On the Analysis of Indexing Schemes

Joseph M. Hellerstein*
Division of Computer Science
UC Berkeley, Berkeley, CA 94720
jmh@cs.berkeley.edu

Elias Koutsoupias†
Department of Computer Science
UCLA, Los Angeles, CA 90095
elias@cs.ucla.edu

Christos H. Papadimitriou‡
Division of Computer Science
UC Berkeley, Berkeley, CA 94720
christos@cs.berkeley.edu

Abstract

We consider the problem of indexing general database workloads (combinations of data sets and sets of potential queries). We define a framework for measuring the efficiency of an indexing scheme for a workload based on two characterizations: *storage redundancy* (how many times each item in the data set is stored), and *access overhead* (how many times more blocks than necessary does a query retrieve). Using this framework we present some initial results, showing upper and lower bounds and trade-offs between them in the case of multi-dimensional range queries and set queries.

1 Introduction

The success and ubiquity of the relational data model arguably owes much to the B-tree, the access method breakthrough that accompanied it with superb timing [2]. It seems likely that access methods will continue to play an important role in, and largely determine the viability of, the novel data models currently under intense scrutiny in the database research community. The B-tree is widely recognized to be an inadequate data structure in many of the novel contexts, and no clear successor has emerged (or is likely, in view of the diversity of the applications and requirements). It is therefore important to develop general methodologies and tools for the design of new indexing methods, as well as mathematical tools and techniques for evaluating their performance and pointing out their limitations.

*Supported by NASA Grant FDNAGW-5198.
†Supported by NSF grant CCR-9521606.
‡Supported by the National Science Foundation.

A systems approach to this "generalized indexing" problem has been proposed and implemented [11]. The need for theoretical tools for the rigorous analysis of indexing problems was one of the main conclusions of that work. What seems to be needed is a kind of *theory of indexability*, a mathematical methodology which, in analogy with tractability, would evaluate rigorously the power and limitations of indexing techniques in diverse contexts. What differentiates such a theoretical approach to indexing from complexity theory and the theory of in-memory data structures is its emphasis on the secondary storage nature of indexing schemes, and on the two aspects that determine their cost and feasibility: storage utilization and disk accesses.

In this paper, we lay out a general framework for formally evaluating the power and limitations of indexing schemes. We identify the salient features of an indexing problem, and define two simple metrics for judging the difficulty of the problem. Based on this framework, we provide some initial results: lower bounds and space/time tradeoffs for multidimensional range queries and for set inclusion queries. One novelty of our results is that they are stated exclusively in terms of a parameter B, the *block size.* This important technological parameter, which is usually ignored in the data structures literature, is at center stage in our work. Interestingly, the size of the instance does not enter the statements of our results at all.

Related Work

There is extensive work on data structures (see, for example, [16, 1, 29, 19]), which only occasionally focuses on the external memory aspects of the problem. Workloads that can be optimized off-line have also been occasionally considered in the data structure literature (see, for example, [31, 12, 21]). A survey of B-trees and their variants appears in [4]; the variant in common use in database systems is the B+-tree, which stores all data in the leaf nodes, and fits our model well. A variety of external memory multidimensional data structures exists, including both hash structures (see, for example, [20]) and tree structures (see, for example, [10, 28, 24, 18]). The Generalized Search Tree (GiST) [11] is an extensible data structure which simplifies the development of tree-based indexing schemes.

Although our framework for studying indexability is new, there are some previous results that fit into it rather naturally. Analyses along the lines we are suggesting here have been emerging in the past few years, most notably in the work of the late Paris Kanellakis and his collaborators [15, 23, 25, 27, 30]. Most of this work involves upper bounds, and is therefore mainly concerned with the analysis of the searching aspect of the problem. There are two exceptions. First, in a recent version of [15] there is an argument (proof of Lemma 2.7) that anticipates our Theorem 1, namely, that the access overhead must be $\sqrt{B}$ *in the special case in which the blocks are restricted to be rectangular.* Second, in the last section of [27], there is an interesting lower bound, where it is shown (by extending a result by Chazelle to the case of block accesses) that storage redundancy $\log n / \log\log n$ is necessary if *additive* (as opposed to our multiplicative) access overhead is to remain polynomial in $\frac{\log n}{\log B}$. The question of lower bounds in multi-dimensional searching has been addressed in [19], without, however, our emphasis on block accesses. Lower bounds for multidimensional searching are also studied in [26], where the bounds are derived in a model involving binary trees with certain further restrictions; the block size is considered in that paper as a function of n, the number of points.

Finally, in the database literature there have been analyses (worst case, expected case, or empirical) of many access methods for multi-dimensional searching (see, for example, [22, 7, 3]). Our emphasis on the two ratios (storage and access) as the salient performance parameters of an indexing scheme reflects influences from the area of on-line algorithms [13].

2 Framework and Definitions

In this section, we set out a simple framework for defining an indexing problem, and for measuring the efficiency of a particular indexing scheme for the problem.

2.1 Workload

Access methods must be evaluated in the context of a particular *workload*. A workload consists of a finite subset of some domain together with a set of queries. More formally, a workload is a triple $W = (D, I, \mathcal{Q})$, where D is the *domain* (a set such as $\mathbb{R}^d$ together with methods such as x-component, order, etc.), I is a finite subset of the domain called an *instance*, and $\mathcal{Q} = \{Q_1, \ldots, Q_q\}$, the set of *queries*, is a set of subsets of I.

For example, one of the workloads we consider extensively (the two-dimensional range queries) consists of the domain $\mathbb{R}^2$, the instance $I = \{(i,j) : 1 \leq i, j, \leq n\}$, and the family of "range queries" $Q[a,b,c,d] = \{(i,j) : a \leq i \leq b, c \leq j \leq d\}$, one for each quadruple (a,b,c,d) with $1 \leq a \leq b \leq n, 1 \leq c \leq d \leq n$. Notice that this is a *family of workloads*, with instances of increasing cardinality, one for each $n \geq 0$. Another family of workloads (the set inclusion queries) has as its domain, for each n, all subsets of $\{1, 2, \ldots, n\}$, and for each subset I of the domain, the set of queries $\mathcal{Q} = \{Q_S : S \subseteq \{1, 2, \ldots, n\}\}$, where $Q_S = \{T \in I : T \subseteq S\}$.

The workload plays in indexability theory the role played by languages in complexity theory: it is the unit whose complexity must be characterized[1].

2.2 Indexing Schemes

For each workload we have a space of possible indexing schemes, the analog of algorithms that decide the language. Intuitively, an indexing scheme is simply a collection of blocks, each block containing some large, fixed number of objects (typically hundreds of objects per block). The union of the blocks exhausts the instance. Each query is answered by retrieving a set of blocks whose union is a superset of the query.

Formally, an *indexing scheme* $\mathcal{S}$ for a workload $W = (D, I, \mathcal{Q})$ is a collection $\mathcal{S} = \{S_1, S_2, \ldots, S_s\}$ of *blocks*, where a block is a subset of exactly B elements of I. B is a constant called the *block size*, assumed fixed for each workload family, and large. In practice B is typically a constant between one hundred and one thousand that corresponds to a disk's block size of 4 to 8 Kbytes.

2.3 Two Performance Measures

Given a workload and an indexing scheme, we identify two basic performance measures that seem to capture the two determining factors of the cost of indexing schemes: storage and access costs.

The storage cost of an index can be expressed as the ratio of the number of blocks used by the index, divided by the number of blocks that are absolutely necessary for storing the instance — the size of the instance divided by B. More formally, we define the *storage redundancy* of an indexing scheme as the maximum number of blocks that contain an element of I. We also define the *average redundancy* of the indexing scheme as the average number of blocks that contain an element of I, that is, $sB/|I|$.

The access cost for queries in an indexing scheme can be defined by a similar ratio. Let Q be a query of $\mathcal{Q}$.

[1] More accurately, the analog of a language is a family of workloads, one for each cardinality of the instance. Such growing families of workloads allow us to focus on asymptotic analysis and ignore additive constants.

An ideal indexing scheme would require $\lceil |Q|/B \rceil$ blocks to answer it. However, this is not in general possible. The *access overhead* of the indexing scheme $\mathcal{S}$ for Q is a measure of how far we are from the ideal situation: it is the minimum number of blocks from $\mathcal{S}$ that cover Q, divided by the ideal cost $\lceil |Q|/B \rceil$. The access overhead of the indexing scheme $\mathcal{S}$ on workload $W = (D, I, \mathcal{Q})$ is the maximum access overhead over all queries in $\mathcal{Q}$.

Notice that the access overhead is never greater than B, which is the worst case, achievable if we can cover a large query by blocks containing only one relevant data item each. Naturally, in one-dimensional range query workloads, B-trees — as well as any access method that partitions data items along the linear order — easily achieve optimality (one) in both storage and access parameters (with an additive constant of one or two in access overhead).

2.4 Notes on the Framework

Our approach suppresses important aspects of indexing, such as the algorithms for determining the partition of the instance into blocks (possibly with repetitions), as well as the algorithms for determining, given a query, the blocks in the index that cover it (e.g. hash lookups, or traversal of a tree to its leaf level). Furthermore, we ignore the storage and retrieval costs due to auxiliary information such as "directories" or "internal nodes". These omissions are justified in three ways. First, we are mostly interested in lower bounds, and therefore we are free to disregard aspects of the complexity of the problem. Second, these aspects do not seem to be the source of design difficulties or of complexity — it appears that good assignment of data items to blocks tend to suggest efficient traversal algorithms, and to have low storage overhead. And third, secondary storage techniques such as buffer management mask and absorb many of these auxiliary cost components.

Though our framework is simple, it captures the essence of previous heuristic approaches taken for indexing complex workloads. For example, a common multidimensional index for database systems is the R-tree [10], which has redundancy 1. Three years after the initial R-tree paper, the R+-tree was proposed as an improvement [28]; the main innovation of the R+-tree was (in our terms) to lower access overhead by increasing storage redundancy. This intuition is mirrored by the framework in this paper.[2]

As another example, many heuristic solutions for indexing in non-traditional domains have proceeded by a process of analogy: rather than designing indices for the non-traditional workloads, they have mapped the workloads into well-understood domains. Typically this is done by mapping objects to points in an n-dimensional space (e.g. based on a binary distance function), and mapping "similarity" queries over the objects to range queries or nearest-neighbor queries over the resulting space [6]. A similar process of analogy motivates our work: by defining a general framework for studying indexability, we can analyze new workloads by showing them to be isomorphic to well-understood workloads. Of course this approach harks back to seminal techniques in complexity theory as well.

As in complexity theory, one must have some basic results in order to drive the analogy process. In the remainder of the paper we present initial results for two canonical workloads: range queries in multidimensional space, and inclusion queries over set objects. Multidimensional range queries are quite natural to a variety of applications, and hence well worth studying in their own right. Set inclusion queries, as we shall see, represent a worst-case scenario in terms of indexability, and hence define the opposite end of the spectrum from the simple B-tree workloads.

3 Lower Bounds and Trade-offs

Two-dimensional queries

We shall consider here the two-dimensional workload with $I = \{(i, j) : 1 \leq i, j, \leq n\}$, and the range queries over this instance. We are interested in determining the minimum possible access overhead when the redundancy r is fixed.

Proposition 1 *For each integer r, there is an indexing scheme $\mathcal{S}_r$ for the 2-dimensional range queries with redundancy r and access overhead $2B^{\frac{1}{2r}} + 2$.*

Proof. The main idea for the indexing scheme $\mathcal{S}_r$ is that each query Q of $x \times y$ points will be covered by disjoint blocks of $\mathcal{S}_r$ that have "almost" the same aspect ratio y/x with Q. The ideal situation is to have blocks with aspect ratio y/x, so that the query Q is tiled nicely by these blocks; compare this with the worst case when the query Q is "long and narrow" and it is covered by "short and wide" blocks. Because of the restriction on the redundancy r of the indexing scheme $\mathcal{S}_r$, it is not possible to have blocks for each aspect ratio. However, we can choose blocks so that any aspect ratio can be approximated.

More precisely, for each $i = 1, 2, \ldots, r$, the indexing scheme $\mathcal{S}_r$ contains all $B^{\frac{2i-1}{2r}} \times B^{\frac{2r-2i+1}{2r}}$ blocks that partition I. The aspect ratios $B^{\frac{r-2i+1}{r}}$, for $i = 1, 2, \ldots, r$, of these blocks are evenly distributed. It is immediate that $\mathcal{S}_r$ has redundancy r (maximum as well as average). It suffices therefore to show that the access overhead is at

[2] The R+-tree has not proved popular, mostly because of the apparent complexity of performing updates (insertions and deletions) in the structure. This motivates a topic which we intend to study in future work (Section 4): extending our results here to consider dynamic aspects of the indexing problem.

most $2B^{\frac{1}{2r}}+2$. Consider a query Q with $B^{\frac{j}{r}} \times B^{\frac{r-j}{r}}$ points, for some integer j. Clearly, the best coverage of the query is by blocks that have almost the same aspect ratio, that is, by blocks of size $B^{\frac{2j-1}{2r}} \times B^{\frac{2r-2j+1}{2r}}$ or by blocks of size $B^{\frac{2j+1}{2r}} \times B^{\frac{2r-2j-1}{2r}}$. In both cases, when the query is "aligned" with the blocks, it requires $B^{\frac{1}{2r}}$ blocks (either one row of $B^{\frac{1}{2r}}$ blocks or one column of $B^{\frac{1}{2r}}$ blocks). For non-aligned queries the number of blocks needed to cover Q can be as high as $2B^{\frac{1}{2r}}+2$; to see this, consider the case where an aligned query is satisfied by a row of $B^{\frac{1}{2r}}$ blocks. If we shift this query out of horizontal and vertical alignment, we need two rows of blocks instead of one, and at one of the ends we need an additional column of two blocks as well. It is not difficult to show that these are the worst queries for this indexing scheme. □

If the access overhead is a, the above scheme has average and maximum redundancy $r = \Theta(\log B/\log a)$. We conjecture that this is the best possible relation between r and a. Indeed, in the remainder of this section we prove that this is the case when the maximum redundancy is one. For the general case, we show that the average — and therefore the maximum — redundancy is $\Omega(\log B/(a^2 \log a))$. This establishes the conjecture for the most interesting case, when a is a fixed constant. It remains an interesting open problem to remove the factor a^2 from the denominator of the above bound. In particular, when the redundancy r is a small constant, our lower bound implies a logarithmic lower bound $a = \Omega(\sqrt{\log B/\log\log B})$, while the above construction guarantees a polynomial upper bound, $O(B^{\frac{1}{2r}})$.

The case $r = 1$

We will show that up to a constant factor the above indexing scheme is optimal when $r = 1$. The result below was implicitly shown for the special case when the blocks are restricted to be rectangular, in [15].

Theorem 1 *Any indexing scheme of redundancy 1 for 2-dimensional range queries has access overhead at least $B^{\frac{1}{2}}$. For the d-dimensional case, the lower bound is $B^{1-\frac{1}{d}}$.*

Sketch of proof. We only sketch the 2-dimensional case, the general case being a straightforward generalization.

We will use only the $1 \times B$ and $B \times 1$ queries. Consider a block $S \in \mathcal{S}$ that intersects x horizontal lines and y vertical lines (by a "line" we mean a set of data points of the form $\{(1,j),(2,j),\ldots,(n,j)\}$ or $\{(i,1),(i,2),\ldots,(i,n)\}$. Since we must have $xy \geq B$, we conclude that $x+y$ is at least $2B^{\frac{1}{2}}$. Therefore, the block intersects at least $2B^{\frac{1}{2}}$ of the above queries. The number of pairs of intersecting blocks and queries is no less than $2B^{\frac{1}{2}}$ times the total number of blocks, which is $2B^{\frac{1}{2}}n^2/B$. Since there are $2n^2/B$ queries in total in the collection being considered, the average number of intersecting blocks per query is $B^{\frac{1}{2}}$. When the maximum redundancy is $r = 1$, all these blocks are needed to cover the query. Notice that we showed not only that there exists a query with access overhead $B^{\frac{1}{2}}$, but that a random query (from the above set) has this access overhead. □

The redundant case

To prove a lower bound for the case in which redundancy is allowed to be greater than one, we will use an interesting result from extremal set theory. A similar result is given as exercise 13.3 in [17], attributed to K. Corrádi; it is also apparently known in coding theory as *Johnson's Lemma* (Z. Furedi, private communication). We give a simple proof below:

Lemma 1 *Let A be a finite set and $S_1, S_2, \ldots, S_k$ be subsets of A, each of size at least $\alpha|A|$, such that the intersection of any two of them is at most $\beta|A|$. If $\beta < \alpha^2/(2-\alpha)$ then the number of subsets k is at most α/β.*

Proof. Since $S_1, S_2, \ldots, S_t$, $t \leq k$, are subsets of A, their union $S_1 \cup S_2 \cup \ldots \cup S_t$ is also a subset of A and therefore

$$|\bigcup_{j=1}^{t} S_j| \leq |A|.$$

It follows that

$$\sum_{j=1}^{t} |S_j| - \sum_{j=1}^{t} \sum_{l=j+1}^{t} |S_j \cap S_l| \leq |A|.$$

By the assumptions about the sizes of the subsets and their pairwise intersection, the last inequality implies that

$$t\alpha|A| - \binom{t}{2}\beta|A| \leq |A|.$$

Therefore, every $t \leq k$ must satisfy the inequality $\alpha t - \beta\binom{t}{2} - 1 \leq 0$. It immediately follows that if a positive integer t does not satisfy this inequality, then the number k of subsets must be less than t. So, in order to upper bound the number k of subsets, we need to guarantee that the above inequality is not satisfied by at least one positive integer. Obviously, the numbers t that do not satisfy the inequality are between the roots of the polynomial $\alpha t - \beta\binom{t}{2} - 1$. We can therefore guarantee that one of them is integer by requiring that the two roots differ by more than 1. Since the roots of the polynomial are

$$\frac{\alpha + \beta/2 \pm \sqrt{(\alpha+\beta/2)^2 - 2\beta}}{\beta},$$

it is easy to verify that they differ by more than 1 when $\beta < \alpha^2/(2-\alpha)$.

But then, the number of subsets is at most equal to the minimum root of the above polynomial. Thus

$$k \le \frac{\alpha + \beta/2 - \sqrt{(\alpha+\beta/2)^2 - 2\beta}}{\beta}.$$

This last inequality implies that $k \le \alpha/\beta$. □

Note that the hypotheses of the above lemma cannot be improved by a factor more than 2, because when $\beta \ge \alpha^2$, the number of possible subsets is unbounded, i.e., it is an increasing function of $|A|$.

We will use the above lemma to give lower bound of the average access overhead a as a function of the redundancy r. But first, we need to prove the following crucial lemma.

Lemma 2 *Let a be the access overhead. For each x and each query Q of dimensions $x \times (B/x)$, there is a block S and a subset $\hat{Q}$ of $Q \cap S$ such that the points of $\hat{Q}$ belong to exactly x/a vertical lines and each one of these vertical lines contains exactly $B/(ax)$ points of $\hat{Q}$.*

Proof.

Consider a query Q of B points and dimensions $x \times (B/x)$, for some x. Let L be a vertical line of Q. Since the access overhead is a and Q has size B, Q, and consequently L, is covered by at most a blocks. One of these blocks, S_L, must contain at least a fraction of $1/a$ of the points of L, that is, S_L must contain at least $B/(ax)$ points of L. Hence, with each vertical line L of Q, we can associate a block S_L that covers at least $B/(ax)$ points of the line. Since at most a blocks cover B, there is a block S that is associated with at least x/a vertical lines. Therefore, $Q \cap S$ contains x/a vertical lines and each one of these vertical lines contains (at least) $B/(ax)$ points. □

For each query Q, there may be many possible subsets $\hat{Q}$, but we can fix once and for all one of them; we will denote it by $\hat{Q}$.

We now have all the necessary ingredients for the main result of this subsection:

Theorem 2 *The access overhead a and the redundancy r must satisfy $r \cdot a^2 \log(2a^2) \ge \frac{1}{2}\log B$.*

Proof. Let $c \ge 2$ be a parameter to be fixed later. For each $j = 0, 1, \ldots, \log_c B$, we can partition the $n \times n$ space into queries $Q_1^j, Q_2^j, \ldots, Q_{n^2/B}^j$ of dimensions $c^j \times B/c^j$. We will concentrate on the set of queries Q_i^j, for $i = 1, 2, \ldots, n^2/B$ and $j = 0, 1, \ldots, \log_c B$. We want to show that "many" blocks are needed to cover all queries Q_i^j. Instead of arguing directly about the queries Q_i^j, we will argue about their subsets $\hat{Q}_i^j$. By Lemma 2, each such subset belongs entirely into some block. It suffices therefore to show that many blocks are required to have all $\hat{Q}_i^j$'s as subsets. To do this, we show that no block can contain many sets $\hat{Q}_i^j$.

Fix a block S and consider all $\hat{Q}_i^j$'s that are subsets of S. We will use Lemma 1 to show that the number of $\hat{Q}_i^j$'s that are subsets of S is small. Since by Lemma 2 each such subset has $B/(xa)$ lines and each line has x/a points, it follows all subsets have B/a^2 points, and we only need to upper bound the maximum size of their pairwise intersection. We claim that this is at most $\frac{B}{ca^2}$. To see this, notice first that for all distinct i, i': $|\hat{Q}_i^j \cap \hat{Q}_{i'}^j| = 0$, because Q_i^j and $Q_{i'}^j$ are members of a partition of the whole space. Furthermore, when $j < j'$: $|\hat{Q}_i^j \cap \hat{Q}_{i'}^{j'}| \le \frac{B}{ca^2}$, because Lemma 2 assures us that $\hat{Q}_i^j$ has c^j/a vertical lines and every vertical line of $\hat{Q}_{i'}^{j'}$ has $\frac{B}{ac^{j'}}$ points.

Hence, we can apply Lemma 1 with $\alpha = \frac{1}{a^2}$ and $\beta = \frac{1}{ca^2}$. We can now fix the parameter $c = 2a^2$, so that the hypotheses of Lemma 1 are satisfied. Therefore, the number of $\hat{Q}_i^j$'s that are subsets of block S is at most $\alpha/\beta = 2a^2$.

There are $\frac{n^2}{B}(1 + \log_c B)$ subsets $\hat{Q}_i^j$, in total. Each block includes at most $2a^2$ of them. Thus, there are at least $\frac{n^2}{B}\frac{1+\log_c B}{2a^2}$ blocks. It follows that the average redundancy is $r \ge \frac{1+\log_c B}{2a^2} \ge \frac{\log_c B}{2a^2} = \frac{\log B}{2a^2\log(2a^2)}$. □

An implicit assumption in the above calculations is that n is sufficiently large: $\Omega(B^2)$ (B^d for the d-dimensional case). Furthermore, we assumed that certain quantities, such as $\log_c B$, are integers. It is easy to see that this assumption does not affect the results by more than a small constant factor when B is a large constant.

Set workloads

Let us now concentrate on workloads in which the domain is the powerset of $\{1, 2, \ldots, n\}$, and we have inclusion queries. We show that these workloads are far worse than 2-dimensional queries; in fact, they have worst-case access overhead regardless of the redundancy.

Theorem 3 *For each redundancy r, there exists a set inclusion workload such that the access overhead is B.*

Proof. Let n be an integer larger than rB^2 and consider a workload where I consists of all singletons $\{1\}$, $\{2\}, \ldots, \{n\}$ and each query is a subset of $\{1, 2, \ldots, n\}$ that has exactly B elements. We claim that the access overhead is B. To see this notice that each element can be in the same block with at most rB other elements. Therefore, there are at least $n/(rB) \ge B$ elements such that no two of them belong to the same block. So, a query of B such elements cannot be covered by less than B blocks. □

It is interesting to note that the number of queries involved in the proof of Theorem 3 is exponential in the size of the instance I. In contrast, the number of range queries is polynomial in the size of the instance I, with the dimension appearing in the exponent of the polynomial. It is hardly surprising that the trade-off between redundancy and access overhead is, in general, worse for workloads with a large number of queries. However, it is easy to see that there are workloads with exponential number of queries that have optimal trade-off. Consider, for example, the workload where the queries are unions of disjoint sets $S_1, S_2, \ldots, S_k$, each of size B, and $I = \cup_i S_i$. Then the indexing scheme with blocks $S_1, S_2, \ldots, S_k$ has redundancy $r = 1$, access overhead $a = 1$, and the number of queries is $a = 2^{\frac{|I|}{B}}$. On the other hand, notice that the instance I of a workload may have elements that do not appear in any query and consequently there are workloads with small number of queries (compared to the size of the instance) and worst trade-off.

4 Discussion and open problems

We believe that our framework will result in a useful theory of indexability and we provide here a detailed research program towards such a theory. This program spans the following directions:

Range queries. In this work, we consider mainly 2-dimensional workloads. It seems that there is space for improvement in the trade-off of Theorem 2, for large access overhead a; we conjecture that the correct trade-off is $r \geq \frac{\log B}{\log a}$. It is also useful to extend the results to 2-dimensional workloads that are not restricted to the lattice. It seems possible that better lower bounds can be shown for this case.

For higher dimensions the problem becomes qualitatively different. Theorem 1 of this paper provides a lower bound for the non-redundant case for lattice d-dimensional workloads. It is open whether a matching upper bound is achievable, even for small $d \geq 3$. Of course, the more general problem of characterizing the trade-off for the non-redundant case is more important.

One possible criticism of our lower bounds could be that they occur at range queries that are "long and narrow," extreme in their aspect ratio. It seems that our results can be extended to the case of queries with aspect ratio A, by replacing B with A in the lower and upper bound expressions. It will be also very interesting to come up with upper bounds for these types of queries.

Set inclusion workloads. The type of set inclusion queries that we consider here is too general and therefore the lower bound of Theorem 3 is only slightly informative. An interesting direction for future research is to consider set inclusion workloads with restricted type of queries. A natural way to do this is either to simply restrict the number of queries or to consider queries where each element appears in a bounded number of them.

In fact, any workload can be mapped to a set inclusion workload with restricted set of queries. In this sense, the class of set workloads is universal. It will be therefore very interesting to map the workloads of important indexing problems (such as range queries, similarity queries, k-nearest neighbors queries, and k-closest pairs queries) to set workloads and seek hidden common characteristics.

As with any theory of computational limitations, our results may help refocus research in indexing towards directions such as formally understanding the (statistical, geometric, or other) properties of workloads which empirically appear to defy our lower bounds. The fractal dimension has been mentioned as a parameter with some explanatory power in this regard [7, 3]; however, this does not address the impact of the *set of queries* on the performance of indexing schemes.

Dynamic and on-line workloads. In this work, we consider only static workloads. In practice, however, the most important type of workloads are dynamic, i.e., elements are inserted or deleted from the instance I and the set $\mathcal{Q}$ of queries changes respectively. It is an interesting open problem to extend our framework to the dynamic case.

In practice also, the set of possible queries is not completely known in advance. Consider an indexing scheme that is allowed to be re-organized in response to queries. The goal is to minimize the sum of the re-organization cost and the access cost. This on-line problem seems to capture the essence of many practical indexing problems and it will be very interesting to seek competitive algorithms for it [13].

Complexity of indexing schemes. Our framework suppresses important aspects of indexing by focusing on the trade-off between redundancy and access overhead. However, it may also help us to refocus our research on the complexity of indexability. More precisely, it allows to separate the two sources of complexity of indexing schemes. The first one is the complexity of designing an indexing scheme: Given a workload, find a set of blocks with a given redundancy that have minimum access overhead. The second one is the time and, perhaps more importantly, the space complexity of answering a query: Given a set of blocks and a query, find a minimum set of blocks (or a set of a blocks) that covers the query.

A possible approach to reduce the complexity of indexing schemes is to use randomization. Consider for example a random indexing scheme, i.e, the set of blocks are drawn randomly from a given distribution. For such a scheme we are interested in the expected redundancy and the expected access ratio. Or, we can fix the redundancy and ask for the expected access overhead. The main advantage of using randomized indexing schemes is that the complexity of designing them may be substantially lower than that of designing a deterministic indexing scheme. Similarly, it may sometimes be easier to show that a randomized indexing scheme has a good trade-off.

There is also an interesting direction for extending our models and results, motivated by some "new computational paradigms". One can imagine a situation where queries are "fuzzy," and need not be answered exactly. One might have a probabilistic distribution of queries, instead of a set, while each query is a weighted subset of the dataset, with weights representing relevance. We are interested in efficient access methods that return a large portion of the weight of each query, with high probability.

Varying size B of blocks. Our model of block size presupposes that all items of a domain have the same storage requirements. This is clearly a simplification in the case of set workloads, and indeed even in the case of simpler workloads such as those of the B-tree, which typically use "key compression" schemes when possible [4]. Even if all items require the same storage, most structures allow some empty space in the blocks. For most index structures, blocks are required to have between $B/2$ and B items.

An immediate extension of our work is to consider workloads where each element of the domain has a an associate weight (size), and the sum of weights in each block is bounded by some constant B. This gives rise to interesting weighted versions even for the problems studied in this paper.

Acknowledgments

The authors wish to thank the reviewers for their feedback.

References

[1] A. V. Aho, J. E. Hopcroft, J. D. Ullman. *Data Structures and Algorithms.* Addison Wesley, 1983.

[2] R. Bayer and C. McCreight. Organization and Maintenance of Large Ordered Indexes. *Acta Informatica* 1(3):173–189, 1972.

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

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

[5] R. A. Finkel and J. L. Bentley. Quad-Trees: A Data Structure For Retrieval On Composite Keys. *Acta Informatica*, 4(1):1–9, 1974.

[6] C. Faloutsos. *Searching Multimedia Databases By Content.* Kluwer Academic, 1996.

[7] C. Faloutsos and I. 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.

[8] C. Faloutsos and V. Gaede. Analysis of n-Dimensional Quadtrees using the Hausdorff Fractal Dimension. In *Proc. 22nd International Conference on Very Large Data Bases*, Mombai(Bombay), pages 40–50, September 1996.

[9] C. Faloutsos, Y. Matias and A. Silberschatz. Modeling Skewed Distribution Using Multifractals and the '80-20' Law. In *Proc. 22nd International Conference on Very Large Data Bases*, Mombai(Bombay), pages 307–317, September 1996.

[10] 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.

[11] 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*, Zurich, September 1995.

[12] L. Hellerstein, P. Klein, R. Wilber. On the Time-Space Complexity of Reachability Queries for Preprocessed Graphs. *Information Processing Letters, 25*, pages 261–267, 1990.

[13] S. Irani, A. Karlin. Online Computation. Chapter in *Approximation Algorithms for NP-hard Problems*, edited by D. Hochbaum. PWS Publishing, pages 261–267, 1996.

[14] H. V. Jagadish. Linear Clustering of Objects With Multiple Attributes. In *Proc. ACM-SIGMOD International Conference on Management of Data*, Atlantic City, May 1990, pages 332–342.

[15] P. C. Kanellakis, S. Ramaswamy, D. E. Vengroff, 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. (Recent version available from the www.)

[16] D. E. Knuth. *The Art of Computer Programming; Volume III: Searching and Sorting*. Addison Wesley, 1973.

[17] L. Lovász. *Combinatorial Problems and Exercises.* North-Holland, Amsterdam, 1979.

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

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

[20] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The Grid File: An Adaptable, Symmetric Multikey File Structure. *ACM Transactions On Database Systems*, 9(1):38–71, 1984.

[21] M. H. Nodine, M. T. Goodrich, and J. S. Vitter. Blocking for External Graph Searching. *Algorithmica*, 16 (2):181–214, August 1996.

[22] B.-U. Pagel, H.-W. Six, H. Toben, and P. 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.

[23] S. Ramaswamy, 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.

[24] 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.

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

[26] M. H. M. Smid, M. H. Overmars. Maintaining Range Trees in Secondary Memory. Part II: Lower Bounds. *Acta Informatica* 27(5):453–480, 1990.

[27] S. Subramanian, S. Ramaswamy. The p-range Tree: A Data Structure for Range Searching in Secondary Memory. *Proc. 6th SODA*, 1995.

[28] 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.

[29] R. E. Tarjan. *Data Structures and Network Algorithms.* SIAM, 1983.

[30] D. E. Vengroff, J. S. Vitter. Efficient 3-d Searching in External Memory. *Proc. 28th STOC*, pages 191–201, 1996.

[31] A. C.-C. Yao. Should Tables Be Sorted? *J.ACM 28*, pages 625–628, 1981.