# Robust Classification of Dynamic Bichromatic Point Sets in $\mathbb{R}^2$

**Erwin Glazenburg**
Department of Information and Computing Sciences, Utrecht University, The Netherlands

**Marc van Kreveld**
Department of Information and Computing Sciences, Utrecht University, The Netherlands

**Frank Staals**
Department of Information and Computing Sciences, Utrecht University, The Netherlands

## Abstract

Let $R \cup B$ be a set of $n$ points in $\mathbb{R}^2$, and let $k \in 1..n$. Our goal is to compute a line that "best" separates the "red" points $R$ from the "blue" points $B$ with at most $k$ outliers. We present an efficient semi-online dynamic data structure that can maintain whether such a separator exists ("semi-online" meaning that when a point is inserted, we know when it will be deleted). Furthermore, we present efficient exact and approximation algorithms that compute a linear separator that is guaranteed to misclassify at most $k$, points and minimizes the distance to the farthest outlier. Our exact algorithm runs in $O(nk + n \log n)$ time, and our $(1 + \varepsilon)$-approximation algorithm runs in $O(\varepsilon^{-1/2}((n + k^2) \log n))$ time. Based on our $(1 + \varepsilon)$-approximation algorithm we then also obtain a semi-online data structure to maintain such a separator efficiently.

**2012 ACM Subject Classification** Theory of computation → Computational geometry

**Keywords and phrases** classification, duality, data structures, dynamic, linear programming

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2024.34

**Related Version** *Full Version*: `https://arxiv.org/abs/2406.19161` [13]

**Funding** *Erwin Glazenburg*: Supported by the Dutch Research Council (NWO) under project OCENW.M20.135.

## 1 Introduction

Classification is a well known and well studied problem: given a training set of $n$ data items with known classes, decide which class to assign to a new query item. Support Vector Machines (SVMs) [8] are a popular method for binary classification in which there are just two classes: *red* and *blue*. An SVM maps the input data items to points in $\mathbb{R}^d$, and constructs a hyperplane $s$ that separates the red points $R$ from the blue points $B$ "as well as possible". Intuitively, it tries to minimize the distance from $s$ to the set $X(s, B \cup R) \subseteq R \cup B$ of points *misclassified* by $s$ while maximizing the distance to the closest correctly classified points. A red point $r \in R$ is misclassified if it lies strictly inside the halfspace $s^+$ above (left of) $s$, whereas a blue point $b \in B$ is misclassified if it lies strictly inside the halfspace $s^-$ below $s$. See Figure 1 for an illustration. An SVM is typically modeled as a convex quadratic programming problem with linear constraints. However, this cannot provide guarantees on the number of misclassifications nor on the running time.[1] In practice, solving such optimization problems is possible, but it is computationally expensive as it involves $n + d$ variables [17]. This problem is magnified as training a high-quality classifier typically requires computing many

[1] When we restrict the coefficients in the SVM formulation to be rational numbers with bounded bit complexity such a problem can be solved in polynomial time [14, 23, 18]. However, it is unclear if they can be extended to allow for arbitrary real valued costs.


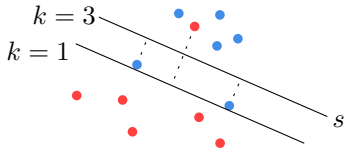


**Figure 1** Red and blue points, and two optimal separators for $M_{\max}$ with 1 and 3 misclassification.

classifiers, each trained on a large subset of the input data, during cross-validation. Similarly, in streaming settings, the labeled input points arrive on the fly, and old data points should be removed due to concept drift [25]. Each such update requires recomputing the classifier. Hence, this limits the applicability of SVMs in these settings, even when the input data is low-dimensional.

**The goal.** We aim to tackle both these problems. That is, we wish to develop an "SVM-like" linear classifier that can provide guarantees on the number of misclassified points $k$, and can be constructed and updated efficiently. As the problem of minimizing $k$ is NP-complete in general [1], we restrict our attention to the setting where the input points are low-dimensional. As it turns out, even for points in the plane, this is a challenging problem.

For a separator $s$, let $M_{\text{mis}}(s) = |X(s, B \cup R)|$ be the number of points misclassified by $s$. Let $S_k(B \cup R) = \{s \mid M_{\text{mis}}(s) \leq k\}$ denote the set of hyperplanes that misclassify at most $k$ points from $B \cup R$, and let $dist(p, q)$ denote the Euclidean distance between geometric objects $p$ and $q$. When the point sets are linearly separable, we want to compute a maximum-margin separator $s_{strip} \in S_0(R \cup B)$ that correctly classifies all points and maximizes the distance $M_{strip}(s_{strip}) = \min_{p \in R \cup B} dist(s_{strip}, p)$ to the closest points, exactly as in an SVM. Moreover, we would like to efficiently maintain such a separator when we insert or delete a point from $B \cup R$. The main challenge occurs when the point sets are not linearly separable. In this case, given a maximum $k$ on the number of misclassified points, our aim is to find a separator $s_{\text{opt}} \in S_k(R \cup B)$ that minimizes the (Euclidean) distance $M_{\max}(s) = \max_{p \in X(s, B \cup R)} dist(p, s)$ to the furthest misclassified point. This thus asks for a minimum width strip containing the $k$ outliers. We again would like to maintain such a separator when points are inserted or deleted. Furthermore, we may want to compute the smallest number $k_{\min}$ for which there exists a separator $s_{\min}$ that misclassifies at most $k_{\min}$ points. Note that decreasing the number of outliers may increase the value of $M_{\max}$, i.e. when $k_{\min} < k$ we may have $M_{\max}(s_{\min}) > M_{\max}(s_{\text{opt}})$, see Figure 1.

By the above discussion we distinguish four general variations of the problem:
**MaxStrip:** find a separator $s_{strip} = \text{argmax}_{s \in S_0(R \cup B)} M_{strip}(s)$
**MinMax:** find a separator $s_{\max} = \text{argmin}_s M_{\max}(s)$
**MinMis:** find a separator $s_{\text{mis}} = \text{argmin}_s M_{\text{mis}}(s)$
***k*-mis MinMax:** given a value $k$, find a separator $s_{\text{opt}} = \text{argmin}_{s \in S_k(B \cup R)} M_{\max}(s)$

**Related Work.** It is well known that for points in $\mathbb{R}^d$, for constant $d$, we can test if $R$ and $B$ can be linearly separated in $O(n)$ time by linear programming (LP) [22]. The problem becomes much more challenging when we allow a limited number of misclassifications. Everett et al. [12] show that for point sets $R$ and $B$ in the plane, one can find a line that separates $R$ and $B$ while allowing for at most $k$ misclassifications in $O(n \log n + nk)$ time. Matoušek [20] shows how to solve LP-type problems while allowing at most $k$ violated constraints. In particular, for linear programming in $\mathbb{R}^2$, his algorithm runs in $O(n \log n + k^3 \log^2 n)$ time. Chan [3] improves this to $O((n + k^2) \log n)$ time, and can compute the smallest number

$k$ for which the points can be separated (the MinMis problem) in the same time. Aronov et al. [2] consider computing optimal separators with respect to other error measures as well. In particular, they consider minimizing the distance $M_{\max}(s)$ from $s$ to the furthest misclassified point, as well as minimizing the average (squared) distance to a misclassified point $M^{\beta}_{\text{avg}}(s) = \sum_{p \in X(s, B \cup R)} (dist(s,p))^{\beta}$. For $n$ points in $\mathbb{R}^2$, their running times for computing an optimal separator vary from $O(n \log n)$ for the $M_{\max}$ measure (the MinMax problem), to $O(n^{4/3})$ for the $M^1_{\text{avg}}$ measure, to $O(n^2)$ for $M_{\text{mis}}$ (the MinMis problem) and the $M^2_{\text{avg}}$ measures. Some of their results extend to points in higher dimensions. Har-Peled and Koltun [16] consider similar measures, and present both exact and approximation algorithms. For example, they present an exact $O(nk^{d+1} \log n)$ time algorithm to find a hyperplane that minimizes the number of outliers (for points in $\mathbb{R}^d$), and an $O(n(\varepsilon^{-2} \log n)^{d+1})$ time algorithm to compute a $(1+\varepsilon)$-approximation of that number.[2] Their exact and approximation algorithms for computing a hyperplane minimizing $M_{\max}$ run in $O(n^d)$ and $O(n\varepsilon^{(d-1)/2})$ time, respectively. Matheny and Phillips [19] consider computing a separating hyperplane $s$, so that the discrepancy (the fraction of red points in $s^-$ minus the fraction of blue points in $s^-$) is maximized. They present an $O(n + \varepsilon^{-d} \log^4(\varepsilon^{-1}))$ time algorithm that makes an additive error of at most $\varepsilon$ (and thus misclassifies at most $\varepsilon n$ points more than an optimal (with respect to discrepancy) classifier).

**Results.** We can show that for points in $\mathbb{R}^1$ we can achieve both our goals: minimizing $M_{\max}$ with a hard guarantee on the number of outliers *and* efficiently supporting updates. In particular, in the full version [13] we present an optimal linear space solution:

▶ **Theorem 1.** *Let $B \cup R$ be a set of $n$ points in $\mathbb{R}^1$. There is an $O(n)$ space data structure that, given a query value $k \in 1..n$ can compute an optimal separator $s_{\text{opt}} \in S_k(R \cup B)$ with respect to $M_{\max}$ in $O(\log n)$ time, and supports inserting or deleting a point in $O(\log n)$ time.*

The main focus of our paper is to establish whether we can achieve similar results for points in $\mathbb{R}^2$. If the points are separable, we can maintain a maximum-margin separator – essentially a maximum width strip – in $O(\log^2 n)$ time per update.

The problem gets significantly more complicated when the point sets are not separable, and we thus wish to compute, and maintain, a separator $s_{\text{opt}} \in S_k(B \cup R)$ minimizing the distance $M_{\max}$ to the farthest misclassified point. We can test whether a separator $s \in S_k(B \cup R)$ exists (and find the smallest $k$ for which a separator exists) using LP with violations. In Section 3 we show how to dynamize Chan's approach [3] to maintain such a separator when the set of points changes. In particular, given a static linear objective function $f : \mathbb{R}^2 \to \mathbb{R}$ and a dynamic set $H$ of halfplanes that is given in a semi-online manner (at the time we insert a halfplane $h$ into $H$ we are told when we will delete $h$), we show how to efficiently maintain a point $p$ minimizing $f$ that lies outside at most $k$ halfplanes from $H$:

▶ **Theorem 2.** *Let $H$ be a set of $n$ halfplanes in $\mathbb{R}^2$, let $f$ be a linear objective function, and let $k \in 1..n$. There is an $O(n + k^2 \log^2 n)$ space data structure that maintains a point $p$ that violates at most $k$ constraints of $H$ (if it exists) and minimizes $f$, and supports semi-online updates in expected amortized $O(k \log^3 n)$ time.*

This then also allows us to maintain whether a separator that misclassifies at most $k$ points exists in amortized $O(k \log^3 n)$ time per (semi-online) update, as well as maintain the minimum value $k$ for which this is the case. Since linear programming queries have

[2] Here and throughout the rest of the paper, $\varepsilon > 0$ is an arbitrarily small constant.

many other applications, e.g. finding extremal points and tangents, we believe this result to be of independent interest. For example, given a threshold $\delta$, our data structure also allows us to maintain a line $\ell$ that minimizes the number of points $k$ at vertical distance exceeding $\delta$ from $\ell$ in amortized $O(k \log^3 n)$ time per update. Note that the best update time we can reasonably expect with this approach is $O((1 + k^2/n) \log n)$. For values of $k$ that are small (e.g. polylogarithmic) or very large (near linear) our approach is relatively close to this bound.

In Section 4, we incorporate finding the best separator from $S_k(B \cup R)$; i.e. a separator that minimizes $M_{\max}$. We first tackle the algorithmic problem of computing such an optimal separator. Our main result here is:

▶ **Theorem 3.** *Let $B \cup R$ be a set of $n$ points in $\mathbb{R}^2$, and let $k \in 1..n$. We can compute a separator $s_{\text{opt}} \in S_k(B \cup R)$ minimizing $M_{\max}$ in*

- *$O(nk + n \log n)$ time,*
- *$O((n + |S_k(B \cup R)| + k^3) \log^2 n)$ time, or*
- *when $k = k_{\min}$ in $O(k^{4/3} n^{2/3} \log n + (n + k^2) \log n)$ time.*

Here $|S_k(B \cup R)|$ denotes the complexity of (the region in the dual plane representing) $S_k(B \cup R)$. The key challenge is that this region $S_k(B \cup R)$ may consist of $\Theta(k^2)$ connected components, and each one has very little structure. While in the linear programming approach we can efficiently find one local minimum per connected component, that is no longer the case here. Instead, we explicitly construct the boundary of this region. Unfortunately, the total complexity of $S_k(B \cup R)$ is rather large: Chan [4] gives an upper bound of $|S_k(B \cup R)| = O(nk^{1/3} + n^{5/6-\varepsilon} k^{2/3+2\varepsilon} + k^2)$. We give two different algorithms to construct $S_k(B \cup R)$, and then efficiently find an optimal separator. When we restrict to the case where $k = k_{\min}$, i.e. finding an separator that minimizes $M_{\max}$ among all lines that misclassify the least possible number of outliers, each connected component of $S_k(B \cup R)$ is a single face in an arrangement of lines. This then gives us a slightly faster $O(k^{4/3} n^{2/3} \log^{2/3}(n/k) + (n + k^2) \log n)$ time algorithm as well.

Unfortunately, even when $k = k_{\min}$, dynamization appears very challenging. In the full version we present an $O((k^{4/3} n^{2/3} + n) \log^5 n)$ space data structure that supports insertions in amortized $O(kn^{3/4+\varepsilon})$ time, provided that the convex hulls of $R$ and $B$ remain the same. While the applicability of this result is limited, we use and develop an interesting combination of techniques here. For example, we develop a near linear space data structure that stores the the lower envelope of surfaces and allows for sub-linear time vertical ray shooting queries.

In Section 5, we slightly relax our goal and consider approximating the distance $M_{\max}$ instead. Our key idea is to replace the Euclidean distance by a convex distance function. This avoids some algebraic issues, as the distance between a point and a line now no longer has a quadratic dependency on the slope of the line. Instead the dependency becomes linear. We now obtain a much more efficient algorithm for finding a good separator:

▶ **Theorem 4.** *Let $B \cup R$ be a set of $n$ points in $\mathbb{R}^2$, let $k \in 1..n$, and let $\varepsilon > 0$. We can compute a separator $s \in S_k(B \cup R)$ that is a $(1 + \varepsilon)$ approximation with respect to $M_{\max}$ in $O(\varepsilon^{-1/2}((n + k^2) \log n))$ time.*

We essentially "guess" the width $\delta$ of a strip "separating" the point sets, and show that we can use the linear programming machinery to efficiently test whether there exists such a strip containing at most $k$ outliers . This involves extending the algorithm to deal with both "soft constraints" that may be violated, as well as "hard constraints" that cannot be violated, and using parametric search [21] to find the smallest $\delta$ for which such a strip exists.

▶ **Theorem 5.** *Let $B \cup R$ be a set of $n$ points in $\mathbb{R}^2$, let $k \in 1..n$, and let $\varepsilon > 0$. There is an $O(\varepsilon^{-1/2}(k^2 \log^2 n + n))$ space data structure that maintains a separator $s \in S_k(B \cup R)$ that is a $(1+\varepsilon)$-approximation with respect to $M_{\max}$, and supports semi-online updates in expected amortized $O(\varepsilon^{-1/2} k \log^4 n)$ time.*

**Applications.** Our data structure from Theorem 5 can reduce the total time in a leave-out-one cross validation process by roughly a linear factor in comparison to Theorem 4. For $m$-fold cross validation we gain a factor $m$. Similarly, in a streaming setting in which we maintain a window of width $w$, we gain a factor of roughly $w$. Note that in both these settings, the semi-online updates indeed suffice.

## 2 Preliminaries

**General definitions.** We use the standard point-line duality that maps any point $p = (p_x, p_y)$ in the primal plane to a line $p^* : y = p_x x - p_y$ in the dual plane, and any line $\ell : y = mx + c$ in the primal plane into a point $(m, -c)$ in the dual plane.

Let $A$ be a set of $n$ lines, and let $k \in 1..n$. Let the *lower* $\leq k$*-level* $L_{\leq k}(A) \subset \mathbb{R}^2$ of $A$ be the set of points for which there are at most $k$ lines below it. Similarly let the upper $\leq k$-level $L'_{\leq k}(A)$ be set of points for which there are at most $k$ lines above it. Let $L_k(A)$ be the *$k$-level*, the boundary of $L_{\leq k}(A)$. Note that a $k$-level lies exactly on existing lines in $A$. Although these terms refer to a region in the plane, with a slight abuse of notation we will also use them to refer to the part of the arrangement $\mathcal{A}$ of the lines in $A$ that lies in this region. The complexity of ($\mathcal{A}$ restricted to) $L_{\leq k}(A)$ is $O(nk)$, and it can be computed in $O(nk + n \log n)$ time [12]. Note that the lower 0-level $L_0(A)$ and the upper 0-level $L'_0(A)$ denote the *lower envelope* and the *upper envelope* of the set of lines, respectively.

In $O(n \log k)$ time we can compute a *concave chain decomposition* [3, 6] of $L_{\leq k}(A)$: a set of $O(k)$ concave chains of total complexity $O(n)$ that together cover all edges of $\mathcal{A}$ in $L_{\leq k}(A)$. See Figure 2a. A convex chain decomposition is defined similarly for $L'_{\leq k}(A)$.

Throughout this paper we assume points above a separating line $s$ should be blue, and points below should be red. In the dual this means that lines above separating point $s^*$ should be red, and lines below should be blue. In particular, we describe algorithms for finding the optimal separator that classifies in this way. We can then repeat the algorithm to find the best separator that classifies the other way around, and finally output the best of the two. For ease of description we assume all points in $R \cup B$ are in general position, meaning that all coordinates are unique, and no three points lie on a line.

**Valid separators.** Fix a value $k \in 1..n$. A separator $s$ and its dual $s^*$ are *valid* with respect to $k$ if (and only if) $s \in S_k(B \cup R)$. Line $s$ misclassifies all red points above $s$ and all blue points below $s$. In the dual, this means all red lines below $s^*$ and all blue lines above $s^*$ are misclassified. Consider the dual arrangement of lines $R^* \cup B^*$. For any two separators $s_1$ and $s_2$ whose duals lie in the same face of the arrangement, $M_{\text{mis}}(s_1) = M_{\text{mis}}(s_2)$. Let a face containing valid points be a *valid face*, and note that points on the boundary of a valid face are also valid. A *valid region* is the union of a maximal set of adjacent valid faces, and the boundary of a valid region is composed of *valid edges* (note that we ignore edges that are fully contained within a valid region). Now observe that $S_k(B \cup R)$ thus corresponds to the union of these valid regions. With some abuse of notation we use $S_k(B \cup R)$ to refer to this union of regions in the dual plane as well.

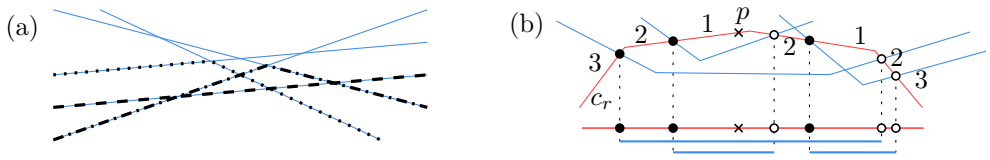


**Figure 2** (a) A concave chain decomposition of $L_{\leq 2}(B^*)$. (b): Blue chains create intervals on a red chain $c_r$. The blue ply of a point $p$ on $c_r$ is the number of endpoints (open) before $p$ plus the number of startpoints (closed) after $p$.

We observe the following useful properties (refer to the full version for omitted proofs):

▶ **Lemma 6** (Chan [4]). *The set $S_k(B \cup R)$ is contained in $L_{\leq k}(R^*) \cap L'_{\leq k}(B^*)$, consists of $O(k^2)$ valid regions, and its total complexity is $O(nk^{1/3} + n^{5/6-\varepsilon}k^{2/3+2\varepsilon} + k^2)$.*

▶ **Lemma 7.** *There may be $\Omega(k^2)$ valid regions of total complexity $\Omega(k^2 + ne^{\sqrt{\log k}})$.*

▶ **Lemma 8.** *There are $O(k^2)$ red-blue intersections in $L_{\leq k}(R^*) \cap L'_{\leq k}(B^*)$.*

▶ **Lemma 9.** *A line $\ell$ has $O(k)$ intersections with $L_{\leq k}(R^*) \cap L'_{\leq k}(B^*)$.*

▶ **Lemma 10.** *A valid region $V$ is bounded by red lines on the top and blue lines on the bottom. The leftmost point of $V$ is a red-blue intersection, or $V$ is unbounded towards the left. The rightmost point in $V$ is a red-blue intersection, or $V$ is unbounded to the right.*

## 3 Dynamic linear programming with violations

In this section we consider the following problem: given a set of $n$ constraints (halfplanes) $H$ in $\mathbb{R}^2$, an objective function $f$, and an integer $k$, find a point $p$ that violates at most $k$ constraints and minimizes $f(p)$. We assume without loss of generality that $f(p) = p_x$, so we are looking for the leftmost *valid* point, that is, a point that violates at most $k$ constraints. Chan solves this problem in $O((n+k^2)\log n)$ time [3]. In the same time bounds, their approach can find the minimum number $k_{\min}$ of constraints violated by any point. We give an overview of their techniques below, and then show how to make the approach semi-dynamic. We maintain an optimal point $p$ under semi-online insertions and deletions of constraints. "Semi-online" means that when a constraint is inserted we are told when it will be deleted. We first do so for a given value $k$, and then extend the result to maintain $k_{\min}$.

The above linear programming problem is a generalization of (the dual of) our MinMis problem. Point $p$ violates a constraint $h \in H$ if it lies outside of the halfplane. Let $R$ be the set of lines bounding lower halfplanes, and $B$ be the set lines bounding upper halfplanes. Point $p$ violates all blue constraints above, and all red constraints below, and thus $p$ violates exactly the $M_{\text{mis}}(p)$ lines in $X(p, R \cup B)$. This means we can solve the MinMis problem and compute $s_{\text{mis}} = \text{argmin}_s M_{\text{mis}}(s)$ in $O((n+k^2)\log n)$ time.

### 3.1 Chan's algorithm

Chan considers the decision version of the problem: given an integer $k$, find the leftmost point that violates at most $k$ constraints. Their algorithm actually generates all local minima that violate fewer than $k$ constraints as well, so by guessing $k = \sqrt{n}, 2\sqrt{n}, 4\sqrt{n}\ldots$ we can find the minimum value $k_{\min}$ for which a valid point exists in the same time bounds.

We first assume that there are no valid regions that are unbounded towards the left. By Lemma 10, the leftmost valid point in a valid region must then be a red-blue intersection, and by Lemma 8 there are only $O(k^2)$ of them. We construct the concave chain decomposition of $L_{\leq k}(R)$ and the convex chain decomposition of $L'_{\leq k}(B)$ in $O(n \log n)$ time, and compute their intersections in $O(k^2 \log n)$ time; this gives us all candidate optima.

Consider a red chain $c_r$, as in Figure 2b. Every blue chain $c_b$ defines a (possibly empty) interval on $c_r$, such that points inside the interval lie above $c_b$ and points outside the interval lie below $c_b$. The *blue ply* of a point $p$ on $c_r$ is the number of blue chains above $p$ (and thus the number of violated blue constraints above $p$). This is the number of blue intervals not containing $p$, and thus the number of intervals ending before $p$ or starting after $p$. By storing the start points and end points of all blue intervals in two balanced binary trees we can thus find the blue ply of any point $p$ on $c_r$ in $O(\log k)$ time. We call this the *chromatic ply data structure* of $c_r$. The chromatic ply data structure of a blue chain $c_b$ is defined symmetrically.

For an intersection point $p$ between red chain $c_r$ and blue chain $c_b$ we can now calculate its $M_{\text{mis}}(p)$ value: query $c_r$ for the blue ply and query $c_b$ for the red ply, both in $O(\log k)$ time, and sum them up. For all $O(k^2)$ red-blue intersections this takes $O(k^2 \log k)$ time. Among them we then find the leftmost valid intersection and return it, if it exists.

If the optimum was unbounded, then part of the leftmost segment of one of the chains must be valid. We can check this in $O(k \log k)$ time using the chromatic ply data structures.

## 3.2 A semi-dynamic data structure for a fixed $k$

We now make the above algorithm dynamic under semi-online insertions and deletions: given a fixed value $k$, we maintain the leftmost point that violates at most $k$ constraints.

We first show how to maintain the concave chain decomposition of $L_{\leq k}(R)$ (and similarly the convex chain decomposition of $L'_{\leq k}(B)$) using an extension of the logarithmic method [10, 26], then show how to maintain the chromatic ply datastructures, and lastly use these chains to actually maintain the leftmost valid separator.

**Maintaining the concave chain decomposition.** We maintain the concave chain decomposition of $L_{\leq k}(R)$ using Dobkin and Suri's extension of the logarithmic method [10, 26]. We maintain a partition of $R$ into $z = O(\log n)$ subsets $R_0, R_1 .. R_z$, such that for each layer $i$:
**(1)** none of the lines in set $R_i$ will be deleted for at least $2^i$ updates after the set is created.
**(2)** $|R_i| = O(2^i)$.

For each set $R_i$ we store the concave chain decomposition of $L_{\leq k}(R_i)$. Since each such structure contains $O(k)$ chains, we have $O(k \log n)$ chains in total. The union of these chains also covers $L_{\leq k}(R)$: if a line $\ell$ is among the lowest $k$ lines in $R$ at some $x$-coordinate, it must also be among the lowest $k$ lines in any subset $R' \subseteq R$, including the subset $R_i$ containing $\ell$.

The basic idea is the following. After set $R_i$ is created, by condition (1) no items will be deleted from it for at least $2^i$ updates, so it remains fixed for $2^i$ updates and gets rebuilt after that. As such, the smaller data structures are rebuilt quite often, and the larger data structures remain fixed for a long time. By construction, deletions happen only at layer 0 from set $R_0$, which contains $O(1)$ lines. Lines are inserted in layer 0, and gradually move to higher layers where they remain fixed for an ever increasing number of updates. When a line is to be deleted soon, it gradually moves down to layer 0 again.

▶ **Lemma 11.** *We can maintain $O(k \log n)$ concave chains of total complexity $O(n)$ that cover $L_{\leq k}(R)$ under semi-online insertions and deletions in $O(\log^2 n)$ amortized time.*

In fact, we can maintain a slightly altered version of the above data structure within the same time and space bounds. Let $2^x$ be the smallest power of two that is at least $k \log n$, i.e. $2^{x-1} < k \log n \leq 2^x$. We store the $2^x$ lines that are the first to be deleted in a separate list, the *leftover* list. We do not build a chain data structure on the leftover lines, but instead we let each leftover line forms a trivial chain, so we still have $O(k \log n)$ chains covering $L_{\leq k}(R)$. This way all sets $R_j$ with $j < x$ are empty. We can thus perform $2^x$ *cheap* updates without having to modify any of the sets $R_j$: we can simply insert directly in (or delete directly from) the leftover list, without having to rebuild any data structure, in $O(1)$ time. Once every $2^x$ updates we perform an *expensive* update, destroying all sets up to some layer $i'$ (the largest set that no longer adheres to invariant (1)), and redistributing all the lines in them over the layers 0 through $i'$ again. Each set $R_i$ still has an amortized update time of $O(i)$, and thus the total amortized update time remains $O(\log^2 n)$.

**Maintaining intersections and chromatic ply data structures.** Next, we show how to maintain the chromatic ply data structures on the chains, which also gives us the set $I_{\leq k}$ of $O(k^2)$ red-blue intersections in $L_{\leq k}(R^*) \cap L'_{\leq k}(B^*)$.

We maintain a concave chain decomposition of $L_{\leq k}(R^*)$ and a convex chain decomposition of $L'_{\leq k}(B^*)$ using Lemma 11, with $O(k \log n)$ leftover lines. Whenever we perform an expensive update on one of the two (i.e. rebuilding the chains), we do so on the other as well. On each red chain $c_r$ we maintain a blue chromatic ply data structure. Similarly on each blue chain $c_b$ we maintain a red ply data structure. Additionally, we maintain a set $I$ of the $O(k^2 \log^2 n)$ red-blue intersections between the chains. This is a superset of $I_{\leq k}$.

Consider the insertion of a line $r$. Depending on the type of update, we do the following:

**Cheap update:** line $r$ is added to the leftover list, and forms a trivial chain $c_r$. We compute all $O(k \log n)$ intersections between $c_r$ and the blue chains, insert them in $I$, and build the blue ply data structure on $c_r$. For each intersected blue chain $c_b$ we insert the interval induced on $c_b$ by $c_r$ into the red ply data structure of $c_b$. This takes $O(k \log^2 n)$ time.

**Expensive update:** some number of chains are rebuilt. We rebuild the set $I$ and the chromatic ply data structures on all chains from scratch. Since we have $O(k \log n)$ red and blue chains, this takes $O(k^2 \log^3 n)$ time.

Every $2^x = O(k \log n)$ updates we have $2^x - 1$ cheap updates, taking $O(k \log^2 n)$ time each, and one expensive update, taking $O(k^2 \log^3 n)$ time. This thus takes $O(k^2 \log^3 n)$ time for $2^x$ updates, making the amortized updates time $O(k \log^2 n)$. We thus have:

▶ **Lemma 12.** *We can maintain a set $I \supseteq I_{\leq k}$ of $O(k^2 \log^2 n)$ bichromatic intersection points under semi-online updates in amortized $O(k \log^2 n)$ time. This uses $O(n + k^2 \log^2 n)$ space.*

**Maintaining the leftmost valid point.** The last step is to maintain the leftmost valid point $s$ for a fixed value $k$. We know $s$ is contained in the set $I$ maintained by Lemma 12, but simply iterating through the entire set each update would take too long. We store $I$ in a data structure that maintains $M_{\text{mis}}(p)$ for each $p \in I$, and can handle the following operations:

**Insertion/Deletion:** Inserting or deleting a point (a red-blue intersection).

**Halfplane update:** Update $M_{\text{mis}}(p)$ for each $p \in I$ after the insertion or deletion of a constraint, e.g. increment $M_{\text{mis}}(p)$ by one for all points $p$ in the halfplane above an inserted line $r$ (or in the halfplane below an inserted line $b$).

**Query:** Given a query value $k' \leq k$, return the leftmost point $p \in I$ with $M_{\text{mis}}(p) \leq k'$.

We can achieve the above using a binary search tree on $I$ sorted by $x$-coordinate, and a partition tree [5] on $I$ where each node $u$ stores (a point attaining) the minimum number of constraints violated by a point in its canonical subset. We use the logarithmic method to

handle insertions on the partition tree, and perform deletion of a point $p \in I$ implicitly by setting $M_{\text{mis}}(p) = \infty$. A halfplane update corresponds to one "query" in the partition tree, where we only recurse on intersected triangles. Using the binary search tree and the partition tree, we can then binary search for the leftmost point that violates at most $k'$ constraints.

▶ **Lemma 13.** *We can build a data structure on a set of $O(k^2 \log^2 n)$ points $I$ that can perform insertions and deletions in $O(\log k \log n)$ amortized time, halfplane updates in expected $O(k \log^2 n)$ time, and queries in expected $O(k \log^3 n)$ time. The data structure uses $O(k^2 \log^2 n)$ space.*

We can now dynamically maintain the solution to an LP with at most $k$ violations by using Lemmas 11, 12 and 13 as follows. For each cheap update, e.g. the insertion of a line $r$, we find the $O(k \log n)$ intersections between $r$ and blue chains, and insert them in $O(k \log^2 n \log k)$ total time. We then perform one halfplane update in $O(k \log^2 n)$ time. For each expensive update we discard the data structures from Lemmas 12 and 13 and rebuild them from scratch. This takes $O(k^2 \log^3 n)$ time, and thus $O(k \log^2 n)$ amortized time. After every update we perform one query with $k' = k$ in $O(k \log^3 n)$ time. This establishes Theorem 2. In the full version, we extend the data structure to maintain the minimum number $k_{\min}$ of constraints violated by any point as well.

## 4 Exact algorithms for $k$-mis MinMax

For the $k$-mis MinMax problem we are given point sets $R$ and $B$ and an integer $k$ and wish to compute a separator $s_{\text{opt}} = \text{argmin}_{s \in S_k(R \cup B)} M_{\max}(s)$ that misclassifies at most $k$ points and minimizes the distance to the furthest misclassified point. In this section we present an exact algorithm for this problem. In Section 4.1 we first discuss some useful geometric properties. In Section 4.2 we then present algorithms to construct the valid regions $S_k(R \cup B)$. Finally, in Section 4.3, we show how we can then compute an optimal separator.

### 4.1 Geometric properties

**The MinMax problem.** We first consider the MinMax problem. Here, we wish to compute $s_{\max} = \text{argmin}_s M_{\max}(s)$, a separator with minimal distance to the farthest misclassified point. Consider the dual plane. At a fixed $x$-coordinate, this point lies exactly in the middle of the envelopes $L_0(R^*)$ and $L'_0(B^*)$. Let the MinMax curve be the polygonal curve in the middle of $L_0(R^*)$ and $L'_0(B^*)$, and observe that it consists of $O(n)$ segments. See Figure 3.

▶ **Lemma 14.** *Let $s$ be a point in the interior of an edge $e$ of the MinMax curve. Moving $s$ left or moving $s$ right along $e$ decreases the error $M_{\max}(s)$.*

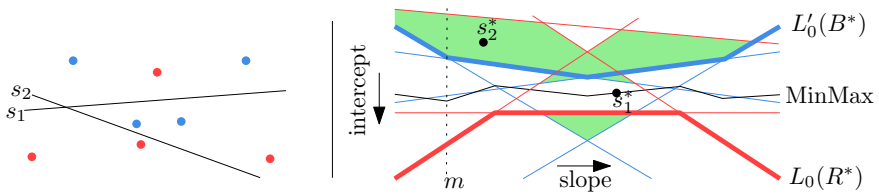


**Figure 3** Some primal points (left) with their dual (right). Valid faces for $k = 2$ are green.

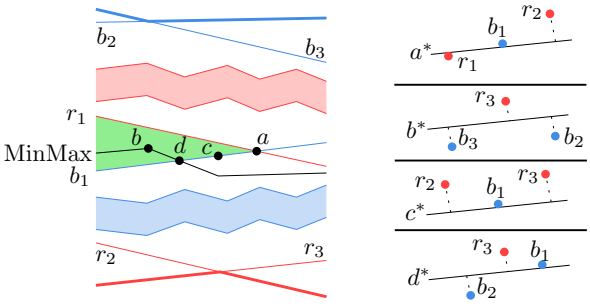


**Figure 4** Left: the cases **a, b, c, d** for $s^*_{\text{opt}}$ in the dual plane; the red/blue regions represent some number of correctly classified lines. Right: the cases **a, b, c, d** for $s_{\text{opt}}$ in the primal plane.

**The $k$-min MinMax problem.** For the $k$-mis MinMax problem, we wish to compute $s_{\text{opt}} = \text{argmin}_{s \in S_k(B \cup R)} M_{\max}(s)$, a valid separator with minimal $M_{\max}(s_{\text{opt}})$. At a fixed $x$-coordinate, this is the valid separator (if it exists) with the smallest vertical distance to the MinMax curve. This fact and Lemma 14 lead to the following characterization:

▶ **Lemma 15.** *A point $s^*_{\text{opt}}$ dual to an optimal separator is one of the following:*

**a.** *A vertex of a valid face, vertically closest to MinMax.*
**b.** *A (valid) vertex of MinMax.*
**c.** *The first valid point directly above or below a vertex of MinMax.*
**d.** *The intersection of a MinMax edge $e$ with a valid edge, closest to one of $e$'s endpoints.*

**Proof sketch.** In the primal plane, the optimal separator $s_{\text{opt}}$ has to be "bounded" by at least three points, otherwise we can rotate or translate $s_{\text{opt}}$ slightly to decrease $M_{\max}(s_{\text{opt}})$. These bounding points can either be extremal points that we want the separator to be as close to as possible, or points that the separator is not allowed to cross because that would make it invalid. The four ways in which $s_{\text{opt}}$ can be bounded are shown in Figure 4. ◀

## 4.2 Constructing the valid regions

We present three algorithms for constructing the valid regions $S_k(B \cup R)$.

First, by Lemma 6 all valid points lie inside $L_{\leq k}(R^*) \cap L'_{\leq k}(B^*)$, so we present a simple algorithm that constructs this part of the arrangement, and prunes all invalid regions. Since $L_{\leq k}(R^*)$ and $L'_{\leq k}(B^*)$ have complexity $O(nk)$, this gives an $O(n \log n + nk)$ time algorithm.

Second, we present a much more involved output sensitive $O((nk^{1/3} + n^{5/6-\varepsilon}k^{2/3+2\varepsilon} + k^3)\log^2 n)$ time algorithm, which is faster for $k < n^{2/3}$. It is based on an approach sketched by Chan [4] to compute the valid region in an output-sensitive manner, by first computing the bichromatic intersection points of $L_{\leq k}(R^*) \cap L'_{\leq k}(B^*)$, and then tracing $S_k(B \cup R)$, starting from these bichromatic intersection points "as in a standard $k$-level algorithm". They claim this results in a running time of $O(|S_k(B \cup R)| \operatorname{polylog} n)$ time, but do not provide details. The $k$-level in an arrangement of lines is connected, whereas here $S_k(B \cup R)$ may consist of $\Omega(k^2)$ disconnected pieces (Lemma 7). This unfortunately provides some additional difficulties in initializing the data structure used in the tracing process. Hence, it is not clear that it can indeed be done in $O(|S_k(B \cup R)| \operatorname{polylog} n)$ time. Instead, we present an algorithm that runs in $O((|S_k(B \cup R)| + n + k^3)\log^2 n)$ time, the stated time bound.

Finally, if we care only about the case where $k = k_{min} = M_{\text{mis}}(s_{\text{mis}})$, i.e. where we are required to misclassify as few points as possible, we observe that each valid region is a single face. By Lemma 6 there are $O(k^2)$ valid regions, so now there are $O(k^2)$ valid faces. Clarkson et al. [7] show that $m$ faces in an arrangement have a complexity of $O(m^{2/3}n^{2/3} + n)$, and we can construct them in $O(k^{4/3}n^{2/3}\log^{2/3}(n/k) + (n + k^2)\log n)$ time [27].

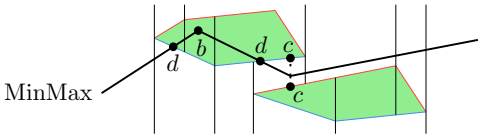


**Figure 5** Two valid faces with their vertical decomposition and type $b$, $c$ and $d$ points.

## 4.3 An algorithm for solving the $k$-mis MinMax problem

We now show how, given the valid regions, we can compute an optimal separator $s_{\text{opt}} \in S_k(B \cup R)$ efficiently. We start by constructing $L_0(R)$ and $L'_0(B)$, and simultaneously scan through them to construct the MinMax curve $s^*_{\max}$. This takes $O(n \log n)$ time [9]. By Lemma 15 an optimal separator is of type $a, b, c$, or $d$. So, we will now compute all these candidate optima, and iterate through them to find the one with lowest error.

**Type a points.** Since we are given $S_k(B \cup R)$, we can simply scan through its vertices, keeping track of the vertex with the smallest error. To calculate the error of a vertex, we need to know which segment of $s^*_{\max}$ it lies above/below; then the error can be calculated in $O(1)$ time. We can compute this in $O(\log n)$ time per vertex using binary search (since MinMax is $x$-monotone). Hence, this step takes $O(|S_k(B \cup R)| \log n)$ time.

**Type b and c points.** Recall type $b$ points are MinMax vertices, and type $c$ points are the first valid points above or below MinMax vertices. We construct the *trapezoidal decomposition* of $S_k(B \cup R)$ in $O(|S_k(B \cup R)| \log n)$ time, which supports $O(\log n)$ time point location queries [24]. Each trapezoid has vertical left and right sides, see Figure 5.

For each vertex of MinMax we perform one point location query, which tells us what trapezoid the vertex lies in. If this trapezoid is inside a valid region, the vertex is a type $b$ point. Otherwise the closest valid edges vertically above and below this vertex are simply the edges bounding that trapezoid, giving us up to two type $c$ points. Since MinMax has $O(n)$ vertices, this gives us all type $b$ and $c$ points in $O(n \log n)$ time. Including the time to build the decomposition, this thus takes $O(|S_k(B \cup R)| + n) \log n)$ time.

**Type d points.** Recall type $d$ points are intersections between MinMax and edges bounding $S_k(B \cup R)$. In particular, for every MinMax edge we care only about its outermost intersection points. We walk along MinMax from left to right through the vertical decomposition until we find the leftmost intersection on an edge; we then continue the walk from the next MinMax vertex. We find the rightmost intersection symmetrically. After locating the MinMax vertices in $O(n \log n)$ time, this takes $O(n + |S_k(B \cup R)|)$ time, since MinMax is $x$-monotone.

▶ **Lemma 16.** *Given $S_k(B \cup R)$, we can compute a separator $s_{\text{opt}} = \operatorname{argmin}_{s \in S_k(B \cup R)} M_{\max}(s)$ in $O((|S_k(B \cup R)| + n) \log n)$ time.*

By combining Lemma 16 with the three algorithms from Section 4.2 we thus obtain an $O((nk + n) \log n)$ (which we can reduce to $O(nk + n \log n)$) and an $O((|S_k(B \cup R)| + n + k^3) \log^2 n)$ time algorithm for the general problem, and an $O(k^{4/3} n^{2/3} \log n + (n + k^2) \log n)$ time algorithm for when $k = k_{\min}$. These results together establish Theorem 3.

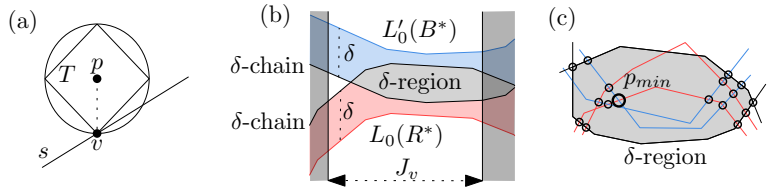


**Figure 6** (a) The Euclidean unit circle and convex unit 4-gon. (b) The convex and concave $\delta$-chain forming a $\delta$-region. (c) A $\delta$-region, with candidate red-blue intersections marked.

## 5 An $\varepsilon$-approximation algorithm

Let $s_{\text{opt}} \in S_k(B \cup R)$ be an optimal valid separator minimizing $M_{\max}$, and let $\varepsilon \in (0,1)$ be some given threshold. Our goal is to compute a $(1+\varepsilon)$-*approximation* of $s_{\text{opt}}$: that is, we want to find a valid separator $\hat{s}$ with $M_{\max}(\hat{s}) \leq (1+\varepsilon)M_{\max}(s_{\text{opt}})$. The main idea is to replace the Euclidean distance function *dist* by some convex distance function $\hat{d}$ that approximates *dist*, and compute a separator $\hat{s}$ that minimizes $\hat{M}(\hat{s}) = \max_{p \in X(\hat{s}, B \cup R)} \hat{d}(p, \hat{s})$.

Let $p$ be a point and $s$ be a line, let $t = \Theta(1/\sqrt{\varepsilon})$, and let $T$ be a convex regular $t$-gon centered at the origin inscribed by a unit disk. See Figure 6a. We then define the convex distance function $\hat{d}(p,s) = \min\{\lambda \mid s \cap (p + \lambda T) \neq \emptyset\}$ to be the smallest scaling factor for which a scaled copy of $T$ centered at $p$ intersects $s$. It can be shown that $dist(p,s) \leq \hat{d}(p,s) \leq (1+\varepsilon)dist(p,s)$ [11, 15], and thus $M_{\max}(s) \leq \hat{M}(s) \leq (1+\varepsilon)M_{\max}(s)$. It follows that the separator $\hat{s}$ minimizing $\hat{M}$ is a $(1+\varepsilon)$-approximation of $s_{\text{opt}}$.

Observe that this distance $\hat{d}(p,s)$ is realized in a corner $v$ of the $t$-gon; i.e. the $t$-gon scaled by a factor $\hat{d}(p,s)$ intersects $s$ in a corner point $v$ of the $t$-gon. We say $v$ is a *realizer* for the line $s$. More specifically, there is some interval of slopes $J_v$ such that $v$ is the realizer for all lines with a slope in the interval $J_v$. For each slope interval $J_v$ we will compute a valid separator $\hat{s}^v$ with slope in $J_v$ minimizing $\hat{M}(\hat{s}^v)$, and finally $\hat{s} = \text{argmin}_v M_{\max}(\hat{s}^v)$.

We consider one such slope interval $J_v$. Assume w.l.o.g. that $v$ is vertically below the center point of the $t$-gon (we can rotate the plane to achieve this). This means interval $J_v$ is centered at slope 0, so $J_v = (-\pi/t, \pi/t)$, and the distance $\hat{d}(p,s)$ between a point $p$ and line $s$ is the vertical distance between $p$ and $s$. Since vertical distance is preserved by dualizing, this means that for all points $s$ in the $x$-interval $J_v$, the value $\hat{M}(s)$ expresses the vertical – and thus convex $t$-gon – distance from $s$ to $L_0(R^*)$ or $L_0'(B^*)$, whichever is larger.

**The algorithmic problem.** Extending Chan's algorithm from Section 3.1, we build a data structure that, for a given value $\delta$, can find a valid separator $s \in J_v \times \mathbb{R}$ with $\hat{M}(s) \leq \delta$ if it exists. We then use parametric search [21] to find the optimal value $\delta$, and a separator $\hat{s}^v$.

Fix a value $\delta$, and observe that all points with error at most $\delta$ lie at most $\delta$ below $L_0'(B)$. This can be imagined as moving $L_0'(B)$ down by $\delta$. Let the resulting chain be the *convex* $\delta$*-chain*, see Figure 6(b). Similarly, let the *concave* $\delta$*-chain* be $L_0(R)$ moved up by $\delta$. All points with error at most $\delta$ must thus lie above the convex $\delta$-chain, and below the concave $\delta$-chain: the $\delta$*-region*. The question now becomes: does a valid point exist in the $\delta$-region?

In Section 3.1 we considered only red-blue intersections. Similarly, we can show that now we need to consider only intersections between convex chains (a blue chain or the convex $\delta$-chain) and concave chains (a red chain or the concave $\delta$-chain). There are $O(k^2)$ such convex-concave intersections (see Figure 6(c)). We can compute them all during preprocessing, find the valid point $p_{\min}$ among them with smallest error, and simply forget about all others.

The data structure consists of three parts. First, a concave chain decomposition of $L_{\leq k}(R)$, and a convex chain decomposition of $L'_{\leq k}(B)$, with a chromatic ply data structure for every chain. Second, the point $p_{\min}$. Third, the envelopes $L_0(R)$ and $L'_0(B)$. This can all be built in $O((n+k^2)\log n)$ time using Chan's method, and uses $O(n+k^2)$ space.

We answer a query with value $\delta$ as follows. We check if $\hat{M}(p_{\min}) \leq \delta$, and if so, return $p_{\min}$. If not, we find the $O(k)$ convex-concave intersections involving the $\delta$-chains, and build a red ply data structure for the convex $\delta$-chain, and a blue ply data structure for the concave $\delta$-chain. For each intersection $p$ we compute $M_{\text{mis}}(p)$ using the chromatic ply data structures, and compute $M_{\max}(p)$ using the envelopes. Finally we return the valid intersection with lowest error if its error is at most $\delta$, otherwise there exists no point with error at most $\delta$.

Using parametric search on the above data structure gives us a valid separator with slope in $J_v$ with the lowest error in $O((n+k^2)\log n)$ time. Doing this for all $t = \Theta(1/\sqrt{\varepsilon})$ slope intervals $J_v$ proves Theorem 4.

**A dynamic data structure.** Using the dynamic chain decomposition data structure from Lemma 11, we can maintain the data structure under semi-online updates, and perform the parametric search after every update to maintain an optimum $\hat{s}$, thus proving Theorem 5.

## References


1. Edoardo Amaldi and Viggo Kann. The complexity and approximability of finding maximum feasible subsystems of linear relations. *Theor. Comput. Sci.*, 147(1&2):181–210, 1995. doi:10.1016/0304-3975(94)00254-G.
2. Boris Aronov, Delia Garijo, Yurai Núñez Rodríguez, David Rappaport, Carlos Seara, and Jorge Urrutia. Minimizing the error of linear separators on linearly inseparable data. *Discret. Appl. Math.*, 160(10-11):1441–1452, 2012. doi:10.1016/j.dam.2012.03.009.
3. Timothy M. Chan. Low-dimensional linear programming with violations. *SIAM J. Comput.*, 34(4):879–893, 2005. doi:10.1137/S0097539703439404.
4. Timothy M. Chan. On the bichromatic $k$-set problem. *ACM Trans. Algorithms*, 6(4):62:1–62:20, 2010. doi:10.1145/1824777.1824782.
5. Timothy M Chan. Optimal partition trees. In *Proceedings of the twenty-sixth annual symposium on Computational geometry*, pages 1–10, 2010. doi:10.1145/1810959.1810961.
6. Timothy M. Chan and Konstantinos Tsakalidis. Optimal deterministic algorithms for 2-d and 3-d shallow cuttings. *Discret. Comput. Geom.*, 56(4):866–881, 2016. doi:10.1007/S00454-016-9784-4.
7. Kenneth L Clarkson, Herbert Edelsbrunner, Leonidas J Guibas, Micha Sharir, and Emo Welzl. Combinatorial complexity bounds for arrangements of curves and spheres. *Discrete & Computational Geometry*, 5(2):99–160, 1990. doi:10.1007/BF02187783.
8. Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, 1995. doi:10.1007/BF00994018.
9. Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008. URL: https://www.worldcat.org/oclc/227584184.
10. David Dobkin and Subhash Suri. Dynamically computing the maxima of decomposable functions, with applications. In *30th Annual Symposium on Foundations of Computer Science*, pages 488–493. IEEE Computer Society, 1989. doi:10.1109/SFCS.1989.63523.
11. Richard M Dudley. Metric entropy of some classes of sets with differentiable boundaries. *Journal of Approximation Theory*, 10(3):227–236, 1974.
12. Hazel Everett, Jean-Marc Robert, and Marc J. van Kreveld. An optimal algorithm for the (≤ k)-levels, with applications to separation and transversal problems. *Int. J. Comput. Geom. Appl.*, 6(3):247–261, 1996.

13 Erwin Glazenburg, Frank Staals, and Marc van Kreveld. Robust classification of dynamic bichromatic point sets in r2, 2024. arXiv:2406.19161, doi:10.48550/arXiv.2406.19161.

14 D. Goldfarb and S. Liu. An $O(n^3L)$ primal interior point algorithm for convex quadratic programming. *Mathematical Programming*, 49(1):325–340, 1990.

15 Sariel Har-Peled and Mitchell Jones. Proof of dudley's convex approximation. *arXiv preprint*, 2019. arXiv:1912.01977.

16 Sariel Har-Peled and Vladlen Koltun. Separability with outliers. In *Algorithms and Computation, 16th International Symposium, ISAAC 2005, Sanya, Hainan, China, December 19-21, 2005, Proceedings*, volume 3827 of *Lecture Notes in Computer Science*, pages 28–39. Springer, 2005. doi:10.1007/11602613_5.

17 Cho-Jui Hsieh, Si Si, and Inderjit Dhillon. A divide-and-conquer solver for kernel support vector machines. In *International conference on machine learning*, pages 566–574. PMLR, 2014. URL: http://proceedings.mlr.press/v32/hsieha14.html.

18 M. K. Kozlov, S. P. Tarasov, and L. G. Khachiyan. The polynomial solvability of convex quadratic programming. *USSR Comp. Math. and Math. Phys.*, 20(5):223–228, 1980.

19 Michael Matheny and Jeff M. Phillips. Approximate maximum halfspace discrepancy. In Hee-Kap Ahn and Kunihiko Sadakane, editors, *32nd International Symposium on Algorithms and Computation, ISAAC 2021, December 6-8, 2021, Fukuoka, Japan*, volume 212 of *LIPIcs*, pages 4:1–4:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ISAAC.2021.4.

20 Jirí Matousek. On geometric optimization with few violated constraints. *Discret. Comput. Geom.*, 14(4):365–384, 1995. doi:10.1007/BF02570713.

21 Nimrod Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM (JACM)*, 30(4):852–865, 1983. doi:10.1145/2157.322410.

22 Nimrod Megiddo. Linear programming in linear time when the dimension is fixed. *J. ACM*, 31(1):114–127, 1984. doi:10.1145/2422.322418.

23 R. D. C. Monteiro and I. Adler. Interior path following primal-dual algorithms. Part II: Convex quadratic programming. *Mathematical Programming*, 44(1):43–66, 1989. doi:10.1007/BF01587076.

24 Neil Sarnak and Robert E Tarjan. Planar point location using persistent search trees. *Communications of the ACM*, 29(7):669–679, 1986. doi:10.1145/6138.6151.

25 Jeffrey C Schlimmer and Richard H Granger Jr. Beyond incremental processing: Tracking concept drift. In *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*, pages 502–507, 1986.

26 Michiel Smid. A worst-case algorithm for semi-online updates on decomposable problems. Technical report, Univeristät des Saarlandes, 1990. doi:10.22028/D291-26450.

27 Haitao Wang. Constructing many faces in arrangements of lines and segments. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3168–3180. SIAM, 2022. doi:10.1137/1.9781611977073.123.