On the Generation of 2-Dimensional Index Workloads

Joseph M. Hellerstein, U.C. Berkeley*
Lisa Hellerstein, Polytechnic University†
George Kollios, Polytechnic University‡
jmh@cs.berkeley.edu, hstein@duke.poly.edu, gkollios@db.poly.edu

Abstract

A large number of database index structures have been proposed over the last two decades, and little consensus has emerged regarding their relative effectiveness. In order to empirically evaluate these indexes, it is helpful to have methodologies for generating random queries for performance testing. In this paper we propose a natural, domain-independent approach to the generation of random queries for experimenting with indexes: choose randomly among all logically distinct queries.

We investigate this idea in the context of a widely-used and widely-studied indexing workload: range queries over 2-dimensional points. We present an algorithm that chooses randomly among logically distinct 2-d range queries. It has constant-time expected performance over uniformly distributed data, and exhibited good performance in experiments over a variety of real and synthetic data sets.

We observe nonuniformities in the way randomly chosen logical 2-d range queries are distributed over a variety of spatial properties. This raises questions about the quality of the workloads generated from such queries. To explore this further, we contrast our approach of choosing random logical range queries with previous work that generates workloads of random spatial ranges. We highlight pros and cons of the alternate approaches, and sketch directions for future work on the robust generation of workloads for studying index performance.

1 Introduction

Multidimensional indexing has been studied extensively over the last 25 years; a recent survey article [GG97] describes over 50 alternative index structures for the two-dimensional case alone. Two-dimensional indexing problems arise frequently, especially in popular applications such as Geographic Information Systems (GIS) and Computer Aided Design (CAD). A frustrating aspect of the multidimensional indexing literature is that among the many proposed techniques, there is still no "clear winner" even for two-dimensional indexing. Performance studies that accompany new index proposals typically offer little help, presenting confusing and sometimes conflicting results. Significantly absent from many of these studies is a crisp description of the distribution of queries that were used for testing the index. The need for rigorous empirical performance methodologies in this domain has been noted with increasing urgency in recent years [SRF97, GOP+97].

Recent work on generalized indexing schemes presents software and analytic frameworks for indexing that are *domain-independent*, *i.e.*, applicable to arbitrary sets of data and queries [HNP95, HKP97]. As noted in [HKP97], there is a simple logical characterization of the space of queries supported by an index over a data set D: they form a set $S \subseteq P(D)$, *i.e.*, a set of subsets of the data being indexed (here $P(D)$ denotes the power set of D). Note that this logical view of the query space abstracts away the semantics of the data domain and considers only the membership of data items in queries. In particular, this definition makes no distinction between query specifications and their outputs – a query is defined by the set of items it retrieves. This abstraction leads to simplified systems [HNP95], frameworks for discussing the hardness of indexing problems [HKP97, SM98, KT98], and domain-independent methodologies for measuring the performance of queries over indexes [KSH98].

A natural extension of this idea is to test indexes in a similarly domain-independent manner, by choosing randomly from the space of logical queries. In particular, a random logical query is simply a randomly chosen element of the set $S \subseteq P(D)$ of queries supported by the index. In this paper we consider randomly generating logical queries in this fashion for indexes that support range queries over two-dimensional points.

We begin by presenting a simple algorithm for generating random logical 2-d range queries, and we study its performance both analytically and empirically. While in the worst case the algorithm takes expected time $\theta(n^2)$ for databases of n points, in the case of uniformly distributed point sets it runs in constant expected time. We conducted experiments over standard geographic databases, and over synthetic data sets of various fractal dimensions: in all these cases, the running time of the algorithm was within a factor of two over its expected time on uniformly distributed points, suggesting that the algorithm performance is efficient and robust in practice.

*Supported by NASA grant 1996-MTPE-00099, NSF grant IRI-9703972, and a Sloan Foundation Fellowship.
†Supported by NSF grant CCR-9501660.
‡Supported by NSF grant IRI-9509527.

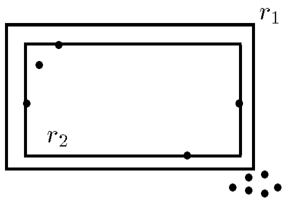

Figure 1: Random Rectangle vs. Random Query Rectangle: the outer and inner rectangles represent the same query, though they are distinct rectangles.

We continue by considering the spatial properties of randomly generated logical queries. We note that the queries in a randomly generated set of logical 2-d range queries, although likely to be diverse with respect to the set of points they contain, may not be diverse with respect to natural properties such as area, cardinality, and aspect ratio. For example, given a data set consisting of uniformly distributed 2-d points, the expected area of a random logical range query over those points is approximately 36% of the total data space (the unit square), and the variance is only approximately 3%. Thus simply testing a data structure using random logical range queries will shed little light on how the data structure performs on queries of differing area. Moreover, doing so is unlikely to expose the performance of the data structure on "typical" queries, which are likely to be more selective.

To illuminate this issue further, we contrast these observations with properties of previously-proposed query workloads. These workloads provide explicit control over domain-specific properties like cardinality and area, but are not necessarily diverse with respect to logical properties like the set of points they contain, or the cardinality of queries. Thus these workloads may not provide good "coverage" of an index's behavior in different scenarios, and may also not be representative of "typical" real queries either. These observations raise a number of research issues, which broadly amount to the following challenge: in order to allow experimentalists to do good work analyzing the performance of indexes, we need better understanding and control of the techniques for generating synthetic workloads.

1.1 Related Work

Previous work on spatial index benchmarking used queries generated from domain-specific distributions, based on geometric properties (area, position, aspect ratio, etc.) For example, many studies generated random rectangles in the plane, and used these to query the index. Note that the spaces of random rectangles in the plane and random queries are quite different: this is illustrated in Figure 1. From a spatial perspective, r_1 and r_2 are distinct two-dimensional *rectangles*. From a strictly logical perspective, however, r_1 and r_2 are identical *queries*, since they describe the same subset of data. We are careful in this paper to distinguish *random rectangles* in the plane from *random query rectangles*, which are chosen from the space of logically distinct rectangular range queries. It is inaccurate to consider random rectangles to be good representatives of randomly chosen logical queries; we discuss this issue in more detail in Section 5.

The original papers on many of the well-known spatial database indexes use domain-specific query benchmarks. This includes the papers on R-trees [Gut84], R*-trees [BKSS90], Segment Indexes [KS91], and hBΠ-trees [ELS95]. For some of those papers, the construction of the random rectangles is not even clear; for example, the R-tree paper describes generating "search rectangles made up using random numbers...each retrieving about 5% of the data", and goes into no further detail as to how those rectangles are chosen. The hBΠ-tree authors gave some consideration to the relationship of queries and data set by forming each rectangle "by taking a randomly chosen existing point as its center" [ELS95]; a similar technique was used in [BF95]. Domain-specific techniques were also used in spatial index analysis papers, including Greene's R-tree performance study [Gre89], and the various papers on fractal dimensions [FK94, BF95] (which only used square and radius query ranges). The domain-specific approach has been studied in greater detail by Pagel et al. Their work provides an interesting contrast to the work presented here, and we discuss this in depth in Section 5.

1.2 Structure of the Paper

In Section 2 we present the algorithm for generating random logical 2-d range queries. We also provide analytical results about expected running time over any data set, and expected running time over uniformly distributed data. In Section 3 we present results of a performance study over other data distributions, which produces average running times within a factor of 2 of the expected time over uniform data. Section 4 considers the spatial properties of randomly chosen range queries, and in Section 5 we discuss the properties of rectangles generated by prior algorithms that explicitly considered spatial properties. Section 6 reflects on these results and considers their implications for the practical task of empirically analyzing index performance.

2 An Algorithm for Generating Random 2-d Range Queries

2.1 Preliminaries

Definition 1 *Let $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ be a set of points in the plane. The* minimal bounding rectangle (MBR) *containing S is*

$$[\min\{x_1, \ldots, x_n\}, \max\{x_1, \ldots, x_n\}] \times [\min\{y_1, \ldots, y_n\}, \max\{y_1, \ldots, y_n\}]$$

The rectangle represented by S *is the MBR containing* S*. A query rectangle, with respect to a data set* I*, is a rectangle that is represented by a set* $S \subseteq I$*.*

Two rectangular regions contain the same set of points iff they have the same MBR. Thus the set of all query rectangles corresponds precisely to the set of logically distinct rectangular region queries.

Although distinct sets of points can represent the same rectangle, we define the canonical set of points representing a rectangle (with respect to a data set) as follows.

Definition 2 *Let* I *be a data set and let* q *be a rectangle represented by a subset of* I*. The* canonical top point of q *(with respect to* I*) is the leftmost point in* I *lying on the top boundary of* q*. Similarly, the canonical bottom point is the leftmost point lying on the bottom boundary. The canonical left and right points of* q *are the topmost points lying respectively on the left and right boundaries of* q*.*

S *is the* canonical set of points representing q *if* S *consists of the canonical bottom, top, left, and right points of* q*.*

Definition 3 *There are four types of query rectangles: 1-point, 2-point, 3-point, and 4-point. An* i-point query rectangle *is one whose canonical set consists of* i *distinct points.*

A 4-point rectangle has one data point on each of its boundaries. A 3-point rectangle has one data point on a corner of the rectangle, and one point on each of the two sides that do not meet at this corner. A 2-point rectangle is either a line segment (a degenerate rectangle with zero height or width) with a data point on each end, or a non-degenerate rectangle with data points on 2 opposite corners. A 1-point rectangle is a single point (a degenerate rectangle, with no height or width).

Definition 4 *For data set* I*, the* logical distribution on rectangular queries *is the uniform distribution on the set of all distinct query rectangles represented by subsets of* I*. A* random query rectangle *is a query rectangle that is generated according to the logical distribution on rectangular queries.*

2.2 Approaches to Generating Random Query Rectangles

We consider the problem of generating random query rectangles. It is easy to generate a random rectangular region $[x_1, x_2] \times [y_1, y_2]$, chosen uniformly from the space of all such regions. Generating a random query rectangle is not as easy. Some of the most natural approaches to generating random query rectangles do not work, or are impractical. Consider for example the idea of generating query rectangles by choosing four data points at random, and taking the minimal bounding rectangle containing those points. Under this scheme, a rectangle which has one data point on each of its four sides could only be generated by choosing precisely those four points. In contrast, a rectangle with two data points at opposite corners, and many data points in its interior, could be generated by choosing the two corner points, and any two points from its interior. As a result, this scheme will be biased towards the latter type of rectangle. Other techniques, such as "growing" or "shrinking" random rectangles until a query rectangle is achieved, have similar biases.

A naive approach to avoiding such bias is to generate all query rectangles and pick one uniformly from among them. However, since there can be $\theta(n^4)$ query rectangles, this approach is grossly impractical for sizable databases. A somewhat less naive method, which uses a range tree, correctly generates random query rectangles, but requires $\theta(n^2 \log n)$ preprocessing time for a preprocessing stage, and $\theta(\log n)$ time to generate each query following the preprocessing stage. The method uses $\theta(n^2 \log n)$ storage. Since n is typically large this is still impractical, and we omit the details of this method here. It is an interesting open question whether there is a scheme for generating random query rectangles that in the worst case uses $O(n \log n)$ space, preprocessing time $O(n \log n)$, and time $O(\log n)$ to generate each random query rectangle following the preprocessing stage. The method we present in the next section does not achieve these bounds in the worst case, but it does work quite well in practice.

2.3 The Algorithm

We present a simple Las Vegas algorithm for generating random query rectangles. The amount of time it takes to run can vary (because of randomness in the algorithm), but when it terminates it produces a correct output—a random query rectangle.

Our algorithm for generating a random query rectangle first chooses, with an appropriate bias, whether to generate a random 1-point, 2-point, 3-point, or 4-point rectangle. It then generates a rectangle of the appropriate type, chosen uniformly from all query rectangles of that type. We present the pseudocode in Figure 2. We call an iteration of the repeat loop in the algorithm a *trial.* We say that a trial is *successful* if the set Z generated in that trial is output.

Proposition 1 *On any input data set* I*, the Las Vegas algorithm in Figure 2 outputs a random query rectangle.*

Proof. We consider the query rectangle output by the algorithm to be the rectangle represented by the set Z output by the algorithm.

Consider a single trial. If the set Z generated during the trial is a 1-point set or a 2-point set, then Z must be the canonical set of points representing the MBR containing Z. If Z is a 3-point set or a 4-point set, then Z is output iff it is the canonical set of points representing the MBR R containing Z. Thus Z is output iff if is the canonical set of points for the query rectangle it represents.

We need to show that the algorithm outputs each possible query rectangle with equal probability. Let R be a j-point query rectangle for some j between 1 and 4. Let Q be the canonical set of points for R. In any trial, the set

Repeat until halted:

1. Generate a number i between 1 and 4, according to the following probability distribution:
   (a) Let $t = \binom{n}{1} + \binom{n}{2} + \binom{n}{3} + \binom{n}{4}$. For $j \in \{1, \ldots, 4\}$, Prob$[i = j] = \frac{\binom{n}{j}}{t}$.
2. (a) Generate a set Z containing i points of I uniformly at random from among all i-point sets in Z
   (b) If $i = 1$ or $i = 2$ output Z.
   (c) If $i = 3$ or $i = 4$, compute the MBR R containing Z, and check whether Z is the canonical set of points for R. If so, output Z.

Figure 2: Las Vegas Algorithm for Generating Queries from the Logical Distribution

Z will be equal to Q iff i is chosen to be equal to j in line 1, and Z is chosen to be equal to Q in line 2. Thus the probability that, in a given trial, Z is the canonical set of points for R is $\frac{\binom{n}{i}}{t} * \frac{1}{\binom{n}{i}} = \frac{1}{t}$.

The probability that a trial is successful is therefore $\frac{r}{t}$, where r is the number of distinct query rectangles R. The conditional probability of generating a particular query rectangle R in a trial, given that the trial is successful, is $\frac{1}{t}/\frac{r}{t} = \frac{1}{r}$. Since the algorithm outputs a query rectangle as soon as it has a successful trial, each distinct query rectangle R is output by the algorithm with the same probability of $\frac{1}{r}$. □

Proposition 2 *On any input data set I consisting of n points, the expected number of trials of the Las Vegas algorithm in Figure 2 is $\frac{t}{r}$, where r is the number of distinct query rectangles represented by subsets of I, and $t = \binom{n}{1} + \binom{n}{2} + \binom{n}{3} + \binom{n}{4}$.*

Proof. This follows immediately from the proof of Proposition 1, in which it is shown that the probability that a given trial is successful is $\frac{r}{t}$. □

2.4 Expected Performance over Worst-Case Data Sets

We now do a worst-case analysis of the expected running time of the algorithm. Our analysis will assume that all the points can be stored in memory; when this is not the case there is an additional constant overhead for random I/Os in a file. and installed in many personal Since each trial will consist of only a few I/Os, we do not expect this overhead to be significant even for very large point sets.

Proposition 3 *The expected number of trials of the Las Vegas algorithm of Figure 2 (on the worst-case input) is $\theta(n^2)$.*

Proof. By Proposition 2, the expected number of trials is $\frac{t}{r}$. This quantity depends on r, which can vary according to the configuration of the points in the data set. We now bound r. On any n-point data set, the number of 1-point query rectangles is n and the number of 2-point query rectangles is $\binom{n}{2}$. Therefore, on any data set, r is at least $n + \binom{n}{2}$. This bound is achieved by data sets in which all points fall on a line [1], because such data sets define no 3-point and 4-point rectangles. Since $t = \theta(n^4)$, the proposition follows. □

Proposition 4 *There is an implementation of the Las Vegas algorithm of Figure 2 that runs in expected time $\theta(n^2)$ (on the worst-case input) with $\theta(n \log n)$ preprocessing.*

Proof. Consider the following implementation of the algorithm. It requires that the data set be preprocessed. The preprocessing can be done as follows. Form two sorted arrays, the first containing the data set sorted by x coordinate (with ties broken by sorting on the y coordinate), and the second containing the data set sorted by y coordinate (with ties broken on the x coordinate). With each data point p in the first array, store a link to the corresponding point in the second array. This preprocessing can easily be implemented to run in time $O(n \log n)$.

The implemention of a trial is as follows. Generate the points Z using the first sorted array and calculate the MBR R containing them. If Z contains 1 or 2 points, then output Z and end the trial.

Otherwise, begin checking whether Z is the canonical set of points for R by first checking whether all points in Z lie on the boundary of R and whether each boundary of R contains exactly one point of Z. If not, end the trial without outputting Z.

If Z passes the above tests, it is sufficient to check, for each boundary of R, whether Z contains the canonical point on that boundary. To check a boundary of R, find the point p in Z lying on that boundary. Then access the point p' immediately preceding p in the first array (for the left and right boundaries) or immediately following p in the second array (for the top and bottom boundaries). Because of the links from p in the first to p in the second array, p' can be accessed in constant time for each of the boundaries. Check whether p' lies on the same boundary of R as p. If not, p is the canonical point on the relevant boundary of R, otherwise it is. If Z contains the canonical point for each boundary of R, then output Z. Otherwise, end the trial without outputting Z.

[1]Another data set where the same bound is achieved by our algorithm is the grid data set ($I = \{(i,j) : 1 \leq i, j \leq k, n = k^2\}$). It 's easy to see that the number of different range queries for this data set is $\theta(n^2)$, since any range query is defined by only one or two points.

Since each of the steps in the above implementation of a trial takes constant time, each trial takes constant time. By the previous proposition, the expected number of trials is $\theta(n^2)$, and thus the expected running time is $\theta(n^2)$. □

Fortunately, the expected number of trials is significantly lower than $\theta(n^2)$ for many data sets. For example consider the "plus" data set, where the data points lie on two line segments, a horizontal and a vertical, intersecting each other in the middle. Thus, the data set is divided into four partitions (up, down, left, right) of approximately the same size. For this data set it can be shown that a set Z generated in Step 2(a) of the above algorithm is a canonical set with significant probability. In particular, if Z is a 4-point set, the probability that Z is a canonical set is equal to the probability that the four points belong to different partitions. Hence:

$$P(Z\ is\ a\ 4-pt\ canonical\ set) \;=\; 1 \cdot \frac{3}{4} \cdot \frac{2}{4} \cdot \frac{1}{4} \;=\; \frac{3}{32} \;>\; \frac{1}{11}$$

Since for the other query types (eg. 1-,2- or 3-point) the above probability is even higher, it follows that the expected number of trials per query rectangle for this data set is most 11.

In the following section, we prove that on a uniform data set, the expected number of trials before the algorithm outputs a query rectangle is less than 6, independent of the number of points in the data set. In Section 3 we present empirical results on both artificial and real data sets, showing that the average number of trials is similarly small.

2.5 Algorithm Analysis: Uniform Data Sets

Definition 5 *A* uniform data set *is a set of points drawn independently from the uniform distribution on the points in the unit square.*

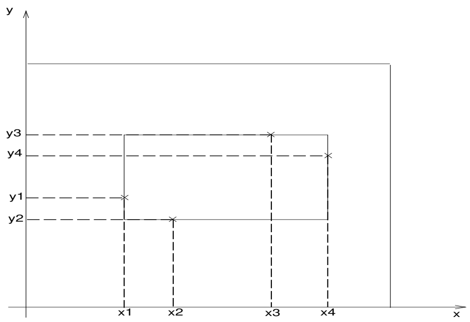

Figure 3: Query Rectangle

Proposition 5 *Let I be a uniform dataset of size n. The expected number of trials of the Las Vegas algorithm in Figure 2 on data set I is less than 6. As n increases, the expected number of trials approaches 6. (The expectation is over the choice of points in I.)*

Proof. Let $\{p_1, p_2, p_3, p_4\}$ be a set of four distinct random points chosen from I ($p_i = (x_i, y_i)$). Since the points in I are chosen from the uniform distribution on the unit square, the probability that any two have a common x or y coordinate is 0. Therefore, we assume without loss of generality that all four points have distinct x coordinates and all have distinct y coordinates.[2]

Let S_4 be the set of permutations of 4 elements x_1, x_2, x_3, x_4. We associate with the permutation $\sigma = (x_i, x_j, x_k, x_l)$ the event $A_\sigma = \{x_i < x_j < x_k < x_l\}$. Clearly the A_σ's partition the sample space, so if B is the event that $\{p_1, \ldots, p_4\}$ are the canonical set of points of a query rectangle, then by Bayes' formula

$$P(B) = \sum_{\sigma \in S_4} P(B|A_\sigma)P(A_\sigma)$$

By symmetry $P(A_\sigma) = \frac{1}{4!}$ for every $\sigma \in S_4$ and $P(B|A_\sigma)$ is the same for any $\sigma \in S_4$, so $P(B)$ is actually $P(B|A_\sigma)$ where σ is any permutation. Taken in particular $\sigma_0 = (x_1, x_2, x_3, x_4)$ (see Figure 3), then

$$\begin{aligned} P(B|A_\sigma) &= P(y_1, y_4\ between\ y_2, y_3) \qquad \text{by symmetry} \\ &= 2P(y_2 < y_1, y_4 < y_3) \\ &= 2\int_0^1 dy_3 \int_0^{y_3} dy_2 \int_{y_2}^{y_3} dy_1 \int_{y_2}^{y_3} dy_4 \end{aligned}$$

[2]In practice, the probability will not be zero because of finite precision in the representation of points on the unit squre, but it will usually be small enough so that its effect on the expected number of trials will be negligible.

Data Sets	Algorithm Performance		Query Properties (average over 1000 queries)		
	Average #Trials per Query Rectangle	*Seconds for 1000 Queries*	*Mean Area*	*Mean Cardinality*	*Mean Aspect Ratio*
Uniform	6.033	3.61	0.36475	35.67%	1.170289
Double Cluster	9.530	4.04	0.3015	39.28%	1.124432
LBCounty	7.440	3.20	0.17285	36.62%	1.138304
MGCounty	6.450	2.95	0.1351	37.19%	1.234010
Lévy 1.46	12.92	4.64	0.06955	33.60%	4.993343
Lévy 1.68	8.741	3.87	0.08729	35.11%	0.360295

Table 1: Performance of the Randomized Algorithm, and Properties of the Random Range Queries

$$= 2\int_0^1 dy_3 \int_0^{y_3} dy_2 (y_3 - y_2)^2$$
$$= \frac{1}{6}$$

A similar argument shows that three distinct randomly selected points from I have probability $\frac{2}{3}$ of being a canonical set. One or two randomly selected points have probability 1 of being a canonical set.

Let I_j denote the set of all subsets of size j of the data set I, for $j = 1, 2, 3, 4$. For all sets S in I_j, let $\chi_S = 1$ if the points in S are the canonical points for the rectangle they represent, and $\chi_S = 0$ otherwise. The expected number r of query rectangles in I can be bounded as follows:

E[Number of Query Rectangles]
$= \sum_{j=1}^{4} E$[Number of j-point Query Rectangles]
$= \sum_{j=1}^{4} E[\sum_{S \in I_j} \chi_S]$
$= \sum_{j=1}^{4} \sum_{S \in I_j} E[\chi_S]$
$> \sum_{j=1}^{4} \sum_{S \in I_j} (1/6)$
$= \sum_{j=1}^{4} \frac{1}{6}\binom{n}{j}$.

Since the expected number of trials of the Las Vegas algorithm is $\frac{t}{r}$, where $t = \sum_{j=1}^{4} \binom{n}{j}$, the expected number of trials on data set I is less than 6. As n increases, the expected number of trials approaches 6, because for large n, $\binom{n}{4} >> \binom{n}{j}$ for $j = 1, 2, 3$. □

3 Empirical Performance

In order to validate the efficiency of the Las Vegas algorithm in practice, we conducted several experiments over standard synthetic and geographic data sets. We also used a technique called Lévy flights [SZE95] to generate sets of points with fractal dimension similar to those found in real-life data [FK94, BF95]. All data sets were normalized to the unit square. In particular, we used the following 2-dimensional data sets:

- Uniform: In this data set 100000 points are uniformly distributed in the unit square.
- Double Cluster: This data set contains two clusters of approximately 50000 points each, one centered near the origin, and the other close to the point $(1, 1)$. The points in each cluster follow a 2-dimensional independent Gaussian distribution with $\sigma^2 = 0.1$.
- LBCounty: This data set is part of the TIGER database [USBC92] and contains 53145 road intersections from Long Beach County, CA.
- MGCounty: The same as above from Montgomery County, MD containing 39231 points.
- Lévy 1.46: This is a data set of 100000 points generated by the Lévy flight generator, with fractal dimension approximately 1.46.
- Lévy 1.68: The fractal dimension here was close to 1.68 with the same number of points as above.

is

We used a main memory implementation of the algorithm to generate 1000 random range queries for each data set and the results are shown in Table 1. The cost of the algorithm is expressed both as the number of trials per random range query, and the elapsed time. Recall that all datasets are the same size except for the two real datasets. For the Uniform data set the cost was very close to 6 trials, as anticipated. For the Double Cluster data set the cost was a little higher but remained small. The algorithm performed very well for the real-life data sets even though these data sets are clearly non-uniform. Finally, for the Lévy Flights data sets, the one with fractal dimension 1.46 had the higher cost[3]. The results of the experiments indicate that our randomized algorithm is quite efficient in practice, and robust across a variety of distributions.

on

[3]This matches the intuition behind the use of fractal dimension in [FK94]: as discussed in Section 2.4, data laid out on a line (fractal dimension 1) results in more trials than data laid out uniformly in 2-space (fractal dimension 2), hence one should expect worse performance with lower fractal dimension. Unfortunately in further experiments we found counter-examples to this trend; additional study of fractal dimension and query generation seems interesting, but is beyond the scope of this paper.

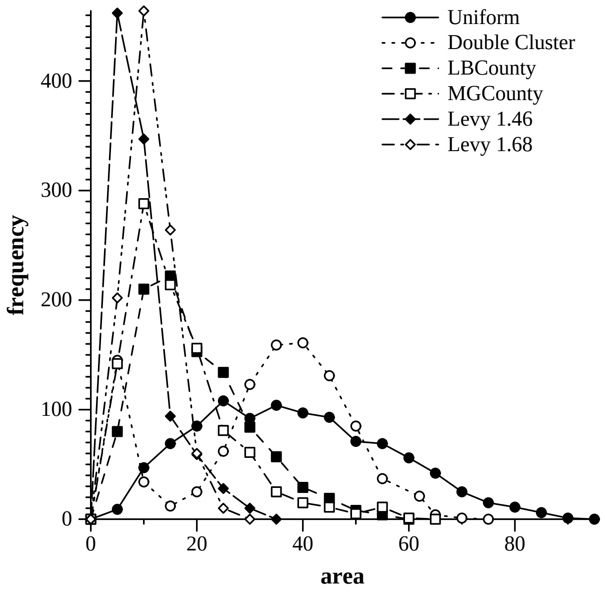

Figure 4: Frequencies of query-area for various distributions.

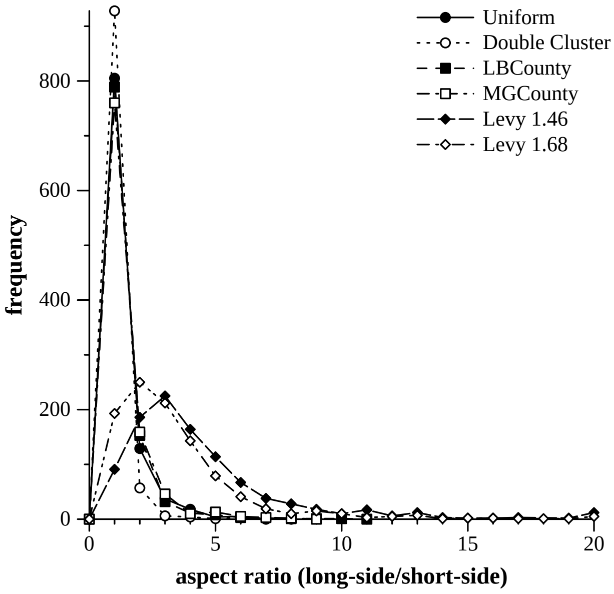

Figure 5: Frequencies of query-aspect-ratio for various distributions.

4 Properties of Random Range Queries

In this section we consider various properties of the random range queries generated by our algorithm; in particular we consider the area, cardinality and aspect ratio of the query rectangles. Note that, given a data set, the areas of the query rectangles defined by that data set may not be uniformly distributed. In fact, as we show here, the contrary is often true.

We first prove a result concerning uniform datasets.

Proposition 6 *Let I be a uniform dataset. Let $\vec{z} = (p_1, \ldots, p_4)$ be four distinct random points chosen from I. The expected area of the rectangle defined by $\vec{z}$, given that $\vec{z}$ is the set of canonical set of points of a rectangle, is $\eta = 0.36$, and the variance is $\sigma^2 = 0.03$*

Proof. Let B be the event that $\vec{z}$ is the canonical set of points for a query rectangle, and let A_σ be the event associated with the permutation σ as above (Proposition 5). Then for $\sigma_0 = (x_1, x_2, x_3, x_4)$ the area of the generated rectangle is $g(\vec{z}) = |x_4 - x_1||y_3 - y_2|$. Hence, the expected area is:

$$\begin{aligned}
Expected\ Area &= E\{g(\vec{z})|B\} = \int_B \frac{g(\vec{z})d\vec{z}}{P(B)} = 6 \times \int_B g(\vec{z})d\vec{z} \\
&= 6 \times \sum_\sigma \int_{B\cap A_\sigma} g(\vec{z})d\vec{z} \\
&= 6 \times 4! \times \int_{B\cap A_\sigma} g(\vec{z})d\vec{z} \\
&= 6 \times 4! \times 2 \times \int_0^1 dx_4 \int_0^{x_4} dx_1 \int_{x_1}^{x_4} dx_2 \int_{x_2}^{x_4} dx_3 (x_4 - x_1) \int_0^1 dy_3 \int_0^{y_3} dy_2 \int_{y_2}^{y_3} dy_1 \int_{y_2}^{y_3} dy_4 (y_3 - y_2) \\
&= \frac{36}{100}
\end{aligned}$$

Similarly, the variance is computed equal to $\sigma^2 = 0.03$. □

If we run our algorithm on a uniform dataset of any reasonably large size, then with high probability, the rectangle returned by the algorithm will be a 4-point query rectangle (because in each trial, the probability that the algorithm generates a 4-point set is very high, and on a uniform dataset, the probability that a 4-point trial is successful is 1/6). Thus, by Proposition 6, the expected area of a rectangle generated by running our algorithm on a reasonably large uniform dataset is approximately .36, with variance approximately .03. A similar analysis shows that the expected aspect ratio of the generated rectangle (defined as the ratio of x-side over y-side) will be approximately 1.2 with variance approximately 0.96.

In light of the above results, we were interested in the spatial properties (namely area and aspect ratio) for all of our data sets. Table 1 shows the mean values of the three spatial properties in experiments of 1000 random range

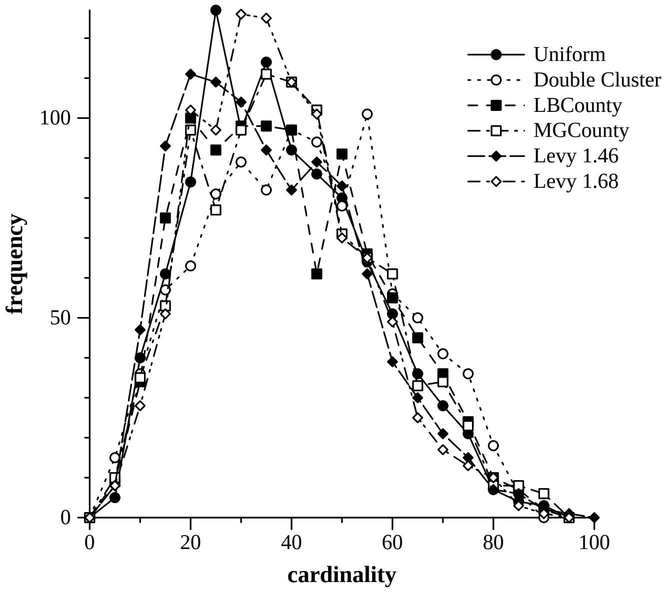

Figure 6: Frequencies of query-cardinality for various distributions.

queries over each of the various data sets. Figure 4 shows the frequency graphs of the areas of the queries for each data set. From Figure 4 is clear that the distribution of the area of the generated random range queries is highly dependent on the underlying data set. The Uniform data set gives a variety of large-area range queries, whereas for the fractal data sets most of the query rectangles are smaller than 10% of the total area. In Figure 5 we present the frequencies of the aspect ratio (long-side/short-side) for the same set of range queries, and in Figure 6 we present the frequencies of cardinalities. Note that cardinality seems to be less sensitive to data distribution than the spatial properties.

One conclusion of this analysis is that a domain-independent "logical" query generator may have unusual properties, and those properties may vary widely in data-dependent ways. This is a cautionary lesson about generating test queries in a domain-independent fashion, which undoubtedly applies to the generation of queries over indexes in domains other than spatial range queries. A natural alternative is to consider using domain-specific query generators when possible. To explore this idea further, we proceed to examine domain-specific spatial query generation techniques from the literature, and demonstrate that they raise complementary and potentially problematic issues of their own.

5 Alternative Techniques for Query Generation

As remarked above, other authors have considered domain-dependent query distributions. The most extensive exploration of such distributions is to due Pagel et al. [PSTW93] They considered the expected cost of 2-d range queries over distributions other than the logical distribution; in particular they proposed four specific classes of distributions.

The first class, called $\mathcal{QM}_1$, consists of distributions that are uniform over the space of all squares of area k (for some fixed k) whose centers fall "within the data space", i.e., within the spatial boundaries of the existing data set. This is a natural scheme with clear domain-specific properties. However, using a distribution from this class to test the performance of an index has notable drawbacks. The workloads generated from this distribution are very sensitive to the data distribution, and may not be statistically sound in *covering* the possible behaviors of the index. For example, if the area k is a relatively small fraction of the data space (as is common in many previous studies), and if the data is clustered in a small region of the data space, then random squares chosen from $\mathcal{QM}_1$ are likely to be empty. Repeatedly testing an index on empty queries may not yield the kind of information desired in experiments.

The other classes of distributions proposed by Pagel et al. attempt to overcome such drawbacks. Each of these classes is defined in terms of a density function D on the data space. One, called $\mathcal{QM}_2$, is like $\mathcal{QM}_1$ in that it consists of distributions over the space of squares of area k, for some fixed k, whose centers are within the the data space. In $\mathcal{QM}_2$ distributions, though, the probability of each square is weighted according to the value of D at the square's center. The remaining two classes of distributions, $\mathcal{QM}_3$ and $\mathcal{QM}_4$ are over the space of squares that enclose a fixed fraction s of the total probability mass (as defined by D), for some s, whose centers are within the data space. Distributions in $\mathcal{QM}_3$ are uniform over such squares, and distributions in $\mathcal{QM}_4$ weight each square according to the value of D at its center.

Pagel et al. point out that the expected cost of performing a random rectangular query in a particular LSD-tree (or R-tree, or similar data structure) is equal to the sum, over all leaves in the tree, of the probability that a random rectangle intersects the rectangular region defined by the points stored in that leaf [PSTW93]. They use this fact to compute analytically, for particular trees, the expected cost of a random rectangular query drawn from a distribution in $\mathcal{QM}_1$. In contrast, to compute the expected cost of a rectangular query drawn from the logical distribution on rectangular queries, we would use an empirical approach; we would generate random queries, determine the cost of each, and take an average. The logical distribution is a difficult one to work with analytically; unlike distributions in $\mathcal{QM}_1$, it is a discrete distribution defined by a data set.

In later experimental work [PSW95, PS96], Pagel et al. use only distributions in $\mathcal{QM}_1$. As they point out, exactly computing expected query cost with respect to distributions in $\mathcal{QM}_3$ and $\mathcal{QM}_4$ can be difficult [PSTW93]. In addition, distributions in $\mathcal{QM}_2$, $\mathcal{QM}_3$, and $\mathcal{QM}_4$ are defined in terms of a distribution D on the data space. Since a real data set does not directly provide a distribution D on the data space, the last three query distributions are not well-defined "in the field," that is, for real data sets (although in some cases it may be feasible to model the distribution D).

In short, the Pagel query distributions present a mirror image of our logical query distribution. The Pagel distributions' domain-specific properties are easily controlled, and they are amenable to analytic analyses – they appeal to the intuitive properties of 2-space. By contrast, the logical properties of these distributions (e.g. query cardinality) are sensitive to data-distribution and often wildly skewed, and some of the available techniques are inapplicable to real data sets. The domain-specific and logical distributions have very different strengths and weaknesses for studying indexes, and we elaborate on this in the next section.

6 On Randomization, Benchmarks, and Performance Studies

One advantage of domain-dependent distributions like $\mathcal{QM}_1$ is that they attempt to model a class of user behavior. This is the goal of so-called *Domain-Specific Benchmarking* [Gra93], and is clearly a good idea: one main motivation for benchmarking is to analyze how well a technique works overall for an important class of users. In the case of spatial queries it makes sense to choose square queries of small area, for example, since users with graphical user interfaces are likely to "drag" such squares with their mouse. This is natural when trying to get detail about a small portion of a larger space shown on screen – e.g., given a map of the world one might drag a square around Wisconsin.

But random workloads can also be used to learn more about a technique – in particular to "stress test" it. To do this, one wants to provide a diversity of inputs to the technique, in order to identify the inputs it handles gracefully, and those it handles poorly. For indexes, a workload is described logically as a set of queries (subsets of the data), and the indexing problem can be set up as a logical optimization problem: an indexing scheme [HKP97] should cluster items into fixed-size buckets to optimize some metric on bucket-fetches over the workload (e.g. minimize the total number of bucket fetches for a set of queries run in sequence). It is by no means clear that domain-specific considerations help set up these stress tests – logical workloads seem more natural for this task.

One can easily quibble with both of these characterizations, however. Clearly the (domain-dependent) $\mathcal{QM}_1$ workloads do a bad job of modeling user behavior when they generate many empty queries – in essence, they ignore the fact that users often do care about logical properties like cardinality. Conversely, the stress imposed by the logical workload on a 2-d index is questionable if most of the queries are actually large and squarish as in Table 1: anecdotal evidence suggests that long, skinny queries are the real nemesis of spatial indexes like R-trees, which were designed to cluster points into squarish regions[4].

What then can one conclude about the pros and cons of logical and domain-specific query generation? In short, that regardless of the technique an experimentalist uses to generate queries, she needs to understand and be able to control the distribution of a workload over domain-specific and logical properties of relevance. Identifying these properties is not a simple problem, and understanding how to generate queries that are well distributed over them is a further challenge. One strong conclusion of our work is that this process has been little explored in index experiments to date, and is in fact fairly complex.

Two-dimensional range queries are probably the best-understood, non-trivial indexing challenge. Thus a natural direction for future work is to attempt and merge the insights from the previous sections to develop a malleable, well-understood toolkit for experimenting with 2-d indexes. We conclude by exploring some ideas in this direction, and raising questions for generating queries in other domains.

7 Conclusion and Future Directions

In this paper we highlight a new approach to generating random queries for index experimentation, which uses a logical distribution on queries. We present an algorithm to generate queries from this distribution, showing that it has good expected performance for some distributions, and good measured performance over a variety of real and synthetic data. A remaining open problem is to devise an algorithm for this task with good guaranteed time- and space-efficiency over all point distributions.

The very different properties of this distribution and previously proposed distributions suggest a new line of research: developing techniques to allow experimentalists to easily understand and control various properties of their workload. A direct attack on this problem is to first map desired domain-dependent properties into a distribution over the space of logical queries, and then devise an efficient algorithm for choosing from that distribution. In general this seems quite difficult, but the original problem tackled in this paper is a simple instance of this approach: it specifies a distribution over all queries $P(D)$, with 100% of the distribution falling into the (domain-specific) category of query rectangles. Perhaps this direct approach will prove tractable for other simple spatial distributions as well.

In addition to this direct theoretical challenge, a variety of potentially useful heuristics suggest themselves as well. Rather than map spatial properties to a distribution over the logical queries, one can simply partition the data set on spatial properties, and use the uniform logical distribution over the partitions. For example, to generate a distribution with smaller average query area, one can tile the data space and run our Las Vegas algorithm over data partitions that correspond to tiles. This seems rather *ad hoc*, but is perhaps easier to reason about logically than the totally domain-specific techniques of Pagel, et al. It is also applicable to extant "real-world" data sets.

[4] In fact our interest in the subject of random 2-d range queries was sparked by an attempt to experimentally validate this hypothesis about R-trees!

Another heuristic is to introduce new points into the query generator's data set in order to achieve biases on spatial properties. For example, to increase the variance in aspect ratio of a uniformly distributed point set, one can insert clusters of points along the extremes of one axis or the other. One can then run the resulting queries over the original data set. This may allow for a better control over the resulting query mix than a domain-specific spatial technique.

Acknowledgments

We thank Dina Kravets, Joe Kilian, and Boris Aronov for useful discussions, and Jeff Sidell for his Lévy flight generator.

References

[BF95] Alberto Belussi and Christos Faloutsos. Estimating the Selectivity of Spatial Queries Using the 'Correlation' Fractal Dimension. In *Proc. 21st International Conference on Very Large Data Bases*, Zurich, September 1995, pages 299–310.

[BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: An Efficient and Robust Access Method For Points and Rectangles. In *Proc. ACM-SIGMOD International Conference on Management of Data*, pages 322–331, Atlantic City, May 1990.

[Come79] D. Comer. The Ubiquitous B-Tree. *Computing Surveys*, 11(2):121–137, June 1979.

[ELS95] Georgios Evangelidis, David B. Lomet, and Betty Salzberg. The hBΠ-tree: A Modified hB-tree Supporting Concurrency, Recovery and Node Consolidation. In *Proc. 21st International Conference on Very Large Data Bases*, Zurich, September 1995, pages 551–561.

[FK94] Christos Faloutsos and Ibrahim Kamel. Beyond Uniformity and Independence: Analysis of R-trees Using the Concept of Fractal Dimension. In *Proc. 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 4–13, Minneapolis, May 1994.

[GG97] V. Gaede and O. Gunther. Multidimensional Access Methods. *ACM Computing Surveys*, 1997. To appear.

[GOP+97] O. Gunther, V. Oria, P. Picouet, J.-M. Saglio, and M. Scholl. Benchmarking Spatial Joins A la Carte. In J. Ferriee (ed.), *Proc. 13e Journees Bases de Donnees Avancees*, Grenoble, 1997.

[Gra93] Jim Gray. *The Benchmark Handbook for Database and Transaction Processing Systems, Second Edition.* Morgan Kauffman, San Francisco, 1993.

[Gre89] Diane Greene. An Implementation and Performance Analysis of Spatial Data Access Methods. In *Proc. 5th IEEE International Conference on Data Engineering*, pages 606–615, 1989.

[Gut84] Antonin Guttman. R-Trees: A Dynamic Index Structure For Spatial Searching. In *Proc. ACM-SIGMOD International Conference on Management of Data*, pages 47–57, Boston, June 1984.

[HKP97] Joseph M. Hellerstein, Elias Koutsoupias, and Christos H. Papadimitriou. On the Analysis of Indexing Schemes. In *Proc. 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 249–256, Tucson, May 1997.

[HNP95] Joseph M. Hellerstein, Jeffrey F. Naughton, and Avi Pfeffer. Generalized Search Trees for Database Systems. In *Proc. 21st International Conference on Very Large Data Bases*, Zurich, September 1995.

[KS91] Curtis P. Kolovson and Michael Stonebraker. Segment Indexes: Dynamic Indexing Techniques for Multi-Dimensional Interval Data. In *Proc. ACM-SIGMOD International Conference on Management of Data*, pages 138–147, Denver, June 1991.

[KSH98] Marcel Kornacker, Mehul Shah, and Joseph M. Hellerstein. amdb: An Access Method Debugging Toolkit. In *Proc. ACM-SIGMOD International Conference on Management of Data*, Seattle, June 1998.

[KT98] Elias Koutsoupias and David Scot Taylor. Tight bounds for 2-dimensional Indexing Schemes. In *Proc. 17th ACM PODS Symposium on Principles of Database Systems*, Seattle, 1998.

[PS96] Bernd-Uwe Pagel and Hans-Werner Six. Are Window Queries Represesntative for Arbitrary Range Queries? In *Proc. 15th ACM PODS Symposium on Principles of Database Systems*, pages 151–160, 1996.

[PSTW93] Bernd-Uwe Pagel, Hans-Werner Six, Heinrich Toben, and Peter Widmayer. Towards an Analysis of Range Query Performance in Spatial Data Structures. In *Proc. 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 214–221, Washington, D. C., May 1993.

[PSW95] Bernd-Uwe Pagel, Hans-Werner Six, and Mario Winter. Window Query-Optimal Clustering of Spatial Objects In *Proc. 14th ACM PODS Symposium on Principles of Database Systems*, pages 86–94, 1995.

[PS85] Franco P. Preparata and Michael Ian Shamos. Computational Geometry: An Introduction Springer Verlag, New York, 1985.

[SM98] Vasilis Samoladas and Daniel P. Miranker. A Lower Bound Theorem for Indexing Schemes and its Application to Multidimensional Range Queries In *Proc. 17th ACM PODS Symposium on Principles of Database Systems*, Seattle, 1998.

[SRF97] Timos K. Sellis, Nick Roussopoulos, and Christos Faloutsos. Multidimensional Access Methods: Trees Have Grown Everywhere. In *Proc. 23rd International Conference on Very Large Data Bases*, Athens, August 1997.

[SZE95] Shlesinger, Zaslavsky, and Frisch (Eds.), editors. *Levy Flights and Related Topics in Physics.* Springer-Verlag, 1995.
Standard

[USBC92] U.S. Bureau of the Census. TiGER Files(TM), 1992 Technical Documentation, 1992.