}$ and $2D_{\max}\sqrt{2}$. This compromise seems difficult to avoid.

Using values of $s < 0.5l$ does not bring any advantage. Actually, it brings disadvantages in terms of computational resources used. In practice, halving the step size leads to about 4x the number of patches, adding considerable extra work.

3.1.2 Brute-force algorithm

Given a patch containing n points, the brute-force algorithm computes the minimum area convex polygon enclosing the maximum number of points, with area bounded by $A_{\max}$ in $O(2^n)$ time and $O(n)$ space.

The algorithm constructs all the convex paths originating from any point p_0 in the patch. Let $p_0, \ldots, p_{i-1}$ be a convex path containing i points. At the $(i-1)$-th step, the algorithm selects the next point p_i from the list of points sorted in ccw order around p_{i-1}, choosing only those that keep the path convex and have not yet been visited.

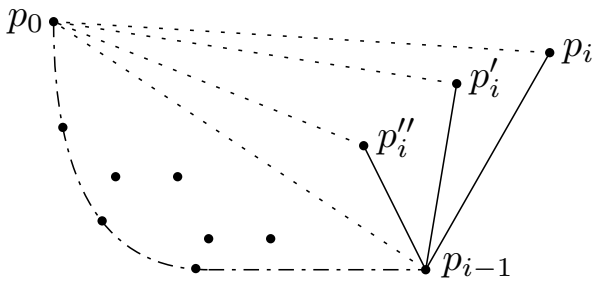


Figure 2: Convex paths generated by the brute-force algorithm from point p_0. The points following p_{i-1} are processed in the order p_i, p'_i and p''_i.

The points on the left side of the line segment $\overline{p_{i-1}p_0}$ are not valid successors of p_{i-1} and any other vertex further in the sequence. For this reason, they are counted as internal points of the current convex path. Figure 2 shows this situation. The base cases for the recursion are two: either the area of the convex path exceeds $A_{\max}$, or there are no points left to visit.

If the points in the patch define a convex set, then the algorithm will process 2^n different convex paths. Even for non-convex sets, the average number of convex paths is nearly exponential.

3.1.3 Heuristic algorithm

Given a patch containing n points, the heuristic algorithm computes the convex hull of these points and uses a greedy approach to shrink it until its area is bounded by $A_{\max}$.

We describe how the greedy approach works. Let $p_0, \ldots, p_i$ be the sequence of ccw vertices defining the convex hull of the input set. The greedy step looks for a shrink that removes as much area as possible while discarding as few internal points as possible. Given three consecutive points p_j, p_{j+1}, p_{j+2} defining the boundary of the convex hull, the greedy approach either removes p_{j+1} or replaces it with q, where q is any point within the triangle $\triangle p_j p_{j+1} p_{j+2}$. Among all, the optimal shrink is the one that is applied. This procedure is repeated until the area of the resulting convex polygon is bounded by $A_{\max}$.

This algorithm runs in $O(n^3)$ time and uses $O(n)$ space. Each greedy step runs in $O(n^2)$ time because there are n possible shrinks to consider, and the number of points excluded by each shrink is determined in $O(n)$. This step is repeated $O(n)$ times, thus leading to the stated time complexity $O(n^3)$.

It is unclear whether this algorithm has an approximation guarantee about the number of enclosed points in the exact solution. In general, it is reasonable to expect that the distribution of the points affects the quality of the result.

3.2 Area-only algorithm

We describe the A algorithm applied to the problem of finding the minimum area convex polygon enclosing k points. Before that, we delve into a preprocessing technique that allows us to determine the number of points inside any triangle in P in constant time. This preprocessing is used in the A and AD algorithms.

Let us define a total order $\prec$ on the set of planar points as

$$p \prec q \Leftrightarrow p_x < q_x \vee (p_x = q_x \wedge p_y < q_y) \tag{2}$$

where p, q are two arbitrary planar points. We will use this ordering to sort points throughout this section.

3.2.1 The number of points in all triangles

Given an input set P containing n points, Eppstein et al. [8] presented a preprocessing technique which allows to count the number of points in any triangle $\triangle abc \in P$ in constant time. This technique originally uses $O(n^2)$ time and $O(n^2)$ space.

In contrast to Eppstein et al.'s technique, our approach does not assume P to be in general position nor that all points in P have different x-coordinates. For this reason, our approach slightly deviates from the one originally proposed in [8]. We are able to preprocess any input set in time $O(n^2 \log n)$. By default, we do not count any point that is collinear with any of the edges of the queried triangle.

The preprocessing technique produces the following three arrays of integers. $B[\cdot,\cdot]$ is an array of n^2 entries storing the number of points within the vertical stripe below any line segment defined in P, excluding points vertically below the segment endpoints. $C[\cdot,\cdot]$ is an array of n^2 entries storing the number of points collinear to any line segment defined in P. $V[\cdot]$ is an array of n entries storing the number of points vertically below any point in P.

Algorithm 1: Count points in triangle $\triangle lmr$ in $O(1)$ time

Input: Triangle points l, m, r sorted using $\prec$
Output: Number of points in triangle $\triangle lmr$
if l, m, r *are collinear* **then**
  return 0;
end
if l_x *equals* m_x **then**
  return $B[m,r] - B[l,r] - C[l,r]$;
end
if m_x *equals* r_x **then**
  return $B[l,r] - B[l,m] - C[l,m]$;
end
if m *lies left of* $\overrightarrow{rl}$ **then**
  return $B[l,r] - B[l,m] - B[m,r] - C[l,m] - C[m,r] - V[m] - 1$;
end
if m *lies right of* $\overrightarrow{rl}$ **then**
  return $B[l,m] + B[m,r] + V[m] - B[l,r] - C[l,r]$;
end

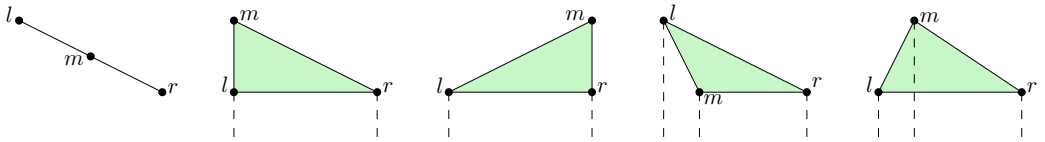


Figure 3: The five possibilities for the triangle $\triangle lmr$

Given three points $l, m, r \in P$ sorted using $\prec$, we now describe a procedure using the previously introduced arrays to compute in $O(1)$ time the number of points in the triangle $\triangle lmr$. Let us focus on the possible disposition of l, m, r. We show these cases in Figure 3. If the three points are collinear, then the number of enclosed points is exactly 0. If l, m have equal x-coordinates, then l must be below m. Therefore, we subtract the points below or collinear to $\overline{rl}$ from the points below $\overline{rm}$. The same reasoning applies when m, r have equal x-coordinates. If m is on the left side of $\overline{rl}$, then m is below $\overline{rl}$. We subtract 1 plus the points below m and the points below or collinear to $\overline{rm}$ and $\overline{ml}$ from the points below $\overline{rl}$. This way we ensure m and any point below it is not counted. The last case is when m is on the right side of $\overline{rl}$. In this case, we subtract the points exactly below m and the points below or collinear to $\overline{lr}$ from the points below or collinear to $\overline{lm}$ and $\overline{mr}$. From now on, we will use $K(a, b, c)$ to refer to this procedure, where a, b, and c are points in P. We summarize this procedure in Algorithm 1.

We now describe how the preprocessing procedure fills the arrays of integers B, C, V. Initially, all the entries $B[\cdot, \cdot], C[\cdot, \cdot]$ and $V[\cdot]$ are set to 0. Then, we sort the points in P using $\prec$. Since points having the same x-coordinate appear consecutively in P, we fill V with a single pass over the sequence. After that, we proceed with the filling of arrays B and C. The main idea here is to process all the line segments in P in a specific processing order that allows to compute the points below or collinear to the next line segment based on a constant amount of information retrieved from the previously processed segments. In order to do so, we treat the line segments from left to right (using $\prec$), according to their right endpoint. Line segments with the same endpoint are treated in clockwise order. This sorting can be easily performed in $O(n^2)$ time and space with the results from Edelsbrunner et al. [6], when P is in general position. In our scenario, we do not require this to be true. Consequently, we decided to slightly sacrifice performance and use a classical approach which works for any point set in $O(n^2 \log n)$ time and $O(n^2)$ space. We first sort the points by angle, and when angles are equal,

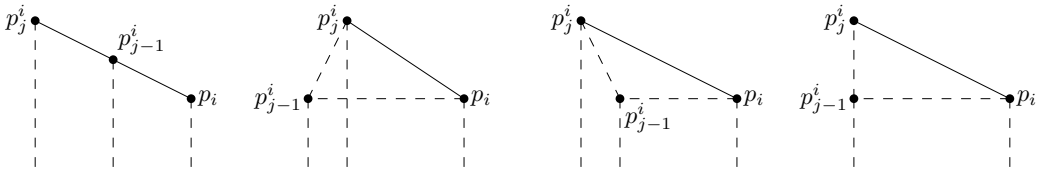


Figure 4: The four possibilities for points p_i, p_{j-1}^i and p_j^i

we sort by distance from the right endpoint. The result is the sequence of sorted points $p_1^i, p_2^i, \ldots, p_{i-1}^i$ for any $p_i \in P$. This enables us to count points below or collinear to any line segment in P by using an approach similar to Algorithm 1. The count relies only on information from one or two previous segments.

At each step, we keep track of three points, namely the right endpoint p_i, and the current and previous left endpoints p_j^i, p_{j-1}^i. Here we also consider the situation in which p_j^i and p_{j-1}^i have same x-coordinate. This introduces several edge cases that we now discuss. We provide the complete pseudocode covering all the edge cases in Appendix A.1. As for the counting procedure, we deal with several dispositions of the points p_i, p_j^i, p_{j-1}^i. The four possible cases we discuss below are shown in Figure 4. While we do not provide visuals for all possible sub-cases, our explanation and the included figures should make them easy to understand.

1. The first case is when p_i, p_j^i, p_{j-1}^i are collinear. In this case, the number of collinear points along the current segment is set to the points collinear to the previous segment plus 1. We also consider the case in which the points are collinear and have unequal x-coordinate. In this case, we also count the number of points below the current segment by considering the points below $\overline{p_i p_{j-1}^i}$ and $\overline{p_{j-1}^i p_j^i}$ plus the points exactly below p_{j-1}^i.

2. The second case is when p_j^i has greater x-coordinate than p_{j-1}^i and p_{j-1}^i is not placed exactly above p_i. The number of points below the current segment is set to the points below or collinear to the segment $\overline{p_i p_{j-1}^i}$ minus the points below the segment $\overline{p_j^i p_{j-1}^i}$ and the points exactly below p_j^i.

3. The third case is when p_j^i has smaller x-coordinate than p_{j-1}^i. The number of points below the current segment is set to the points below

or collinear to the segments $\overline{p_i p^i_{j-1}}$ and $\overline{p^i_{j-1} p^i_j}$. In addition, if p^i_{j-1} does not lie exactly below p_i, we sum 1 plus the points exactly below p^i_{j-1}.

4. If none of the previous cases apply, then p^i_j has same x-coordinate as p^i_{j-1}. In this case, we simply copy the number of points below to the previous segment to the new one.

The proof of correctness in Eppstein et al.'s work also applies for our preprocessing technique. We have tested our technique with point sets containing all the possible degenerate cases and we have validated the results using a brute force approach.

3.2.2 Finding a minimum area convex polygon enclosing k points

Given a point set P with n points, let us consider the optimal convex polygon $\mathcal{C}$ enclosing the maximum number of points k with area bounded by $A_{\max}$. $\mathcal{C}$ is a valid solution for the diameter-free version of Problem 1.

Let p_1 be the bottommost vertex of $\mathcal{C}$ and let p_2 and p_3 be the next two vertices in counterclockwise order from p_1. We can always cut $\mathcal{C}$ along the line segment $\overline{p_1 p_3}$ and obtain the convex polygon $\mathcal{C}'$ with $k' = k - K(p_1, p_2, p_3) - 1$ points, and the triangle $\triangle p_1 p_2 p_3$. $\mathcal{C}'$ is optimal among all the possible convex polygons enclosing k' points with p_1 as the bottommost vertex, p_2 as the next ccw vertex, and with all of its points left of (or collinear to) the directed line through p_2 and p_3, and in this direction.

The A algorithm exploits this observation and uses a dynamic programming technique to compute the minimum area convex polygon in $O(kn^3)$ time and $O(kn^2)$ space. They use a four-dimensional array such that the entry $T[p_i, p_j, p_l, m]$ contains the area of the minimum area convex polygon such that:

1. p_i is the bottommost vertex.
2. p_j is next counterclockwise vertex after p_i.
3. The subset of points generating the convex region has exactly m points.
4. All the points are left of the directed line through p_j and p_l, and in this direction.

We can reduce the amount of storage needed for the array T by observing that we do not need to store the subarray $T[p_i, \cdot, \cdot, \cdot]$ for all $p_i \in P$. We need to store the subarray for the $p_i^\star$ that led to the current optimum plus another subarray to store the computations for the current p_i. This approach allows us to reduce by a factor n the space allocated for T. Therefore, from now on, we will consider the array $T[\cdot, \cdot, \cdot]$.

The goal is to fill the array T up to $m = k$ for any point p_i. In the initialization phase, we set $T[\cdot, \cdot, 2] = 0$ (as the area of a 2-gon is always zero). The remaining entries are all set to ∞.

Assume we have already filled all entries in T up to $m - 1$. We provide a technique to fill all the entries in T for m and some fixed points p_i and p_j in $O(n)$ time. Given that p_i is the bottommost point, all points p_j and p_l must lie above the horizontal line passing through p_i. Fix the attention to one point p_i at the time. We process these points sorted by increasing y-coordinate.

We reuse the array of points sorted in cw order around all points from Section 3.2.1. For each p_i, we process the clockwise sorted sequence of points p_j that are above or collinear to the horizontal line passing through p_i. For each p_j, we process the sequence of points p_l sorted in descending order by the slope of the half-lines passing through p_l and p_j, with wrap-around. That is, we treat slopes cyclically—points defining half-lines with slope $-\infty$ follow those defining half-lines with slope ∞, and we do not distinguish on which side of $\overline{p_i p_j}$ the point p_l lies. We start processing this sequence from the point after p_i. Let $\text{prec}(p_l)$ be the point before p_l in the sequence and let $A(p_i, p_j, p_l)$ be the area of triangle $\triangle p_i p_j p_l$. The dynamic programming step works as follows.

Consider the case where p_l is right of the directed line passing through p_i and p_j, and in this direction. Then, no new point can be used to generate a new convex polygon. Therefore, $T[p_j, p_l, m]$ is equal to $T[p_j, \text{pred}(p_l), m]$.

Now, let $m' = 1 + K(p_i, p_j, p_l)$ and consider the case where p_l is left or collinear to the directed line passing through p_i and p_j, and in this direction. Then, attaching the triangle $\triangle p_i p_j p_l$ to the previously found optimal convex polygon at $T[p_l, p_j, m - m']$ could generate a new optimal convex polygon with m points as shown in Figure 5. We briefly explain why this is the case. First, the resulting shape is guaranteed to be convex, as by definition, the solution in $T[p_l, p_j, m - m']$ has all of its points left of the directed line passing through p_j and p_l, and in this direction. Second, every triangle adds at least one point (namely p_j) to the overall cardinality of the solution plus the

points enclosed in it. Given we look at the entries stored in $T[\cdot, \cdot, m - m']$, our new solution is guaranteed to contain exactly m points. If the area of this new convex polygon is also smaller than the current best solution found while sweeping around p_j, then $T[p_j, p_l, m]$ is equal to $A[p_j, p_l, m - m']$ plus $A(p_i, p_j, p_l)$. Otherwise, $T[p_j, p_l, m]$ is set to the previous optimum value at $A[p_j, \text{pred}(p_l), m]$ and nothing changes. We summarize the dynamic programming step in Equation 3 and provide the pseudocode of the A algorithm in Appendix A.2.

$$
\begin{aligned}
T[p_j, p_l, m] &= \begin{cases} 0 & \text{if } m = 2 \\ \min\{T[p_j, p_l, m], \triangle^{\text{new}}\} & \text{if } p_l \text{ left of (or collinear to) } \overline{p_i p_j} \\ T[p_j, \text{pred}(p_l), m] & \text{otherwise} \end{cases} \\
\triangle^{\text{new}} &= \begin{cases} \min_q\{T[q, p_j, m - m'] + A(p_i, p_j, p_l)\} & \text{if } m' < m \\ \infty & \text{otherwise} \end{cases} \\
m' &= K(p_i, p_j, p_l) + 1 \\
q &= \text{any predecessor of } p_l \text{ in the sorted sequence around } p_j
\end{aligned}
\tag{3}
$$

Therefore, we are able to process all points p_l in $O(n)$ time. Given that there are $O(n^2)$ segments $\overline{p_i p_j}$ and the entire procedure is repeated for increasing values of m up to $m = k$, the overall amount of work required by the A algorithm is $O(kn^3)$.

Let us now examine how to reconstruct the optimal convex polygon enclosing k points. We show this can be done in $O(kn)$ time. The main idea is to walk backward from $T[p_j, p_l, k]$ and decompose the optimal solution back into the triangles that we initially assembled together. We start with the points p_i and p_l that led to the optimal solution with k points. In the initialization step, we insert p_i in the array of vertices defining the optimal convex region. Then, we repeat the following steps until $k = 0$. Let p_{prev} be the last vertex inserted in the array. We look for the point q minimizing $T[p_j, q, k]$, that is left of the directed line passing through p_{prev} and p_j, in this direction. This is done in $O(n)$ time. Then, we update the remaining number of points to $k = k - K(p_i, p_j, q) - 1$, we append p_j to the array of vertices and set $p_j = q$. Once $k = 0$, we append the last point p_j to the array of vertices. The optimal solution consists of $k - 2$ triangles. Finding each triangle takes $O(n)$ time. This gives us a total runtime of $O(kn)$. You can

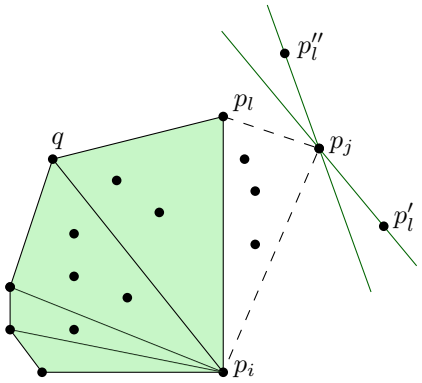


Figure 5: The triangle $\triangle p_i p_j p_l$ is attached to the solution stored at $T[p_l, q, m - K(p_i, p_j, p_l) - 1]$. Table entries are computed for all p_l in $O(1)$ by rotating around p_j.

find the pseudocode for this reconstruction algorithm in Appendix A.2.

3.2.3 Implementation details

We now discuss several details of our implementation of the A algorithm.

We have considered a technique that allows us to reduce the memory allocations in the A algorithm. Consider the dynamic programming step, which processes the sorted sequence of points p_l around p_j. If p_l does not generate a new optimal result, the value stored in the current entry is copied from the previous entry. This could lead to many duplicate values in T. If we limit ourselves to store only the unique optimum values generated for each segment $\overline{p_i p_j}$ in the same order as we process the points p_l, then for an additional $O(\log n)$ factor, we can binary search over this smaller sequence to find the minimum area. We have not implemented this, but we keep track of the number of unique optimum areas to evaluate the potential memory saving using this approach.

We mentioned that we process the points p_i by increasing y-coordinate. This allows us to devise an early exit strategy that does not affect the optimality of the result but likely provides a speedup. The early exit strategy is based on the following observation. If the current optimal solution has m points and the next p_i has only $m - 1$ points above it, we already know that we will not find any better solution than the current one. This means we can stop the procedure and return our current optimal value.

Finally, we consider processing point sets that are not in general position. Our implementation can process any input set, as it deals with collinearity and points having the same x-coordinates. In this context, it is important to consider how to sort the points p_l around p_j by slopes of the lines $\overline{p_j p_l}$ when some of these lines may be identical. We deal with this issue by prioritizing the left side. This allows to count the points in the middle of a side.

4 Area-Diameter algorithm

4.1 Finding a minimum area convex polygon with bounded diameter enclosing k points

We present the Area-Diameter (AD) algorithm, a novel algorithm for finding the convex polygon enclosing the maximum number of points k, with area bounded by $A_{\max}$ and diameter bounded by $D_{\max}$ in $O(kn^6)$ time and using $O(kn^3)$ space. The algorithm finds the optimal region by processing all pairs $p, q \in P$ such that $|pq| \leq D_{\max}$, each time assuming that $|pq|$ is the diameter.

Let C_{pq} be any convex polygon with diametral pair p, q defined solely by the points within the intersection region of two disks of radius $|pq|$ centered at p and q, as shown in Figure 6. Using any point outside this region would create a diameter for C_{pq} larger than $|pq|$, contradicting our initial assumption.

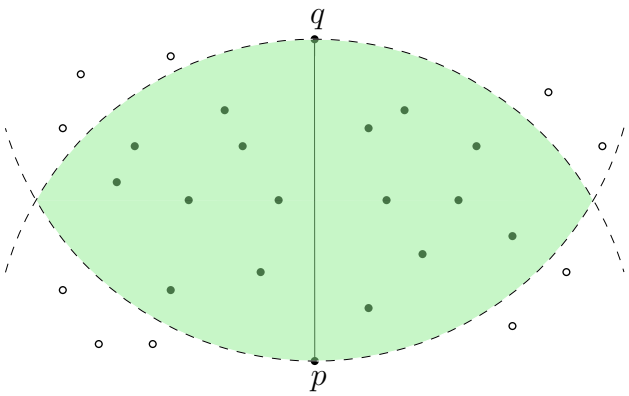


Figure 6: The intersection region of the disks with radius $|pq|$ centered at p and q.

The goal is to find the sequences of counterclockwise (ccw) vertices from p to q and q to p defining the optimal C_{pq}. These sequences are defined right and left of the directed line passing through p and q, and in this direction. The approach we use to construct these sequences is based on several properties of antipodal pairs. An antipodal pair is two vertices of a convex polygon that allow parallel supporting lines to contain all points of the polygon between them. We now discuss some properties of antipodal pairs in the following lemma.

Lemma 1. *Let C_{pq} be a convex polygon with diametral pair p, q. Let $v_1, .., v_t$ be the right sequence of ccw vertices of C_{pq}, where $v_1 = p$ and $v_t = q$. Analogously, let $v_t, .., v_s$ be the left sequence of ccw vertices of C_{pq}, where $v_s = p$. Let v_i, v_j be an antipodal pair with $1 \leq i \leq t$ and $t \leq j \leq s$. We assume there are no parallel edges in C_{pq}. Then, we must have one of the following cases.*

1. *v_{i+1} is antipodal to v_j.*
2. *v_{j+1} is antipodal to v_i.*

Proof. See Figure 7. We rotate ccw the two parallel lines passing through v_i and v_j until one of the lines touches either v_{i+1} or v_{j+1}. Touching both vertices at the same time is not possible, as this would imply that two edges in C are parallel.

□

The lemma above implies that for any edge $\overline{v_i v_{i+1}}$, there is at most one point v_j such that v_i, v_j and v_{i+1}, v_j are antipodal pairs. The same applies to the edge $\overline{v_j v_{j+1}}$ and vertex v_i.

Let r, r' be two points right of the directed line passing through p and q, in this direction. Let ℓ be a point left of this same directed line. We allow r to be p, r' to be q and ℓ to be either q or p. We can use these three points plus an integer k to index the convex polygon C_{pq} with the following properties.

1. C_{pq} has ℓ as a vertex and $\overline{rr'}$ as a counterclockwise edge.
2. ℓ is antipodal to r and r'.
3. The distances $|\ell r|$ and $|\ell r'|$ are both less than or equal to $|pq|$.
4. C_{pq} has minimum area and contains exactly k points.

Analogously, we can define the same indexing for two left points ℓ, ℓ' and a right point r. These properties are symmetrical.

Based on Lemma 1, we also derive the incremental property of antipodal pairs. Let v_i, v_j and v_c, v_{j+1} be two antipodal pairs. Then, $c \geq i$. Using Figure 7, one can visually understand why this is the case. In other words, considering a new vertex after v_i cannot generate new antipodal pairs with vertices before v_j.

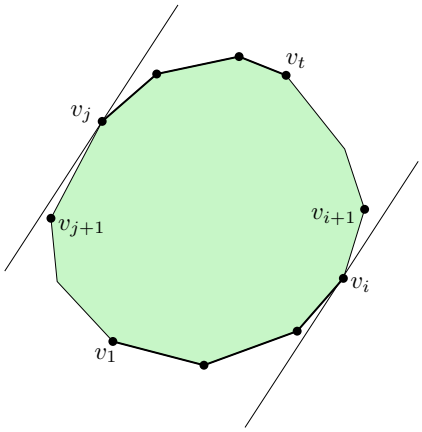


Figure 7: If v_i, v_j is an antipodal pair, then either v_i, v_{j+1} or v_{i+1}, v_j is an antipodal pair.

We exploit these properties to devise a dynamic programming algorithm finding the minimum area convex polygon C_{pq} with maximum cardinality for a given pair $p, q \in P$ in $O(kn^4)$ time and $O(kn^3)$ space. We use the rotating calipers approach to guarantee that all antipodal pairs define line segments of length bounded by $|pq|$.

Let $R[r, r', \ell, k]$ be the array used to index the minimum area convex polygon C_{pq} with counterclockwise edge rr', ℓ antipodal to r and r', containing k points. Analogously, let $L[\ell, \ell', r, k]$ be the array used to index the minimum area convex polygon C_{pq} with counterclockwise edge $\ell\ell'$, r antipodal to ℓ and ℓ', containing k points. We now describe how the algorithm works.

In the initialization phase, the entries $R[\cdot, \cdot, \cdot, \cdot]$ and $L[\cdot, \cdot, \cdot, \cdot]$ are set to ∞. Then, for any possible value of r', we try to compute the solutions $R[r = p, r', \ell = q, k]$. Analogously, for any possible value of ℓ', we try to compute the solutions $L[\ell = q, \ell', r = p, k]$.

The main idea is to use a recursive procedure to find the sequence of triangles generating a triangulation of the minimum area convex polygon C_{pq} having maximum cardinality. Figure 8b shows the final result. We only show how to perform this step from an entry stored in R since the approach is symmetrical for L.

At any step, we can expand C_{pq} in one of two ways:

1. Expand right: Attach a new triangle $\triangle r'r''q$ to the edge $r'q$, where r''

is the counterclockwise successor of r'.

2. Expand left: Attach a new triangle $\triangle \ell\ell' p$ to the edge $\ell\ell'$, where ℓ' is the counterclockwise successor of ℓ. The new entry in L is ℓ, ℓ', r'.

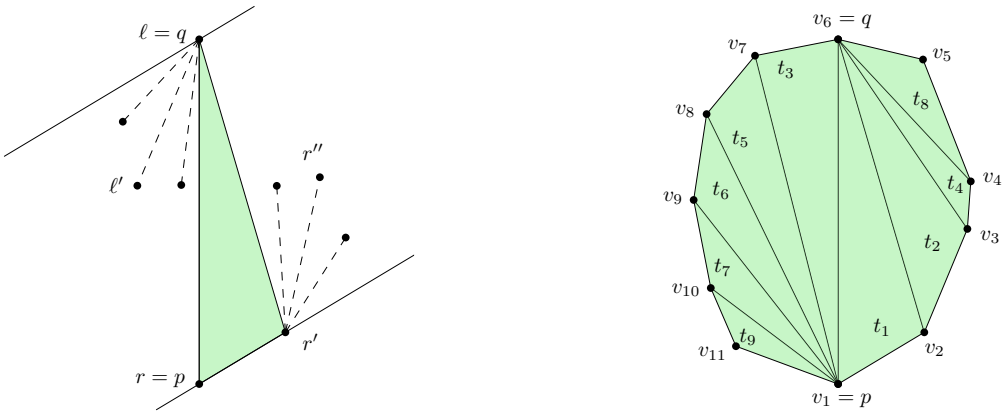
When we expand on the right side, we choose all r'' satisfying the following requirements.

1. $r''r'r$ makes a right turn.
2. $qr''r'$ makes a right turn.
3. r'' is antipodal to ℓ.
4. $|r''\ell| \leq |pq|$.

When we expand on the left side, we choose all ℓ' satisfying the following requirements.

1. ℓ' is on the right side of the line through ℓ that is parallel to the line through r and r'.
2. $p\ell'\ell$ makes a right turn.
3. ℓ' is antipodal to r'.
4. $|\ell' r'| \leq |pq|$.

The cases are shown in Figure 8a. These requirements are sufficient to guarantee that the shape obtained by attaching either $\triangle r'r''q$ or $\triangle \ell\ell' p$ to $R[r, r', \ell, m]$, is convex and maintains p, q as diametral pair. Based on the incremental property of antipodal pairs, a successor of r' or ℓ can only create new antipodal pairs with either ℓ, r' or one of their successors. Therefore, we only need to verify that the segment length of the new antipodal pair is bounded by $|pq|$. We summarize this recursive procedure for R with the following recurrence.



(a) (b)

Figure 8: (a) We can either expand on the left side with $\triangle \ell \ell' p$ or again on the right side with $\triangle r r' q$. (b) A valid solution C_{pq} seen as a decomposition into triangles.

$$R[r, r', \ell, m] = \begin{cases} 0 & \text{if } r' = q \wedge m = 2 \\ \infty & \text{if } r' = q \wedge m > 2 \\ \infty & \text{if } m < 2 \\ \infty & \text{if } \min\{\triangle_R^{\min}, \triangle_L^{\min}\} > A_{\max} \\ \min\{\triangle_R^{\min}, \triangle_L^{\min}\} & \text{otherwise} \end{cases} \tag{4}$$

$$\triangle_R^{\min} = A(r, r', q) + \min_{r''} R[r', r'', \ell, m - K(r, r', q) - 1]$$

$$\triangle_L^{\min} = A(r, r', q) + \min_{\ell'} L[\ell, \ell', r', m - K(r, r', q) - 1]$$

We use a memoization technique to avoid recomputing the entire recursive sub-tree whenever the same entry in R or L is reached. We do this by storing, for all recursive calls, the one that led to the valid solution with minimum area. We are able to use this approach thanks to the incremental property of antipodal pairs we discussed.

We now discuss an edge case our approach needs to consider. Suppose we compute a solution C_{pq} with all its vertices right of the directed line passing through p and q, and in this direction. Based on our approach, this implies that we never go from entries in R to entries in L, and therefore, ℓ would

always be equal to q. This does not allow to obtain C_{pq}, as we would reach a situation where any possible r'' is not antipodal with ℓ (i.e. p left of the directed line passing through $\ell = q$ and parallel to $r'r''$, and in this direction). We deal with this by allowing ℓ to be p. The same issue occurs when C_{pq} has all of its vertices on the left side. Analogously, we allow r to be q. These are the only situations in which we allow ℓ to be p and r to be q.

Our recurrence deals with three base cases that we now discuss.

1. The first happens when the only possible value for r'' is q and the current solution contains exactly $k-2$ points. In this case, we found a valid solution with k points. This is because the missing two points are p and q, which are in C_{pq} by definition.

2. The second happens when the only possible value for r'' is q and the current solution contains less than k points. In this case, the solution is not valid.

3. The third happens whenever $m < 0$. This condition is possible only if we limit our search to regions with $k < n$ points, where n is the number of input points. In this case, the solution is not valid as it exceeds the maximum number of points.

If all pairs of points $p, q \in P$ have distances $|pq|$ greater than $D_{\max}$, then no polygon can be formed. In this case, the solution is a degenerate region consisting of a single point with zero area.

Let us now examine the overall complexity of this approach. There are $O(n^2)$ diametral pairs $p, q \in P$ and for each one, we fill the tables L and R with $O(kn^3)$ entries. Each entry is computed in $O(n)$ time. Hence, we are able to solve Problem 1 in $O(kn^6)$ time and $O(kn^3)$ space.

The approach used to reconstruct the optimal convex polygon is essentially the same as the one described for the A algorithm, with the only difference that during the reconstruction, we occasionally switch from entries in R to entries in L and vice versa. During the computation of R and L, we keep track of the best next triangle from any valid entry. We use this information to select the next optimal triangle in $O(1)$ until the optimal shape is complete. Therefore, we can reconstruct the optimal solution in $O(k)$ time.

4.2 Implementation details

We now discuss several details of our implementation of the AD algorithm.

We do not use two arrays for L and R as that would always use the worst-case amount of memory. Instead, we use one array with k hash tables. We store the entry, r, r', ℓ, m in the m-th hash table, using the indices of the three points as key. The same applies for entries ℓ, ℓ', r, m. We use an iterative approach which, from a given entry with m points, generates all the valid entries with $m' > m$. Whenever an entry is already in the hash table, we keep the one with the minimum area. This way, we allocate memory only for the entries strictly needed for the computation. In addition, this approach allows processing only the entries generated from smaller values of m. Thus, we do not need to iterate through all the possible triples of points. In practice, this brings a noticeable speedup.

As for the A algorithm, we have implemented an early exit strategy. We process pairs p, q in decreasing order based on the number of points in the lune. If our current best solution contains more points than the next pair can provide, we can stop since no better solution is possible.

Suppose we have an entry defined as $R[r, r', \ell, \cdot]$. We want to expand this entry with ℓ', where ℓ' is a valid counterclockwise successor of ℓ on the left side of the oriented line that passes first through point p and then through point q. In practice, we use the following approach: we only consider the points ℓ' having perpendicular distance with the line $\overline{rr'}$ up to the perpendicular distance between ℓ and the line $\overline{rr'}$. Our implementation uses a perpendicular pseudo-distance function to determine which of two points has the smallest perpendicular distance from a line. The perpendicular pseudo-distance is simply the non-normalized version of the perpendicular distance, which lets us avoid the computationally expensive square root calculations.

We now consider how to count the points in any triangle when the input set is not in general position. When we attach the first triangle $\triangle pr'q$, we count $K(p, r', q)$ plus the points collinear to any edge of the triangle. For any other triangle $\triangle rr'q$ further in the sequence, we count $K(p, r', q)$ plus the points collinear to rr' and $r'q$. The approach is symmetrical for triangles with two left points.

5 Experimental setup

In this section, we describe the experimental setup used to evaluate the area selection (AS) algorithm, the Area-only (A) algorithm [8] and the new Area-Diameter (AD) algorithm. We perform two experiments: one running the A and AD algorithms on synthetically generated data for several values of $D_{\max}$; another one, running the A, AS and AD algorithms on real-world data. Our experiments focus on answering the following questions.

1. How does increasing the size of the point set affect the performance (runtime and memory usage) of A and AD algorithms?
2. How does increasing the standard deviation (spread) of the point set affect the performance (runtime and memory usage) of A and AD algorithms?
3. How does increasing the maximum allowed diameter $D_{\max}$ affect the performance of AD algorithm?
4. How does the number of enclosed points and diameter compare in the optimal solutions found by the A and AD algorithms?
5. What are the output features (area, diameter, and number of enclosed points) of the optimal solutions found by the three algorithms, including the AS algorithm [16], when applied to the real-world data of the application that motivated this research?

To answer these questions, we implemented the algorithms and generated (or selected) the data to be used. We provide more information about this process in the following subsections. Section 6 presents the experimental results and answers the questions. We provide our implementations, the data used and supporting scripts in our github repository [14].

5.1 Implementation

We implemented the A algorithm [8] and the AD algorithm in C++. We implemented all other necessary data structures and algorithms that are not already in the std ourselves. The implementations do not use parallelism and run in a single thread. The entire codebase is hosted on a github repository [14]. Our experiments have been run on a MacBook Pro using an Apple M1

Pro (3228 MHz) and 16GB (6400 MT/s LPDDR5) RAM. It runs macOS Sonoma 14.2.1, and we compiled it for 64 bits using clang 19.1.6 with the -O3 option. We used the library Google Benchmark to setup our experiments and perform data collection. The data is then post processed in Python with several scripts that we provide in our repository. Regarding the area selection method, we used the Python implementation made available by [16]. We use this implementation only to compare the output features, not to compare performance.

The early-exit strategy for the A and AD algorithms is enabled only when processing the real-world data.

5.2 Data

We use synthetic and real-world data for the experiments.

The synthetic data is made of three sub-datasets. The first one contains uniformly distributed point sets of increasing size $n = 100, 110, \ldots, 200$ inside a square of size 20×20 mm. The second one uses the same settings, except for $n = 100, 200, ..., 500$. The third one contains normally distributed point sets of increasing standard deviations raging from $\sigma = 0.5, 1.0, \ldots, 6.5$ mm inside a square of size 20×20 mm^2.

We run the A and AD algorithms with maximum area set to 4mm^2. This area is larger than what was reported in [16], but the authors have since confirmed in personal communication that they now use 4mm^2. The AD algorithm is run for $D_{\max} = 2, 3, 4, 5$, and 6 mm. For each setting, we generate 100 point sets and report the averages over 100 runs. The exception is the second sub-dataset, where we generate only 5 point sets and report averages from these 5 runs. We added this dataset to our experiments shortly before the project deadline, which limited our time to perform more repetitions.

The standard deviation is omitted whenever the observed variation is no more than 2% of the total value.

The real world data is the medical data used in [16]. The dataset is provided by the Department of Digital Pathology at UMC Utrecht and contains 1982 point sets representing the mitotic figures identified in Whole-Slide-Images (WSIs) from real patients. The detection and localization of the mitotic figures was carried out with the classification model presented in [18], which generates 2-dimensional point sets along with the detection accuracy for each point (figure). We apply a preprocessing step to filter out any point with detection accuracy below 0.86 and convert the coordinates from

pixels to millimeters. We consider the top 10 most populated point sets for our experiments. Table 1 provides an overview of this data. We include the Average Nearest Neighbour (ANN) to provide an indication of how clustered the points are.

We run the AS algorithm [16] using patches generated with step size $s = 1.98, 0.5$, the A algorithm applied to each generated patch with step size $s = 0.5$ and the AD algorithm (with $D_{\max} = 4$mm) without patching. Patches have a size of 3×3 mm, therefore the maximum diameter found by AS and A is at most $3\sqrt{2}$ mm. We allow solutions with area up to 4mm^2.

Both the synthetic generation and preprocessing of real-world data are fully automated in a Python script we provide in our github repository.

I	n	ANN	I	n	ANN
0	699	0.21 ± 0.14	5	474	0.30 ± 0.23
1	623	0.32 ± 0.22	6	412	0.35 ± 0.23
2	594	0.28 ± 0.19	7	382	0.34 ± 0.23
3	526	0.29 ± 0.21	8	377	0.33 ± 0.27
4	492	0.32 ± 0.25	9	362	0.48 ± 0.34

Table 1: The real-world point sets used in our experiments. We report the index (I), number of points (n), and Average Nearest Neighbor (ANN) features.

6 Experimental results

This section presents the experimental results obtained using synthetic and real-world data. We first discuss the performance and output features obtained with the A and AD algorithms on synthetic data. Then, we discuss the output features obtained with the A, AS and AD algorithms on real-world data. The referenced figures are located in Appendix B.

6.1 Performance

Figure 9a reports the runtime, memory and number of entries used by the A and AD algorithms for the uniformly distributed point sets with $n = 100, 110, ..., 200$. We observe that the AD algorithm has a better runtime than the A algorithm, even if its worst-case time complexity is $O(kn^6)$. This is made possible by the pruning of all pairs with length above $D_{\max}$ which practically outperforms the $O(kn^3)$ time complexity of the A algorithm. Therefore, an increase in the maximum diameter implies potentially more pairs to check and thus an higher runtime for AD. We are able to process up to 500 points in reasonable time. However, as we further increase n, we expect a diminishing advantage of AD over A, given the high density of points which largely cancels out the benefits of pruning. In terms of number of entries, we observe that the AD algorithm uses less entries for all tested values of $D_{\max}$.

Figure 10a reports the runtime, memory and number of entries used by the A and AD algorithms for the uniformly distributed point sets with $n = 100, 200, ..., 500$. Even for this dataset, we see that the AD algorithm is still faster than the A algorithm. In terms of memory and table entries used, we notice that the values for the A and AD algorithms get considerably closer as we increase n.

A different result is observed with the normally distributed point sets. Figure 11a reports the runtime, memory and number of entries used. For standard deviations $\sigma \leq 1.5$ the points are so dense and close that the AD algorithm is not able to prune many pairs and has to process most points, thus having a worse performance than the A algorithm. This is reflected in the memory usage (and number of entries used) as well.

As expected, using higher diameters for the AD algorithm implies the usage of more computational resources for both uniform and gaussian data.

We also consider a variation of the A algorithm using fewer entries and

based on the approach discussed in Section 3.2.3. This allows to store about a factor 4 fewer entries for uniformly distributed point sets but yields much worse memory savings—about a factor 0.3—for normally distributed point sets with low standard deviations.

6.2 Output features

Figures 9b, 10b and 11b report the features (number of enclosed points, area and diameter) of the solutions found by the A and AD algorithms for uniformly and normally distributed point sets.

As expected, we can clearly see that the A algorithm finds solutions with more points, followed by the AD algorithm with decreasing diameters. This is indeed the case for both uniform and gaussian data.

Based on the diameter and area of the solutions found by the A algorithm, we see that the shapes are typically much more elongated than those found by the AD algorithm, even for $n \geq 200$. This goes against our initial hypothesis, which supposed that applying the A algorithm to more dense areas would typically result in roundish regions.

We note that the values for area, diameter and number of enclosed points obtained with the uniform dataset with $n = 100, 200, ..., 500$ are similar but have a higher standard deviation due to the low number of repetitions.

As expected, increasing the standard deviation of the point sets has the effect of increasing the diameter of the solutions found by the A algorithm. This increase is initially observed also for the AD algorithm using $D_{\max} >$ 2mm.

6.3 Real-world data

We report the runtime, number of enclosed points, area and diameter for the algorithms applied to the ten real-world point sets in Table 2. As expected, the solutions found by the AD_4 algorithm are the ones enclosing most points, followed by the A algorithm and the AS algorithms sorted by increasing step size. This confirms that using a sufficiently smaller step size for the patching step matters and could increase the number of enclosed points in the solution.

We also see that the A algorithm is an effective replacement of the hybrid approach (brute-force and heuristic algorithms) used in the AS algorithm. This allows to consistently find optimal solutions within a patch while also establishing an upper bound on the maximum possible diameter.

The regions found for ten real-world point sets are shown in Figures 12, 13 and 14. For the majority of these point sets, the four algorithms typically roughly find the same region but rarely find the identical convex polygon. This is visible for the point set with index I = 7 in Figure 13.

I	**AD**				**AS** ($s = 1.98$)				**AS** ($s = 0.5$)				**A** ($s = 0.5$)			
	time	k	area	dia.	time	$\hat{k}$	area	dia.	time	$\hat{k}$	area	dia.	time	$\hat{k}$	area	dia.
0	494.19	**76**	3.99	3.15	5.24	68	3.42	2.89	72.83	70	3.74	2.88	13.7	74	3.9	3.07
1	14.7	**42**	3.83	3.99	17.15	36	3.84	2.74	274.82	39	3.97	2.9	2.84	40	3.94	2.87
2	29.38	**39**	3.93	2.72	19.89	33	3.99	2.85	218.36	**39**	3.98	2.66	3.66	**39**	3.93	2.72
3	24.43	**42**	3.94	3.1	19.12	34	3.99	2.71	263.07	39	3.97	3.15	3.17	40	3.88	3.1
4	8.83	**42**	3.88	2.99	13.83	38	3.87	2.82	229.33	40	3.99	2.83	2.08	**42**	3.98	2.88
5	21.76	**44**	3.88	3.7	9.16	37	3.82	3.31	141.13	39	3.76	3.17	4.51	39	3.82	3.01
6	2.08	**31**	3.98	3.85	9.32	27	3.72	2.73	161.07	28	4.0	3.45	1.63	28	3.94	3.38
7	6.19	**34**	3.89	3.22	13.66	30	3.94	2.65	174.71	32	3.92	3.22	1.82	**34**	3.92	3.22
8	37.18	**61**	3.91	3.94	9.66	57	3.86	2.67	77.41	59	3.97	2.72	2.58	60	3.92	3.01
9	0.19	**28**	3.86	3.49	4.83	23	3.97	3.41	52.02	27	3.97	2.74	1.47	27	3.97	2.74

Table 2: Comparison of the results obtained by applying the AD_4 algorithm, the AS algorithm and the A algorithm to all tiles of the point sets summarized in Table 1. AS is run for step sizes $s = 1.98, s = 0.5$. A is run for step size $s = 0.5$. The values for area and diameter are expressed in mm^2 and mm respectively. We report the runtime of the AD_4 algorithm expressed in seconds.

7 Conclusion and future work

We have presented the Area-Diameter (AD) algorithm, a new dynamic programming procedure for finding the convex region enclosing the maximum number of points k, with area bounded by $A_{\max}$ and diameter bounded by $D_{\max}$ in $O(kn^6)$ time and using $O(kn^4)$ space.

The AD algorithm has a worst case time complexity that is a cubic factor higher than the area (A) algorithm [8]. In practice, we have shown with our experiments that the AD algorithm is faster than the A algorithm in many circumstances thanks to the significant pruning guided by the maximum allowed diameter $D_{\max}$. We acknowledge that this advantage heavily depends on the maximum allowed diameter and the specific point set (i.e. size and distribution). We have seen that processing point sets with standard deviation $\sigma = 1$mm is about 35x faster than processing point sets with standard deviation $\sigma = 0.5$mm. The results obtained using the real-world data shows that our algorithm finds exact solutions for real-world instances of Problem 1 within a reasonable amount of time.

We have tested several variations of the Area Selection (AS) algorithm [16] on real-world data. Our results show that the A algorithm is an effective replacement of the hybrid approach (brute-force and heuristic) used in the AS algorithm. It finds solutions with the minimum possible area, diameter bounded by the diagonal of the patch and the optimum number of enclosed points with respect to the current patch. Nonetheless, the usage of patches brings several limitations, the most important of which is that by definition it is impossible to bound the diameter to a specific value and guarantee that the global optimum is found, even if the patches overlap. For this reason, we consider this approach a good compromise to find solutions in difficult point sets (i.e. very big and dense) which the AD would not be able to process within an acceptable timeframe.

Our results could be improved in several ways.

One could implement and evaluate the performance of the more recent algorithm from Eppstein [7] offering a runtime improvement over the A algorithm whenever $k = O(\sqrt{n})$, one could try to reduce by a factor n or more the complexity of the AD algorithm or perhaps devise ϵ-approximation algorithms for this problem.

We spent some time discussing about this last possibility in the early phases of our research but in the end, given the strict time schedule, we decided to exclude this task from our research. In general, we believe that an

ϵ-approximation algorithm for this problem is plausible. The output features of the approximation could then be used to speed up the search performed by the AD algorithm to find the optimal solution.

Finally, we see good potential in the usage of multi-threading in the A and AD algorithms, and it would be interesting to assess their performance.

Appendix A Selected algorithms

A.1 The number of points in all triangles

Algorithm 2: Preprocessing of the input point set in $O(n^2 \log n)$ time

Sort the points $p_1, p_2, ..., p_n \in P$ using $\prec$;
For any $pi \in P$, sort the points to the left of p_i in clockwise order around p_i. This gives the sequence $p_1^i, p_2^i, ..., p_{i-1}^i$;
$B[\cdot,\cdot], C[\cdot,\cdot], V[\cdot] \leftarrow 0$;
for $p_i = p_2$ *to* p_n **do**
  if p_{i-1} *is below* p_i **then**
    $V[p_i] = 1 + V[p_{i-1}]$;
  end
  for $j = 2$ *to* $i - 1$ **do**
    if p_i, p_j^i, p_{j-1}^i *are collinear* **then**
      $C[p_i, p_j^i] \leftarrow 1 + C[p_i, p_{j-1}^i]$;
      if p_j^i *lies left of* p_i **then**
        $B[p_i, p_j^i] \leftarrow B[p_i, p_{j-1}^i] + B[p_{j-1}^i, p_j^i] + V[p_{j-1}^i]$;
      end
    end
    else if p_j^i *lies right of* p_{j-1}^i *and* p_{j-1}^i *lies left of* p_i **then**
      $B[p_i, p_j^i] \leftarrow B[p_i, p_{j-1}^i] + C[p_i, p_{j-1}^i] - B[p_j^i, p_{j-1}^i] - V[p_j^i]$;
    end
    else if p_j^i *lies left of* p_{j-1}^i **then**
      $B[p_i, p_j^i] \leftarrow$
       $B[p_i, p_{j-1}^i] + C[p_i, p_{j-1}^i] + B[p_j^i, p_{j-1}^i] + C[p_j^i, p_{j-1}^i] + V[p_{j-1}^i]$;
      if p_{j-1}^i *lies left of* p_i **then**
        $B[p_i, p_j^i] \leftarrow 1 + B[p_i, p_j^i]$;
      end
    end
    else
      $B[p_i, p_j^i] \leftarrow B[p_{j-1}^i, p_i]$;
    end
  end
end

A.2 Area-only algorithm

Algorithm 3: Fill array T up to $m = k$ in $O(kn^3)$ time

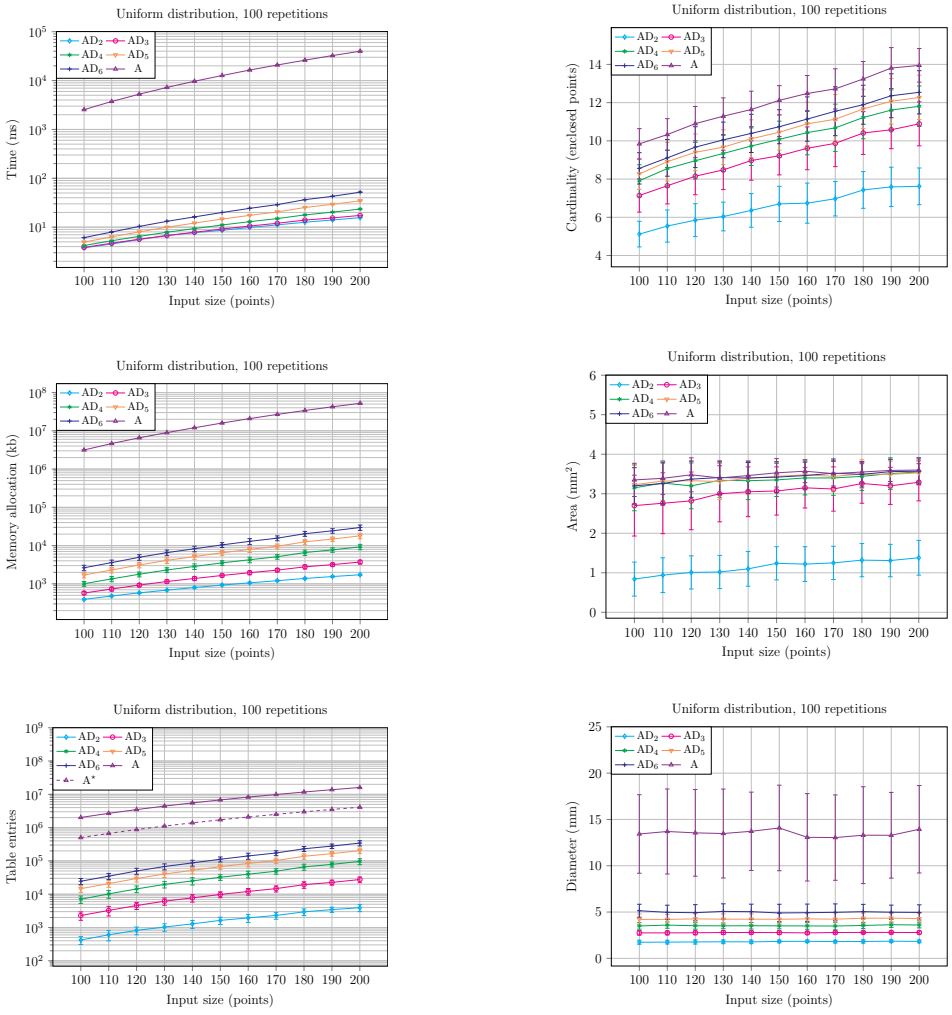
```
T[n, n, n, k] ← ∞;
TotalMinimum ← ∞;
for all points p_i ∈ P do
    T[p_i, ·, ·, 2] ← 0;
    for m ← 3 to k do
        for all points p_j above p_i, in clockwise order around p_i by angle do
            MinArea ← ∞;
            for all points p_l above p_i, in clockwise order around p_j by slope
              of the lines passing through p_j and p_l as explained in Section
              3.2.2 do
                if p_l is left of p_i p_j then
                    s ← K(p_i, p_j, p_l) + 1;
                    if s ≤ m then
                        area ← T[p_i, p_l, p_j, m − s] + A(p_i, p_j, p_l);
                        if area ≤ A_max then
                            MinArea ← min(MinArea, area);
                        end
                    end
                end
                T[p_i, p_j, p_l, m] ← MinArea;
            end
        end
    end
    TotalMinimum ← min(TotalMinimum, min T[p_i, ·, ·, k]);
end
```

Algorithm 4: Reconstruct optimal solution from T in $O(kn)$ time

Let p_i, p_j be the points indexing the optimal solution with k points in T;
Let $\mathcal{C}$ be the array storing the sequence of ccw vertices defining the optimal solution witj k points;
$\mathcal{C} \leftarrow \emptyset$;
Append p_i to $\mathcal{C}$;
while $k > 2$ **do**
    $p_{\text{prev}} \leftarrow$ get the last point inserted in $\mathcal{C}$;
    $p_l \leftarrow$ get the point q such that $A[p_i, p_j, q, k]$ is the minimum. q is left of the directed line passing through p_{prev} and p_j, and in this direction;
    $k \leftarrow k - K(p_i, p_j, p_l) - 1$;
    Append p_j to $\mathcal{C}$;
    $p_j \leftarrow p_l$;
end
Append p_j to $\mathcal{C}$;

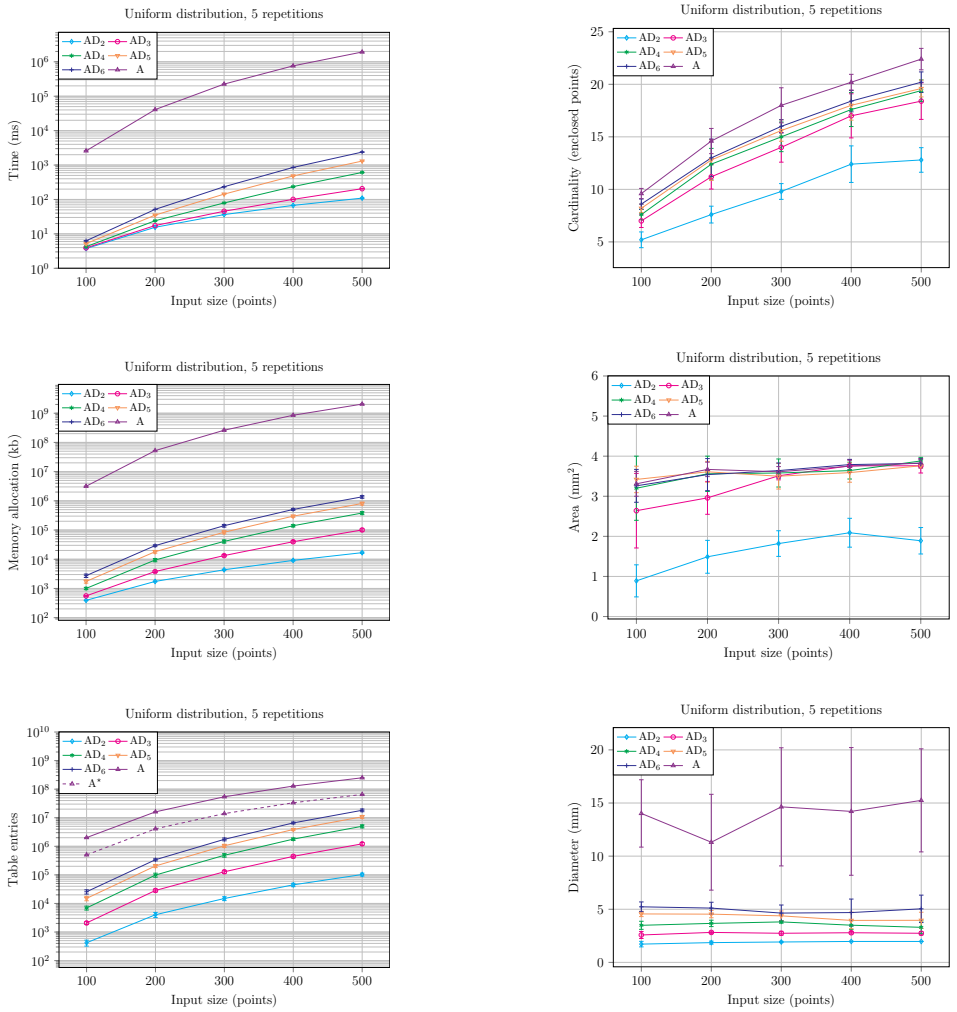
Appendix B Experimental results

B.1 Uniformly distributed data



(a) (b)

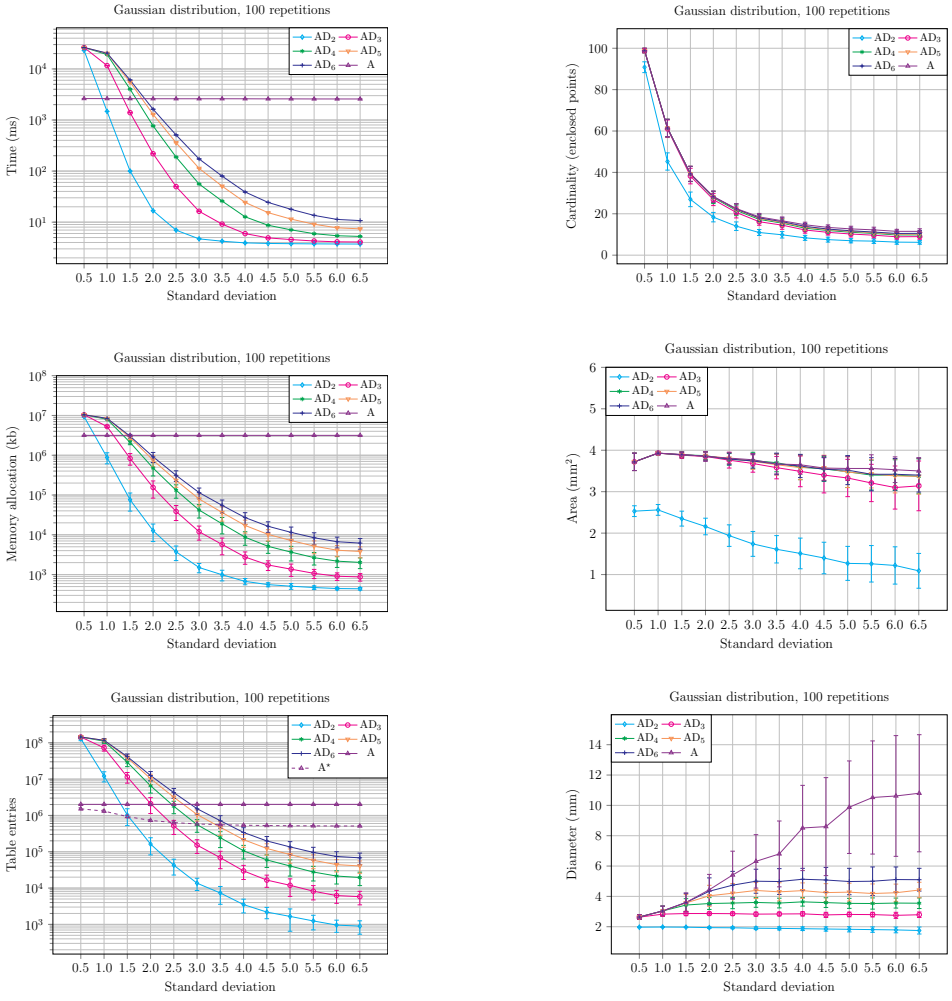
Figure 9: (a) Average runtime, memory and number of entries of the AD and A algorithms over 100 uniformly distributed point sets of sizes 100 to 200. (b) Average number of enclosed points, area and diameter of solutions generated by the A and AD algorithms over 100 uniformly distributed point sets of sizes 100 to 200.



(a) (b)

Figure 10: (a) Average runtime, memory and number of entries of the AD and A algorithms over 5 uniformly distributed point sets of sizes 100 to 500. (b) Average number of enclosed points, area and diameter of solutions generated by the A and AD algorithms over 5 uniformly distributed point sets of sizes 100 to 500.

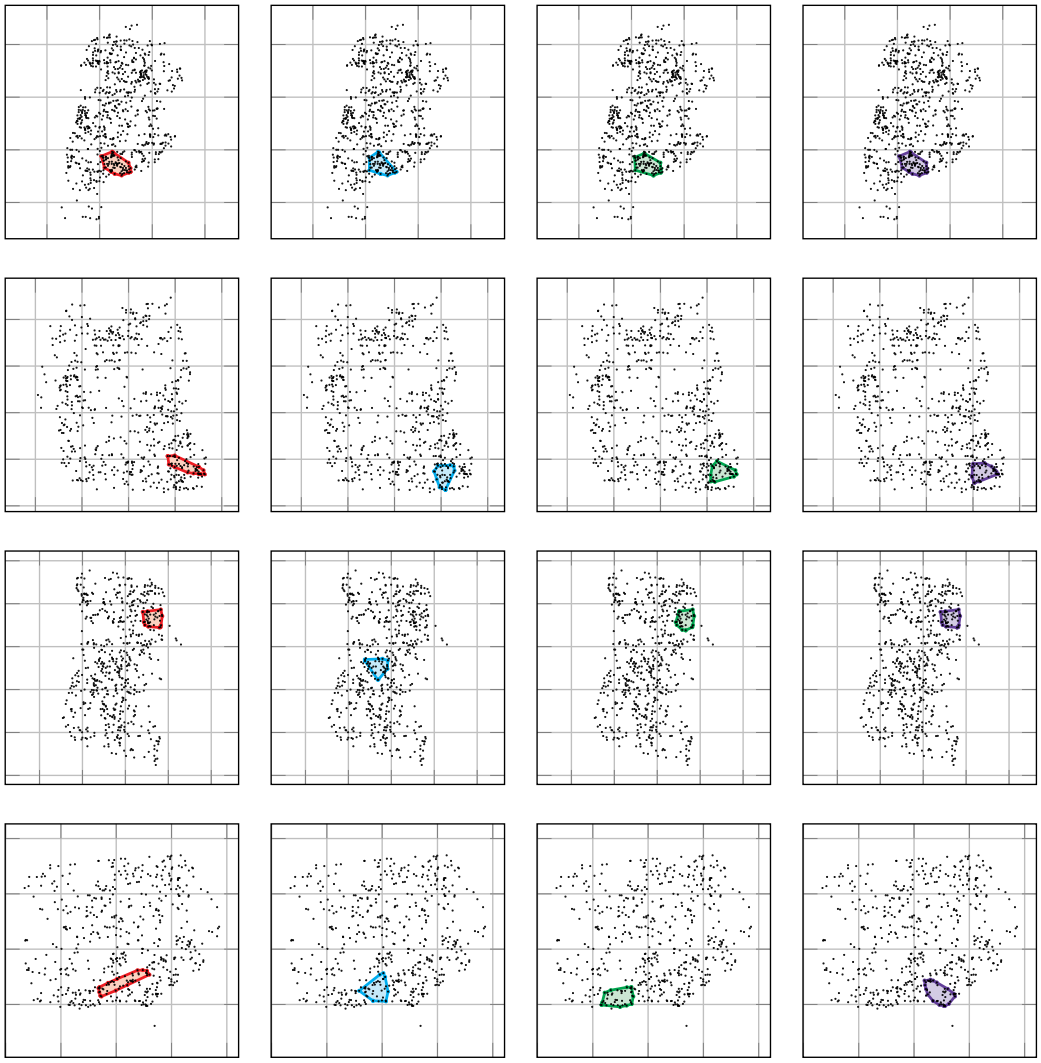
B.2 Normally distributed data



(a) (b)

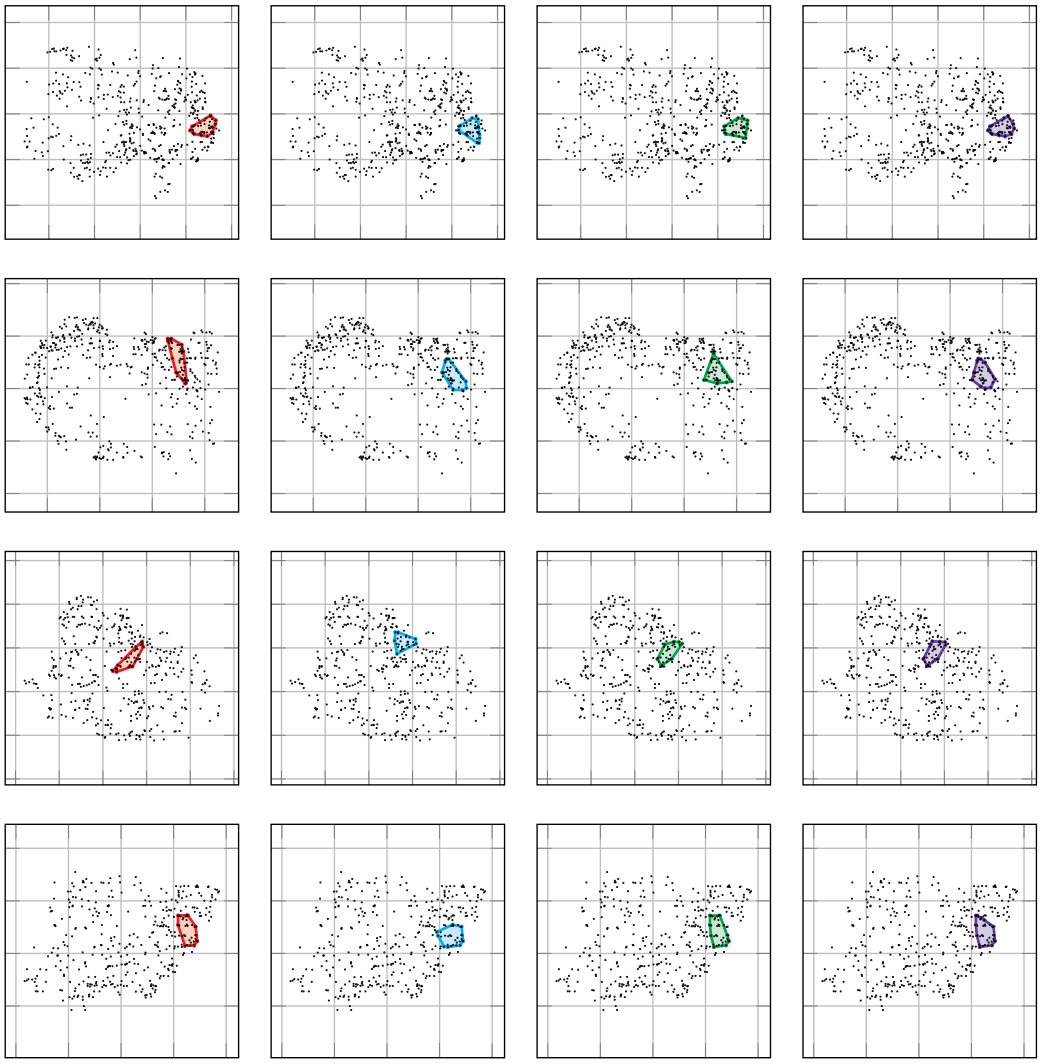
Figure 11: (a) Average runtime, memory and number of entries of the AD and A algorithms over 100 normally distributed point sets with standard deviation varying from $\sigma = 0.5$ to $\sigma = 6.5$. (b) Average cardinality, area and diameter of solutions generated by the A and AD algorithms over 100 normally distributed point sets with standard deviation varying from $\sigma = 0.5$ to $\sigma = 6.5$.

B.3 Real-world data



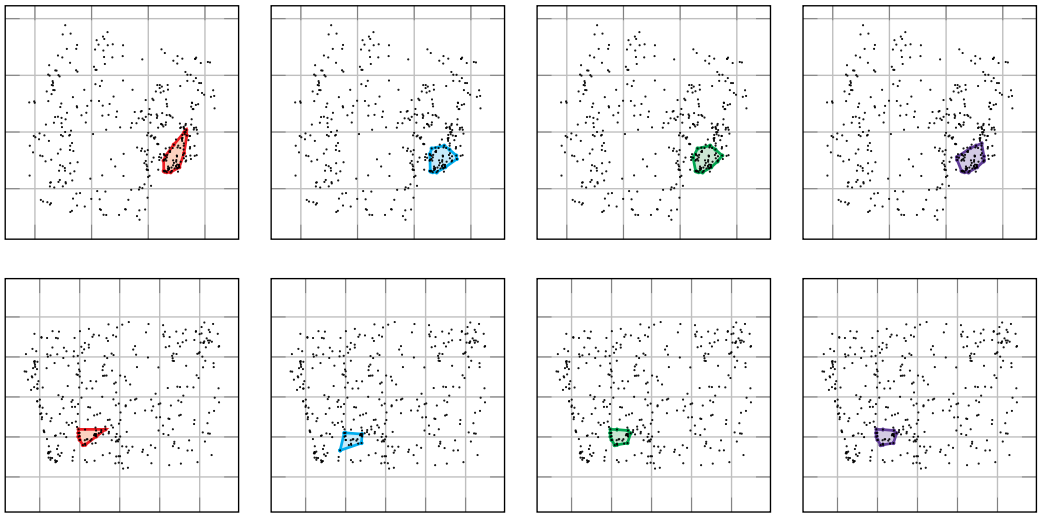
(a) (b) (c) (d)

Figure 12: (a) Mitotic hotspots found by the AD_4 algorithm. (b) Mitotic hotspots found by the AS algorithm using $s = 1.98$. (c) Mitotic hotspots found by the AS algorithm using $s = 0.5$. (d) Mitotic hotspots found by the A algorithm using $s = 0.5$. We report the results for the real-world point sets with index $\mathrm{I} = 0, 1, 2, 3$. The patch size is 3×3 mm. The line spacing is set to 5 mm.



(a) (b) (c) (d)

Figure 13: (a) Mitotic hotspots found by the AD_4 algorithm. (b) Mitotic hotspots found by the AS algorithm using $s = 1.98$. (c) Mitotic hotspots found by the AS algorithm using $s = 0.5$. (d) Mitotic hotspots found by the A algorithm using $s = 0.5$. We report the results for the real-world point sets with index $\mathrm{I} = 4, 5, 6, 7$. The patch size is 3×3 mm. The line spacing is set to 5 mm.



(a) (b) (c) (d)

Figure 14: (a) Mitotic hotspots found by the AD_4 algorithm. (b) Mitotic hotspots found by the AS algorithm using $s = 1.98$. (c) Mitotic hotspots found by the AS algorithm using $s = 0.5$. (d) Mitotic hotspots found by the A algorithm using $s = 0.5$. We report the results for the real-world point sets with index $\mathrm{I} = 8, 9$. The patch size is 3×3 mm. The line spacing is set to 5 mm.

Bibliography

[1] A. Aggarwal, H. Imai, N. Katoh, and S. Suri. Finding k points with minimum diameter and related problems. In: *Journal of Algorithms* 12.1 (1991), pp. 38–56.

[2] E. M. Arkin, S. Khuller, and J. S. Mitchell. Geometric knapsack problems. In: *Algorithmica* 10.5 (Nov. 1993), pp. 399–427. issn: 0178-4617.

[3] M. Aubreville, C. A. Bertram, N. Stathonikos, M. Veta, T. Donovan, N. ter Hoeve, F. Ciompi, C. Marzahl, F. Wilm, K. Breininger, A. Maier, and R. Klopfleisch. Mitosis domain generalization challenge (miccai midog 2021) training data. Version 1.0. Zenodo, Mar. 2021.

[4] M. Bonert and A. J. Tate. Mitotic counts in breast cancer should be standardized with a uniform sample area. In: *BioMedical Engineering OnLine* 16 (2017).

[5] J. M. Chang, A. E. McCullough, A. C. Dueck, H. E. Kosiorek, I. T. Ocal, T. K. Lidner, R. J. Gray, N. Wasif, D. W. Northfelt, K. S. Anderson, and B. A. Pockaj. Back to basics: traditional nottingham grade mitotic counts alone are significant in predicting survival in invasive breast carcinoma. In: *Annals of Surgical Oncology* 22 (2015), pp. 509–515.

[6] H. Edelsbrunner. Algorithms in combinatorial geometry. Berlin, Heidelberg: Springer-Verlag, 1987. isbn: 038713722X.

[7] D. Eppstein. New algorithms for minimum area k-gons. In: *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '92. Orlando, Florida, USA: Society for Industrial and Applied Mathematics, 1992, pp. 83–88. isbn: 089791466X.

[8] D. Eppstein, M. Overmars, G. Rote, and G. Woeginger. Finding minimum area k-gons. In: *Discrete & Computational Geometry* 7.1 (Jan. 1992), pp. 45–58. issn: 1432-0444.

[9] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD'96. Portland, Oregon: AAAI Press, 1996, pp. 226–231.

[10] P. Fischer. Sequential and parallel algorithms for finding a maximum convex polygon. In: *Comput. Geom. Theory Appl.* 7.3 (Feb. 1997), pp. 187–200. issn: 0925-7721.

[11] P. L. Fitzgibbons, S. Bose, Y.-Y. Chen, J. L. Connolly, M. E. de Baca, M. Edgerton, D. F. Hayes, K. A. Hill, C. Kleer, S. C. Lester, F. P. O'Malley, D. L. Page, J. F. Simpson, R. Simpson, B. L. Smith, L. K. Tan, D. L. Weaver, and E. Winer. Protocol for the examination of specimens from patients with invasive carcinoma of the breast. CAP Protocol, College of American Pathologists, 2018.

[12] V. Keikha. On optimal w-gons in convex polygons. In: *CoRR* abs/2103.01660 (2021). arXiv: 2103.01660.

[13] J. S. Mitchell, G. Rote, G. Sundaram, and G. Woeginger. Counting convex polygons in planar point sets. In: *Information Processing Letters* 56.1 (1995), pp. 45–49. issn: 0020-0190.

[14] G. Picarella. Finding dense and well-shaped convex clusters in 2d point sets. url: https://github.com/gianmarcopicarella/master-thesis.

[15] E. A. Rakha, M. E. El-Sayed, A. H. Lee, C. W. Elston, M. J. Grainge, Z. Hodi, R. W. Blamey, and I. O. Ellis. Prognostic significance of nottingham histologic grade in invasive breast carcinoma. In: *Journal of Clinical Oncology* 26.19 (2008). PMID: 18490649, pp. 3153–3158.

[16] N. Stathonikos, M. Aubreville, S. de Vries, F. Wilm, C. Bertram, M. Veta, and P. van Diest. Breast cancer survival prediction using an automated mitosis detection pipeline. In: *Journal of Pathology: Clinical Research* 10.6 (Nov. 2024). The Journal of Pathology: Clinical Research published by The Pathological Society of Great Britain and Ireland and John Wiley & Sons Ltd. issn: 2056-4538.

[17] M. Veta, P. van Diest, M. Jiwa, S. Al-Janabi, and J. Pluim. Mitosis counting in breast cancer: object-level interobserver agreement and comparison to an automatic method. In: *PLoS ONE* 11.8 (Aug. 2016), pp. 1–13. issn: 1932-6203.

[18] F. Wilm, C. Marzahl, K. Breininger, and M. Aubreville. Domain adversarial retinanet as a reference algorithm for the mitosis domain generalization challenge. In: *Biomedical Image Registration, Domain Generalisation and Out-of-Distribution Analysis*. Ed. by M. Aubreville, D. Zimmerer, and M. Heinrich. Cham: Springer International Publishing, 2022, pp. 5–13. isbn: 978-3-030-97281-3.

[19] Y. Xu, M. Gong, Y. Wang, Y. Yang, S. Liu, and Q. Zeng. Global trends and forecasts of breast cancer incidence and deaths. In: *Scientific Data* 10.1 (May 2023), p. 334. issn: 2052-4463.