Indexing Schemes for Random Points

Elias Koutsoupias* David Taylor†

Abstract

We investigate the tradeoff between storage redundancy and access overhead for indexing random d-dimensional point sets. We show that with high probability a range-query workload of n random points has polylogarithmic trade-off; more precisely, there is a constant $c_{B,d}$ such that every indexing scheme with storage redundancy $c_{B,d} \log^{d-1} n$ has the worst possible access overhead (equal to the block size B). We also show that this result is almost tight and that the trade-off even exhibits a threshold behavior: on a random set of points, expected storage redundancy $O(\log^{d-1} n)$ achieves access overhead $2d-1$.

1 Introduction

Indexing schemes introduced in [3] attempt to capture the intrinsic difficulty of storing large database workloads for efficient retrieval of requested data from secondary memory. Informally, an indexing scheme is a way to organize the data into a collection of disk blocks that facilitates efficient retrieval of data. In an ideal indexing scheme, when answering a query, all items of each retrieved block are useful; in the other extreme, each retrieved block contains only one relevant item. The question we address in this paper is whether a random Euclidean workload admits indexing schemes that come close to the ideal.

Indexing schemes are defined with respect to database workloads: a workload $W = (D, I, \mathcal{Q})$ consists of a domain D, a finite subset $I \subseteq D$ (the instance), and a set $\mathcal{Q} \subseteq 2^I$ of queries. The workloads considered here are the d-dimensional range-queries, for which I is a finite set of points of the d-dimensional Euclidean space ($D = R^d$), and $\mathcal{Q}$ consists of all intersections of I with rectilinear d-dimensional rectangles. Intuitively, a range-query is defined by a minimum and maximum value in each dimension, and the points that fall within the given range in every dimension make up the query.

For a workload W, an indexing scheme is simply a collection of blocks of data; each block consists of at most B elements of I. Copies of elements are allowed to be stored in multiple blocks. An important proposal of [3] was that an indexing scheme can be evaluated in terms of two simple parameters: the first parameter is the *storage redundancy* which measures how many times an element is stored in disk blocks (there are two kinds, the maximum redundancy and the average redundancy). The blocks are chosen so that we can answer (cover) queries by only using a few blocks. The second parameter is the *access overhead* which captures the quality of an indexing scheme: the access overhead of a query Q measures the ratio of how many blocks are needed to cover Q, compared with the ideal $\lceil |Q|/B \rceil$. The access overhead of the indexing scheme is the worst access overhead of all queries $Q \in \mathcal{Q}$ (see [3] for an extensive discussion of indexing schemes). Predictably, there is a trade-off between the two parameters of an indexing scheme: the higher the storage redundancy, the lower the access overhead should be.

The problem of whether a workload admits an indexing scheme with low storage redundancy and access overhead is easy for 1-dimensional range-queries. For example, B-trees give an ideal indexing scheme where both storage redundancy and access overhead are almost 1. However, no such simple solution is possible in higher dimensions. Extensive research on secondary memory data structures that perform well in higher dimensions [1, 2, 4, 7, 12, 14, 15] led [4] to note the need for a more structured definition of the power of indexing techniques. In response, [3] proposed that the difficulty of indexing complex workloads lies in the complexity of the data itself. It was argued that in many cases it is worth focusing solely on the trade-off between storage redundancy and access overhead of indexing schemes. This approach concentrates on the "data complexity" of workloads and ignores the computational complexity of organizing the data into the secondary memory (selecting an indexing scheme for a given workload), as well as the search complexity of finding which blocks are needed to best answer a given query. A concrete example is given in [3] on workloads of 2-dimensional range-queries where the instance I consists of the regular grid (the $k \times k$ grid points). It was shown that these workloads have non-trivial trade-off; an indexing scheme that achieves access overhead a must have storage redundancy $r = \Omega(\log B/(a^2 \log a))$.

*Computer Science Department, University of California, Los Angeles, CA 90095, USA. Email: elias@cs.ucla.edu

†Computer Science Department, University of California, Los Angeles, CA 90095, USA. Email: dstaylor@cs.ucla.edu

This result was improved in [13], which gave the exact trade-off for these workloads: $r = \Theta(\log B/\log a)$. For the d-dimensional range-queries where the instance I consists of the regular grid: the trade-off is given by $r = \Theta((\log B/\log a)^{d-1})$.

In another direction, [6] studied workloads of range-queries on arbitrary Euclidean points (as opposed to regular grid points). In particular, the Fibonacci workload of n points —the regular 2-dimensional grid rotated by the golden ratio— was analyzed. It was shown that there exists a constant c_B, such that any indexing scheme with storage redundancy $c_B \log n$ has access overhead $a = B$. There is also a matching upper bound for the trade-off: *any* 2-dimensional workload has an indexing scheme with storage redundancy $r = \Theta(\log n)$ and access overhead $a = 4$. These results provided the first concrete example of a trade-off that grows with the workload size, and also showed that simple workloads, such as the Fibonacci workload, are hard with respect to indexing. Furthermore, the trade-off exhibits a threshold behavior: increasing the storage redundancy by a constant factor can dramatically improve the access overhead from the worst possible $a = B$ to almost optimal $a = 4$.

Our model

We consider only workloads of range-queries on a Euclidean space. These workloads are simply determined by their set of points. Consider some d-dimensional workload C of size $|I| = n$. With respect to range-queries it is homeomorphic to some workload whose points have integer coordinates $1, \ldots, n$. Therefore, without loss of generality we consider only workloads with points in $\{1, \ldots, n\}^d$. In this paper we study random workloads. There are two natural uniform probability distributions of random workloads: the first results from uniformly choosing a set of exactly n points from $\{1, \ldots, n\}^d$. The second results from independently selecting each point from $\{1, \ldots, n\}^d$ with probability $1/n^{d-1}$ so that the resulting workload has expected size n. The two distributions are very similar, but it is usually more convenient to work with the second one. This is the distribution we use in this paper. With minimal work, our results could be adapted to apply to the first distribution as well.

To keep expressions simple, we treat the block size B as a constant, so, for example, the expression $O(n)$ may conceal some multiplicative factor that depends on B and d. Our aim is to show the dependency of the indexing trade-off on the workload size, not the block size.

Our results

We continue the work of [6] by extending it in two directions. We prove that logarithmic trade-off is not a property of only structured workloads such as the Fibonacci workload, but it is a property of the vast majority of 2-dimensional workloads: with high probability, a random 2-dimensional workload will require logarithmic redundancy to achieve anything better than the worst possible access overhead. We also extend the result to higher dimensions for which we know of no simple workload with the properties of the Fibonacci workload. We show that a random d-dimensional workload needs polylogarithmic redundancy, with high probability. More precisely, for d-dimensional workload with range-queries, there is a constant $c_{B,d}$ such that with high probability, every indexing scheme with storage redundancy $c_{B,d} \log^{d-1} n$ has the worst possible access overhead (equal to the block size B). In the process, we provide a general lower bound theorem for indexing schemes (Theorem 2.1).

We also show that random workloads exhibit the threshold behavior of the Fibonacci workload. In particular, we show that for a random d-dimensional workload the expected number of queries of size B or less is $\Theta(\log^{d-1} n)$ which implies that there exists an indexing scheme that has expected storage redundancy $\Theta(\log^{d-1} n)$ and access overhead $a = 2d - 1$.

2 A general lower-bound

We will make use of the following general theorem, which provides a lower bound of the trade-off between storage redundancy and access overhead. This theorem is valid for any workload, and the constants involved can be strengthened when given specific workload structure (e.g. range-queries).

THEOREM 2.1. *Let $W = (D, I, \mathcal{Q})$ be a workload such that $\mathcal{Q}$ contains a collection of m queries each with size a multiple of B with the property that no $k > 1$ elements of I belong to more than c of these queries. If $m > r\frac{|I|}{B}\binom{B}{k}c$ then any indexing scheme with redundancy r has access overhead at least $B/(k-1)$.*

Proof. Consider a query Q that has size a multiple of B and access overhead strictly less than $B/(k-1)$. It is covered by less than $\lceil |Q|/B \rceil B/(k-1) = |Q|/(k-1)$ blocks. It follows that at least one of these blocks shares k or more points with the query. Hence, to each query that has access overhead less than $B/(k-1)$, we can associate a block that shares with it k or more points.

We can now give an upper-bound of the number of queries that have access overhead less than $B/(k-1)$. There are $r|I|/B$ blocks in total, each block contains $\binom{B}{k}$ subsets of k elements, and each such set can be associated with at most c queries. Therefore, the number of queries with access overhead less than $B/(k-1)$ is at most $r\frac{|I|}{B}\binom{B}{k}c$. If the number of queries m is larger that this, it follows that at least one query has access overhead at least $B/(k-1)$.

The restriction of considering only queries of size a multiple of B is a technical one. A slightly weaker statement is true for queries that do not have size a multiple of B. Here, we will use only the special case of the theorem, when $k = 2$; it was first used (implicitly) in [6].

COROLLARY 2.1. *Let $W = (D, I, \mathcal{Q})$ be a workload such that $\mathcal{Q}$ contains a collection of m queries of size B with the property that no two elements of I belong to more than c of these queries. If $m > r\frac{|I|}{B}\binom{B}{2}c$ then any indexing scheme with redundancy r has access overhead at least B.*

We remark that Corollary 2.1 guarantees only that some query has access overhead B. However, it follows from the proof of Theorem 2.1 that if $m >> r\frac{|I|}{B}\binom{B}{2}c$, almost all m queries have access overhead B.

3 Lower bound for random workloads

Our main theorem shows that indexing a random range-query workload has an inherent $\log^{d-1} n$ trade-off. To simplify our proofs, we assume that both n and B are integer powers of 2, although the results hold even without this assumption.

THEOREM 3.1. *With probability $1-o(1)$, every indexing scheme for a random d-dimensional workload requires $\Omega(\log^{d-1} n)$ average redundancy to achieve access overhead $a < B$.*

We remark that the term $o(1)$ in the theorem is exponentially small in n. The proof of the theorem is a direct application of Corollary 2.1 and the following lemma.

LEMMA 3.1. *With probability $1 - o(1)$, the following holds for a random d-dimensional workload: there is a collection of $\Omega(n \log^{d-1} n)$ queries of B points each, such that no pair of points belongs to more than $c_{B,d}$ queries in the set, where the constant $c_{B,d}$ is independent of n.*

Before we proceed with the proof of Lemma 3.1, we need to develop some terminology. We define the *volume* of a query to be the number of grid points within (and including) the ranges given. The *size* of a query is the number of data points (rather than grid points) in it. For our purposes, it suffices to consider only queries of B points.

In Lemma 3.1, we need to show that there are $\Omega(n \log^{d-1} n)$ range-queries of size B. Estimating the expected number of such queries is a much easier task than what Lemma 3.1 requires. The difficulty arises when we want to estimate the number of queries with high probability $(1 - o(1))$. In order to decrease dependency among queries, we consider multiple partitions of the space into rectangles. Any statement (event) about the points of a rectangle in a partition is independent of the other rectangles from the same partition; here our choice of the probability distribution comes handy. The limited dependency between intersecting rectangles from different partitions will not cause any problems.

Each partition divides the workload into n/B identical rectangles of volume Bn^{d-1}. Furthermore, the dimensions of these rectangles will all be powers of 2. More precisely, we consider all possible partitions with rectangles of dimensions $x_1, \ldots, x_d$ with $x_i = 2^{k_i}$ and $\prod_i x_i = Bn^{d-1}$. The number of partitions is equal to the number of tuples $k_1, \ldots, k_d$ of non-negative integers k_i such that $\sum_{i=1}^{d} k_i = \log(Bn^{d-1})$. It is easy to see that there are $\binom{d-1+\log(Bn^{d-1})}{d-1}$ partitions. This is at least $\binom{(d-1)(1+\log n)}{d-1} \geq \log^{d-1} n$.

There are n/B rectangles within each partition. Each of these rectangles is a candidate query for Lemma 3.1. We identify 3 sufficient conditions for a rectangle to be a query for Lemma 3.1:

- First, a rectangle must contain exactly B points.
- Second, to ensure that we do not encounter the same query in a different partition, we will require that the rectangle has at least one point in each half of its volume, in all d dimensions. We will call such a rectangle *balanced*. A balanced rectangle is guaranteed to be considered at most once, since rectangles in 2 different partitions always have at least one dimension where one rectangle overlaps only with half of the other rectangle.
- Finally, we need to guarantee that no pair of points belongs to more than a constant number of these queries. To enforce this condition, we require that the queries are *well-spaced*. A query is said to be well-spaced if no two of its points are "close" in the following sense: the smallest rectilinear rectangle that contains both points has volume at least $v_{B,d}n^{d-1}$ for some fixed constant $v_{B,d}$ to be specified later (in the proof of Lemma 3.4).

We first verify that the third property does indeed achieve its purpose, that is, well-spaced queries have the property that no pair of points is contained in more than a constant number of them.

LEMMA 3.2. *Any given pair of points of a well-spaced query can be contained in at most $c_{B,d}$ partition rectangles, where $c_{B,d}$ is independent of n.*

Proof. Fix some pair of points of a well-spaced query of size B. Let $y_1, \ldots, y_d$ be the dimensions of the minimum rectilinear rectangle that contains both points. Since the query is well-spaced, the volume $\prod_i y_i$ of this minimum rectangle is at least $v_{B,d}n^{d-1}$. Consider a partition into rectangles of dimensions $x_1, \ldots, x_d$, where each x_i is a power of 2, and the volume $\prod_i x_i$ of each rectangle is Bn^{d-1}. Clearly, at most one rectangle from the partition can contain the given pair. Furthermore, such a rectangle cannot exist unless $x_i \geq y_i$, for $i = 1, \ldots, d$. This in turn implies that $x_i = Bn^{d-1}/(\prod_{j\neq i} x_j) \leq Bn^{d-1}/(\prod_{j\neq i} y_j)$. Thus, x_i can take values, which are powers of 2, in the interval $[y_i, Bn^{d-1}/(\prod_{j\neq i} y_j)]$. Clearly, there are at most $1+\log(Bn^{d-1}/(\prod_j y_j)) = 1+\log(B/v_{B,d})$ possible values for x_i. Consequently, there are at most $c_{B,d} = (1+\log(B/v_{B,d}))^{d-1}$ partitions that have a rectangle that contains both points.

To finish the proof of Lemma 3.1, it suffices to show that a lot of partition rectangles satisfy all three conditions (size B, balanced, and well-spaced), with high probability. For this, we first compute the probability that a rectangle satisfies each condition. The first two conditions are easy:

LEMMA 3.3. *The probability that any given rectangle of volume Bn^{d-1} contains exactly B points is at least e^{-B}. Furthermore, given that a rectangle has exactly B points, the probability that it is balanced is at least $(1-2^{-B+1})^d$.*

Proof. There are Bn^{d-1} grid points in the rectangle, and each has probability $1/n^{d-1}$ of being a data point. The probability of having exactly B data points is

$$\binom{Bn^{d-1}}{B}\left(\frac{1}{n^{d-1}}\right)^B\left(1-\frac{1}{n^{d-1}}\right)^{Bn^{d-1}-B}.$$

Using $\binom{Bn^{d-1}}{B} \geq n^{B(d-1)}$, this probability is at least $\left(\frac{n^{d-1}-1}{n^{d-1}}\right)^{(n^{d-1}-1)B} \geq e^{-B}$.

Given that the rectangle has exactly B points, the probability distribution of these points results by uniformly choosing a set of B points of the rectangle. However, we get a lower bound on the probability that the rectangle is balanced by assuming that the B points are chosen uniformly and independently with repetitions. The reason is that the probability that these B *or fewer* points are balanced is no bigger than the probability that *exactly* B points of the rectangle are balanced. Now, the probability that all the points B would fall in one half of a particular dimension is 2^{-B+1}, and therefore the probability that there is a point in each half of the rectangle in this dimension is $1-2^{-B+1}$. All dimensions are independent, so the probability that the rectangle has a point in each half, along every dimension, is $(1-2^{-B+1})^d$.

Predictably, the probabilities in the above lemma do not depend on n. We now turn our attention to the third condition. We will show that a rectangle of a partition that has B data points is well-spaced also with constant probability. If two grid points violate the well-spaced requirement —the minimum rectangle that contains both points has volume less than $v_{B,d}n^{d-1}$— we will say that one point *covers* the other. We will need the following lemma.

LEMMA 3.4. *In a rectangle of volume Bn^{d-1}, a grid point can cover at most $1/(B(B-1))$ of the volume of the rectangle.*

Proof. Fix some grid point p. To get an upper bound on the volume covered by it, we will suppose that the point p is in one of the corners of the query, consider the number of grid points covered by it, and multiply this answer by 2^d. This last multiplication is needed because a point in the center of the query can cover grid points in any of the 2^d orthants for which it acts as a corner. The number of grid points covered by p is exactly equal to the number of grid points $(x_1, x_2 \ldots x_d)$ below the surface $\prod_{i=1}^d x_i \leq v_{B,d}n^{d-1}$. We can upper bound this by the volume below the surface. It is easy to see that the volume under the surface is proportional to the volume of the rectangle, and it is independent of its shape. It works out that the grid point p covers $(v_{B,d}\sum_{i=0}^{d-1}\ln(B/v_{B,d}))n^{d-1}$ of the grid points in the rectangle.

Therefore, a grid point covers at most $2^d(v_{B,d}\sum_{i=0}^{d-1}\ln(B/v_{B,d}))/B$ of the volume of the rectangle. By choosing an appropriate constant $v_{B,d}$, we can guarantee that a grid point covers at most $1/(B(B-1))$ of the volume of the query.

We can now estimate the probability that a balanced rectangle that has B data points is a well-spaced query.

LEMMA 3.5. *A balanced rectangle that contains B data points is well-spaced with probability at least $1/2$.*

Proof. The only information we have about the distribution of points inside the rectangle is that there is at least one point in every half of the rectangle, in every dimension. We want to find the probability that two points p_1 and p_2 cover each other. Fix the point p_1. It certainly accounts for one point in its half of the rectangle (in all d dimensions), so the only information that we have about the distribution of the remaining $B-1$ points is that there is at least one in every other half of the rectangle (in all d dimensions). These points are no closer to p_1 than random points, so the probability of p_2 being covered by p_1 is no worse than it would be if p_2 were uniformly distributed in the rectangle. By Lemma 3.4, the probability that p_1 covers p_2 is at most $1/(B(B-1))$. Since there are $\binom{B}{2}$ pairs of points, the probability that there exists a pair whose points cover each other is at most $\binom{B}{2}\frac{1}{B(B-1)} = 1/2$.

In summary, Lemmata 3.3 and 3.5 show that a rectangle within a partition is a balanced well-spaced query of size B with probability at least $(1-2^{-B+1})^d e^{-B}/2$. Since they are balanced, any two of these queries are guaranteed to be distinct. We can now give a lower bound of the number of these queries.

Lemma 3.6. *With probability* $1-o(1)$*, the partitions contain* $\Omega(n \log^{d-1} n)$ *balanced, well-spaced rectangles with exactly* B *points.*

Proof. The crucial property is that within a partition one rectangle is completely independent of another. The probability that a rectangle is a balanced well-spaced query of size B is at least $p = (1-2^{-B+1})^d e^{-B}/2$. It is safe to assume that the probability is exactly p. We can use the Chernoff bounds [9] to limit the probability of having few such rectangles within a partition. In particular, let X_i be a random variable that denotes the number of rectangles of partition i that are balanced well-spaced queries of size B. We have that its expectation is $\mu = pn/B = \Theta(n)$ and the Chernoff bound gives $P[X_i \le (1-\delta)\mu] \le e^{-\delta^2\mu/2}$. For a constant δ, the probability that one partition fails to have enough balanced well-spaced queries is exponentially small. Fix some $\log^{d-1} n$ partitions (there are at least so many). The probability that there exists one partition among them that fails to have $(1-\delta)\mu$ balanced well-spaced queries is at most $e^{-\delta^2\mu/2}\log^{d-1} n = o(1)$. Therefore, with probability $1-o(1)$ there are $(1-\delta)\mu\log^{d-1} n = \Omega(n\log^{d-1} n)$ balanced well-spaced queries of size B.

This concludes the proof of Lemma 3.1 and of Theorem 3.1.

4 Upper bound for random workloads

In this section, we will give a matching upper bound of Theorem 3.1. For 2 dimensions, it was shown in [6] that any 2-dimensional range-query workload admits an indexing scheme with redundancy $\Theta(\log n)$ and constant access overhead ($a = 4$). This trade-off matches (up to a constant factor) the trade-off of Theorem 3.1. Hence, for 2 dimensions, a random workload has (up to a constant factor) the worst possible trade-off between storage redundancy and access overhead.

Does this result hold for higher dimensions? More precisely, is it true that for any d-dimensional range-query workload there exists an indexing scheme with storage redundancy $r = \Theta(\log^{d-1} n)$ and constant access overhead a (or $a = 2^d$)? We do not know the answer to this question, although we conjecture that it is positive. However, we can show that the statement holds for random workloads in the following sense: for a random d-dimensional workload, there is an indexing scheme that is expected to have storage redundancy $O(\log^{d-1} n)$ and access overhead $a = 2d-1$.

To this end, we will show that the expected number of range-queries of size at most B in a random workload is $O(n\log^{d-1} n)$. This means that we can store all queries of size B or less in separate blocks. We then show how to inductively break any query Q into $1 + (2d-1)\lfloor(|Q|-1)/B\rfloor$ subqueries of size B or less, so that the resulting indexing scheme has expected redundancy $O(\log^{d-1} n)$ and access overhead $a = 2d-1$.

Theorem 4.1. *In a random* d*-dimensional range-query workload, the expected number of queries of size* B *or less is* $O(n\log^{d-1} n)$*.*

Proof. The proof is a straightforward calculation. We first compute an appropriate upper bound of the probability that a given rectilinear rectangle is a query of size X and use it to compute the expected number of queries of size X.

Let $x_1,\ldots,x_d$ be the dimensions of a rectilinear rectangle R. We want to find the probability that the rectangle is a query of size X. To avoid counting the same query twice, we require that the rectangle has a point on every side, that is, we identify a query with the minimum rectangle that contains all its points. The probability that the rectangle has exactly X points is simply $\binom{x_1\cdots x_d}{X}(\frac{1}{n^{d-1}})^X(1-\frac{1}{n^{d-1}})^{x_1\cdots x_d - X}$. Using the inequalities $\binom{a}{b} \le (\frac{ea}{b})^b$ and $1-x \le e^{-x}$, we get that this probability is at most $\frac{e^X}{X^X}\frac{(x_1\cdots x_d)^X}{(n^{d-1}-1)^B}e^{-\frac{x_1\cdots x_d}{n^{d-1}}}$.

Given now that the rectangle has X points, we want to estimate the probability that every side has at least one point. Let us denote the events that there is at least one point on the two sides of the rectangle R

perpendicular to x_i axis by E_{i1} and E_{i2}. It is easy to see that $P[E_{i1}] = 1 - \binom{x_1 x_2 \cdots (x_i - 1) \cdots x_d}{X} / \binom{x_1 \cdots x_d}{X} \leq \frac{X}{x_i}$. Also, $P[E_{i2} \mid E_{i1}] \leq P[E_{i2}] = P[E_{i1}]$, since knowing at least one of the points is on the opposite side can only decrease the probability that there is a point on this side. Since events E_{ix} and E_{jy} are independent for $i \neq j$, the probability that there are points on all $2d$ sides of the rectangle R is at most $X^{2d}/(x_1 \cdots x_d)^2$.

Putting everything together, we get that the probability that the rectangle R is a query of size X is $O(\frac{(x_1 \cdots x_d)^{X-2}}{n^{X(d-1)}} e^{-\frac{x_1 \cdots x_d}{n^{d-1}}})$. Fix now a grid point q. The expected number of queries of size X for which q is the corner closest to the origin is bounded, within a constant factor $t_{X,d}$, by

$$\int_1^n \cdots \int_1^n \frac{(x_1 \cdots x_d)^{X-2}}{n^{X(d-1)}} e^{-\frac{x_1 \cdots x_d}{n^{d-1}}} \, dx_d \cdots dx_1.$$

Using the fact that $\int_0^\infty x^k e^{-ax} dx = k!/a^{k+1}$, it is not hard to verify that the integral is bounded by $\frac{(X-2)!}{n^{d-1}} \ln^{d-1} n$. Accounting for the n^d possible choices for q, we get that the number of queries of size $X \leq B$ is $O(n \log^{d-1} n)$.

Lemma 3.6 shows that Theorem 4.1 is tight: the expected number of range-queries of size B is $\Theta(n \log^{d-1} n)$.

Now, if we could guarantee that every hyperplane of the space had only one point on it, then every query could be covered using $\lceil |Q|/B \rceil$ queries of size B, and we could use those queries to achieve access overhead $a = 1$. However, since multiple points per hyperplane are allowed, it becomes harder to break a query into small subqueries, and we will need to use more blocks.

LEMMA 4.1. *A range-query in a d-dimensional workload can be covered using at most $1 + (2d-1)\lfloor (|Q|-1)/B \rfloor$ queries, each of size B or less.*

Proof. By induction on the dimension and the size of the query. The base cases $|Q| \leq B$ and $d = 1$ are both obvious. Consider now a query Q of dimension $d \geq 2$ and size $|Q| > B$. We partition the query along the d-th dimension into three subqueries. If x_d is the size of the query along the d-th dimension, the new queries have sides $y-1, 1, x_d - y$ such that the first subquery (side $y-1$), which may be empty, contains at most B points, while the first two subqueries (sides $d-1$ and 1) contain $Y > B$ points in total. By the induction hypothesis, the second query can be covered with at most $1 + (2(d-1)-1)\lfloor (Y-1)/B \rfloor$ small (size B or less) queries. Similarly, the third query can inductively be covered with at most $1 + (2d-1)\lfloor (|Q|-Y-1)/B \rfloor$ small queries. Taking into account that the first query is by itself small, it is straightforward to show that the sum of small queries used to cover Q is bounded by the formula of the lemma.

COROLLARY 4.1. *A random d-dimensional workload admits an indexing scheme with expected storage redundancy $r = \Theta(\log^{d-1} n)$ and access overhead $a = 2d-1$.*

Proof. By Theorem 4.1, we know that using expected storage redundancy $r = \Theta(\log^{d-1} n)$ we can store every range-query of size B or less in a block. By Lemma 4.1 we can cover any query Q using at most $1 + (2d-1)\lfloor (|Q|-1)/B \rfloor$ of those blocks. Thus, for a query that is optimally answered using $\text{opt} = \lceil |Q|/B \rceil$ blocks, we use at most $1 + (2d-1)(\text{opt}-1)$. This gives an access overhead of at most $a = 2d-1$.

5 Open problems

The constants in Theorem 3.1 that result from our proofs are small for practical values of B and d. Similarly, the constants of Corollary 4.1 are extremely large. We have chosen to provide simple proofs, instead of obtaining better constants. However, we don't see how to tighten our proofs so that the constants are relevant for practical values of B and d. It will be interesting to improve these constants.

The lower bound of Theorem 3.1 holds with high probability, i.e., the probability that a random workload has less than $\Theta(\log^{d-1} n)$ trade-off between storage redundancy and access overhead is exponentially small. On the other hand, the upper bound in Corollary 4.1 is about expectations. Of course, using the Markov Inequality, we can translate expectations to probability. This however shows only that the upper bound of the trade-off holds with probability $1-\epsilon$, for any constant ϵ. Is it true that the upper bound also holds with high probability? For 2 dimensions, the answer is certainly positive since *any* 2-dimensional workload admits an indexing scheme of redundancy $r = \Theta(\log n)$ and constant access overhead $a = 4$ [6]. We were unable to extend this result to higher dimensions, although we believe that it is true. More precisely, we conjecture that *any* d-dimensional range-query workload admits an indexing scheme with redundancy $r = \Theta(\log^{d-1} n)$ and access overhead $a = 2^d$.

References

[1] L. Arge and J. S. Vitter. Optimal Dynamic Interval Management in External Memory. In *37th Annual Symposium on Foundations of Computer Science (FOCS '96)*, pages 560–569, Burlington, VT, Oct 1996.

[2] A. 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.

[3] J. M. Hellerstein, E. Koutsoupias, and C. H. Papadimitriou. On the analysis of indexing schemes. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 249–256, Tucson, Arizona, 12–15 May 1997.

[4] J. M. Hellerstein, J. F. Naughton, and A. Pfeffer. Generalized Search Trees for Database Systems. In *Proc. 21st International Conference on Very Large Data Bases*, pages 562–573, Zurich, September 1995.

[5] P. C. Kanellakis, S. Ramaswamy, D. E. Vengroff, and J. S. Vitter. Indexing for Data Models with Constraints and Classes. In *Proc. 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 233–243, Washington, D.C., May 1993.

[6] E. Koutsoupias and D. S. Taylor. Tight bounds for 2-dimensional indexing schemes In *Proc. 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 52–58, Seattle, June 1998.

[7] D. B. Lomet and B. Salzberg. The hB-Tree: A Multiattribute Indexing Method. *ACM Transactions on Database Systems*, 15(4), pages 625–658, December 1990.

[8] K. Mehlhorn. *Data Structures and Algorithms 3: Multidimensional Searching and Computational Geometry.* Springer-Verlag, Berlin, 1984.

[9] R. Motwani and P. Raghavan. *Randomized Algorithms* . Cambridge University Press, Cambridge Mass., 1995.

[10] S. Ramaswamy and P. C. Kanellakis. OODB Indexing by Class Division. In *Proc. 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 233–243, 1993.

[11] S. Ramaswamy and S. Subramanian. Path Caching: A Technique for Optimal External Searching. In *Proc. 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 25–35, Minneapolis, 1994.

[12] J. T. Robinson. The k-D-B-Tree: A Search Structure for Large Multidimensional Dynamic Indexes. In *Proc. ACM-SIGMOD International Conference on Management of Data*, pages 10–18, Ann Arbor, April/May 1981.

[13] V. Samoladas and D. P. Miranker. A Lower Bound Theorem for Indexing Schemes and its Application to Multidimensional Range Queries. In *Proc. 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 44–51, Seattle, June 1998.

[14] S. Subramanian and S. Ramaswamy. The p-range Tree: A Data Structure for Range Searching in Secondary Memory. In *Proc. 6th ACM-SIAM Symposium on Discrete Algorithms*, pages 378–387, 1995.

[15] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-Tree: A Dynamic Index For Multi-Dimensional Objects. In *Proc. 13th International Conference on Very Large Data Bases*, pages 507–518, Brighton, September 1987.

[16] D. E. Vengroff and J. S. Vitter. Efficient 3-d Searching in External Memory. In *Proc. 28th ACM Symposium on the Theory of Computing* , pages 191–201, 1996.