# Computing the Maximum Bichromatic Discrepancy, with applications to Computer Graphics and Machine Learning

David P. Dobkin [1]

Department of Computer Science, Princeton University,
35 Olden St., Princeton, NJ 08540, USA
e-mail: dpd@cs.princeton.edu

Dimitrios Gunopulos [1]

Department of Computer Science, Princeton University,
35 Olden St., Princeton, NJ 08540, USA
e-mail: dg@cs.princeton.edu

Wolfgang Maass

Institute for Theoretical Computer Science,
Technische Universitaet Graz,
Klosterwiesgasse 32/2, A-8010 Graz, Austria
e-mail: maass@igi.tu-graz.ac.at

## Abstract

Computing the maximum bichromatic discrepancy is an interesting theoretical problem with important applications in computational learning theory, computational geometry and computer graphics. In this paper we give algorithms to compute the maximum bichromatic discrepancy for simple geometric ranges, including rectangles and halfspaces. In addition, we give extensions to other discrepancy problems.

## 1. Introduction

The main theme of this paper is to present efficient algorithms that solve the problem of computing the maximum bichromatic discrepancy for axis oriented rectangles. This problem arises naturally in different areas of computer science, such as computational

[1]The research work of these authors was supported by NSF Grant CCR93-01254 and the Geometry Center.

learning theory, computational geometry and computer graphics ([Ma], [DG]), and has applications in all these areas.

In computational learning theory, the problem of agnostic PAC-learning with simple geometric hypotheses can be reduced to the problem of computing the maximum bichromatic discrepancy for simple geometric ranges. In computational geometry, efficient computation of the discrepancy of a two-colored point set is useful for the construction of $\epsilon$-approximations of point sets. Finally in computer graphics, the maximum numerical discrepancy of a point set is a good measure on how well a sampling pattern captures details in a picture.

In the next three parts of the introduction we give the background and present three views of the same algorithmic problem from the perspective of learning theory (the minimizing disagreement problem), computer graphics (the numerical discrepancy) and computational geometry (the bichromatic discrepancy). The subject of section 2 is Algorithm 2, an $O(n^2 \log n)$ algorithm that computes the maximum bichromatic discrepancy in two dimensions. Here, we first prove that that the minimizing disagreement problem and the computation of the maximum bichromatic discrepancy are equivalent. Then, we develop Algorithm 2 working in one and then in two dimensions. In section 3 we modify Algorithm 2 to compute the maximum numerical discrepancy. In section 4 we give an approximation algorithm, and we extend our results in higher dimensions.

Finally, we note here that we are using the RAM model of computation ([HU], [P]) throughout this paper.

### 1.1 Agnostic PAC-Learning and the minimizing disagreement problem

One goal of computational learning theory is to provide tools for the design and analysis of learning algorithms that provide satisfactory solutions for real-world learning problems. There are a lot of experimental results regarding the performance of various heuristic learning algorithms on a number of "benchmark"-datasets for real world classification problems ([Min], [WK90], [WGT], [WK91], [BN], [Ho]), but the set of learning algorithms that are examined in these applications is virtually disjoint from the set of learning algorithms that are traditionally considered in computational learning theory.

Haussler in [Hau] provided an important link between theoretical and applied machine learning, when he introduced a variation of Valiant's ([V84]) well known model for Probably Approximately Correct learning ("PAC-learning"). The new model, agnostic PAC-learning, provides an adequate format for the generic formulation of real-life classification problems. The shortcoming of the original PAC-learning model of Valiant is that it relies on the assumption that the labels of the training examples arise from a target concept with an apriori known specific simple structure, an assumption rarely met in practice. Valiant later extended the basic model to include the possibility that some labels come from noise rather than from the target concept. However, so far positive learning results in this model have been proven only under the assumption that the noise is of specific structure ([AL], [KS]) or that the percentage of the noisy labels is very small in comparison to the desired error bound $\epsilon$ of the learner.

In the agnostic PAC-learning model of Haussler we assume that the learner gets a sequence of training examples, each consisting of an instance $x \in X$ and an outcome (we will also call it a label) $y \in Y$. The sets $X$ and $Y$ are arbitrary sets, called instance space and outcome space respectively. However for the most part of this paper we will set $Y = \{0, 1\}$. The examples are generated according to an arbitrary distribution $D$ on $X \times Y$ unknown to the learner. In a real-world application, $D$ may simply reflect the distribution of data as they occur in nature (possibly including contradictions, i.e. for some $x \in X$ both $\langle x, 0\rangle$ and $\langle x, 1\rangle$ may occur as examples) without assuming that the labels are generated by any rule. The learner is also given a set $\mathcal{H}$ of hypotheses. Each hypothesis $H \in \mathcal{H}$ is a function from $X$ to $\{0, 1\}$, and we will also call it a decision rule. The *true error* $\text{Error}_D(H)$ of a hypothesis $H$ is:

$$\text{Error}_D(H) = E_{\langle x,b\rangle \in D}(|H(x) - b|)$$

or in other words, the probability that $H$ fails to predict the label $b$ of an example $\langle x, b\rangle$ drawn according to $D$.

The goal of the learner is to compute, for given parameters $\epsilon, \delta > 0$, a hypothesis $H^* \in \mathcal{H}$ whose true error $\text{Error}_D(H^*)$ is with probability at least $1 - \delta$ not larger than $\epsilon + \inf_{H\in\mathcal{H}} \text{Error}_D(H)$.

To achieve this goal, the learner is allowed to specify a minimum size $m(\epsilon, \delta)$ for the training sequence. So the input for the computation of $H^*$ by the learner is a training sequence $T$ of at least $m(\epsilon, \delta)$ examples that are drawn from $X \times \{0, 1\}$ according to $D$. This input $T$ may be atypical for the actual distribution $D$, so we allow the learner to fail with probability at most $\delta$.

A learner which can carry out this task for *any* distribution $D$ over $X \times \{0, 1\}$ is called an *efficient agnostic PAC-learner for hypothesis class* $\mathcal{H}$ if its sample bound $m(\epsilon, \delta)$ and its number of computation steps can be bounded by a polynomial in the parameters involved (in particular in $1/\epsilon$ and $1/\delta$).

For a given training sequence $T = \{\langle x_i, b_i\rangle | 1 \leq i \leq n\}$ and hypothesis $H$, we define the *empirical error*:

$$\text{Error}_T(H) = |\{i | (\langle x_i, b_i\rangle \in T) \wedge (H(x_i) \neq b_i)\}|/n$$

that is, the number of positive (labeled 1) examples outside of $H$ plus the number of negative (labeled 0) examples inside $H$ over the size of the set. The empirical error measures how well a hypothesis predicts $D$ for the given training sequence. The following two *uniform convergence results* provide a connection (first given by Haussler, [Hau]) between the required size of the training sequence and the difference of true and empirical errors. They say that we can bound the difference between the true error and the empirical error (with high probability) if we pick a large enough training sequence at random. Thus, if we have a random training sequence large enough, we can compute the empirical error of any $H \in \mathcal{H}$ and obtain in this way a good approximation of the true error. The first result is applicable when the set $\mathcal{H}$ is finite, and the second when it has a finite VC-dimension.

1. Theorem 1 in [Hau] states that for any sample $T$ with at least $m(\epsilon, \delta) = (\ln|\mathcal{H}| + \ln(2/\delta))/(2\epsilon^2)$ examples drawn with regard to some arbitrary distribution $D$ over $X \times \{0,1\}$, the following holds with probability at least $1-\delta$:

$$\forall H \in \mathcal{H} \ \ (|\mathrm{Error}_T(H) - \mathrm{Error}_D(H)| \leq \epsilon)$$

2. A recent result by Talagrand ([T], which slightly improves [Hau]) implies that under some rather harmless measurability conditions, the same claim holds for

$$m(\epsilon, \delta) = (\mathrm{VCdim}(\mathcal{H})(\ln K + \ln\ln 2K + \ln(1/\epsilon) + \ln(1/\delta))/(2\epsilon^2)$$

   where $\mathrm{VCdim}(\mathcal{H})$ is the VC-dimension of $\mathcal{H}$, and $K$ is some absolute constant that is conjectured to be not larger than 1000.

These results show that in order to prove a positive result for efficient agnostic PAC-learning with a specific hypothesis class $\mathcal{H} \subseteq 2^X$ of bounded VC-dimension, it suffices to design an efficient algorithm for a related finite optimization problem, for which an efficient solution is also very desirable from the point of view of applied machine learning: the *minimizing disagreement problem for* $\mathcal{H}$. This is the problem of computing, for any given finite training sequence $T$ of labeled points, some hypothesis $H \in \mathcal{H}$ whose empirical error is minimal among all hypotheses in $\mathcal{H}$ . An algorithm that solves the minimizing disagreement problem for $\mathcal{H}$ is, together with the bounds for the minimum number of training examples given above, an agnostic PAC-learner for hypothesis class $\mathcal{H}$. By the same reasoning, an efficient $\epsilon_1$-approximation algorithm for the minimizing disagreement problem for $\mathcal{H}$ can produce a hypothesis $H$ with true error up to $\epsilon_1 + \epsilon$ from the optimal.

It should be pointed out that it has been shown in [KSS] that for any hypothesis class $\mathcal{H}$, the existence of an efficient algorithm for the minimizing disagreement problem for $\mathcal{H}$ is in fact also a necessary condition for efficient agnostic PAC-learning with hypothesis class $\mathcal{H}$. There are cases where the minimizing disagreement problem, and therefore efficient agnostic PAC-learning, is very hard. For example it is NP-hard to solve the problem for the class of monomials ([KSS]) and halfspaces in arbitrary dimensions ([HSV]).

In this paper we look at the minimizing disagreement problem when the class of hypotheses is the set $\mathcal{R}$ of axis aligned, but otherwise arbitrary, rectangles. A rectangular hypothesis $R \in \mathcal{R}$ defines a natural function from $X$ to $\{0,1\}$. $R(x) = 1$ if $x$ lies in the interior of $R$, and $R(x) = 0$ otherwise. The problem is, given a labeled training sequence $T$

(in two dimensions), find the rectangular hypothesis $R$ that minimizes the empirical error (Fig. 1):

$$\min_{R \in \mathcal{R}} \mathrm{Error}_T(R)$$

In the following sections we give a relatively fast polynomial solution to this problem. It is easy to show that the VC dimension of $\mathcal{R}$ is finite (see §1.3) and therefore our algorithm gives an efficient agnostic PAC-learner for rectangular hypotheses. We also consider related minimizing disagreement problems, in particular for the union of two disjoint rectangles, the complement of rectangles and unions of rectangles, and halfspaces in low dimensions.

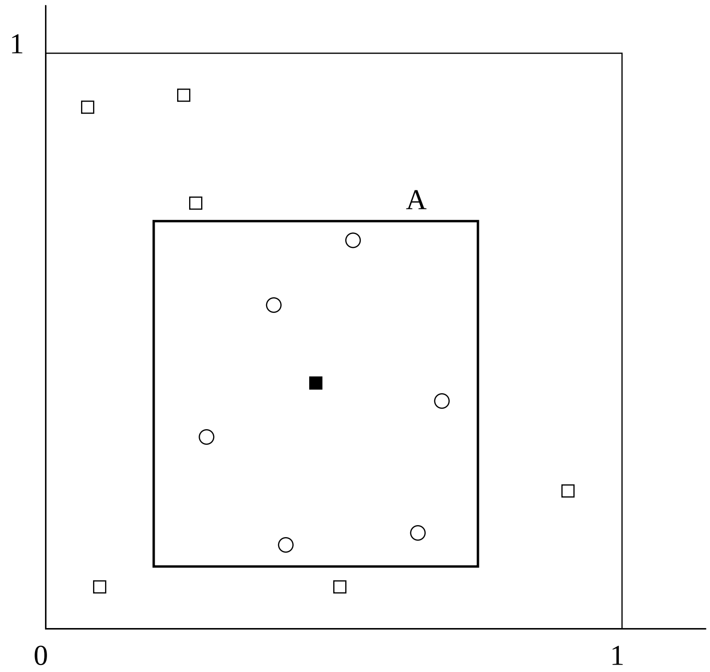



Figure 1: The hypothesis $A$ minimizes the empirical error for the point set shown (circles are labeled 1, and squares are labeled 0, $A$ fails to predict the filled point).

Simple hypotheses classes of this type have turned out to be quite interesting from the point of view of applied machine learning. Weiss et al. ([WK90], [WGT], [WK91]) have shown through experiments that for many of the standard benchmark datasets a short rule that depends on only two of the attributes, and which is a boolean combination of expressions of the form "$a_j > c$" or "$a_j = c$", provides the best available prediction-rule. For example it is reported by Weiss et al. ([WK91]) that for their appendicitis dataset the complement of a rectangle (in 2 of the 8 attributes of the particular dataset) is the prediction rule that performs best. Finally we would like to point out that optimal hypotheses of a simple type like the ones considered in this paper have the additional advantage that they provide a human user valuable heuristic insight into the structure of a real-world learning problem. Our goal is to contribute tools for the design of algorithms that compute optimal prediction rules of this kind.

## 1.2 Numerical discrepancy and sampling patterns in graphics

The importance of the sampling technique in computer graphics is clearly demonstrated in the following examples.

The synthesis of realistic images of scenes is one of the most important applications of computer graphics. Often the method of choice to produce the image of a computer modeled scene is ray tracing. In ray tracing's basic form we find the value of each pixel by sampling the radiance at a set of points in the area of the pixel. For each sample we cast a ray in the computer modeling scene and we try to trace it to light sources. The value of the pixel is then computed from a combination, typically averaging, of the value of the samples. In distributed ray tracing we are sampling in higher dimensions to produce other effects, for example motion blurring or depth of view perception. The idea in ray tracing is to find an approximate solution of the rendering equation, an integral equation derived by [Ka]. Instead of solving the continuous problem, we solve a approximate discrete version by using a set of samples. The error in this approximation, and consequently the quality of the picture, depends on how well the sampling point set approximates the area function. Therefore the error depends both both on the size and the quality of the sampling point set.

One of the most general ways of attacking antialiasing is supersampling. In this approach we sample the picture at a rate much higher than the pixel rate. So there are many samples (called supersamples) in each pixel, and to compute the pixel values we resample by averaging the supersamples within each pixel area. If we define the points we are sampling using a uniform pattern we get very visible aliasing artifacts like the Moire patterns. Therefore a stochastic or a semi-random point set is preferable because it produces less prominent random noise.

A number of different approaches have been proposed to find good quality sampling patterns. [Mit] (see also [DM]) includes an extensive search for good patterns, and shows that their use is important in practice. For example, one approach is to use sampling patterns with high frequency spectrum. These patterns drive aliasing noise to higher frequencies, where it is less visible, and can be reduced with supersampling. One promising approach is the application of the theory of discrepancy or irregularities of distribution, introduced by [BC], and applied to computer graphics first by [S] and [N92]. The discrepancy theory focuses on the problem of approximating one measure (typically a continuous one) with another (typically a discrete one). It's main application is in Quasi Monte-Carlo numerical integration, where we use a point set in order to apply finite-element techniques ([N78]).

In our graphics applications we want to approximate the area function (or a weighted area function) and our objective is to find a sampling pattern that provides good coverage for all kinds of images. Assume that we have a sample set $S$ of points in $[0,1]^d$. Let $\mathcal{F}$ be a family of regions in $[0,1]^d$. For any region $R$ in $\mathcal{F}$, let $\mu(R)$ be the Euclidian measure of $R \cap [0,1]^d$ (the area of $R$), and $\mu_S(R)$ be the discrete measure $|R \cap S|/|S|$ (the fraction of $S$ in $R$). Then the *numerical discrepancy* of $R$ with respect to $S$ is:

$$\mathcal{D}_S(R) = |\mu(R) - \mu_S(R)|$$

and the *maximum numerical discrepancy* of $S$ with respect to the family $\mathcal{F}$ is:

$$Max\mathcal{D}_{\mathcal{F}}(S) = \max_{R \in \mathcal{F}}(\mathcal{D}_S(R))$$

Discrepancy is a geometric data structures problem of broader interest. The problems that arise typically require the introduction of new techniques or extensions of existing ones ([dB], [DM], [DE]). Intuitively it is a good quality measure for sampling point sets because it provides a direct measurement on how well a given pattern estimates certain simple integral types. Ideally we would like to compute the maximum numerical discrepancy for the most general model, where the set of regions $\mathcal{F}$ is the family of convex polygons. While this problem cannot be done in reasonable complexity, other families provide useful approximations. Such families include halfspaces, stripes and axis aligned left anchored rectangles.

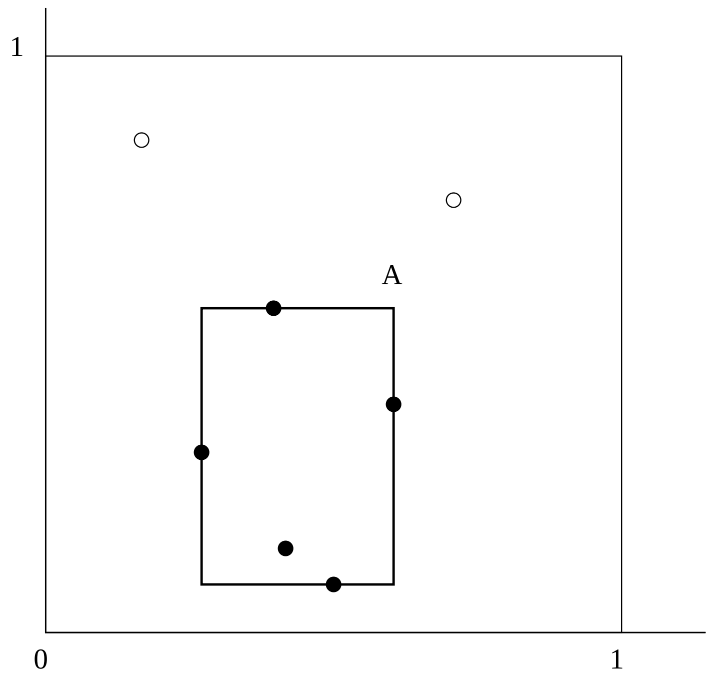



Figure 2: The rectangle $A$ maximizes the numerical discrepancy for the point set shown (the filled points are the ones inside $A$).

The family of regions that we will consider in the following sections is the set $\mathcal{R}$ of $d$-dimensional axis-aligned boxes (rectangles) in $[0,1]^d$. We will concentrate on the two-dimensional case (Fig. 2). This model, the *rectangle discrepancy*, is a good approximation to the real problem of interest, because in two dimensions it gives a good measure on how well a sample pattern, when applied to the area of one or more pixels, captures small details in the picture. Results by [Ko] and [TW] which show that the error in evaluating an integral (under bounded variation conditions) is proportional to the discrepancy of the sampling point set, further justify its use.

A lot of work has to be done to produce point sets with low discrepancy in this model (see [BC]), and in fact almost optimal sequences (Hammersley points, [Hal]) are known. Algorithms that compute the exact or approximate maximum numerical discrepancy of point sets are useful however to compare point sets, to find patterns with very low discrepancy and to produce point sets when other properties (for example a random distribution) are also important.

### 1.3 Bichromatic discrepancy and $\epsilon$-approximations

Many new results in computational geometry have used probabilistic techniques and algorithms. [HW] gave a very useful abstract framework for their development and analysis, the concept of set systems with bounded (finite) VC-dimension.

A set system is a pair $(S, \mathcal{R})$, where $S$ is a set of points, and $\mathcal{R}$ is a set of subsets (we will call them ranges) of $S$. For a set $Y \subset S$, we call the set system $(Y, \{U \mid (U = R \cap Y) \wedge (R \in \mathcal{R})\})$ the subspace *induced* by $Y$. We say that $Y$ is *shattered* by $\mathcal{R}$ if, in the subspace induced by $Y$, every possible subset of $Y$ is a range (in other words, if $|\{U \mid (U = R \cap Y) \wedge (R \in \mathcal{R})\}| = 2^{|Y|}$). The Vapnik-Chervonenkis dimension, or VC-dimension of the set system $(S, \mathcal{R})$ is the maximum size of a shattered subset of $S$ ([VC]).

A subset $A \subset S$ is an $\epsilon$-approximation for the set system $(S, \mathcal{R})$ if $||A \cap R|/|A| - |R|/|S|| \leq \epsilon$, for all $R \in \mathcal{R}$. A subset $N \subset S$ is an $\epsilon$-net for $(S, \mathcal{R})$ if $S \cap R \neq \emptyset$ for any $R \in \mathcal{R}$ with $|R|/|S| > \epsilon$. Remarkably, if $(S, \mathcal{R})$ has finite VC-dimension, it has $\epsilon$-approximations and $\epsilon$-nets with sizes independent from $|S|$, as the following result shows:

Theorem ([HW]): Let $d$ be fixed and let $(S, \mathcal{R})$ be a set system of VC-dimension $d$. Then for every $\epsilon > 0$, there exists an $1/\epsilon$-net for $(S, \mathcal{R})$ of size $O((\epsilon)\log(\epsilon))$.

Set systems with finite VC-dimension occur naturally in geometry and in learning theory. Let's consider for example the set system we primarily examine in this paper. The set $S$ is a finite set of two dimensional points ($S \subset [0,1]^2$), and for $A \subset S$, $A$ is a range ($A \in \mathcal{R}$) if and only if there exists an axis aligned rectangle that contains exactly the points in $A$. It is easy to see that the VC-dimension of $(S, \mathcal{R})$ is at least 4. If we pick a subset

$Y$ with 4 points arranged in a diamond, $Y$ is shattered by $\mathcal{R}$. Suppose $Y$ has 5 or more points. Take the smallest enclosing axis aligned rectangle and, for each of its edges take exactly one point that intersects it. This subset has at most 4 points and cannot be in the subspace induced by $Y$ because any axis aligned rectangle that contains this set of points contains all points of $Y$. So the VC-dimension of our set system cannot be more than 4 (or $2d$ for $d$-dimensional points).

Because of their properties, $\epsilon$-nets have been used in many geometric algorithms and applications. Their use is also instrumental in the derandomization of divide and conquer algorithms ([BCM]). The derandomization of random algorithms is a general problem that allows us a better understanding of the importance of randomness as a computational resource. It also produces algorithms with guaranteed worst case performance. The only known way to deterministically and efficiently compute $\epsilon$-nets is via $\epsilon$-approximations ([MWW]). Recently [MWW] gave a strong connection between the discrepancy of a set system, and the deterministic construction of $\epsilon$-approximations and $\epsilon$-nets.

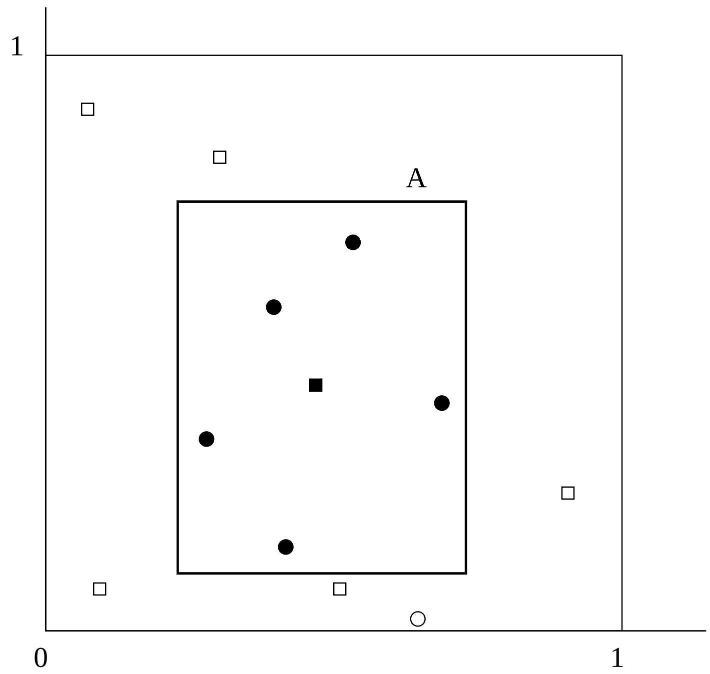



Figure 3: The rectangle $A$ maximizes the bichromatic discrepancy for the point set shown (circles are mapped to +1, and squares are mapped to -1, the filled points are the ones in the interior).

Let us define the bichromatic discrepancy first. Let $(S, \mathcal{R})$ be a set system and let $\chi : S \rightarrow \Re$ be a mapping. For a set $R \in \mathcal{R}$, let $\Delta(R) = \sum_{x \in R} \chi(x)$ be the *bichromatic* *discrepancy* of $R$. We define the *maximum bichromatic discrepancy* of $\chi$ on $(S, \mathcal{R})$ by:

$$\Delta(S, \chi, \mathcal{R}) = \max_{R \in \mathcal{R}} |\Delta(R)|$$

Usually $\chi$ is a mapping to $\{-1, +1\}$, and is called a *coloring* of $S$ (Fig 3). This is where the name bichromatic comes from.

The *discrepancy* of $(S, \mathcal{R})$ is:

$$\mathrm{disc}(S, \mathcal{R}) = \min_{\chi : X \to \{-1,+1\}} \Delta(S, \chi, \mathcal{R})$$

[MWW] have shown that a set system with discrepancy $\delta$ has a $2\delta/|S|$-approximation $A$, with $|A| = \lceil |S|/2 \rceil$. Obviously $A$ is also a $2\delta/|S|$-net. Furthermore, this approximation $A$ can be constructed from the coloring that has discrepancy $\delta$. Therefore an algorithm that computes the maximum bichromatic discrepancy of a coloring $\chi$ also gives an accurate bound on the quality of the resulting $\epsilon$-approximation.

# 2. Computing the maximum bichromatic discrepancy

### 2.1 The problem

In this section we concentrate on the problem of computing the maximum bichromatic discrepancy:

$$\Delta(S, \chi, \mathcal{R}) = \max_{R \in \mathcal{R}} |\Delta(R)| = \max_{R \in \mathcal{R}} |\sum_{x \in R} \chi(x)|$$

for a given set $S$, a given weight function $\chi : S \to \Re$ and the set $\mathcal{R}$ of axis aligned rectangles. When $\chi$ is a coloring, this is simply the rectangle that maximizes the difference between the number of the red and the blue points it contains.

To simplify the definition of $\Delta(S, \chi, \mathcal{R})$ (and avoid the absolute value), we define two auxiliary functions:

$$\Delta_{max}(S, \chi, \mathcal{R}) = \max_{R \in \mathcal{R}} \Delta(R)$$

and

$$\Delta_{min}(S, \chi, \mathcal{R}) = \min_{R \in \mathcal{R}} \Delta(R)$$

In the following, unless we mention otherwise, we drop the $\mathcal{R}$ from the notation with the assumption that we are considering the family of axis oriented rectangles.

*Theorem 1:* Solving the minimizing disagreement problem for rectangle hypotheses for a training sequence $T$ is equivalent to the problem of finding the rectangle that maximizes

$\Delta$ for the associated set of points $S$ and the mapping $\chi : S \to \{-1, +1\}$ which is defined by the labels in $T$.

*Proof:* Suppose we are given a sequence $T$ of labeled examples. Let $S$ be the set of points we get if we remove the labels from the examples in $T$. We use the labels to construct the coloring $\chi$. An example labeled 1, is mapped to +1 (a red point), and an example labeled 0 is mapped to -1 (blue point). The rectangle $R$ that maximizes $\Delta(R) = \sum_{x \in R} \chi(x)$ maximizes the red points inside $R$ minus the blue points inside $R$, so it minimizes the blue points inside $R$ minus the red points inside $R$. But the number of red points inside $R$ is equal to the total number of red points (a constant) minus the red points outside $R$, so we have that the rectangle that maximizes $\Delta(R) = \sum_{x \in R} \chi(x)$ minimizes the number of of blue points inside $R$ plus the number of red points outside $R$. But this is equal to $\text{Error}_T(R)$. The other direction is similar. □

The bichromatic discrepancy provides an approximation for the numerical discrepancy model with the use of a sufficiently fine grid of blue points. It is the discrete equivalent of the numerical discrepancy, since we measure how well a point set approximates another. In fact, some upper bound results for the numerical discrepancy given in [BC] were obtained via theorems for the bichromatic discrepancy. So, as we will see in section 3, an algorithm for the bichromatic discrepancy is a good foundation to solve the numerical discrepancy case as well.

We begin our investigation with a look at the one-dimensional case, to build intuition and set ideas.

### 2.2 The 1-d case

In one dimension, axis oriented rectangles become intervals that have both endpoints in the unit interval, and the set $X$ is a set of $n$ distinct numbers between 0 and 1. Our goal here is to develop an algorithm that finds the interval that maximizes $\Delta_{max}(S, \chi)$.

We consider the static case first, where both the point set $S$ and the mapping $\chi$ are fixed and known beforehand. We assume that the point set is sorted. If not, sorting is a preprocessing step of the algorithm.

First we show that we have only a finite number of intervals to consider.

*Lemma 1:* $\Delta_{max}(S, \chi)$ is maximized by an interval whose endpoints are in $S$.

*Proof:* If an interval has a free endpoint, we can move it until it meets the closest point inside without changing the discrepancy of the interval. With the exception of the trivial case of no red points, the interval that maximizes $\Delta_{max}(S, \chi)$ cannot be empty. □

¿From this lemma we know that there are only $O(n^2)$ different intervals to consider. The most naive algorithm would compute the discrepancy for all of them to find the maximum bichromatic discrepancy in $O(n^3)$ time. Using an incremental algorithm we can compute all the disrepancies in $O(n^2)$ time. Below we show that, with a splitting strategy, further improvement is possible. First we show another simple lemma.

*Lemma 2:* If $0 \le l \le m \le r \le 1$ and $m \notin S$, then $\Delta([l,r]) = \Delta([l,m]) + \Delta([m,r])$.

*Proof:* Since $m$ is not a point, each point in $[l,r]$ has to be in exactly one of $[l,m]$ and $[m,r]$. So $\sum_{x\in(S\cap[l,r])}\chi(x) = \sum_{x\in(S\cap[l,m])}\chi(x) + \sum_{x\in(S\cap[m,r])}\chi(x)$ and the lemma follows. $\square$

The following lemma shows that we can find the endpoints of the interval that maximizes $\Delta$ over all intervals that contain $m$ locally in $[0,m]$ (for the left endpoint) and in $[m,1]$ (for the right endpoint).

*Lemma 3:* Let $A = [l,r]$ be an interval, and take a point $m \in A$ with $m \notin S$. Assume that $[x_i, x_j]$ maximizes $\Delta$ over all subintervals of A that contain the point $m$. Then $[x_i, m]$ maximizes $\Delta$ over all subintervals of A that have $m$ as their right endpoint.

*Proof:* Suppose that there is an interval $[y,m]$ (with $y \in A$) such that $\Delta([y,m]) > \Delta([x_i,m])$. ¿From the hypothesis we know that $\Delta([y,x_j]) \le \Delta([x_i,x_j])$. ¿From lemma 3 we have $\Delta([x_i,x_j]) = \Delta([x_i,m]) + \Delta([m,x_j])$ and $\Delta([y,x_j]) = \Delta([y,m]) + \Delta([m,x_j])$.

¿From the equalities it follows that $\Delta([y,m]) \le \Delta([x_i,m])$ which is a contradiction.

Similarly we can show that $[m, x_j]$ must maximize $\Delta$ over all subintervals whose left endpoint is $m$. $\square$

The next lemma shows that we can find the interval that maximizes $\Delta$ over all intervals with a given point $m$ as their right (or left) boundary if we split $[l,r]$ into two parts, and solve two subproblems.

*Lemma 4:* Assume that we split into interval $A = [l,r]$ into $A_l = [l,m]$ and $A_r = [m,r]$ so that $m \notin S$. Also assume that $[x_i, r]$ maximizes $\Delta$ over all subintervals of $A_r$ that have $r$ as their right endpoint, and that $[x_j, m]$ maximizes $\Delta$ over all subintervals of $A_l$ that have $m$ as their right endpoint. Then either $[x_i, r]$ or $[x_j, r]$ maximizes $\Delta$ over all subintervals of $A$ that have $r$ as their right endpoint.

*Proof:* Suppose there is a subinterval $[y,r]$ of $A$ such that $\Delta([y,r]) > \Delta([x_i,r])$ and $\Delta([y,r]) > \Delta([x_j,r])$. Obviously $y$ cannot be in $A_r$, since in that case $[y,r]$ would be contained in $A_r$ and $\Delta([y,r])$ would be at most $\Delta([x_i,r])$. So it has to be in $A_l$. If we use lemma 3, we get that $\Delta([y,m]) > \Delta([x_j,m])$, a contradiction. $\square$

This suggests that we can compute the maximum discrepancy while we are building a tree of intervals in a hierarchical fashion. We partition $[0,1]$ into intervals, that we call regions. For each region $A$ we compute three maxima. $A_{max}$ is the interval that maximizes $\Delta$ over all subintervals of $A$. By definition, $[0,1]_{max}$ is the maximum we are looking for. $A_{left}$ is the interval that maximizes $\Delta$ over all subintervals of $A$ that share $A$'s left endpoint. And $A_{right}$ is the interval that maximizes $\Delta$ over all subintervals of $A$ that share $A$'s right endpoint.

*Lemma 5:* Assume $L$ and $R$ are adjacent non-overlapping regions that are merged to form $LR$. Then $LR_{max}$, $LR_{left}$ and $LR_{right}$ can be computed in constant time from $R_{max}$, $R_{left}$, $R_{right}$, $L_{max}$, $L_{left}$ and $L_{right}$.

*Proof:* ¿From lemma 3 we know that $LR_{max}$ is either $L_{max}$ or $R_{max}$ or $L_{right} \cup R_{left}$, so we can compute it in constant time. ¿From lemma 4 we know that $LR_{left}$ is either $L_{left}$ or $L \cup R_{left}$, and $LR_{right}$ is either $R_{right}$ or $L_{right} \cup R$. It follows that we can perform a merge operation in constant time. □

The algorithm to compute $[0,1]_{max}$ takes $\log n$ steps, halving the number of regions in each one.

*Algorithm 1:*

1. Partition $[0,1]$ into $\lceil n/2 \rceil$ non-overlapping regions, each properly containing at most 2 points of $S$, so that the boundaries of the regions are not points in $S$. We find the three maxima for each of these regions.
2. Merge consecutive even and odd regions to produce half the number of new regions. In each merge operation find the three new maxima for the new region.
3. If only $[0,1]$ is left, output $[0,1]_{max}$, otherwise go back to step 2.

*Theorem 2:* We can compute the maximum bichromatic discrepancy of a sorted one-dimensional point set $S$ for any given coloring in linear time and space.

*Proof:* Algorithm 1 computes $[0,1]_{max}$ in linear time and space. It's correctness follows from lemmata 3 and 4. The running time of the first step is $\Theta(n)$ since there are $O(n)$ intervals, each containing a constant number of points of $S$. Each merge operation takes $O(1)$ time, and, since we halve the number of intervals in each iteration, there are a total of $\Theta(n)$ merges. In the same way we can find $[0,1]_{max}$ (the interval that minimizes $\Delta$), and so compute the maximum bichromatic discrepancy. □

A corollary of Theorem 1 and Theorem 2 is that we can solve the minimizing disagreement problem for arbitrary interval hypotheses in linear time.

### 2.3. The dynamic 1-d case

Algorithm 1 can be easily modified to be dynamic with respect to insertions and deletions of points.

We explicitly construct a binary tree by subdividing the intervals, with the leaves corresponding to intervals that contain a constant number of points of $S$. Each node corresponds to a region, and in each node we keep the largest bichromatic discrepancy recorded in this region and also the other two maxima, information which requires constant size per node. The total size is thus linear.

To delete a point, we follow the path down to the leaf that contains the deleted point, and then retrace the path to the root, recording the new values in each visited note. The previous lemmata show that this can be done in constant time per node. Insertion is similar.

If we use a balanced binary tree (for example a red-black tree) each update costs $O(\log n)$ time (where $n$ is the current number of points), plus the time required to keep the tree balanced. Each tree rotate is a local operation however and can be performed in constant time, so the total time of each update is $O(\log n)$.

*Theorem 3:* We can recompute the maximum bichromatic discrepancy of a colored one-dimensional point set $S$ after an update in $O(\log n)$ time and in linear space.

*Proof:* From the discussion above. □

### 2.4. The 2-d case

In two dimensions the input set $S$ is a set of $n$ points in the unit square $[0,1]^2$ and a mapping $\chi : S \rightarrow \Re$. To make the presentation simpler we assume that all $x$ and $y$ coordinates are distinct. This is not an important restriction however.

The 2-d case builds on the one dimensional algorithm. To solve the problem we combine a sweeping technique along the $y$ axis with an application of the dynamic algorithm of §2.3 on the $x$ axis. Let's begin with some simple but important lemmata.

The following lemma (like Lemma 2) shows that we have to search among a finite number of rectangles.

*Lemma 6:* There exists a rectangle that maximizes $\Delta$ such that all its edges pass through a point of $S$.

*Proof:* Let $I_{max} = \{(x_{min}, y_{min}), (x_{max}, y_{min}), (x_{max}, y_{max}), (x_{min}, y_{max})\}$, be a rectangle that maximizes $\Delta$. Suppose that it has an edge $e$ that does not pass through a point. We can then freely move $e$ along the axis perpendicular to it until it meets the closest point inside, without changing the discrepancy of the rectangle. $\Box$

It follows that there are at most $O(n^4)$ candidates to consider in the computation of the maximum discrepancy, and, with a smart data structure ([Me], [PS]), we can search through this range in $O(n^4 \log n)$ time. We use a plane sweep technique to improve upon it.

Let $Y_{coord}$ be the set of all $y$ coordinates of points in $S$. There are $O(|Y_{coord}|^2) = O(n^2)$ different pairs of values for the $y$ coordinates of $I_{max}$. For each such pair we are going to find the rectangle that maximizes $\Delta$ among all rectangles with these $y$ coordinates. The following lemma shows that for a given pair of points the problem can be reduced to a one-dimensional problem.

*Lemma 7:* The problem of finding the rectangle that maximizes $\Delta$, among all the rectangles with given $y$ coordinates, is equivalent to the problem of finding the interval that maximizes $\Delta$ for a set of one-dimensional points with similar cardinality.

*Proof:* Assume that the fixed $y$-coordinates are $y_b, y_t$, with $y_b < y_t$ (Fig. 4).

Let $I_m = \{(x_i, y_b), (x_j, y_b), (x_j, y_t), (x_i, y_t)\}$ be the rectangle that maximizes $\Delta$ for all $x_i$, $x_j$, $x_i < x_j$. By the definition, $\Delta(I_{max}) = \sum_{x \in (S \cap I_{max})} \chi(x)$. But $S \cap I_{max} = \{(x_k, y_k) \mid x_k \in [x_i, x_j] \text{ and } y_k \in [y_b, y_t]\}$. Obviously only the points with $y$ coordinates between $y_b$ and $y_t$ have to be considered in the computation of $I_m$.

Now consider the set $S_{[y_b, y_t]} = \{x_k \mid (x_k, y_k) \in S \text{ and } y_k \in [y_b, y_t]\}$. Let $A_{max} = [x_l, x_r]$ be the interval that maximizes $\Delta$ for the set $S_{[y_b, y_t]}$ and the restriction of the original $\chi$ to the new set. Then take the rectangle $I'$ that is defined by $A_{max}$ and $[y_b, y_t]$ (that is, $I' = \{(x_l, y_b), (x_r, y_b), (x_r, y_t), (x_l, y_t)\}$). Since $I_m$ is maximum, we have that $\Delta(I_m) \geq \Delta(I')$. But from the construction of $S_{[y_b, y_t]}$ we have that $\Delta([x_i, x_j]) = \Delta(I_m)$ and $\Delta(I') = \Delta(A_{max})$. Finally we have that $A_{max} = [x_l, x_r]$ is a maximum and so $\Delta([x_i, x_j]) \leq \Delta([x_l, x_r])$. It follows that $\Delta([x_i, x_j]) = \Delta([x_l, x_r])$. $\Box$

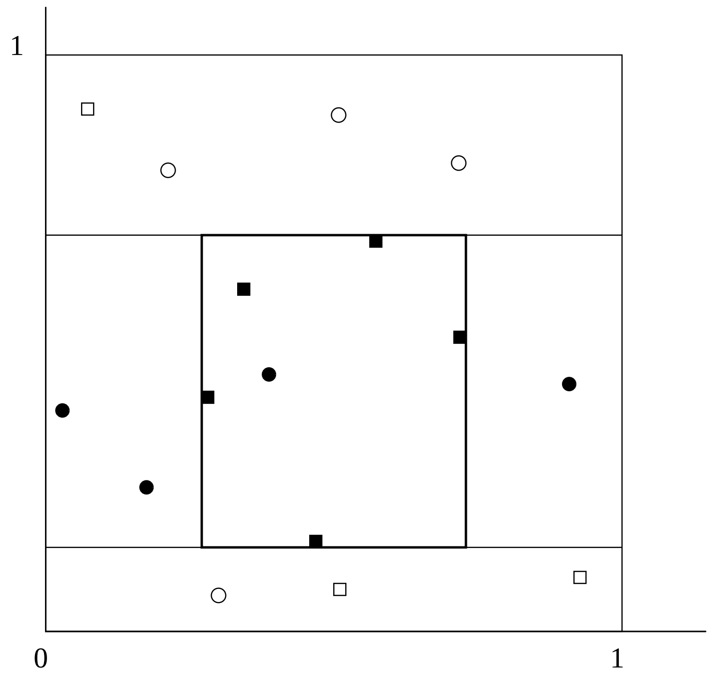



Figure 4.a: The $y$-coordinates of the rectangle are fixed.

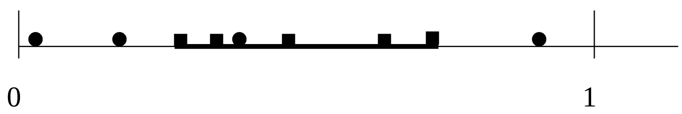



Figure 4.b: The equivalent one-dimensional problem.

Lemma 7 shows that we can easily compute the maximum rectangle when the $y$ coordinates are fixed by a pair of points. There are a total of $O(n^2)$ pairs, and we can search through all of them using the dynamic linear algorithm of §2.3.

The following algorithm finds the rectangle that maximizes $\Delta$ for an input set $S \subset [0,1]^2$ and a coloring $\chi$.

*Algorithm 2:*

1. Compute $Y_{coord}$ and sort it.

2. For each element $y_i$ of $Y_{coord}$ do:

(a) Find the set $S_i = \{x_j \mid (x_j, y_j) \in S \text{ and } y_i \leq y_j\}$ and sort it.

(b) While $S_i$ is not empty, do:

i. Remove from $S_i$ the point $x_j$ with the smallest $x$ coordinate, and insert it into the region.

ii. Use the dynamic linear algorithm to find the rectangle that maximizes $\Delta$ over all rectangles that have $y$ coordinates between $y_i$ and $y_j$.

iii. If the new value is larger than the largest seen so far, record the rectangle.

3. Return the best rectangle found.

*Theorem 4:* We can compute the maximum bichromatic discrepancy of a set $S$ ($|S| = n$) with an arbitrary coloring $\chi$ in $O(n^2 \log n)$ time and $O(n)$ space.

*Proof:* The outlined algorithm finds the maximum of $\Delta$. Its correctness follows from lemmata 6 and 7. The algorithm considers all pairs of points. For each one it finds the rectangle that maximizes $\Delta$ while it contains the lower point. In the inner loop of the algorithm we use the dynamic algorithm of §2.3 for the left and the right region separately.

For the running time, we analyze each step. The first step takes $O(n \log n)$ time and then the outer loop is executed $O(n)$ times. The first step of the outer loop also takes $O(n \log n)$ time. The second step takes linear time. The inner loop is executed $O(n)$ times. The dynamic linear algorithm takes $O(\log n)$ time for insertions and $O(\log n)$ time for queries. In the second step we do one insertion and two queries, so the second step takes $O(\log n)$ time. Finally the third step takes constant time.

The total time is $O(n^2 \log n)$. At each point we have to maintain the data structure for the dynamic linear algorithm. The used space is therefore linear. $\Box$

A corollary of Theorem 1 and Theorem 4 is that we can solve the minimizing disagreement problem for rectangular hypotheses in $O(n^2 \log n)$ time and linear space.

# 3. Computing the maximum numerical discrepancy

Here we extend the results of section 2 to the numerical discrepancy problem. Again we consider the low dimension cases separately.

### 3.1. The static 1-D case

The input is a set $S$ of $n$ distinct points with coordinates $0 \leq x_1 < \ldots < x_n \leq 1$. The family $\mathcal{R}$ is the set of intervals in $[0,1]$. Following the definition of §1.2, the numerical discrepancy of a given interval is the difference between its length and the ratio of the points it contains, over $|S|$.

The problem is to find the interval that maximizes the numerical discrepancy.

We give some convenient notation first.

We define the function $Count : \mathcal{R} \rightarrow \mathcal{N}$, where $Count([l,r])$ is the number of points inside the interval $[l,r]$, endpoints inclusive. We will also apply $Count$ to open intervals, and then $Count((l,r))$ gives the number of points in the open interval $(l,r)$. For an interval $A$, if $x_i$ is the point in $A$ that is the closest to its left border, and if $x_j$ is the point in $A$ that is the closest to its right border, then we have $Count(A) = (j-i+1)$. Given this, the numerical discrepancy $\mathcal{D}$ of some interval $A = [l,r]$ can be expressed as

$$\mathcal{D}(A) = |(r-l) - Count(A)/n|$$

This definition of the discrepancy gives a different expression depending on whether $(r-l)$ or $Count(A)/n$ is greater, and to simplify matters we define the following two functions that operate on intervals. The first function, $\mathcal{D}_s$, computes the discrepancy of an interval when the ratio of the points in it is smaller that the length of the interval. In this case, if the endpoints of the interval are also in $S$, it is obvious that the discrepancy increases if we consider the equivalent open interval, and so we have:

$$\mathcal{D}_s([l,r]) = (r-l) - Count((l,r))/n$$

The second function, $\mathcal{D}_l$, computes the discrepancy of an interval when the ratio of the points in the interval is larger than its length. In this case all points, endpoints included, are counted, so:

$$\mathcal{D}_l([l,r]) = Count([l,r])/n - (r-l)$$

Clearly, the interval that maximizes the discrepancy must also maximize one of $\mathcal{D}_s$ or $\mathcal{D}_l$.

Again it is easy to show that we only have to consider a finite number of intervals.

*Lemma 8:* The numerical discrepancy of an one-dimensional point set $S$ can be maximized only by a interval with endpoints in $S \cup \{0,1\}$ .

*Proof:* If a interval has a free endpoint, we can both increase and decrease its length without changing its intersection with $S$, and one of the two operations has to increase the discrepancy. □

The static one dimensional case is exactly equivalent to the bichromatic discrepancy. We can follow the same approach and develop a linear time algorithm that computes the maximum numerical discrepancy of an one dimensional point set.

We only state the main lemmata.

*Lemma 9:* If $0 \leq l \leq m \leq r \leq 1$ and $m \notin S$, then $\mathcal{D}_l([l,r]) = \mathcal{D}_l([l,m]) + \mathcal{D}_l([m,r])$.

*Proof:* Since $m$ is not a point, each point in $[l,r]$ has to be in exactly one of $[m,r]$ and $[l,m]$. Then $Count([l,r]) = Count([l,m]) + Count([m,r])$. Also, $(r-l) = (m-l) + (r-m)$. ¿From these two equalities we have that: $Count([l,r])/n - (r-l) = Count([l,m])/n - (m-l) + Count([m,r])/n - (r-m) \Rightarrow \mathcal{D}_l([l,r]) = \mathcal{D}_l([l,m]) + \mathcal{D}_l([m,r])$. □

*Lemma 10:* Let $m \in [l,r]$, $m \notin S$, and assume that $[x_i, x_j]$ maximizes $\mathcal{D}_l$ over all subintervals of $[l,r]$ that contain $m$. Then $[x_i, m]$ maximizes $\mathcal{D}_l$ over all such subintervals that have $m$ as their right endpoint, and $[m, x_j]$ maximizes $\mathcal{D}_l$ over all such subintervals hat have $m$ as their left endpoint.

*Proof:* We only prove the lemma for the $[x_i, m]$, the other case is similar. Suppose that there is such a subinterval $[y,m]$ that $\mathcal{D}_l([y,m]) > \mathcal{D}_l([x_i,m])$. ¿From the hypothesis we know that $[x_i, x_j]$ is maximum, therefore $\mathcal{D}_l([y,x_j]) \leq \mathcal{D}_l([x_i,x_j])$. ¿From lemma 9 we have $\mathcal{D}_l([x_i,x_j]) = \mathcal{D}_l([x_i,m]) + \mathcal{D}_l([m,x_j])$ and $\mathcal{D}_l([y,x_j]) = \mathcal{D}_l([y,m]) + \mathcal{D}_l([m,x_j])$. ¿From the last two equalities it follows that $\mathcal{D}_l([y,m]) \leq \mathcal{D}_l([x_i,m])$ which is a contradiction. □

*Lemma 11:* Assume that we split $[l,r]$ into $[l,m]$ and $[m,r]$, (with $m \notin S$). Also assume that $[x_i, r]$ maximizes $\mathcal{D}_l$ over all intervals in $[m,r]$ that have $r$ as their right endpoint, and that $[x_j, m]$ maximizes $\mathcal{D}_l$ over all intervals that have $m$ as their right endpoint. Then either $[x_i, r]$ or $[x_j, r]$ maximizes $\mathcal{D}_l$ over all intervals that have $r$ as their right endpoint.

*Proof:* Suppose there is an interval $[y,r]$ such that $\mathcal{D}_l([y,r]) > max(\mathcal{D}_l([x_i,r]), \mathcal{D}_l([x_j,r]))$. Obviously $y$ cannot be in $[m,r]$, since in that case $[y,r]$ would be contained in $[m,r]$ and by the assumption that $[x_i,r]$ is maximum we have $\mathcal{D}_l([y,r]) \leq \mathcal{D}_l([x_k,r])$. So it have to be in $[l,m]$. But then, if we use lemma 9, we get that $\mathcal{D}_l([y,m]) > \mathcal{D}_l([x_j,m])$, a contradiction. □

*Theorem 5:* We can compute the maximum numerical discrepancy of a sorted point set $S$ on a line in linear time and space.

*Proof:* Algorithm 1, suitably modified, finds the interval that maximizes $\mathcal{D}_l$. Its correctness follows from lemmata 10 and 11. The running time of the first step is $\Theta(n)$ since there are $O(n)$ intervals of constant size each. Each merge operation takes $O(1)$ time, and, since we halve the number of intervals in each iteration, there are a total of $\Theta(n)$ merges. Similarly we can maximize $\mathcal{D}_s$, and compute the maximum numerical discrepancy. □

### 3.2 The dynamic 1-D case

The dynamic case is quite more difficult because when we insert or delete a point the cardinality of $S$ ($n$) changes. As a consequence, the function $\mathcal{D}_l$ changes and we have to compute new maxima for every region. The following example shows that the insertion of a new point in a region does change the maximal points of other regions.

Let $S = \{0.1, 0.24, 0.4, 0.6, 0.7, 0.8\}$, and assume we have two regions, $A = [0, 0.5]$, and $B = [0.5, 1]$. We can see that $A_{max} = [0.1, 0.4]$, with $\mathcal{D}_l([0.1, 0, 4]) = 3/6 - 0.3 = 0.2$ and $\mathcal{D}_l([0.1, 0.24]) = 2/6 - 0.14 = 0.193$.

But if we insert a point 0.9 in $R_2$, we have: $\mathcal{D}_l([0.1, 0.4]) = 3/7 - 0.3 = 0.129$ and $\mathcal{D}_l([0.1, 0.24]) = 2/7 - 0.14 = 0.146$ Now $A_{max}$ is $[0.1, 0.24]$ (Fig. 5).

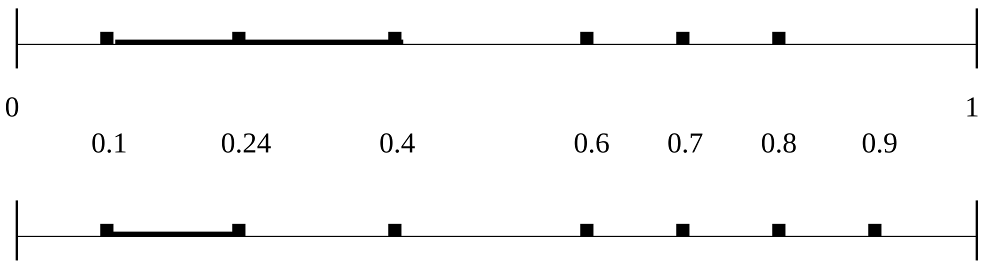



Figure 5: The insertion of a point can change many maximal points.

An approach based on separate regions is still valuable however. When a point is inserted (deleted) only one region is directly affected. For the rest of the section we will assume that $n$ is the maximum cardinality of $S$, for the entire sequence of updates.

First the interval [0, 1] is divided in $r$ regions, each containing $O(n/r)$ points. We do not allow points of $S$ to become endpoints of regions.

Consider such an interval $A = [a_l, a_r]$, and assume that the points that lie inside $A$ are $x_{min}, \ldots, x_{max}$. Recall (§ 2.2) that $A_{max}$ is the subinterval that maximizes $\mathcal{D}_l$, $A_{left}$ is the subinterval that maximizes $\mathcal{D}_l$ and shares $A$'s left endpoint, and $A_{right}$ is the subinterval that maximizes $\mathcal{D}_l$ and shares $A$'s right endpoint. In order to compute $A_{max}$, $A_{left}$ and $A_{right}$, we use the techniques of the following theorem.

Theorem ([DE]): We can insert or delete points from a set $S \subset [0,1]$, and recompute the halfspace discrepancy after each update, in time $O(\log^2 n)$ per update, and $O(n \log n)$ space.

This theorem gives an algorithm to compute $A_{left}$ and $A_{right}$. To compute $A_{left}$ we have to find the point $x_i$ that maximizes $\mathcal{D}_l([x_{min}, x_i]) = (i - min + 1)/n - (x_i - a_l)$ over all points in $A$. This function is a linear function of $(x_i, i)$. So we construct the convex hull of the $O(n/r)$ points $\{(x_i, i) | x_i \in A\}$. We can find the point that maximizes $(i - min + 1)/n - (x_i - a_l)$ with a binary search on the convex hull (Fig. 8.c and 8.d). In same way, we find the point $x_j$ that maximizes $(max - j + 1)/n - (a_r - x_j)$, and this point is the left endpoint of $A_{right}$.

The computation of $A_{max}$ is a little more complicated. The endpoints of $A_{max}$ must be points in $S$, so in fact we want to find that pair of points that maximizes the function $\mathcal{D}_l([x_i, x_j]) = (j - i + 1)/n - (x_j - x_i)$, over all pairs of points in $A$.

An alternative way is to view $(j - i + 1)/n - (x_j - x_i)$ as a linear function of the two-dimensional points $(l(i,j), q(i,j))$, with $l(i,j) = (x_j - x_i)$, and $q(i,j) = (j - i + 1)$. In other words, the first coordinate is the length of the interval $[x_i, x_j]$, and the second coordinate, the difference of the ranks, is the the number of points of $S$ that lie in the same interval. To compute $A_{max}$ we construct the convex hull of the $O((n/r)^2)$ points $\{(l(i,j), q(i,j)) \mid x_j \in A, x_i \in A, x_i < x_j\}$, and perform a binary search to find the point that maximizes the function $(j - i + 1)/n - (x_j - x_i)$. The query time is then $O(\log(n/r))$.

So for each region $A$, we keep two convex hull structures using $O((n/r)^2 \log(n/r))$ space. From these we extract four points of $S$ that define $A_{max}$, $A_{left}$ and $A_{right}$. Let $T$ be the set of such points for all such regions. Of course with each update $T$ changes. The following lemma shows that we can find the new $T$ in $O(r \log(n/r) + (n/r)^2 \log^2(n/r))$ time.

*Lemma 12:* The set $T$ can be recomputed in $O(r \log(n/r) + (n/r)^2 \log^2(n/r))$ time after each update.

*Proof:* With an update only one region is directly affected. The convex hulls are maintained using the dynamic algorithm of [OvL]. The update time is $O(\log^2(n/r))$. For an insertion of a new point $x_k$ of $S$ in the region $A$, we have to make a total of $O(n/r)$ insertions to the convex hull structures because for each $x_i$ in $A$, a new point $(|x_i - x_k|, |i - k|)$ is defined. In addition to that, the second coordinate of up to $(n/r)^2$ convex hull points may change, because the insertion of a new point changes the rank of up to $(n/r)$ points. The cost for the structure update is then $O((n/r)^2 \log^2(n/r))$.

Now we can find $T$ doing three binary searches for each region with a total cost of $O(r \log(n/r) + (n/r)^2 \log^2(n/r))$. $\square$

The following lemma shows that we can find the interval that maximizes $\mathcal{D}_l$ from $T$.

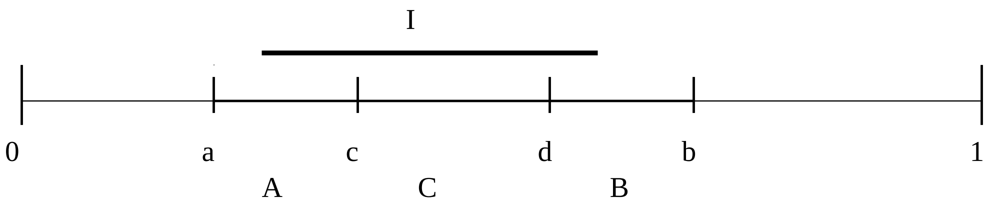



Figure 6: The maximum interval $I$ intersects regions $A$ and $B$, and contains $C$.

*Lemma 13:* An interval in $[0,1]$ that maximizes $\mathcal{D}_l$ must have as endpoints points of $T$.

*Proof:* If this maximum interval $I$ lies in one region, then its endpoints are in $S$. So suppose its left and right endpoint are in regions $A$ and $B$ respectively (Fig. 6).

Let $AB = A \cup C \cup B$ be the region that starts at $a$ and stops at $b$ ($C = AB \setminus A \setminus B$). Then $I = (I \cap A) \cup C \cup (I \cap B)$, and $\mathcal{D}_l(I) = \mathcal{D}_l(I \cap A) + \mathcal{D}_l(C) + \mathcal{D}_l(I \cap B)$. Lemma 11 shows that $\mathcal{D}_l(I \cap A)$ is maximized when $(I \cap A) = A_{right}$ and similarly, $\mathcal{D}_l(I \cap B)$ is maximized when $(I \cap B) = B_{left}$. By the construction of $T$, in includes the endpoints of $A_{right}$ and $B_{left}$ and the lemma follows. $\Box$

So we can then use the algorithm for the static case and find the maximum in $O(|T|) = O(r)$ time.

*Theorem 6:* We can insert or delete points from a point set $S$ on a line, and recompute the maximum numerical discrepancy after each update in time $O(n^{2/3}\log^2 n)$ per update, and space $O(n^{4/3}\log n)$.

*Proof :* We can compute $T$ in $O(r\ \log(n/r)) + O(n^2\ \log^2(n/r)/r^2)$ time (from lemma 12) after each update. Once the new maxima are found, we can use the linear time algorithm for sorted inputs, and find the updated maximum of $\mathcal{D}_l$ for $S$ in an additional $O(r)$ time. The same procedure is applied to $\mathcal{D}_l$. we find the maximum numerical discrepancy from the maximum of the two maxima. If $r = n^{2/3}$, the total update cost is $O(n^{2/3}\ \log^2 n)$.

This cost does not include the time required to split a region if after an insertion it contains more that $2n^{1/3}$ points, or to combine two consecutive regions to one if, after an insertion they both contain less than $n^{1/3}$ points. Each such operation costs $O(n^{2/3}\ \log^2 n)$ because it requires complete rebuilt of the convex hull data structures. So the maintenance of the regions does not raise the asymptotic complexity of an update.

The space requirement for each region is $O((n/r)^2\log(n/r)) = O(n^{2/3}\log n)$, so the space used is $O(n^{4/3}\log n)$. $\Box$

Finally we note that the dynamic algorithm also works when the weight of each point is not uniformly 1 but is assigned by a weight function of the form $\mathcal{R} : S \rightarrow \Re_+$. For this we have to change the index of every point. For the $i$-th point, instead of using $i$ as its index, we use the sum of its weight and the weights of the $i-1$ points on its left.

### 3.3. The 2-D case

In two dimensions the input set $S$ is a set of $n$ points in the unit square $[0,1]^2$. We assume that the points are distinct and no pair of them has the same $y$ coordinate. This assumption makes the derivation easier but is not essential for the proof of correctness and the running time analysis of the algorithm. As in §3.2, we give some definitions first. The function $In : \mathcal{R} \rightarrow \mathcal{N}$ gives the number of points inside a rectangle, and the function $Area : \mathcal{R} \rightarrow \Re_+$ gives the area of a rectangle. The numerical discrepancy $\mathcal{D}$ of a rectangle $R \in [0,1]^2$ is given by $\mathcal{D}(I) = |Area(R) - In(R)/n|$. The objective is to find the rectangle that maximizes $\mathcal{D}$.

Since the definition of $\mathcal{D}$ is somewhat awkward, we also define the following two functions $\mathcal{D}_i(I) = Area(R) - In(R)/n$ and $\mathcal{D}_o(I) = In(R)/n - Area(R)$.

The maximum of $\mathcal{D}$ has to maximize at least one of $\mathcal{D}_o$ and $\mathcal{D}_i$. In this section we give an algorithm that finds the rectangle in $\mathcal{R}$, that maximizes $\mathcal{D}_o$, but the same approach can also find the maximum of $\mathcal{D}_i$.

Our approach to the problem is based to the algorithm we developed for the bichromatic discrepancy. Again it is clear that only $O(n^4)$ rectangles have to be considered, as the following lemma shows.

*Lemma 14:* Let $I_{max} = \{(x_{min}, y_{min}), (x_{max}, y_{min}), (x_{max}, y_{max}), (x_{min}, y_{max})\}$, be the rectangle that maximizes $\mathcal{D}_o$. Then each edge must either pass through a point of $S$.

*Proof:* Suppose that that is not the case for an edge $e$. We can then freely move $e$ along the axis perpendicular to it without changing the number of points that are inside $I_{max}$. One direction of movement increases the area of the rectangle, and the other decreases it, so a movement on the second direction results in a rectangle $I'$ with $\mathcal{D}_o(I') > \mathcal{D}_o(I_{max})$. Furthermore, the direction that decreases the area cannot be blocked even if $e$ lies on the boundary of the unit square. □

A sweeping technique to search through all $O(n^2)$ possible different $y$ coordinates for the best rectangle is again the technique of choice. The following lemma shows that the technique we used in §2.4 follows through.

*Lemma 15:* The problem of finding the rectangle that maximizes $\mathcal{D}_o$ among all the rectangles with given $y$ coordinates is equivalent to the problem of finding the interval that maximizes $\mathcal{D}_l$ for a set of one-dimensional points.

*Proof:* Assume that the fixed $y$-coordinates are $y_b, y_t$, with $y_b < y_t$. We want to find the rectangle $I = \{(x_i, y_b), (x_j, y_b), (x_j, y_t), (x_i, y_t)\}$ that maximizes $\mathcal{D}_o$ for all $i, j, (x_i < x_j)$. By the definition $\mathcal{D}_o(I) = In(I)/n - Area(I)$, and $Area(I) - (x_j - x_i)(y_t - y_b)$. Since $(y_t - y_b) = \Delta y$ is a positive constant, we can equivalently maximize the function

$$\mathcal{D}'_o(I) = In(I)/(n\Delta y) - (x_j - x_i)$$

We also know that $In(I) = \{(x_k, y_k) \mid x_k \in [x_i, x_j] \text{ and } y_k \in [y_b, y_t]\}$, which shows that to find $I_m$ we have to consider only the points with $y$ coordinates between $y_b$ and $y_t$.

Again we take the set $S_{[y_b,y_t]} = \{x_k \mid (x_k, y_k) \in S \text{ and } y_k \in [y_b, y_t]\}$. ¿From the construction of $S_{[y_b,y_t]}$ follows that $Count([x_i, x_j]) = In(\{(x_i, y_b), (x_j, y_b), (x_j, y_t), (x_i, y_t)\})$. Recall that the definition of $\mathcal{D}_l$ is $\mathcal{D}_l([x_i, x_j]) = Count([x_i, x_j])/|S_{[y_b,y_t]}| - (x_j - x_i)$ (§3.1). We can however modify the algorithm we gave in §3.1 to use the constant $n\Delta y$ instead of $|S_{[y_b,y_t]}|$. The modified algorithm maximizes the function $\mathcal{D}_{l,[y_b,y_t]}([x_l, x_r]) = Count([x_l, x_r])/(n\Delta y) - (x_m - x_k)$ . Lets assume that the maximum interval is $[x_l, x_r]$. From it we get the rectangle $I_m = \{(x_l, y_b), (x_r, y_b), (x_l, y_t), (x_l, y_t)\}$ , which maximizes $\mathcal{D}'_o$ and consequently $\mathcal{D}_o$. $\Box$

Lemma 15 shows that we can easily compute the maximum rectangle when the $y$ coordinates are fixed by a pair of points. However the performance of the dynamic one dimensional algorithm does not allow for a fast two dimensional algorithm. The two following lemmata provide an additional idea that leads to a faster algorithm.

*Lemma 16:* Assume that for two given points of $S$, $(x_b, y_b)$ and $(x_t, y_t)$ (with $0 \leq y_b < y_t \leq 1$), the rectangle $I$ maximizes $\mathcal{D}_o$ over all the rectangles with these $y$ coordinates. Then for $I$ to maximize $\mathcal{D}_o$ over all rectangles, it has to include the point $(x_b, y_b)$.

*Proof:* ¿From the initial assumption no two points in $S$ have the same $y$ coordinate. If $(x_b, y_b) \notin I$, then the lower horizontal edge of $I$ does not pass through a point, and, from lemma 14, $I$ can't maximize $\mathcal{D}_o$. $\Box$

It follows from lemma 16 that we don't have to find the best rectangle for every pair of points. Instead, for each pair, we find the best rectangle that contains the lower point of the pair. For a given pair of points, the latter maximum can be smaller than the former, but the maximum rectangle overall has to include both points of the pair.

The following lemma shows that this problem is equivalent to a simpler one dimensional problem.

*Lemma 17:* Suppose that we are given a pair of points that define the $y$ coordinates, and we want to find the rectangle with these $y$ coordinates that maximizes $\mathcal{D}_o$ and passes from the lower point. This problem can be reduced to the problem of finding the halfspace discrepancy for two sets of one dimensional points.

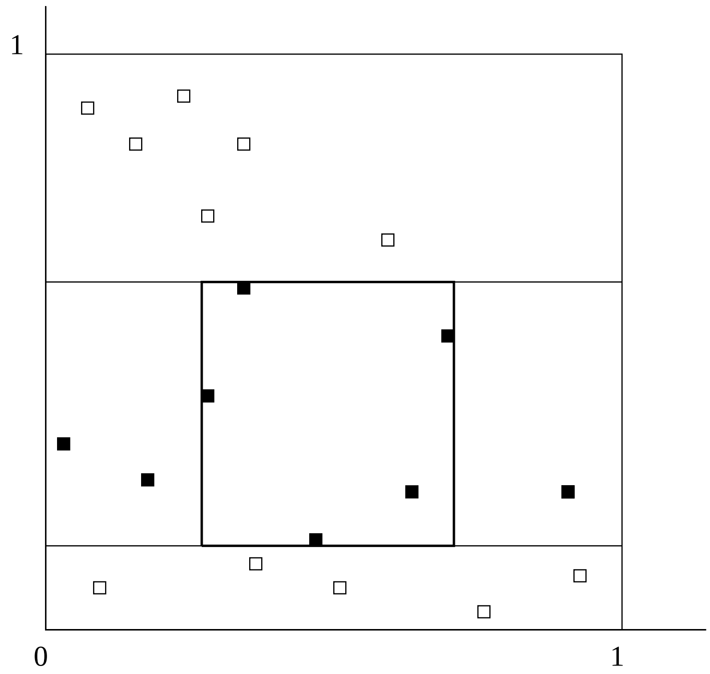



Figure 7.a: The $y$-coordinates of the rectangle are fixed.

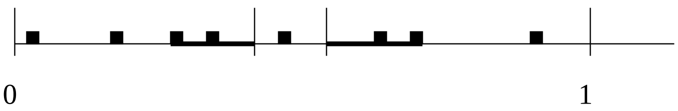



Figure 7.b: The two equivalent one-dimensional problems.

*Proof:* Assume that the given points are $(x_b, y_b)$ and $(x_t, y_t)$, with $y_b < y_t$. We want to find the rectangle $I_m = \{(x_i, y_t), (x_j, y_t), (x_j, y_b), (x_i, y_b)\}$ with $x_i \geq x_b \geq x_j$, that maximizes $\mathcal{D}_o$ (Fig. 7). Following the proof of lemma 15, we take the set $S_{[y_b, y_t]} = \{x_k \mid (x_k, y_k) \in S \text{ and } y_k \in [y_b, y_t]\}$. We also know that we can equivalently maximize the function $Count([x_i, x_j])/(n\Delta y) - (x_j - x_i)$, for all $x_i \leq x_b \leq x_j$.

We divide $[0, 1]$ in three regions, $[0, r_1], [r_1, r_2], [r_2, 1]$ so that $r_1 < x_b < r_2$, and, for all $x_i \in S_{[y_b, y_t]}$, either $x_i < r_1$ or $x_i > r_2$. In other words, only $x_b$ is in the region $[r_1, r_2]$. This partitioning divides $S_{[y_b, y_t]}$ into three disjoint sets, the points on the left of $x_b$ (i.e.

$(S_{[y_b,y_t]} \cap [0,r1]))$, $\{x_b\}$, and the points on the right (i.e. $(S_{[y_b,y_t]} \cap [r2,1]))$. We sort the two sets, and rename the points according to their rank after the sorting: $S_{[y_b,y_t],l} = \{z_1, \ldots, z_{|S_{[y_b,y_t]} \cap [0,r1]|}\}$ and $S_{[y_b,y_t],r} = \{z_1, \ldots, z_{|S_{[y_b,y_t]} \cap [r2,1]|}\}$.

Suppose that the maximum interval is $[x_l, x_r]$. Since it must contain $x_b$, it can have at most one endpoint in each of $[0, r_1]$ and $[r_2, 1]$. ¿From lemma 14 it follows that both $x_l$ and $x_r$ must be points of $S_{[y_b,y_t]}$. So $x_l$ is either in $S_{[y_b,y_t],r}$, or is equal to $x_b$. Assume that the first is true. If we apply lemma 10, we see that $[r2, x_r]$ has to maximize the function $Count([r_2, x_j])/(n\Delta y) - (x_j - r_2)$, for all $[r_2, x_j]$, $x_j \in (S_{[y_b,y_t]} \cap [r2, 1])$. Equivalently, it must maximize the function $i/(n\Delta y) - (z_i - r2)$, for all $z_i \in S_{[y_b,y_t],r}$. This is a linear function of the two dimensional points $\{(z_i, i) \mid z_i \in S_{[y_b,y_t],r}\}$. The maximum of this function can be computed with the technique of [DE] that we gave in § 3.2. The only modification is that we maximize for a different linear function.

This gives us the only two points in $S_{[y_b,y_t]}$ that can form the right endpoint of the maximum interval. Similarly we can find the two possible choices for the left endpoint, and from them we find the maximum interval in constant time. □

The following algorithm finds the rectangle that maximizes $\mathcal{D}_o$ for the input set $S \subset [0,1]^2$. The algorithm uses a modified version of the dynamic linear half-space discrepancy algorithm given in [DE] as a subroutine (Fig. 8).

*Algorithm 3:*

1. Compute $Y_{coord}$ and sort it.
2. For each element $y_i$ of $Y_{coord}$ do:
   (a) Find the set $S_i = \{x_j \mid (x_j, y_j) \in S \text{ and } y_i \leq y_j\}$ and sort it.
   (b) Partition $[0, 1]$ in three regions so that the middle one contains only $x_i$ among all the points of $S_i$.
   (c) While $S_i$ is not empty, do:
      i. Remove from $S_i$ the point $x_j$ with the smallest $x$ coordinate, and insert it into the appropriate (left or right) region.
      ii. Use the modified linear algorithm to find the rectangle that maximizes $\mathcal{D}_o$ over all rectangles that have $y$ coordinates between $y_i$ and $y_j$, and include the point $(x_i, y_i)$.
      iii. If the new value is larger than the largest seen so far, record the rectangle.
3. Return the best rectangle found.

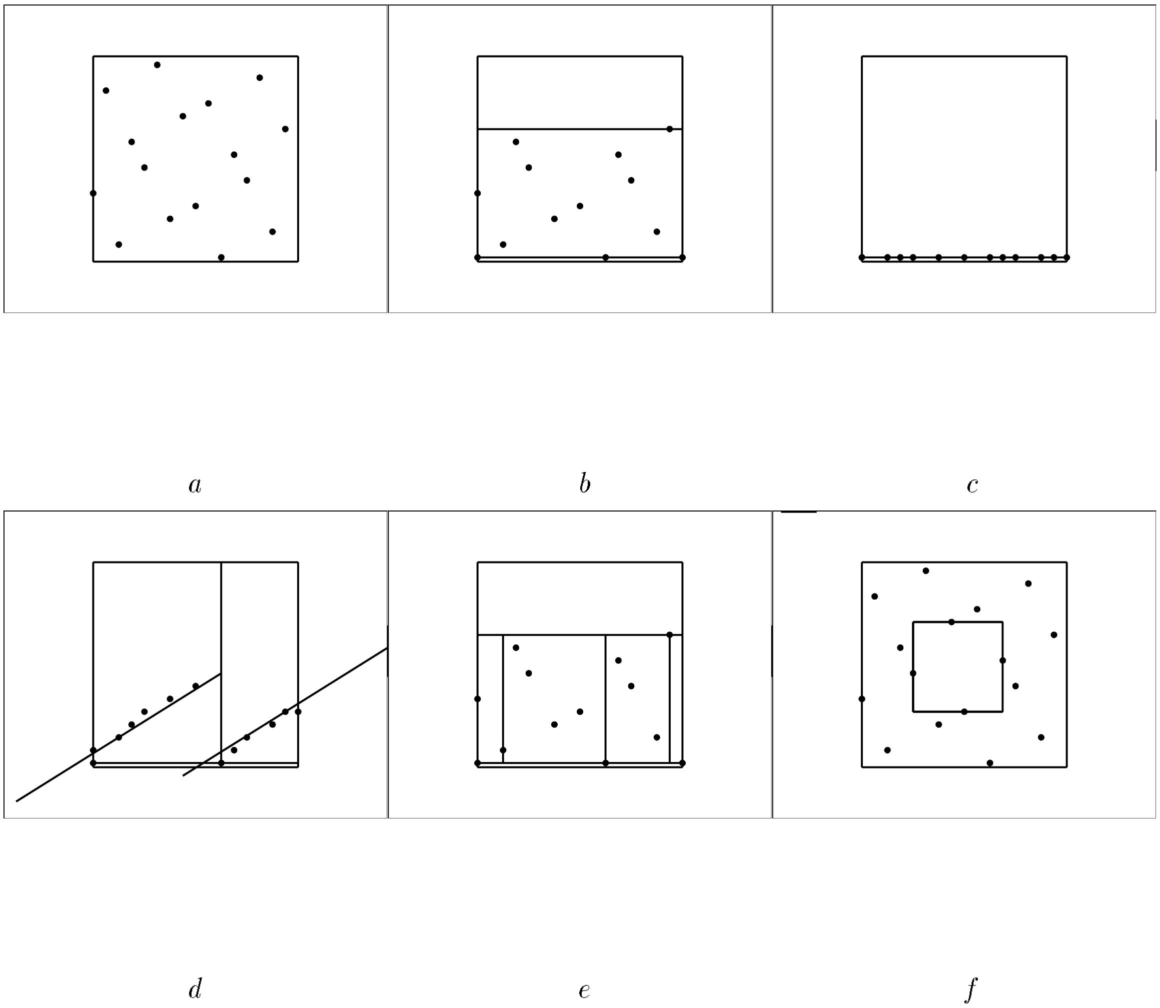



Figure 8: The 2-D algorithm.

$a$: The input point set.

$b$-$e$: Step 2 of the algorithm. In $b$, a pair of points is chosen, in $c$ the 1-D problem is shown, in $d$, the maxima are computed from the convex hulls of the lifted points, in $e$, the best rectangles (that include the lower point and have these $y$ coordinates) are shown.

$f$: The rectangle that maximizes the numerical discrepancy for this point set.

*Theorem 7:* We can compute the maximum numerical discrepancy of a point set $S \subset [0,1]^2$ with $n$ points in $O(n^2 \log^2 n)$ time and $O(n \log n)$ space.

*Proof:* The correctness of the algorithm follows from lemmata 15, 16 and 17. The algorithm considers all pairs of points. For each one it finds the rectangle that maximizes $\mathcal{D}_o$ while it contains the lower point. In the inner loop of the algorithm we use the modified dynamic algorithm of [DE] for the left and the right region separately. We find the maximum of $\mathcal{D}_i$ in a similar way, and from there the discrepancy maximum.

For the running time, we analyze each step. The first step takes $O(n \log n)$ time and then the outer loop is executed $O(n)$ times. The first step of the outer loop also takes $O(n \log n)$ time. The second step takes linear time. The inner loop is executed $O(|S_i|) = O(n)$ times. We are only doing insertions to the convex hull, and we can maintain the convex hulls at a cost of $O(\log^2 n)$ time per insertion [OvL]. Consequently, the dynamic linear algorithm takes $O(\log^2 n)$ time for insertions and $O(\log^2 n)$ time for queries. In the second step we do one insertion and two queries, so the second step takes $O(\log^2 n)$ time. Finally the third step takes constant time.

The total time is $O(n^2 \log^2 n)$. At each point we have to maintain the data structure for two dynamic convex hulls and two sorted arrays of linear size. The used space is therefore $O(n \log n)$. $\square$

So far we implicitly assumed that each point has weight 1, Algorithm 3 can also be used for points with arbitrary non-negative weights. Since all discrepancy computations are being done in the subroutine that calls the one-dimensional algorithm, only this part has to be modified.

# 4. Extensions

In this chapter we give an approximate 2-D algorithm for computing the maximum bichromatic discrepancy and the maximum numerical discrepancy for axis oriented rectangles. We also examine different sets of geometric regions, in particular halfspaces and unions of disjoint rectangles.

## 4.1 An approximation algorithm

It is in many cases desirable (especially when the input sets are very large) to trade off some of the quality of the solutions to reduce the computation time. As we have already seen in the introduction, it is sufficient to find hypotheses with almost optimal empirical error. Approximate algorithms are also useful from the point of view of applied machine learning.

In this section we develop algorithms that find an approximate solutions to the two-dimensional problem. The basic idea is to divide the unit square in a number of rectangles, replace the points in each rectangle with a single point of the appropriate weight, and use the algorithms of sections 2 and 3 to compute the maximum discrepancy of the new, smaller point set $S$ and the new weight function $w : S \rightarrow \Re$.

Let's consider first the bichromatic discrepancy. For a given $S$ , $\chi$ and approximation factor $1/r$, the first step is the partitioning of the unit square in $O(r^2)$ regions (clearly $r$ has to be $o(\sqrt{n})$ for something meaningful to happen). We partition the unit square recursively, by dividing each existing rectangle with more than $O(n/r^2)$ points into two rectangles so that each contains at least $1/3$ of the points. We divide along the $x$ and the $y$ axis alternatively. If we cannot do that for some rectangle, then at least $1/3$ of the points of this rectangle have to have the same $x$ or the same $y$ coordinate. In this case we create a 0-width rectangle, and we subdivide it along its length so that each resulting 0-width rectangle has $\Theta(n/r^2)$ rectangles.

*Lemma 18:* This process is completed in $O(n \log n)$, and in the end we have $\Theta(r^2)$ rectangles, each having $\Theta(n/r^2)$ points.

*Proof:* The running time comes from the fact that first we have to sort the points by $x$ and $y$ coordinates. Since each split reduces the number of each rectangle by a constant, in $O(\log r)$ iterations each rectangle has $O(n/r^2)$ points. Finally, since we stop subdividing a rectangle when it has $\Theta(n/r^2)$ points, we end up with $\Theta(r^2)$ rectangles. $\square$

The next step, after the partition of the unit square, is the reduction of the size of the input set from $n$ to $O(r^2)$. For each rectangle $R$, we compute the weight of the points that lie in it $w(R) = \sum_{x \in (S \cap R)} \chi(x)$, and replace all the points with a point on the lower left corner of the rectangle that has weight $w(R)$. If a point of $S$ lies on a boundary between two rectangles, we assign it to only one rectangle, ensuring that the sum of all the weights is equal to $\sum_{x \in S} \chi(x)$. This step also takes $O(n \log r)$ time.

*Lemma 19:* After the partition of the unit square into rectangles that contain $\Theta(n/r^2)$ points, a line segment parallel to one of the axes can intersect $\Theta(r)$ rectangles.

*Proof:* Assume the line is horizontal, the vertical case is symmetrical. From the construction of the partition, the number of non 0-width rectangles that interset a fixed horizontal line doubles every second iteration, so that line intersects $\Theta(2^{\log r}) = \Theta(r)$ of the initial rectangles. For each one, it might intersect at most one vertical 0-width rectangle. A horizontal line segment can contain many horizontal 0-width rectangles, but can intersect at most two, one with each endpoint. $\square$

*Theorem 8:* Given a point set $S$, and a coloring $\chi$, we can compute in $O(n \log n + r^4 \log r)$ time an approximation of the maximum bichromatic discrepancy within $O(n/r)$ of the optimal.

*Proof:* We apply Algorithm 2, suitably modified to work with the arbitrary weights, to the set $S_r$. Let's assume that the rectangle $I_m$ maximized $\Delta$ for $S$ and $\chi$. From the construction of $S_r$, each edge of $I_m$ can intersect at most $\Theta(r)$ rectangles. Let $S_r \cap I_m$ be the set of points in $S_r$ that are inside $I_m$, and take $I'$ to be the minimum rectangle that contains all of them. The difference between $\sum_{x\in(S\cap I_m)} \chi(x)$ and $\sum_{x\in(S_r\cap I')} w(x)$ is at most the weight of the rectangles intersected by $I_m$. There are $\Theta(r)$ of them, and each has weight $\Theta(n/r^2)$, so $\Delta_{S_r,w}(I')$ differs at most $\Theta(n/r)$ from $\Delta_{S,\chi}(I_m)$. Similarly, if the rectangle $I'_m$ maximizes the discrepancy for $S_r$ and $w$, there exists a rectangle $I$ with $\Delta_{S,\chi}(I)$ that differs at most $\Theta(n/r)$ from $\Delta_{S_r,w}(I'_m)$. Therefore, $|\Delta(S,\chi) - \Delta(S_r,w)| = O(n/r)$. □

When we consider the numerical discrepancy, we realize that we have to add another step in the construction of the set $S_r$ to account for the area of the rectangles. After we do the subdivision, we have $O(r^2)$ rectangles, and the weight of each is at most $O(n/r^2)$. If the area of any of these rectangles is larger than $4r^{-2}$, we divide it in half along its shortest edge, and continue to do so until the area of all rectangles is at most $4r^{-2}$.

The whole operation is performed in $O(n \log n)$ time, and, as the following lemma shows, we end up with $O(r^2)$ rectangles.

*Lemma 20:* After this process, the number of the rectangles that partition the unit square is $\Theta(r^2)$.

*Proof:* The number of the initial rectangles is $\Theta(r^2)$. Every rectangle that is produced by the breaking up of a large initial one has an area of at least $r^{-2}$, and therefore there are at most $r^2$ additional rectangles produced. □

*Lemma 21:* The discrepancy of each one of these rectangles is $O(r^{-2})$.

*Proof:* The discrepancy of any such rectangle $R$ is $|Area(R) - In(R)/n|$. But we have that $0 \leq Area(R) \leq 2r^{-2}$ for every rectangle. We also have that $0 \leq In(R) \leq n/r^2 \Rightarrow 0 \leq In(R)/n \leq r^{-2}$. It follows that $|Area(R) - In(R)/n| \leq 2r^{-2}$. □

After the subdivision, we can reduce the size of the input set from $n$ to $O(r^2)$. For each rectangle $R$, we find the number of points $In(R)$ that lie inside it, and replace them with a point on the lower left corner of the rectangle that has weight $In(R)$ (Fig. 9). Again we have to make sure to count each point only once, so that the sum of all the weights is $n$. This step takes $O(n \log r)$ time.

The final step is the application of the two-dimensional algorithm on the new point set. As we noted in the previous sections, the algorithm can be modified to run for weighted points without affecting its asymptotic performance, so this step takes $O(r^4 \log^2 r)$ time.

*Lemma 22:* After the partition of the unit square into rectangles with area $O(r^{-2})$ and containing $\Theta(n/r^2)$ points, a line parallel to one of the axis can intersect $\Theta(r)$ rectangles.

*Proof:* Assume the line is horizontal, the vertical case is symmetrical. From the construction of the partition, the number of rectangles that interset a fixed horizontal line doubles every second iteration, so that line intersects $\Theta(2^{\log r}) = \Theta(r)$ of the initial rectangles. Some of these are replaced by a number of smaller rectangles, which are now intersected by the horizontal line. There are two kinds of these rectangles, depending on when their intersected edges were created.

There are these that their two intersected edges belong to the same initial rectangle. They cannot be more than $\Theta(r)$, because each one takes the place of one initial rectangle.

And there are these that at least one of their two intersected edges was created during the partitioning of large initial rectangles. But such edges divide the largest dimension of the old rectangle, so each of these rectangles is at least $\sqrt{2}/2r$ wide, and there cannot be more than $\sqrt{2}\, r$ of them that intersect the horizontal line.

The total number of intersected rectangles is then between $r$ and $(1+\sqrt{2})\, r$. $\square$

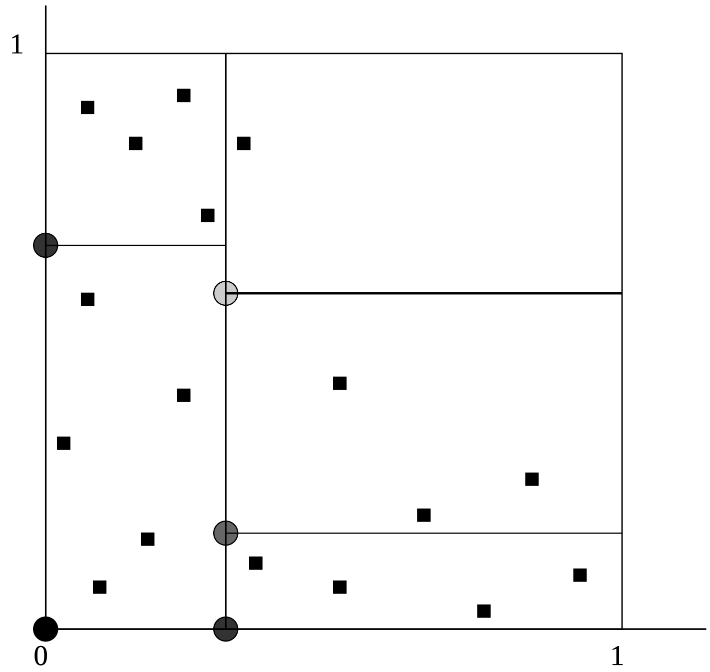



Figure 9: A sample input set (square points), the subdivision of the unit square into rectangles with bounded numerical discrepancy (here $r = 2$), and the input to the approximation algorithm (round points, the darker points have higher weight).

*Theorem 9:* Given a point set $S \subset [0,1]^2$, we can compute in $O(n \log n + r^4 \log^2 r)$ time an approximation of the maximum numerical discrepancy within $O(1/r)$ of the optimal (for $r = o(\sqrt{n})$).

*Proof:* ¿From the discussion above and the proof of Theorem 8. □

We note here that both of these algorithms can be easily modified to work for point sets with arbitrary weights (for the bichromatic discrepancy) or arbitrary positive weights (for the numerical discrepancy).

### 4.2 Higher Dimensions

Both Algorithm 2 and Algorithm 3 can be easily extended to handle $d$ dimensions. The running times are $O(n^{2d-2}\log n)$ for the bichromatic discrepancy and $O(n^{2d-2}\log^2 n)$ for the numerical discrepancy) and the space requirement is linear.

The main idea of the extension is to project the $d$ dimensional points to 2 dimensions. Clearly the box with the maximum (numerical or bichromatic) discrepancy must have a point on each hyperplane of its boundary. So the algorithm examines every possible pair of points as a pair of boundaries in every one of the first $d-2$ dimensions. For each of the $O(n^{2(d-2)})$ combinations of boundaries, it projects all points that are inside the boundaries to the remaining 2 dimensions, and applies the two-dimensional algorithm to the set of the projections.

### 4.3 Union of two rectangles

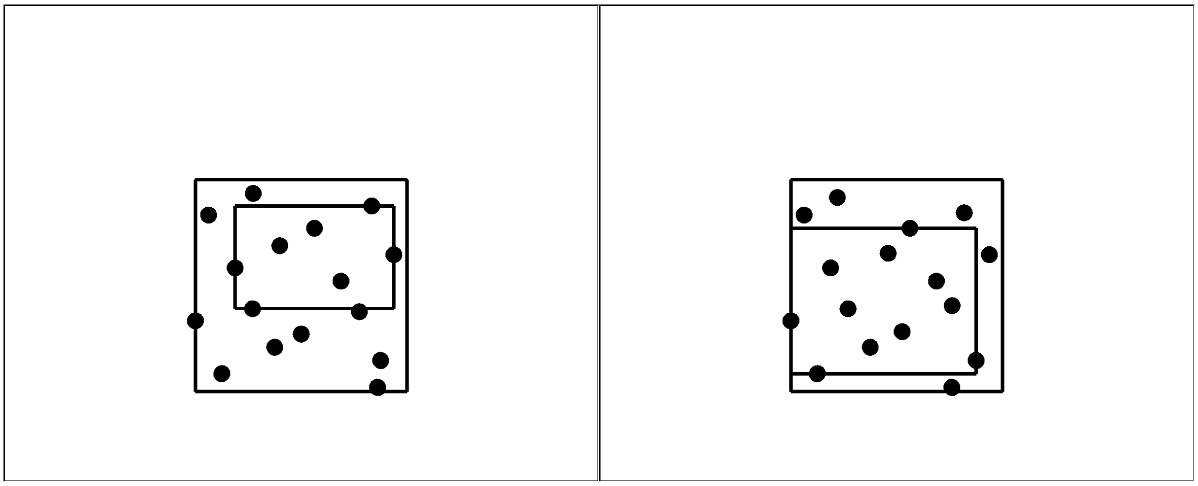
Here we consider a natural extension to the set of axis-aligned rectangles, the set of unions of two disjoint axis aligned-rectangles $\mathcal{R}_2$.

*Theorem 10:* We can compute the range in $\mathcal{R}_2$ that maximizes the bichromatic discrepancy of a given point set $S$ with an arbitrary given coloring $\chi$ in $O(n^2\log n)$ time. We can also compute the range in $\mathcal{R}_2$ that maximizes the numerical discrepancy of a given point set $S$ in $O(n^2\log^2 n)$ time.

*Proof:* Let us consider the case for the bichromatic discrepancy first, the other case is similar. The two rectangles of the optimum range are axis-aligned and disjoint, therefore they are separated by either a horizontal or a vertical line. Consider the case of the horizontal line. There are $n$ possible cases for such a line, the $y$ coordinates of the input points. For each such line we compute the optimum rectangle above it and the optimum rectangle below it. We can find the best rectangle for all $n$ lines with one run of Algorithm 2. Each time we find a new optimum rectangle we record it in the line that passes right through the top of the rectangle. After we have found the optimum rectangle that touches each line, we can find in linear time the optimum rectangle that touches or lies below the line. Similarly we find the optimum rectangle that lies above each line. Then we form the $n$ pairs and find the optimum one. The case for vertical lines is the same, but we have to modify the algorithm to perform a horizontal sweep. □

### 4.4 Halfspaces

[DE] looked at the problem of computing the maximum numerical discrepancy for the set of halfspaces. They gave an algorithm that finds the hyperplane that maximizes the numerical discrepancy of a point set $S \in [0,1]^d$ ($|S| = n$) in $O(n^d)$ time. We note here that this algorithm can be easily modified to solve the corresponding minimizing disagreement problem for hyperplane hypotheses in the same time bound.



*a* *b*

Figure 10: Sampling sets with small maximum numerical discrepancy. The rectangles shown maximize the numerical discrepancy.

*a*: Euclidian area. *b*: Weighted area (§4.5).

### 4.5 Weighted area functions

Algorithm 3 can be used to find sampling patterns that approximate different measures rather than the Euclidean area. Such point sets can be useful in many cases. For example, we may know that some parts of an image have more details and so we can sample these parts at a higher rate. Also, point sets with more weight in the middle can be used for low pass filtering. In a particular case that we examined (Fig. 10.b), the area of a rectangle was given by the function $WArea(A) = \int_A sinc(x-0.5)sinc(y-0.5)dxdy$.

# 5. Conclusions and Problems

In this paper we examine two closely related algorithmic problems that arise in different areas of computer science.

The algorithms we present for computing the maximum discrepancy have direct applications in machine learning and computer graphics, and their near quadratic running time allows them to be useful with the large data sets common in many real-life problems. We also implemented a simple version of the two-dimensional algorithm to compute the maximum numerical discrepancy, and used it to find sampling patterns with very low rectangle numerical discrepancy for ray tracing (Fig. 8, Fig. 10.a). We use a probabilistic procedure that starts with a good pattern of low maximum discrepancy (e.g. Zaremba points) and randomly picks and replaces points so that the maximum numerical discrepancy decreases. This implementation provided the basis for visualization of the way the algorithm works ([DG]). For this purpose we used GASP ([DT]), a system for scientific visualization. Figures 8 and 10 are produced from our visualization.

Finally we present a number of interesting open questions related to discrepancy.

1. What is the lower bound for an algorithm that computes the maximum bichromatic or numerical discrepancy for axis aligned rectangles in the $d$-dimensional case (for $d > 1$)? We conjecture that $\Omega(n^2)$ is a lower bound for both problems in the 2-D case.

2. A related problem is that of finding a fast dynamic algorithm for the $d$-dimensional case ($d > 1$). There is none known so far that asymptotically improves on the simple approach that just uses the static algorithm after each update.

3. Extend these algorithms to arbitrarily oriented boxes and different families of regions. One problem here is that the search space we have to explore increases with the complexity of the regions. For example there are $O(n^5)$ possible solutions when we consider rectangles with arbitrary orientation. A fuller treatment on this subject will appear in [Gu].

4. There is also the problem of finding better approximation algorithms for both the static and the dynamic cases.

5. Finally, there are problems in different areas that can be recast as variants of discrepancy problems. For example, the algorithm that orients polygonal parts [Go] has to solve the following problem. Given a set of linear points in the unit interval, and a number $k$, find the smallest interval that includes $k$ points. This is a disguised discrepancy problem, because a different way to state it is, find the intervals of minimum discrepancy that contain a given number of points. Advanced algorithms and related data structures developed in discrepancy theory could offer solutions for this and similar problems.

# Acknowledgement

The authors would like to thank Bernard Chazelle and David Haussler for helpful communications regarding the research for this paper.

# References

[AL] D. Angluin and P. Laird, Learning from noisy examples. *Machine Learning*, 2 (1988), 343-370.

[BC] J. Beck and W.W.L. Chen, *Irregularities of distribution.* Cambridge University Press (1987).

[BCM] H. Bronnimann, B. Chazelle and J. Matousek, Product range spaces, sensitive sampling, and derandomization. *Proc. 34th Ann. IEEE Symp. Foundat. of Comp. Sci.* (1993), 143-155.

[BN] W. Buntine and T. Niblett, A further comparison of splitting rules for decision-tree induction. *Machine Learning*, 8 (1992), 75-82.

[C] B. Chazelle, Geometric Discrepancy Revisited. *34rth Symposium on Foundations of Computer Science* (1993).

[dB] M. deBerg, Computing Half-Plane and Strip Discrepancy of Planar Point Sets. To appear (1993).

[DE] D. Dobkin and D. Eppstein, Computing the Discrepancy. *Proceedings of the ninth annual Symposium on Computational Geometry* (1993), 47-52.

[DG] D. Dobkin and D. Gunopulos, Computing the rectangle discrepancy. *3rd Annual Video Review of Computational Geometry* (1994) 385-386.

[DM] D. Dobkin, and D. Mitchell, Random-Edge Discrepancy of Supersampling Patterns. *Graphics Interface '93* (1993).

[DT] D. Dobkin and A. Tal, GASP – A System to Facilitate Animating Geometric Algorithms. *Third Annual Video Review of Computational Geometry,* (1994), 388-389.

[Go] K.Y. Goldberg, Orienting Polygonal Parts Without Sensors. *Algorithmica 10,* (1993) 201-226.

[Gu] D. Gunopulos, Ph.D. Thesis, Princeton University, Department of Computer Science, to appear.

[Hal] J.H. Halton, A retrospective and prospective survey of the Monte Carlo method. *SIAM Review* 12 (1970), 1.

[Hau] D. Haussler, Decision theoretic generations of the PAC-model for neural nets and other applications. *Inf. and Comp.*, 100 (1992), 78-150.

[HW] D. Haussler and E. Weltz, $\epsilon$-nets and simplex range queries. *Disc. Comput. Geom.* 2 (1987), 127-151.

[HSV] K.U. Hoeffgen, H.U. Simon and K.S.Van Horn, Robust Trainability of single neurons. *Preprint* (1993).

[Ho] R.C. Holte, Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11 (1993), 63-91.

[HU] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley (1979).

[Ka] J.T. Kajiya, The Rendering Equation. *Computer Graphics,* 20 (1986) 143-150.

[Ke] M. Kearns, Efficient noise-tolerant learning from statistical queries. *Proc. of the 25th ACM Symp. on the Theory of Computing* (1993), 392-401.

[KL] M. Kearns and M. Li, Learning in the presence of malicious errors. *SIAM J. Comp.* 22 (1993), 807-837.

[KS] M. Kearns and R.E. Schapire, Efficient distribution-free learning of probabilistic concepts. *Proc. of the 31st Annual IEEE Symp. on Foundations of Computer Science* (1990), 382-391.

[KSS] M. Kearns, R.E. Schapire and L.M. Sellie, Toward efficient agnostic learning. *Proc. of the 5th ACM Workshop on Computational Learning Theory* (1992) 341-352.

[Ko] J.F. Koksma, Een algemeena stelling uit de theorie der gelijkmtige verdeeling modulo 1. *Mathematica B (Zutphen)* 11 (1942/43) 7-11.

[Ma] W. Maass, Efficient Agnostic PAC-Learning with Simple Hypotheses. *Proc. of the 7th Annual ACM Conference on Computational Learning Theory* (1994), 67-75.

[MWW] J. Matousek, E. Welzl and L. Wernish, Discrepancy and $\epsilon$-approximations for bounded VC-dimension. *Proc. of the 25th ACM Symp. on the Theory of Computing* (1993), 424-430.

[Me] K. Mehlhorn, *Multi-Dimensional Searching and Computational Geometry*, Springer 1984.

[Mit] D. Mitchell, Ph.D. Thesis, Princeton University, Department of Computer Science, to appear.

[Min] J. Mingers, An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4 (1989), 227-243.

[N78] H. Niederreiter, Quasi-Monte Carlo methods and pseudo-random numbers. *Bulletin of the American Mathematical Society*, 84 (1978), 6, 957-1041.

[N92] H. Niederreiter, Quasirandom sampling computer graphics. *Proc. 3rd Internat. Seminar on Digital Image Processing in Medicine,* Riga, Latvia (1992), 29-33.

[OvL] M. Overmars and J. van Leeuwen, Maintenance of configurations in the plane. *J. Comput. Sys. Sci.* 23 (1981) 166-204.

[P] F. Preparata, An optimal real time algorithm for planar convex hulls. *Comm. ACM* 22 (1979).

[PS] F. Preparata and M.I. Shamos, *Computational Geometry,* Springer (1985).

[S] P. Shirley, Discrepancy as a quality measure for sample distributions. *Proceedings of Eurographics '91,* 183-193.

[T] M. Talagrand, Sharper bounds for empirical processes. To appear in *Annals of Probability and its Applications.*

[TW] J.F. Traub and H. Wozniakowski, Recent progress in information-based complexity. *Bulletin of the EATCS 51* (Nov. 1993).

[V84] L.G. Valiant, A theory of the learnable. *Comm. of the ACM* 27 (1984), 1134-1142.

[V84] L.G. Valiant, Learning disjunctions of conjunctions. *Proc. of the 9th Intern. Joint Conf. on Art. Int.* (1985), 560-566.

[VC] V.N. Vapnik and A.Ya. Chervonenkis, On the uniform convergence of relative frequencies of events to their probabilities. *Theory Probab. Applic.* 16 (1971), 264-280.

[WGT] S.M. Weiss, R. Galen and P.V. Tadepalli, Maximizing the predictive value of production rules. *Art. Int.* 45 (1990), 47-71.

[WK90] S.M. Weiss and I. Kapouleas, An empirical comparison of pattern recognition, neural nets, and machine learning classification methods. *Proc. of the 11th Int. Joint Conf. on Art. Int.* (1990), Morgan Kauffmann, 781-787.

[WK91] S.M. Weiss and C.A. Kulikowski, *Computer Systems that Learn* (1991), Morgan Kauffmann.

[W] H. Wozniakowski, Average Case Complexity of Multivariate Integration. *Bulletin Amer. Math. Soc. 24* (1991).