Spatial Scan Statistics: Approximations and Performance Study

Deepak Agarwal
Yahoo! Research

Andrew McGregor
University of Pennsylvania

Jeff M. Phillips
Duke University

Suresh Venkatasubramanian
AT&T Labs – Research

Zhengyuan Zhu
University of North Carolina

ABSTRACT

Spatial scan statistics are used to determine hotspots in spatial data, and are widely used in epidemiology and bio-surveillance. In recent years, there has been much effort invested in designing efficient algorithms for finding such "high discrepancy" regions, with methods ranging from fast heuristics for special cases, to general grid-based methods, and to efficient approximation algorithms with provable guarantees on performance and quality.

In this paper, we make a number of contributions to the computational study of spatial scan statistics. First, we describe a simple exact algorithm for finding the largest discrepancy region in a domain. Second, we propose a new approximation algorithm for a large class of discrepancy functions (including the Kulldorff scan statistic) that improves the approximation versus runtime trade-off of prior methods. Third, we extend our simple exact and our approximation algorithms to data sets which lie naturally on a grid or are accumulated onto a grid. Fourth, we conduct a detailed experimental comparison of these methods with a number of known methods, demonstrating that our approximation algorithm has far superior performance in practice to prior methods, and exhibits a good performance-accuracy trade-off.

All extant methods (including those in this paper) are suitable for data sets that are modestly sized; if data sets are of the order of millions of data points, none of these methods scale well. For such massive data settings, it is natural to examine whether small-space streaming algorithms might yield accurate answers. Here, we provide some negative results, showing that any streaming algorithms that even provide approximately optimal answers to the discrepancy maximization problem must use space linear in the input.

Categories and Subject Descriptors

G.3 Mathematics of Computing - Probability and Statistics

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
KDD'06, August 20–23, 2006, Philadelphia, Pennsylvania, USA.
Copyright 2006 ACM 1-59593-339-5/06/0008 ...$5.00.

General Terms

Algorithms, Design, Performance

Keywords

Spatial Scan Statistics, Kulldorff Scan Statistic, Discrepancy

1. INTRODUCTION

With the availability of massive data and cheap computing power, the problem of detecting "hotspots" has become ubiquitous and has received a lot of attention in data mining [7, 15, 14]. In particular, the special case of detecting "bumps" or local clusters in spatial data has found numerous applications in epidemiology, bio-surveillance, astronomy etc. While a wide range of methods have been proposed to test for spatial clustering (after adjusting for an inhomogeneous background population), the *spatial scan statistic* is by far the most popular. The original method proposed by [11] computes the maximum discrepancy region obtained by *scanning* the spatial region under study with a set of circular regions of various radii. The discrepancy score for each region is based on a likelihood ratio test statistic constructed to detect significant overdensity under the Poisson or Bernoulli model. The test was shown to satisfy the optimality property of being the *individually most powerful test*. Roughly speaking, this means if the model is correct and the main focus is to find the actual location of clusters rather than just detect an overall clustering effect, the spatial scan statistic is optimal. However, due to the dependencies introduced by considering overlapping regions, the analytical distribution of the spatial scan statistic is often intractable and hence one takes recourse to randomization tests [5]. Such a test computes the distribution of the scan statistic by simulating data under the null hypothesis (no clustering) and callibrating the observed value relative to this distribution (using a p-value) to determine the statistical significance of the most discrepant region. In general, 1000 simulations from the null distribution are enough to determine significance. In practice, when the null hypothesis holds, one may be able to conclude statistical insignificance with a lot fewer repetitions. Originally applied to small sample size problems in epidemiology, the technique has attracted interest in the post 9/11 era for surveillance of massive geographic databases leading to growing interest in computationally efficient algorithms. Recently, the scan statistic has also been used in other areas like bioinformatics [10] and for detecting chatter in massive social networks [17].

Friedman and Fisher [7] introduced an alternative approach which greedily computes a high discrepancy rectangle, but has no guarantees as to how it compares to the optimal. Their approach is quite general, and works in arbitrary dimensional spaces, but is not conservative: many regions will remain unexplored. A series of papers by Neill and Moore [15, 14, 16] developed a grid-based heuristic that uses pruning strategies to avoid enumerating all $\binom{n}{4}$ rectangular ranges; similar ideas work for square ranges as well. Most recently, Agarwal *et al.* [1] presented approximation algorithms that run in time $O(\frac{1}{\epsilon}n^2 \log^2 n)$ while guaranteeing a solution that is at most ϵ less than the optimal solution. Their algorithm extends to any convex discrepancy function; the above bound is for the Kulldorff scan statistic.

1.1 Our Contribution

In this paper we continue the investigation of efficient computational strategies for scan statistics. Our contributions are as follows:

- We present a new approximation heuristic for computing scan statistics based on convex discrepancy functions.
- We present an exact algorithm running in time $O(g^4)$ and approximation algorithms running in time $O(g^3 \operatorname{poly}(\log g, 1/\epsilon))$ for grid-based data.
- We implement all of the above algorithms and the algorithm by Agarwal *et al.*, and compare them with an implementation of the algorithm of Neill and Moore. We show that our methods are superior in practice, and scale better than prior methods on both gridded and non-gridded data.
- We examine the problem of computing scan statistics on streams. We show that it is hard to approximate discrepancy functions to within a constant factor using a stream algorithm without using linear space. We also have related lower bounds for additive approximations.

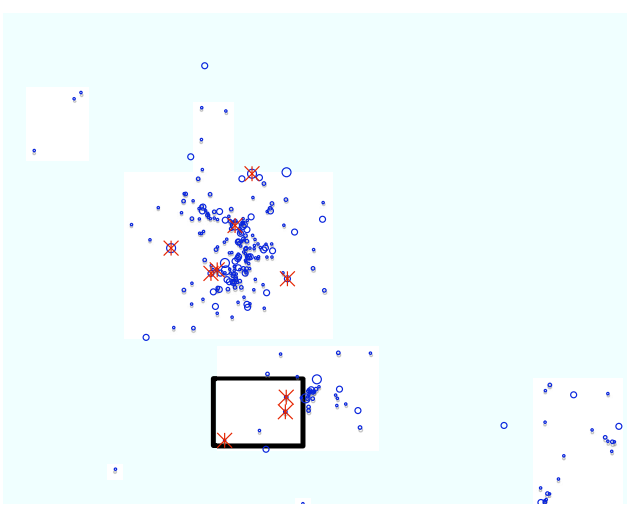


Figure 1: Example of maximal discrepancy range on a data set. Xs are measured data and Os are baseline data.

2. PRELIMINARIES

Let P be a set of n points in the plane. Measurements and baseline measures over P will be represented by two functions, $m : P \to \mathbb{R}$ and $b : P \to \mathbb{R}$. $\mathcal{R}$ denotes a range space over P. A *discrepancy function* is defined as $d : (m, b, R) \to \mathbb{R}$, for $R \in \mathcal{R}$. For instance, in an epidemiology application where the goal is to find disease clusters, the points in space could be a collection of counties. The measurement m associated with each county is the number of cases of some rare disease and the baseline measure b is the population at risk. If one assumes a Poisson distribution for the number of disease cases, the optimal discrepancy measure d obtained in this scenario is the well known Kulldorff scan statistic.

Let $m_R = \sum_{p \in R} m(p)/M, b_R = \sum_{p \in R} b(p)/B$, where $M = \sum_{p \in U} m(p)$, $B = \sum_{p \in U} b(p)$, and U is some box enclosing all of P. *We will assume that d can be written as a convex function of m_R, b_R.* All the discrepancy functions that we consider in this paper satisfy this condition; most discrepancy functions considered prior to this work are convex as well. We can write $d(m, b, R)$ as a function $d' : [0, 1]^2 \to \mathbb{R}$, where $d(m, b, R) = d'(m_R, b_R)$. We will use d to refer to either function where the context is clear. With these notations, the Kulldorff scan statistic (ignoring constants) is given by

$$d(m_R, b_R) = m_R \log\left(\frac{m_R}{b_R}\right) + (1 - m_R) \log\left(\frac{1 - m_R}{1 - b_R}\right)$$

if $m_R > b_R$ and 0 otherwise.

Linear discrepancy functions are a special class of discrepancy functions where $d = \alpha \cdot m_R + \beta \cdot b_R + \gamma$. It is easy to see that *combinatorial (bichromatic) discrepancy*, the difference between the number of red and blue points in a region, is a special case of a linear discrepancy function.

The main problem we study in this paper is:

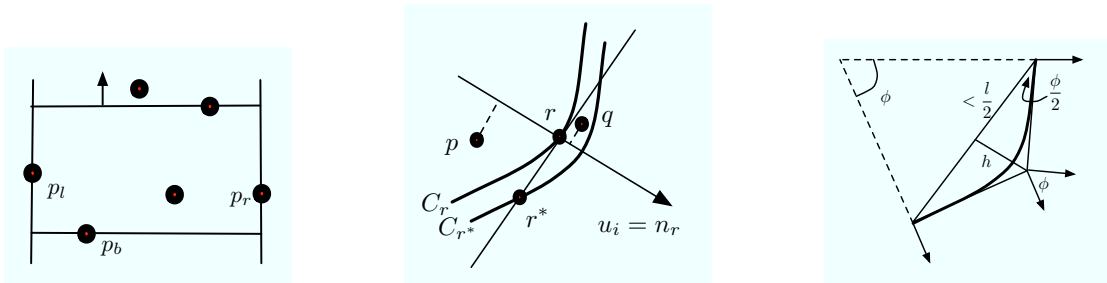
PROBLEM 2.1 (MAXIMIZING DISCREPANCY). *Given a point set P with measurements m, baseline measure b, a range space $\mathcal{R}$, and a convex discrepancy function d, find the range $R \in \mathcal{R}$ that (approximately) maximizes d.*

An equivalent formulation, replacing the range R by the point $r = (m_R, b_R)$ is:

PROBLEM 2.2. *(Approximately) Maximize convex discrepancy function d over all points $r = (m_R, b_R)$, where the range $R \in \mathcal{R}$.*

In this paper we will only consider range spaces consisting of axis-parallel rectangles. Two rectangles that contain the same set of points are equivalent for the purpose of discrepancy calculation. Therefore there are $O(n^4)$ distinct axis-parallel rectangles.

Boundary conditions. As is customary to avoid overfitting, we remove from consideration any range that has very small support in either the baseline or measurements. Formally, we require that any range and its compliment has a measure of at least C, for some arbitrary constant $C \geq 1$. In our mapping from ranges to points, this is equivalent to saying that the domain we maximize over is $S_n = [C/M, 1 - C/M] \times [C/B, 1 - C/B]$. Often we only care about ranges with proportionally more measured data than baseline data. These points are defined by $S_n^+ = \{(m_R, b_R) \in S_n \mid m_R > b_R\}$.



(a) Sweep Line in Algorithm **Exact**.

(b) Error between contours.

(c) Error in approximating an arc with a segment.

Figure 2: Sweep lines, contours, and arcs.

Grid algorithms. For some algorithms, the data is assumed to lie on a grid, or is accumulated onto a set of grid cells. For such algorithms, we will assume a grid of size $g \times g$, with measurement and baseline values associated with each grid point as before. Note that in such a grid, the effective number of points is g^2, and the number of distinct axis-parallel rectangles is $O((g^2)^2) = g^4$, which differs from the corresponding numbers n and $O(n^4)$ for points and axis-parallel rectangles in general position. It will be important to keep this distinction in mind when comparing grid algorithms with those taking inputs in general position.

3. A SIMPLE EXACT ALGORITHM

In this section we present a simple algorithm running in time $O(n^4)$ that computes the maximum discrepancy rectangle exactly. Even though there are $O(n^4)$ rectangles to be considered, a naive search strategy might end up taking linear time for each rectangle (to estimate m_R, b_R) yielding a net running time of $O(n^5)$. We use a simple sweep line technique and incremental updates to avoid this problem.

Any set of four points defines a unique bounding rectangle, with one point defining each side. See Figure 2(a). Fix a pair of points $p_r, p_l \in P$, and consider the set of all rectangles whose left and right extremes are defined by this pair. Choose a third point $p_b \in P$ in between these two; this point defines the bottom edge of the rectangle if it is below otherwise one of p_r, p_l does. Now let a horizontal line segment spanning the rectangle sweep the plane upwards starting from p_b. Every time the sweep line encounters a point, we update m_R, b_R in constant time and recompute the discrepancy, maintaining the largest value. Each sweep takes linear time, and there are $O(n^3)$ choices of triples (p_l, p_r, p_b). Thus, the algorithm runs in time $O(n^4)$. The details of this algorithm are presented in Algorithm 1.

If the points lie in a $g \times g$ grid, a similar argument yields an algorithm (**Exact-Grid**) that runs in time $O(g^4)$. This algorithm has the same asymptotic running time as the algorithm of Neill and Moore [15], has comparable performance in practice (see Section 7), and is much simpler.

4. AN APPROXIMATION HEURISTIC

The basis for the heuristic we present in this section is the following *linearization* lemma, proved in [1].

LEMMA 4.1 ([1]). *A discrepancy function of the form $d(m_R, b_R) = \alpha m_R + \beta b_R + \gamma$ can be maximized over axis parallel rectangles in time $O(n^2 \log n)$.*

Algorithm 1 Algorithm **Exact**

```
maxd = -1
Sort all points by y-coordinate.
for all pairs of points (p_l, p_r) do
  for i = 1 to n do
    Let p_b be the point with i^th smallest y-coordinate.
    m = 0, b = 0
    for j = i + 1 to n do {This is the sweep line}
      Let p be point with j^th smallest y-coordinate
      m = m + m(p), b = b + b(p)
      d = d(m, b).
      if (d > maxd) then
        maxd = d
```

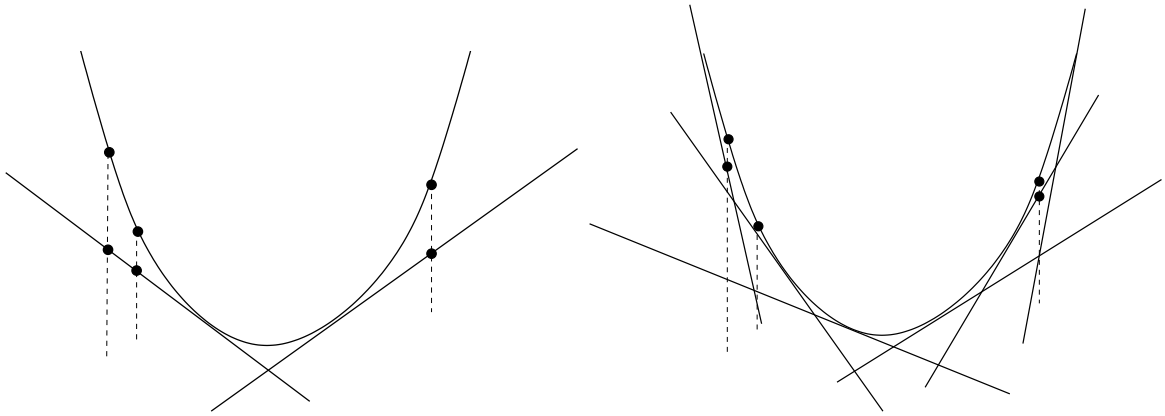
One way of exploiting linearization is to represent the discrepancy function as the upper envelope of a collection of linear functions. The resulting piece-wise linear function closely approximates the true discrepancy, and thus any computation performed using the linear functions will yield an answer close to the true optimum. We refer to this as the **Approx-Linear** Algorithm. This was the approach used in [1].

However, a better approach exploits two facts: first, we only wish to approximate the value of the maximum discrepancy rectangle and second, the function being optimized is monotone in S_n^+. Recall that each range $R \in \mathcal{R}$ can be represented as a point $r = (m_R, b_R) \in [0,1]^2$. We wish to find $r^* = \arg\max_{r \in \mathcal{R}} d(r)$, the maximum discrepancy range, or to find a range which closely approximates r^*. To this end we will approximate the convex function with a set of t linear functions: $\mathcal{L} = \{\ell_i\}_{i=1}^t$, where each $\ell_i(m_R, b_R) = \alpha_i m_R + \beta_i b_R + \gamma_i$. By taking the largest point $r_i = \arg\max_{r \in \mathcal{R}} \ell_i(r)$ for each linear function ℓ_i and then returning $r' = \arg\max_{r_i} d(r_i)$ we can approximate the maximum discrepancy range on d.

Let $C_z = \{(x,y) \in S_n^+ \mid d(x,y) = z\}$ be the contour of f at value z. For optimal point r^*, all points in $C_{d(r^*)}$ are also optimal, and any point $r \in C_z$ such that $d(r^*) - \epsilon \leq z \leq d(r^*)$ gives an ϵ-approximation to r^*.

Let n_r be the normal direction to $C_{d(r)}$ at r. A linear function ℓ_i defines a direction u_i, and sorts all points along that direction.

LEMMA 4.2. *If $u_i = n_{r^*}$, then ℓ_i correctly identifies r^* as $\arg\max_{r \in \mathcal{R}} d(r)$.*



(a) To preserve the maximum discrepancy region (the highest point), we need few functions.

(b) To approximate the function in its entirety we need many functions.

Figure 3: Projecting onto a linear function

PROOF. By definition of r^*, there is no point $r \in \mathcal{R}$ such that $d(r) > d(r^*)$. Since d is convex, any point $p \in C_{d(r^*)}$ is less than or equal to r^* along direction $u_i = n_{r^*}$. Thus r^* is the maximum point along u_i. □

Thus, if we can find a direction u_i such that $n_{r^*} = u_i$, then a single invocation of the linearization lemma yields the optimal solution. Figure 3 illustrates this idea, one dimension lower. The plots depicts a convex discrepancy function defined over a single input parameter. A straightfoward linear approximation of the function would require us to use multiple linear functions, illustrated in Figure 3(b). However, the direction approach described above requires us only to preserve the ordering of points along this direction, and thus two linear functions suffice (Figure 3(a)).

However, we would like to bound the error caused by a $u_i \neq n_{r^*}$, since we do not want to place an ℓ_i such that $u_i = n_r$ for every point $r \in \mathcal{R}$.

LEMMA 4.3. *Consider the point r such that $u_i = n_r$ and $\ell_i(r) = \ell_i(r^*)$. If $d(r) \geq d(r^*) - \epsilon$, then ℓ_i identifies a point $r' \in \mathcal{R}$ that gives an ϵ-approximation to r^*.*

PROOF. Any point $p \in S_n^+$ such that $d(p) \leq d(r) = d(r^*)$ will have $\ell_i(p) < \ell_i(r)$. See Figure 2(b). For any point $q \in S_n^+$ such that $\ell_i(q) > \ell_i(r)$, then $d(q) > d(r)$. Thus ℓ_i identifies a point $q \in \mathcal{R}$ such that $d(q) \geq d(r) \geq d(r^*) - \epsilon$. □

LEMMA 4.4. *Consider a piece of a convex curve that is of arc length l and the angle of the normal to the curve changes by no more than $\phi < \frac{\pi}{2}$. This curve segment can be approximated by a line segment such that the maximum error is no more than $l\phi/2$.*

PROOF. Set the line so that it connects both of the end points of the curve. Since the curve is convex, its error can be maximized at the mid-point of the segment when the curve is two segments that bends an angle of $\pi - \phi$ at its mid-point: see Figure 2(c). Let the height from the mid-point of the segment to the mid-point of the curve to be h.

We now have a right triangle with angle $\phi/2$, adjacent side length less than $l/2$, and opposite side length h. Thus we know that $\tan(\phi/2) = h/(l/2)$. Thus $\phi/2 = \arctan(2h/l) \geq h/l$, for ϕ less than π. Thus $h \leq l\phi/2$. □

Now let r^* be the maximal discrepancy range. It can lie anywhere in S_n^+. We want to consider the maximal error allowed by some linear function ℓ_i. Let $r \in C_{d(r^*)}$ have $n_r = u_i$. Also let $\Delta_\theta(r^*, i)$ be the difference in angles between n_{r^*} and $n_r = u_i$. Let $g(r^*, r)$ be the maximal gradient anywhere on $C_{d(r^*)}$ between r^* and r. Now we can bound the error ϵ incurred by approximating the maximum discrepancy range on d with ℓ_i.

$$\epsilon \leq |r^* - r| \cdot \Delta_\theta(r^*, i) \cdot g(r^*, r), \tag{4.1}$$

since $\Delta_\theta(r^*, i) < \pi/2$ and thus $|r^* - r| < 2 \cdot l(r^*, r)$, the arclength of $C_{d(r^*)}$ between r^* and r. Thus, we need to place a set of linear functions to minimize this quantity for any placement of r^* in S_n^+.

4.1 Approx-Extents Algorithm

Using this intuition we describe a set of linear functions which exploits these properties. For each of t linear functions $\ell_i(m_R, b_R) = \alpha_i m_R + \beta_i b_R + \gamma_i$ let

$$\begin{aligned} \alpha_i &= \cos\left(\sin(h_i)\frac{\pi}{2}\right) \\ \beta_i &= -\sin\left(\sin(h_i)\frac{\pi}{2}\right) \\ \gamma_i &= 0 \end{aligned} \tag{4.2}$$

where $h_i = (i-1) \cdot \pi/(2t-1)$. For $t \leq 2$, set $h_1 = \pi/4$, as this single function often gives a very good approximation just by itself.

In summary, the **Approx-Extents** algorithm runs by creating t linear functions according to (4.2) and then invoking the algorithm described by Lemma 4.1 in [1] on each of them. Now let the maximal range for each linear function be $r_i = \arg\max_{r \in \mathcal{R}} \ell_i(r)$. We return the maximum r_i on d defined $r' = \arg\max_{r_i} d(r_i)$. The details of this algorithm are presented in Algorithm 2.

The running time of **Approx-Extents** is $O(tn^2 \log n)$ because we invoke Lemma 4.1 t times.

5. GRID ALGORITHMS

As we mentioned earlier, algorithms like those presented in [15, 14] aggregate data to a regular $g \times g$ grid. Since such

Algorithm 2 Algorithm **Approx-Extents**

```
maxd = −1
for i = 1 to t do
  φ_i = sin(i · π/(2t−1)) π/2
  ℓ_i = cos(φ_i) m_R − sin(φ_i) b_R
  Find r_i = arg max_{r∈R} ℓ_i(r) using Lemma 4.1.
  d_i = d(r_i)
  if (d_i > maxd) then
    maxd = d_i; r' = r_i
```

a grid contains g^2 points, one can run any of the above mentioned algorithms, setting $n = g^2$. However, this is very inefficient, and ignores the special structure of the grid. For example, algorithm **Exact** would then run in time $O((g^2)^4) = O(g^8)$. In this section, we present two algorithms that take advantage of grid structured data.

5.1 Exact-Grid Algorithm

The first algorithm returns the maximum discrepancy rectangle in time $O(g^4)$. It is quite similar to the algorithm of Section 3, using a sweep line to explore the space of rectangles. The basic idea is as follows. We maintain four sweep lines, two horizontal and two vertical. The two vertical sweep lines move from left to right. At any moment, one of them is at x position i, and the other at x position $j > i$. As the second sweep line moves from i to the right most position, we maintain a count, for each row, of the total measured and baseline mass in this row between i and j. This can be done in time $O(g)$ for each move of the second sweep line. Once the two vertical sweep lines are fixed, two horizontal sweep lines move from bottom to top. Since we maintain counts of the total mass in each row, the discrepancy function for the range bounded by the four sweep lines can be computed in constant time every time the higher horizontal sweep line is moved. A detailed description is presented in Algorithm 3.

Algorithm 3 Algorithm **Exact-Grid**: Input is $g \times g$ grid with values $m(i,j), b(i,j)$

```
for i = 1 to g do {Left sweep line}
  Initialize m[y] = m(i,y), b[y] = b(i,y) for all y
  for y = 2 to g do
    m[y]+= m[y−1], b[y]+= b[y−1]
  for j = i+1 to g do {Right sweep line}
    m = 0, b = 0
    for y = 1 to g do
      m+= m(j,y), b+= b(j,y), m[y]+= m, b[y]+= b
    for k = 1 to g do {Bottom sweep line}
      for l = k to g do {Top sweep line}
        if k = 1 then
          m = m[k], b = b[k]
        else
          m = m[l] − m[k−1], b = b[l] − b[k−1]
        if (d(m,b) > max) then
          max = d(m,b)
```

5.2 Approx-Grid Algorithm

Our second algorithm is approximate, and builds upon the approximate schemes developed in [1] and in Section 4. In all our approximate schemes, the main subroutine is an $O(n^2 \log n)$ time algorithm for maximizing a linear discrepancy function over the space of all axis-parallel rectangles. It is easy to extract from this algorithm an $O(n)$ algorithm **Linear1D** for finding the interval in one dimension that maximizes any linear discrepancy function. Naively transferring the algorithm over rectangles to the grid would yield an algorithm running in time $O(g^4 \log g)$. We improve this to $O(g^3)$. In brief, the $O(n^2 \log n)$ procedure [1] uses two horizontal sweep lines going from bottom to top. For any position of the two sweep lines, the maximum discrepancy rectangle among rectangles bounded by these lines can be found by projecting all points onto the lower sweep line and solving a one-dimensional problem (the resulting interval defines the x-extents of the optimal rectangle). In the modified grid variant, we maintain two arrays $m[]$, $b[]$, each of size g, such that $m[i]$ stores the sum of all values $m(i,j)$ between the lower and upper sweep lines. Note that this can be maintained in constant time per entry as the upper sweep line moves. For each such movement, we run **Linear1D** on the values of $m[]$ and $b[]$. The total running time is therefore g positions of the bottom sweep line $\times$ g positions of the top sweep line $\times$ $O(g)$ for updating counts and running **Linear1D**, for a net running time of $O(g^3)$.

We describe the algorithm in detail in two parts. First we give the $O(g^3)$ gridded algorithm for linear discrepancy functions on a grid: Algorithm 4.

Algorithm 4 Algorithm **Linear-Grid**: Input is $g \times g$ grid with values $m(i,j), b(i,j)$, and linear function ℓ

```
maxd = −1
for i = 1 to g do {Left sweep line}
  Initialize m[y] = m(i,y), b[y] = b(i,y) for all y
  for j = i+1 to g do {Right sweep line}
    for y = 1 to g do
      m[y]+= m(j,y), b[y]+= b(j,y)
    (d, y_b, y_t) = Linear1D(ℓ, m[], b[]).
    if (d > maxd) then
      maxd = d; r = [i,j] × [y_b, y_t]
```

This algorithm is then used as the core subroutine in Algorithm 5.

Algorithm 5 Algorithm **Approx-Grid**

```
maxd = −1
for i = 1 to t do
  Generate ℓ_i according to (4.2).
  (d, r_i) = Linear-Grid (m[], b[], ℓ_i).
  d_i = d(r_i)
  if (d_i > maxd) then
    maxd = d_i; r' = r_i
```

The runtime of **Approx-Grid** is $O(tg^3)$, since there are t calls of **Linear-Grid** which runs in $O(g^3)$. This algorithm could also use a family of linear functions as in Agarwal *et al.* [1]. Then it would give an ϵ-approximation to the maximum discrepancy range on the grid and would run in $O(\frac{1}{\epsilon} g^3 \log g)$. We use the former version because it is more efficient as is demonstrated in Section 7

6. STREAMING ALGORITHMS

In this section we consider algorithms for the data stream model [9, 6, 2]. Here the data points arrive in some, possibly

adversarial, order. An algorithm in the streaming model has limited space, S, to catalog all the points in the stream. Unfortunately most of our results will be lower bounds.

As is typical for lower bounds in the stream model, our lower bounds are proved using reductions from communication complexity problems [12]. We denote $C_\delta(f)$ as the δ-error randomized communication complexity of function f. Also, let $C_\delta^{\text{1-way}}(f)$ be the one-way δ-error randomized communication complexity of function f.

Definition 1 (Indexing). *There are two player P_1 and P_2. P_1 has an n bit string x and P_2 has an index $j \in [n]$. The* indexing *function returns* $\text{INDEX}(x, j) = x_j$.

Definition 2 (Multi-Party Set Disjointness). *There are t players $P_1, \ldots, P_t$. P_i has an n bit string x^i. The t-party set disjointness [3] function returns* $\text{DISJ}_{n,t}(x^1, \ldots, x^t) = \bigvee_{j=1}^n \bigwedge_{i=1}^t x_j^i$.

Theorem 6.1. *For any $0 < \delta < 1/4$, $C_\delta^{\text{1-way}}(\text{INDEX}_n) = \Omega(n)$. The result remains true for instances (x, j) where x has exactly $n/2$ entries which are 1.*

Theorem 6.2 (Chakrabarti *et al.* [4]). *For any $0 < \delta < 1/4$,*

$$C_\delta(\text{DISJ}_{n,t}) = \Omega\left(\frac{n}{t \log t}\right) \quad .$$

This result remains true for the following family $\mathcal{F}$ of instances $(x^1, \ldots x^t)$ satisfying

$$|\{j : x_j^i = 1\}| = n/2t \qquad \forall i \in [t] \tag{6.1}$$

$$|\{i : x_j^i = 1\}| \in \{0, 1, t\} \qquad \forall j \in [n] \tag{6.2}$$

$$|\{j : |\{i : x_j^i = 1\}| = t\}| \leq 1 \qquad . \tag{6.3}$$

For a linear discrepancy function,

$$d(R) = \alpha \cdot m_R + \beta \cdot b_R + \gamma \quad (\alpha > 0, \beta < 0) \ .$$

we make the assumptions that $m : P \to \mathbb{N}$, $b : P \to \mathbb{N}$ and that $m^* = \max_{p \in P} m(p)$ and $b^* = \max_{p \in P} b(p)$ are constant. As a preprocessing step to any algorithm, we construct two point sets P_m and P_b: for each $p \in P$ place $m(p)$ copies of the point in P_m and $b(p)$ copies of the point in P_b. For each $p \in P_m$ let $m(p) = 1$ and $b(p) = 0$. Similarly, for each $p \in P_b$ let $m(p) = 0$ and $b(p) = 1$. Henceforth we will refer to a point p being colored *red* if $p \in P_m$, or *blue* if $p \in P_b$. Note that $|P_m \cup P_n| = O(n)$ and that this construction can be achieved in $O(n)$ time. Finally note that discrepancy for any $R \in \mathcal{R}$ is the same with respect to P as it is to $P_m \cup P_b$.

We will also consider the problem of maximizing *numerical discrepancy*. Here we assume that the P points are drawn from some universe U. For all $p \in P$, $m(p) = 1$. Then the numerical discrepancy is,

$$d(R) = m_R - \frac{|R \cap U|}{|U|} \quad .$$

Theorem 6.3. *Any* P *pass streaming algorithm returning a t relative approximation to the numerical discrepency with probability at least 3/4 requires $\Omega(n/(t^6 \text{P} \log t))$ space. Alternatively, any* P *pass streaming algorithm returning an ϵ additive approximation with probability at least 3/4 requires $\Omega(1/(\epsilon \text{P}))$ space.*

Proof. Let $(x^1, \ldots, x^{t'}) \in \mathcal{F}$ be an instance of $\text{DISJ}_{n',t'}$ where $n' = n/(3t^2)$ and $t' = 3t^2$. We will show how to transform $(x^1, \ldots, x^{t'})$ into a size $n't' = n$ instance of the numerical discrepancy problem such that t-approximating the maximum numerical discrepancy problem determines the value of $\text{DISJ}_{n',t'}$.

The stream we define consists of $n't'$ elements E where elements will come from a universe $[n'(t'+1)]$. We partition the universe into regions $R_1, \ldots, R_{n'}$ where $R_i = [(i-1)(t'+1)+1, i(t'+1)]$. Each player P_i determines a size n' subset of the elements,

$$\begin{aligned} E_i &= \{(i-1)(t'+1)+j+1 : x_j^i = 0, j \in [n']\} \\ &\quad \cup \{(i-1)(t'+1)+1 : x_j^i = 1, j \in [n']\} \ . \end{aligned}$$

Note that every region contains t' elements from E. We next show how the maximum discrepancy of the set depends on the value of $\text{DISJ}_{n',t'}$.

1. If $\text{DISJ}_{n',t'} = 1$ then the maximum numerical discrepancy is at least
$$\frac{t'}{n't'} - \frac{1}{n'(t'+1)} = \frac{t'}{n'(t'+1))} \ ,$$
since there exists an element with multiplicity t'.

2. If $\text{DISJ}_{n',t'} = 0$ then each element occurs at most once. Consider any interval $R \subseteq [n'(t'+1)]$. The numerical discrepancy in any R_i is exactly 0. Furthermore, the numerical discrepancy in any subinterval of R whose length $l \leq t'$ is at most
$$\frac{l}{n't'} - \frac{l}{n'(t'+1)} = \frac{l}{n't'(t'+1)} \ .$$
Hence the numerical discrepancy in interval R is at most $2/(n'(t'+1))$.

Hence, if an algorithm disambiguates between the maximum numerical discrepancy being greater than $t'/(n'(t'+1))$ or less than $2/(n'(t'+1))$ then the value of $\text{DISJ}_{n',t'}$ is also determined. Therefore, a relative approximation better than $\sqrt{t'/2} > t$ determines $\text{DISJ}_{n',t'}$.

Assume that there exists a P pass algorithm $\mathcal{A}$ that returns a t relative approximation to the maximum numerical discrepancy of n points (with probability at least 3/4) and uses at most $S(n,t)$ bits of memory. This algorithm gives rise to a communication protocol for $\text{DISJ}_{n',t'}$ as follows. Let the stream be ordered as $E_1, E_2, \ldots, E_{t'}$. Let $m_{i,j}$ be the memory state of $\mathcal{A}$ after the last elements from E_i has gone past in the j pass. Each player P_i constructs E_i from x^i. P_1 runs $\mathcal{A}$ on E_1 and sends the memory state $m_{1,1}$ to P_2. P_2 initializes $\mathcal{A}$ with memory state $m_{1,1}$, runs $\mathcal{A}$ on E_2 and sends the memory state, $m_{1,2}$, to P_3. They continue in this way where $m_{i,j}$ is the $(i+t'(j-1))$th message sent. The memory state $m_{t',\text{P}}$ determines a t approximation to the maximum discrepancy and, therefore, the value of $\text{DISJ}_{n',t'}$. Each message is at most $S(n,t)$ bits long and there are at most $t'\text{P}$ messages. Hence the total communication is $O(t'S(n,t)\text{P})$ bits. By appealing to Theorem 6.2, we deduce that,

$$S(n,t) = \Omega\left(\frac{n'}{t'^2 \text{P} \log t'}\right) = \Omega\left(\frac{n}{t^6 \text{P} \log t}\right) \quad .$$

The second lower bound uses a similar reduction to the first except that $t' = 3, n' = 1/(8\epsilon)$ and every point in the above construction is replaced by $8\epsilon n/3$ identical points. $\square$

Note that the above result also applies to approximating the maximum linear discrepancy where $\alpha = -\beta = 1$. This is because their may be exactly 1 baseline point at every location in the discretized universe. Although the data in this lower bound lies on the grid, it applies when the data need not lie on a grid; shifting each point slightly gives the same discrepancy values.

COROLLARY 6.1. *Any* P *pass streaming algorithm returning a t relative approximation to the maximum linear discrepency with probability at least* $3/4$ *requires* $\Omega(n/(t^6 \mathrm{P} \log t))$ *space. Alternatively, any* P *pass streaming algorithm returning an ϵ additive approximation with probability at least* $3/4$ *requires* $\Omega(1/(\epsilon \mathrm{P}))$ *space.*

The next lower bound gives a dependence on β when approximating the maximum linear discrepancy.

THEOREM 6.4. *Any one pass streaming algorithm that ϵ additively approximates the maximum linear discrepancy with probability at least* $3/4$ *requires* $\Omega(|\beta|/\epsilon)$ *space.*

PROOF. Consider an instance (x, j) of $\textsc{index}_{|\beta|/\epsilon}$. Let $w = |\beta|/(2\epsilon)$ be the number of 1's in x. We will show how to transform (x, j) into a size $n + 1$ instance of the linear discrepancy problem such that an additive ϵ-approximation of the maximum linear discrepancy problem determines the value of $\textsc{index}_{|\beta|/\epsilon}(x, j)$.

The stream starts with elements determined by P_1: for each $i \in [|\beta|/\epsilon]$ such that $x_i = 1$ there are two blue points with value i. The stream ends with one red point j. Note that the maximum value of $\alpha m_R + \beta b_R + \gamma$ is $\alpha + \gamma$ if $\textsc{index}_{|\beta|/\epsilon}(x, j) = 0$ and is $\alpha - 2\epsilon + \gamma$ if $\textsc{index}_{|\beta|/\epsilon}(x, j) = 1$.

Then, by appealing to Theorem 6.1, we deduce that the space required is $\Omega(|\beta|/\epsilon)$. □

6.1 Sampling Algorithms

We now present an algorithm that finds an additive ϵ approximation to the maximum linear discrepancy. It is based upon a sampling approach related to the construction of ϵ-nets and ϵ-approximations [8].

THEOREM 6.5. *Consider a set of points S in the plane. Let $\mathcal{R}$ be a set of axis-aligned rectangles. An ϵ-approximation is a subset A of S such that, for any $R \in \mathcal{R}$,*

$$\left| \frac{|S \cap R|}{|S|} - \frac{|A \cap R|}{|A|} \right| < \epsilon \ .$$

With probability at least $1 - \delta$, a random subset of size,

$$O\left(\frac{1}{\epsilon^2} \log\left(\frac{1}{\delta\epsilon}\right)\right)$$

is an ϵ-approximation.

THEOREM 6.6. *Let $\tau = \max(\alpha, |\beta|)$. There exists an algorithm running in time*

$$O\left(n + \left(\frac{\tau}{\epsilon}\right)^4 \log^2\left(\frac{\tau}{\delta\epsilon}\right)\left(\log\left(\frac{\tau}{\epsilon}\right) + \log\log\left(\frac{\tau}{\delta\epsilon}\right)\right)\right)$$

that returns an estimate E such that with probability at least $1 - \delta$, $|E - \max_{R \in \mathcal{R}} d(R)| \le \epsilon$.

PROOF. We first preprocess the point set as described above. This takes $O(n)$ time. We randomly construct a sample P' of points as follows: Randomly select a size $O((\alpha/\epsilon)^2 (\log(\alpha/(\delta\epsilon))))$ random subset A' of P_m. Similarly, construct B', a size $O((|\beta|/\epsilon)^2 \log(|\beta|/(\delta\epsilon)))$ random subset of P_b. Let $P' = A' \cup B'$. Estimate $d(R)$ by $\tilde{d}(R) = \alpha \frac{|A' \cap R|}{|A'|} + \beta \frac{|B' \cap R|}{|B'|} + \gamma$. Then, by Theorem 6.5, with probability at least $1 - \delta$, for all $R \in \mathcal{R}$,

$$\left| m_R - \frac{|A' \cap R|}{|A'|} \right| < \frac{\epsilon}{2\alpha} \text{ and } \left| b_R - \frac{|B' \cap R|}{|B'|} \right| < \frac{\epsilon}{2|\beta|} \ .$$

Hence with probability at least $1 - \delta$, $|d(R) - \tilde{d}(R)| \le \epsilon$ for all $R \in \mathcal{R}$. We then appeal to Lemma 4.1. □

It should be noted that a related algorithm can be used for numerical discrepancy or when the data is known to lie on a grid. Further, observe that this algorithm (when used in conjunction with the reservoir sampling technique [18]) can be viewed as a streaming algorithm that uses

$$O\left(\left(\frac{\tau}{\epsilon}\right)^2 \log\left(\frac{\tau}{\delta\epsilon}\right)\right)$$

space.

7. EXPERIMENTS

We now present a performance study of the schemes described in this paper, and compare them to prior work.

Algorithms. We implemented the simple exact algorithms **Exact** and **Exact-Grid**, as well as the approximation algorithms **Approx-Extents** and **Approx-Grid**. We compare these to two algorithms from prior work; the grid algorithm **NM-Grid** of Neill and Moore [15], and the approximation algorithm **Approx-Linear** of Agarwal *et al.* [1].

Code for **NM-Grid** was provided by the authors [13]. Their code is a black box that solves the maximum discrepancy problem and then runs N randomization tests. It only returns a range if it is statistically significant. In order to compare their algorithm we set $N = 0$ and use the discrepancy generated by **Exact-Grid**: both solve the problem exactly on the grid. The code for **NM-Grid** has an additional parameter allowing it to find an approximate solution. Neill and Moore [15] demonstrate this giving 5× to 20× speedups while only misidentifying $< 10\%$ of the regions. We did not investigate this additional parameter due to difficulties in extracting the discrepancy values. The other algorithms were implemented by us. All experiments were run on a machine with 3GHz Pentium IV processor and 1Gb SD-RAM running CentOS.

It should be noted that given the maximum discrepancy range for a given set of data, the problem remains to determine whether it is statistically significant. This is traditionally done by running about 1000 randomization tests, where the experiment is repeated on randomly generated examples from a null hypothesis. Only if the maximum discrepancy range is larger than 95% of the maximum discrepancy ranges from the randomization tests is it deemed significant. Thus the problem we describe in this paper, is repeatedly solved on the order of 1000 times in practical applications, making an efficient solution paramount for massive data sets. Here we study solving the maximum discrepancy problem once.

Data sets. We used a combination of real and synthetic data to evaluate the algorithms. We start with the `example-city-in` data file provided with the code for **NM-Grid** which contains 61291 data points of the form (x, y, b, m) where (x, y)

lies on a 175×203 integer grid distributed according to a relevant population, and where $b, m \in \{0, 1\}$. The population data represents the emergency patients' home locations in Western Pennsylvania from 1999 through 2002, and the measured data are the locations corresponding to patients from a two month span. We generate data sets of size $n = \{256, 512, 1024\}$ first by sampling (x, y) coordinates from `example-city-in`. We then let $x = x + u_1$ and $y = y + u_2$ where u_1, u_2 are drawn uniformly at random from $[0, 1]$, in order to take the points off the grid. We next generate b using an exponential distribution to model the population: we set $b = \lfloor \exp(6u) \rfloor$ where u is uniform random in $[0, 1]$. We then generate a random rectangle R of size 7×9 somewhere in the (x, y) domain. Finally we generate $m = Poisson(b \cdot f_2)$ (where $f_2 = .005$) if the point is in R and $m = Poisson(b \cdot f_1)$ (where $f_1 = .001$) if the point is outside R. The sample in R should indicate a larger discrepancy range. We tested various sizes of R and various values for f_1 and f_2, but these do not significantly affect the runtimes or accuracies of our algorithms.

Both the gridded algorithms and the approximation algorithms can tradeoff their accuracy for runtime. For the gridded algorithms we use a $g \times g$ grid where $g = \{64, 128, 256, 512\}$. For the approximation algorithms we set ϵ, the maximum error, to $\epsilon = \{.01, .1, 1, 5, 10, 20, 40, 100\}$ for **Approx-Linear** and the number of linear functions $t = \{16, 8, 4, 2, 1\}$ for **Approx-Extents** and **Approx-Grid**.

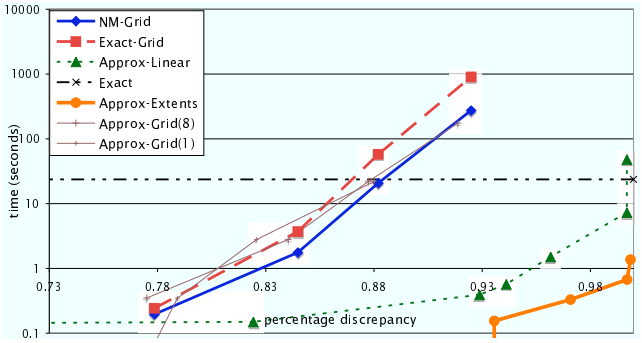
7.1 Overall Performance

Some of the algorithms we evaluate are approximate, and others are defined on a grid (which incurs its own error). To compare these algorithms, we compared their performance versus error curves. In other words, we looked at, for a fixed error bound (percentage discrepancy), how fast each algorithm ran, and for a fixed budget in time, what error was obtained by the algorithms.

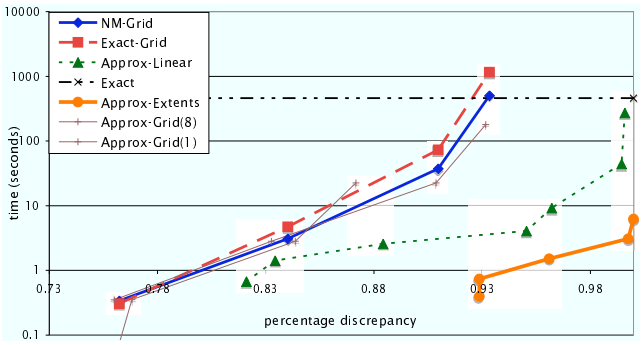
For each data set we use the returned maximum discrepancy value $d_{\mathbf{Exact}}$ of **Exact** as ground truth. We measure error as the *percentage discrepancy* for an algorithm A by $E_A = d_A / d_{\mathbf{Exact}}$ where d_A is its returned maximum discrepancy value of that specific data set. We used 30 data sets for size $n = \{256, 512, 1024\}$ which are generated as described above. We do not test larger values of n in this manor because **Exact** becomes prohibitively slow. For each algorithm at each parameter setting we average the E_A values and the runtime values over all 30 data sets of each size. Figure 4 presents the results, for all three values of n.

Approximate versus grid algorithms. The first observation we can make is that the approximation algorithms are consistently faster than the grid-based algorithms, if one wishes to get within roughly 20% of the true maximum. **Approx-Extents** performs the best overall, in all cases.

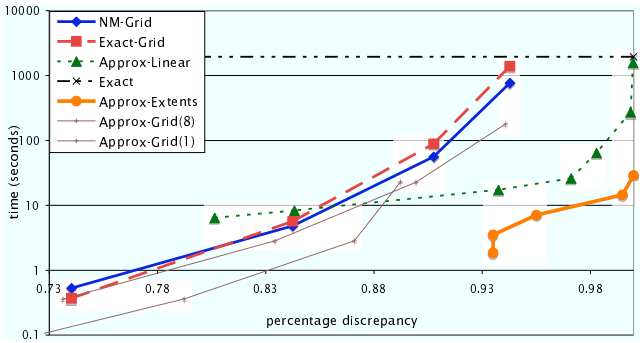
As the desired accuracy increases, the approximation algorithms **Approx-Extents** and **Approx-Linear** scale better, and thus the disparity between their performance and that of the grid algorithms increases. At the 90% accuracy level, approximation algorithms are 3400×, 94×, and 31× faster than the gridded algorithms, for point sets of size 256, 512, 1024. No gridded algorithm has an expected 95% accuracy, even for a 512×512 grid, while the **Approx-Extents** algorithm can return this expected approximation with 1 linear function. This is further demonstrated in Figure 5.



(a) $n = 256$



(b) $n = 512$



(c) $n = 1024$

Figure 4: Running time (in seconds) vs error (as a percentage of the true answer) for all the algorithms.

As n increases, this disparity decreases: the approximate algorithms degrade in performance; however, their behaviour remains superior to the grid-based schemes.

Variation in solutions. The values plotted in Figure 4 represent averages over 30 trials each of the algorithms. For both **NM-Grid** and **Exact-Grid**, the standard deviation of run times are about 25% of the total time, where as for **Approx-Extents** and **Approx-Linear** the standard deviations are only about 2% of the total time. This is likely because the true asymptotic behavior might be governed by the fact that **NM-Grid** and **Exact-Grid** vary in time depending on how they scan over the data and how many times they do the updates (if statements), whereas the approximation algorithms perform the same operations independent of the positioning of the data.

1024	Exact		Exact-Grid		NM-Grid		Approx-Grid		Approx-Linear		Approx-Extents	
99.0%	4091	284.1x	-	-	-	-	-	-	272	18.9x	14.4	1.0x
95.0%	4091	576.2x	-	-	-	-	-	-	26	3.7x	7.1	1.0x
92.5%	4091	2272.8x	1388	771.1x	768	426.7x	180	100.0x	17.3	9.6x	1.8	1.0x
90.0%	4091	2272.8x	88	48.9x	56	31.1x	22.7	12.6x	17.3	9.6x	1.8	1.0x
80.0%	4091	2922.1x	5.6	4.0x	4.8	3.4x	1.4	1.0x	6.4	4.6x	1.8	1.3x

512	Exact		Exact-Grid		NM-Grid		Approx-Grid		Approx-Linear		Approx-Extents	
99.0%	459	148.1x	-	-	-	-	-	-	44.2	14.3x	3.1	1.0x
95.0%	459	306.0x	-	-	-	-	-	-	4.1	2.7x	1.5	1.0x
92.5%	459	1176.9x	1153	2956.4x	500	1282.1x	178	456.4x	4.1	10.5x	0.39	1.0x
90.0%	459	1176.9x	73	187.2x	37	94.9x	22.4	57.4x	4.1	10.5x	0.39	1.0x
80.0%	459	1176.9x	4.7	12.1x	3.1	7.9x	1.4	3.6x	0.67	1.7x	0.39	1.0x

256	Exact		Exact-Grid		NM-Grid		Approx-Grid		Approx-Linear		Approx-Extents	
99.0%	23.5	35.1x	-	-	-	-	-	-	7.2	10.7x	0.67	1.0x
95.0%	23.5	73.4x	-	-	-	-	-	-	1.51	4.7x	0.32	1.0x
92.5%	23.5	293.8x	900	11250.0x	273	3412.5x	350	4375.0x	0.57	7.1x	0.08	1.0x
90.0%	23.5	293.8x	900	11250.0x	273	3412.5x	174	2175.0x	0.39	4.9x	0.08	1.0x
80.0%	23.5	293.8x	3.7	46.3x	1.75	21.9x	1.38	17.3x	0.15	1.9x	0.08	1.0x

Figure 5: Running time (in seconds) vs error (as a percentage of the true answer) for all the algorithms. For each algorithm, the left column is the minimal time to achieve the percentage discrepancy; the right column is the percentage slowdown over the fastest algorithm to reach that percentage discrepancy.

7.2 Behaviour of Exact-Grid

From the above graphs, we also note that **Exact-Grid**, although usually worse than **NM-Grid**, is comparable in speed. This is noteworthy since **NM-Grid** is a fairly complicated algorithm that uses a recursive prune and search to determine the optimal solution. On the other hand, **Exact-Grid** can be written in a few lines of code. Moreover, the code provided for **NM-Grid** cannot be modified easily if one wishes to use it for other functions.

As the size of the point sets increase the algorithms that are not on a grid become slower at a faster pace than those on a grid. We demonstrate this further by generating points sets with $n = 10,000$ in the same way as before and comparing the gridded algorithms: **Exact-Grid**, **NM-Grid**, and **Approx-Grid**. We also run **Approx-Extents** with $t = \{1, 8\}$. Again we plot, in Figure 6, the curves demonstrating the tradeoff between percentage discrepancy and time. However, since it takes too long to run **Exact**, we use the maximum discrepancy returned by any algorithm in place of the ground truth for each data set.

Note that **NM-Grid** still perform better than **Exact-Grid**. However, the difference is small, and might often be outweighed by the simplicity of the code for **Exact-Grid**. Also **Approx-Grid** is now working much faster than **Exact-Grid** and **NM-Grid**. This is probably because the true asymptotic behavior is somehow governed by the number of grid cells that have nonzero values, and **Approx-Grid** is faster by a factor of g/t asymptotically. So when g is large and t small this pays off.

Also notice how **Approx-Extents** continues to perform faster than **NM-Grid** and **Exact-Grid** for $g = 512$, and has much less error. Hence, for large data sets, **Approx-Extents** is better if minimizing error is important. However, if performance is critical, and error bounds on the order of 20% are tolerable, then gridded algorithms are superior, and **Approx-Grid** is the best algorithm to use.

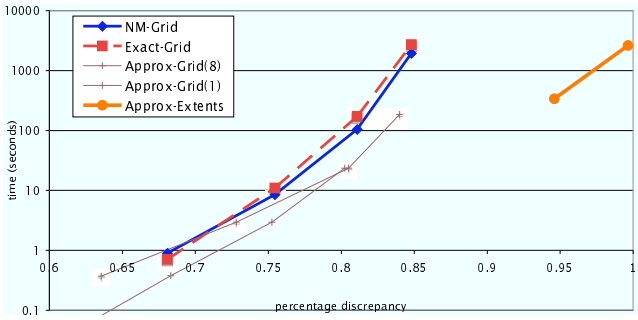


Figure 6: Running time (in seconds) vs error (as percentage of best known answer) for 10000 points.

7.3 Number of Linear Functions

Both **Approx-Linear** and **Approx-Extents** approximate the convex discrepancy function with a set of linear functions. **Approx-Extents** requires far fewer linear functions to get the same expected error. We verify this by plotting the percentage discrepancy (averaged over the data sets used above with $n = \{256, 512, 1024\}$) versus the number of linear functions used to approximate d (see Figure 7).

Approx-Linear requires about 80 linear functions to get an expected error of about 1%, where as **Approx-Extents** only needs 8: a 10× speedup. Also **Approx-Extents** never drops below 93% expected percentage discrepancy even with 1 linear function whereas **Approx-Linear** drops below 90% with less than 5 linear functions.

7.4 Variance of Percentage Discrepancy

To assess the effect of our approximation parameters on the statistical power of the resultant scan statistic, we analyzed the variation in the distribution of percentage discrep-

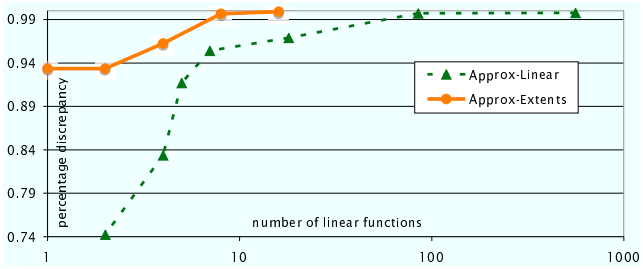


Figure 7: Number of linear functions needed by Approx-Linear and Approx-Extents to achieve and expected error.

ancy. We note that high variation in this distribution would lead to erroneous p-values and invalidate our approximation algorithms. We find consistently small variance estimates both for **Approx-Extents** and **Approx-Linear**. In fact, **Approx-Extents** consistently provides small variance for $t = \{8, 16\}$, as does **Approx-Linear** for $\epsilon = \{.01, .1\}$. The coefficient of variation drops off to zero rapidly with increase in E_A (see Figure 8). Also, the coefficient of variation tends to get smaller with increase in the number of points. A direct evaluation of the power curves involving large scale simulation experiments will be reported in future work.

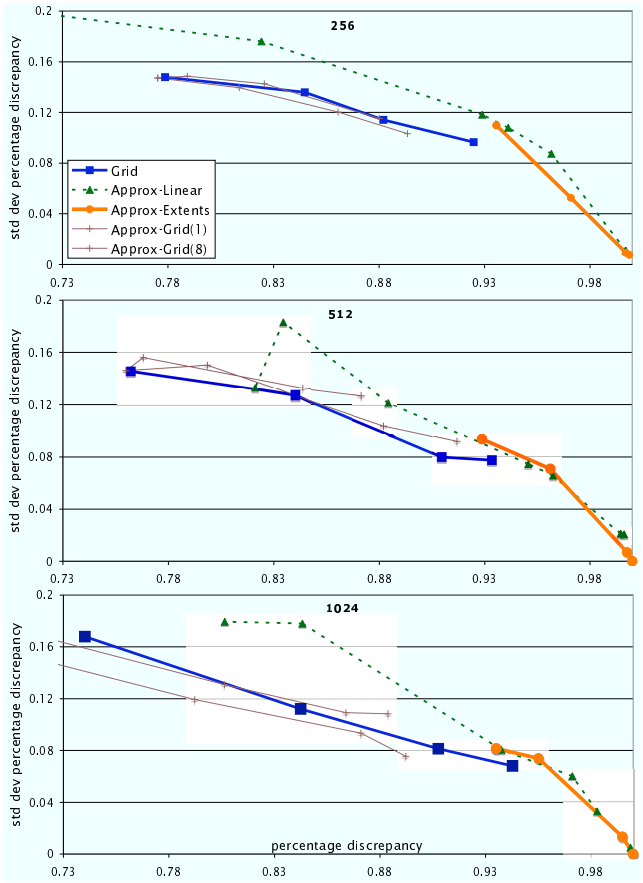


Figure 8: Variance of percentage discrepancy.

Acknowledgements

We thank the Anomaly Detection working group at SAMSI and Daniel Neill for comments and observations and Neill *et al.* for providing their code [13] for comparison.

8. REFERENCES

[1] D. Agarwal, J. M. Phillips, and S. Venkatasubramanian. The hunting of the bump: on maximizing statistical discrepancy. *Proc. 17th Ann. ACM-SIAM Symp. on Disc. Alg.*, pages 1137–1146, 2006.

[2] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Jnl. Comp. Sys. Sci.*, 58(1):137–147, 1999.

[3] Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *J. Comput. Syst. Sci.*, 68(4):702–732, 2004.

[4] A. Chakrabarti, S. Khot, and X. Sun. Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In *IEEE Conf. Comp. Compl.*, pages 107–117, 2003.

[5] M. Dwass. Modified randomization tests for nonparametric hypotheses. *An. Math. Stat.*, 28:181–187, 1957.

[6] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. An approximate L^1 difference algorithm for massive data streams. In *IEEE Symp. on Foun. of Comp. Sci.*, pages 501–511, 1999.

[7] J. H. Friedman and N. I. Fisher. Bump hunting in high-dimensional data. *Stat. and Comp.*, 9(2):123–143, April 1999.

[8] D. Haussler and E. Welzl. epsilon-nets and simplex range queries. *Disc. & Comp. Geom.*, 2:127–151, 1987.

[9] M. R. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. *TR 1998-001, DEC Sys. Res. Center*, 1998.

[10] J. Hoh and J. Ott. Scan statistics to scan markers for susceptibility genes. *Proc. Natl. Acad. Sci. USA*, 97(17):9615–9617, 2000.

[11] M. Kulldorff. A spatial scan statistic. *Comm. in Stat.: Th. and Meth.*, 26:1481–1496, 1997.

[12] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.

[13] D. Neill, A. Moore, K. Daniel, and R. Sabhnani. Scan statistics. `http://www.autonlab.org/autonweb/software/10474.html`, Sep 2005.

[14] D. B. Neill and A. W. Moore. A fast multi-resolution method for detection of significant spatial disease clusters. *Adv. Neur. Info. Proc. Sys.*, 10:651–658, 2004.

[15] D. B. Neill and A. W. Moore. Rapid detection of significant spatial clusters. In *KDD*, 2004.

[16] D. B. Neill, A. W. Moore, F. Pereira, and T. Mitchell. Detecting significant multidimensional spatial clusters. *L.K. Saul, et al., eds. Adv. Neur. Info. Proc. Sys.*, 17:969–976, 2005.

[17] C. Priebe, J. Conroy, D. Marchette, and Y. Park. Scan statistics on enron graphs. *Comp. and Math. Org. Theory*, 11(3):229–247, 2005.

[18] J. S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985.