RANGE SEARCHING DATA STRUCTURES WITH CACHE LOCALITY

by

Christopher H. Hamilton

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

at

Dalhousie University
Halifax, Nova Scotia
March 2011

© Copyright by Christopher H. Hamilton, 2011

DALHOUSIE UNIVERSITY

FACULTY OF COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "RANGE SEARCHING DATA STRUCTURES WITH CACHE LOCALITY" by Christopher H. Hamilton in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Dated: March 17, 2011

External Examiner: ____________________

Research Supervisor: ____________________

Examining Committee: ____________________

____

____

Departmental Representative: ____________________

DALHOUSIE UNIVERSITY

DATE: March 17, 2011

AUTHOR: Christopher H. Hamilton

TITLE: RANGE SEARCHING DATA STRUCTURES WITH CACHE LOCALITY

DEPARTMENT OR SCHOOL: Faculty of Computer Science

DEGREE: Ph.D. CONVOCATION: May YEAR: 2011

Permission is herewith granted to Dalhousie University to circulate and to have copied for non-commercial purposes, at its discretion, the above title upon the request of individuals or institutions. I understand that my thesis will be electronically available to the public.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

The author attests that permission has been obtained for the use of any copyrighted material appearing in the thesis (other than brief excerpts requiring only proper acknowledgement in scholarly writing), and that all such use is clearly acknowledged.

Signature of Author

Table of Contents

List of Tables

List of Figures

Abstract

This thesis focuses on range searching data structures, an elementary problem in computational geometry with research spanning decades. These problems often involve very large data sets. Processor speeds increase faster than memory speeds, thus the gap between the rate at which CPUs can process data and the rate at which it can be retrieved is increasing. To bridge this gap, various levels of cache are used. Since cache misses are costly, algorithms should be cache-friendly.

The *input-output* (I/O) model was the first model for constructing cache-efficient algorithms, focusing on a two-level memory hierarchy. Algorithms for this model require manual tuning to determine optimal values for hardware dependent parameters, and are only optimal at a single level of a memory hierarchy. *Cache-oblivious* (CO) algorithms are built without knowledge of the hierarchy, allowing them to be optimal across all levels at once.

There exist strong theoretical and practical results for I/O-efficient range searching. Recently, the CO model has received attention, but range searching remains poorly understood. This thesis explores data structures for CO range counting and reporting. It presents the first space and worst-case query-time optimal approximate range counting structure for a family of related problems, and associated $O(N \log N)$-space query-optimal reporting structures. The approximate counting structure is the first of its kind in internal memory, I/O and CO models. Researchers have been trying to create linear-space query-optimal CO reporting structures. This thesis shows that for a variety of problems, linear space is in fact impossible.

Heuristics are also used for building cache-friendly algorithms. Space-filling curves are continuous functions mapping multi-dimensional sets into one-dimensional ones. They are used to build search structures in the hopes that objects that were close in the original space remain close in the resulting ordering. This results in queries incurring fewer page swaps when traversing the structure. The Hilbert curve is notably good at this, but often imposes a space or time penalty. This thesis introduces compact Hilbert indices, which remove the ineffiency inherent for input point sets with bounding boxes smaller than their bounding hypercubes.

List of Abbreviations and Symbols Used

$[\cdot]_{[2]}$	Used to denote non-negative integers written in base 2
$\|\cdot\|$	Number of '1' bits in the binary representation of a non-negative integer (the parity)
$\vee$	Bitwise *or* operator
$\oplus$	Bitwise *exclusive-or* operator
$\wedge$	Bitwise *and* operator
$\overline{\wedge}$	Bitwise *not-and* operator
$\neg$	Bitwise *not/negation* operator
$\triangleleft$	Bitwise *shift-left* operator
$\triangleright$	Bitwise *shift-right* operator
$\circlearrowleft$	Bitwise *left-rotation* operator
$\circlearrowright$	Bitwise *right-rotation* operator
2-d	Two-dimensional
3-d	Three-dimensional
B	Memory block size
$\mathcal{B}$	A set of block sizes
$\mathbb{B}$	Set of boolean integers $\{0, 1\}$
$B(P)$	Smallest n-dimensional box $\mathbb{B}^{m_0} \times \cdots \times \mathbb{B}^{m_{n-1}}$ such that $P \subseteq B(P)$
B-cover	A block cover
$\mathbb{B}^k$	Set of positive integers of k bits, $\mathbb{Z}_{2^k}$
$\mathrm{bit}\,(a, k)$	Represents the value of the kth bit of a non-negative integer a
CO	Cache-oblivious
CPU	Central processing unit
d	A *direction* in $\mathbb{Z}_n$

$\mathrm{d}(0), \ldots, \mathrm{d}(2^n - 1)$	Sequence of *directions* in $\mathbb{Z}_n$ such that $\mathrm{e}(i) \oplus 2^{\mathrm{g}(i)} \oplus 2^{\mathrm{d}(i)} = \mathrm{e}(i+1)$
ε	A small positive quantity
e	An *entry point* in $\mathbb{B}^n$
$\mathrm{e}(0), \ldots, \mathrm{e}(2^n - 1)$	Sequence of *entry points* in $\mathbb{B}^n$
f	An *exit point* in $\mathbb{B}^n$
$\mathrm{f}(0), \ldots, \mathrm{f}(2^n - 1)$	Sequence of *exit points* in $\mathbb{B}^n$
FIFO	First in, first out
$\mathrm{g}(0), \ldots, \mathrm{g}(2^n - 2)$	Sequence of integers in $\mathbb{Z}_n$ such that $\mathrm{gc}(i) \oplus 2^{\mathrm{g}(i)} = \mathrm{gc}(i+1)$
gc	*Binary reflected Gray code* function
gcr	*Gray code rank* function
GIS	Geographic information system
h	A Hilbert index in $\mathbb{B}^M$
$H(P)$	Smallest n-dimensional hypercube $\mathbb{B}^m \times \cdots \times \mathbb{B}^m$ such that $P \subseteq H(P)$
$\mathcal{I}$	An indexing scheme
I/O	Input-output
K	Number of reported points/output size
L1	Level 1
L2	Level 2
L3	Level 3
LRU	Least-recently used
LV	Las Vegas

M	Size of main memory
m	Maximum precision, $m = \max_i\{m_i\}$
$m_0, \ldots, m_{n-1}$	Precision (number of bits) of each of the n dimensions
MC	Monte-Carlo
μ	A *mask* in $\mathbb{B}^n$
N	Number of points/problem size
$\mathbb{N}$	Set of natural integers, $\{0, 1, \ldots\}$
n	Number of dimensions
n-d	n-dimensional
P	A set of points
$\mathbf{p} = [p_0, \ldots, p_{n-1}]$	A point in the space $B(P)$
$\mathrm{perm}(N)$	External permutation complexity $\Theta(\min \mathrm{sort}(N), N)$
Q	A set of queries/subsets of S or P
q	A query
$\mathbb{R}$	Set of real numbers
RAM	Random access memory
S	A set
$\mathrm{scan}(N)$	External scanning complexity $\Theta\left(\frac{N}{B}\right)$
$\mathrm{sort}(N)$	External sorting complexity $\Theta\left(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B}\right)$
$T_{(e,d)}(b)$	Hilbert curve transformation function
tsb	*Trailing set bits* function

$\mathcal{U} = \{u_0 < \ldots < u_{\|\mu\|-1}\}$	Set of unconstrained bits associated with a given mask μ
$\mathcal{W}$	A workload
WC	Worst-case
X	Net precision, $X = \sum_i m_i$
$\mathbb{Z}$	Set of integers, $\{\ldots, -1, 0, 1, \ldots\}$
$\mathbb{Z}_+$	Set of positive integers, $\{1, 2, \ldots\}$
$\mathbb{Z}_k$	Set of integers modulo k, $\{0, \ldots, k-1\}$

Acknowledgements

During my work on this thesis I had the honour and pleasure of being supervised by Norbert Zeh, whose depth of knowledge came to my rescue on many occasions. I am also in debt to his dedication and invaluable guidance; he always had time to discuss new ideas and to review drafts, even while dealing with the upheaval of a sabbatical on another continent. Without his intellectual input, this work would have been impossible. I would also like to thank Peyman Afshani whose own thesis work brought to light the key insight allowing the main results of this thesis to unfold after almost two years of fruitless efforts.

I extend thanks to my committee members Andrew Rau-Chaplin and Alex Brodsky for their willingness to work through this thesis in detail, providing valuable feedback.

I thank NSERC, the Killam Foundation, Norbert Zeh and Dalhousie University for their generous financial support without which my continued studies would have been an impossibility.

I wish also to thank my friends at Dalhousie for making my time there memorable, specifically Mason Macklem, Glenn Hickey and Greg Zaverucha. They were always ready to talk about new ideas, and more importantly, to take my mind off of research when required.

My family deserves my gratitude for their continued encouragement and support. Specifically, my sister Sara Hamilton whose own doctoral journey provided me with much needed context and motivation.

Last but not least, my partner Caroline Piché has been with me through this entire process. She has coped with my distracted nature when buried in work, all the while wearing a smile. Her love and easy laughter has been a much needed source of energy and inspiration day after day.

Chapter 1

Introduction

1.1 Motivation

This thesis focuses on the development of algorithms and data structures for cache-efficient range searching. Range searching problems are among the most natural problems in computational geometry. As a consequence, they are among the most well-studied. Range searching exists in many forms (What space does the problem live in? What is being reported? What is the shape of the query regions?), with most variants finding practical applications. For example, orthogonal range searching problems are directly motivated by queries in database systems, while circular range searching queries find immediate use in *geographic information systems* (GIS). Many seemingly unrelated problems may also be reduced to range searching problems through appropriate geometric reductions, as will be discussed in more detail in Chapter 2.

Practical applications of range searching problems often involve extremely large data sets, particularly in the domains of database systems and GIS. This creates unique challenges. Computer systems have evolved to contain numerous layers of data storage (on-processor storage, RAM, hard-disk, network, etc), with layers further away from the processor typically having greater capacities but slower access times. Processor speeds have been increasing faster than memory speeds, thus the gap between the rate at which CPUs can process data and the rate at which data can be loaded from memory is constantly increasing. To bridge this gap, various levels of cache were introduced. In fact, modern computers typically have at least six such layers (L1, L2, L3, RAM, hard-drive cache, hard-drive itself). The hope is that most memory accesses can be served from the nearest cache. For most algorithms, this is not true. Since cache misses are quite expensive, it becomes extremely important for algorithms to manage data in a cache-friendly manner. The cost of a cache miss is smaller the higher up the hierarchy it occurs, as memory speeds increase as we climb the hierarchy. An obvious first approach is to minimize cache misses at the slowest level, but a good algorithm will be one that is simultaneously efficient across all levels in a cache hierarchy.

The *input-output* (I/O) model was the first model for designing cache-efficient algorithms. It ignores the higher levels of cache and only distinguishes between data stored in memory and that stored on the disk. In this manner, it minimizes cache misses at the single level of the hierarchy where they incur the largest cost. Algorithms designed in this model explicitly control the swapping of data between the two levels, and implementations require manual tuning to determine optimal values for what are essentially hardware dependent parameters.

The *cache-oblivious* (CO) model is a simple abstraction that allows the construction of cache-friendly algorithms without specific knowledge of the cache hierarchy. In the CO model algorithms are designed without referencing parameters of the model hierarchy, however, they are analyzed as in the I/O model. This abstraction means that the analysis holds for any memory hierarchy parameters, and thus simultaneously across all levels of a multi-level memory hierarchy. The I/O and CO models will be discussed in more detail in Section 2.1.

The database and GIS communities have long recognized the importance of cache-efficiency, with several I/O-efficient algorithms finding real application in these domains [21, 34, 68]. In particular, the problem of range searching has been well studied in the I/O model, with excellent practical results [27, 68, 119]. In the last decade, CO algorithms have received a lot of attention from the computational geometry community, however the problem of range searching is still poorly understood with many open questions.

Another approach to building cache-friendly algorithms is the use of various tried and proven heuristics [61, 106, 109]. Space-filling curves are continuous self-similar functions that map compact multi-dimensional sets into one-dimensional ones. They are commonly used in the creation of range searching data structures in the hopes that objects that are close in the original space remain close in the resulting one-dimensional data structure [51, 75, 78], usually a type of search tree. The end result is that leaves containing a point in a query very likely contain other points belonging to the same query. Thus, queries need to visit fewer leaves in the resulting data structure, and fewer page swaps are incurred. The Hilbert curve has been shown as being particularly well suited to this task [100]. However, depending on the specifics of their use, Hilbert curves impose either a space or time penalty. This thesis solves this problem with compact Hilbert indices, which remove the inefficiency inherent in Hilbert indices for input point sets with bounding boxes significantly smaller than their bounding hypercubes. Although this thesis focuses on Hilbert curves, many other space-filling curves exist [72].

1.2 Overview

This thesis focuses on cache-efficient range searching, with a focus on axis-aligned range searching problems. The work is divided into two parts. The first part discusses heuristics for data locality in range searching structures. Specifically, Chapter 4 presents a practical improvement to Hilbert index calculations and evaluates its real-world utility.

The second part of this thesis focuses on results in the cache-oblivious model. While both the CO model and the problems we discuss are practically motivated, the results presented in the latter part of this thesis are purely theoretical. Chapter 5 discusses lower bound results for range searching in the CO model, while Chapter 6 discusses data structures for range searching in the CO model.

1.2.1 Hilbert curves

In Chapter 4 we explore the Hilbert curve, reproducing classical algorithms [39, 43] for their generation and manipulation through an intuitive and rigorous geometric approach. We then extend these basic results to construct *compact Hilbert indices* which are able to capture the ordering properties of the regular Hilbert curve but without the associated inefficiency in representation for data sets whose bounding boxes are significantly smaller than their bounding hypercubes. Consider a set P of points in $\mathbb{Z}^n$. For simplicity and without loss of generality, we assume that the point set has been translated such that all coordinates are positive, and the bounding box has one corner at the origin. The points in P lie within a *power-of-two-sided bounding box* $B(P) = [0, 2^{m_1}] \times \cdots \times [0, 2^{m_k}]$ (letting $m_i \in \mathbb{N}$ be the minimum values such that this is true) and therefore can be naturally represented using $X = \sum_i m_i$ bits. However, the Hilbert curve is naturally defined over a *power-of-two-sided bounding hypercube* $H(P) = [0, 2^m] \times \cdots \times [0, 2^m]$, where $m = \max_i(m_i)$, with each position on the curve encoded by an mn-bit integer. For many real world data sets mn is significantly larger than X. By considering only the portion of the curve intersecting $B(P)$, compact Hilbert indices allow the ordering of the Hilbert curve to be preserved while only using X bits to represent each position on the compact curve. This leads to reductions in space-bounds for algorithms with precomputed Hilbert indices, or reductions in computation time for algorithms that repeatedly calculate the Hilbert index as needed. In particular, these results provide the first optimal time in-place Hilbert-order sort for multi-dimensional data sets whose power-of-two-sided bounding boxes are smaller than their power-of-two-sided bounding hypercubes. Our main technical contribution in this chapter is in the creation of compact Hilbert curves, generalizations of Hilbert curves that are naturally defined over $B(P)$ rather than $H(P)$.

1.2.2 Lower bounds

The focus of Chapter 5 is on 3-sided range reporting, 3-d dominance reporting, and 3-d halfspace range reporting. We prove the novel result that any cache-oblivious data structure for these problems that achieves the optimal (or in fact a much weaker) query bound has to use asymptotically more space than a structure with the same query bound in the I/O model. There exist linear- and $O(N \log^* N)$-space data structures that achieve the optimal query bound of $O(\log_B N + K/B)$ for these range searching problems in the I/O model [2, 7]. In contrast, the best known data structures achieving the same query bound in the cache-oblivious model use $O(N \log N)$ space [13, 25, 30]. Our lower bound shows that any cache-oblivious data structure for 3-sided range reporting, 3-d dominance reporting or 3-d halfspace range reporting that achieves a query bound of $f(\log_B N, K/B)$, for any monotonically increasing function $f(\cdot, \cdot)$, has to use $\Omega(N(\log \log N)^{\varepsilon})$ space. The only previously existing separation results between these two models are for sorting and searching. The sorting result used an adversarial argument over only two discrete block sizes [64]; the searching result applies in the limiting sense over many block sizes, but results in only a constant factor separation result [36]. Our approach is novel in that it is an explicit construction that argues simultaneously across many block sizes and yields the first input-size-dependent separation result between the I/O and cache-oblivious models.

1.2.3 Upper bounds

Chapter 6 focuses on the construction of a generic approximate range counting and range reporting data structure. Early I/O-efficient data structures for orthogonal range reporting are based on fairly direct externalizations of internal-memory data structures and techniques [12, 24, 107]. These approaches generally involve nested structures, where non-trivial secondary search structures are embedded in the nodes of a B-tree, with each secondary structure being queried on a root-to-leaf path. However, the entire technique of non-trivial secondary data structures breaks down in the CO model as it is impossible to ensure they are cache aligned for all possible values of B. This incurs a page swap per accessed secondary data structure, resulting in a non-optimal $O(\log N)$-cost search tree traversal. More recent I/O-efficient data structures employ shallow cutting and shallow partitions [7], but require explicit knowledge of B. Shallow partition techniques require the indexing of multiple secondary data structures and as such do not appear promising in a CO context. Chapter 6 shows how to exploit shallow cuttings to guarantee data locality in a cache-oblivious manner, something that has not been previously done.

Investigations into 3-sided range searching techniques yielded the insight that existing data structures in the I/O and CO model use an approach reminiscent of shallow cuttings [30]. Shallow cuttings being more general and powerful, this led to the development of a unifying framework that can additionally handle 3-d halfspace and 3-d dominance queries. The results are constructions for $O(N \log N)$-space cache-oblivious data structures with optimal query bounds across a family of problems. In the case of 3-sided range reporting, the results match those previously obtained in [13,25,30], but they are the first of their kind for the problems of 3-d halfspace, circular, 2-d K-nearest neighbour and 3-d orthogonal range reporting. These reporting data structures are fairly easy to obtain using a standard construction once the output size of a query can be efficiently determined or at least approximated.

Our main technical contribution is a general framework for constructing cache-oblivious data structures for the approximate counting versions of the above problems. These data structures use linear space and provide guaranteed relative $(1+\varepsilon)$-approximate answers using $O(\log_B(N/K))$ block transfers in the worst case, which is optimal. This is in contrast to previous results even in internal memory, where the optimal query bound was not achieved in the worst case before, even using superlinear space. The only previous data structure with the optimal query bound, by Afshani and Chan [6], achieves this bound only in the expected case. Thus, our construction also provides new worst-case optimal data structures for approximate 3-d halfspace range counting and approximate 3-d dominance counting in the pointer machine model. Tables 3.1 and 3.2 contrast our results with previous work.

Chapter 2

Preliminaries

In this chapter we introduce the common terminology, notation and concepts that will be used throughout this thesis. Definitions that apply only to particular chapters or sections will be provided as needed.

2.1 Models of Computation

Since the analysis of algorithms from an I/O-complexity point of view is not as well established as the analysis of their running time or their space requirements, we dedicate a few pages to the discussion of various models of computation. We discuss the relationship between external memory models and internal memory algorithms, allowing the comparison between existing algorithms for the problems we consider and our algorithms.

2.1.1 The Random Access Machine Model

The most commonly studied internal memory model is the *random access machine* (RAM) model. Algorithms designed for this model execute instructions sequentially and all operations are performed on data items stored in a conceptually unlimited main memory. Valid instructions consist of elementary arithmetic and logical operations, instructions to read and write data from or to memory and control instructions allowing the realization of branching, looping and recursion. Each elementary operation is assumed to take $O(1)$ time on a data item representable by $O(\log N)$ bits, where N is the size of the input. Thus, in order to estimate the amount of time necessary for an algorithm to solve a given problem, it suffices to count the number of steps executed by the algorithm.

A number of variants to this model have been proposed, most of which can be distinguished based on the set of operations that are considered primitives of the machine. Since most operations in more powerful RAM models can be emulated at a small (although non-constant) cost in a weaker RAM model, these variations are of little relevance to us, as computation cost is ignored in the analysis of external memory algorithms.

We will be confining ourselves to the standard *algebraic model* of computation, which allows only multiplication, division, addition and subtraction as primitive arithmetic operations. In particular, the floor function is not considered a primitive of this model, although it may be emulated in $O(\log N)$ time. The geometric algorithms discussed in this thesis will generally (unless stated otherwise) be assuming that the fundamental machine words are real numbers. This avoids the hassle of dealing with precision problems. However, these issues must be addressed when actually implementing these algorithms[1].

In the real world the size of main memory is limited and it is possible that the problem does not fit entirely in memory. In this case, it is possible for every read or write operation to cause data to be swapped between main memory and external memory (disk). Since a memory swap incurs significant cost (orders of magnitude slower than a primitive computation operation as permitted by the model [120, 121]), it is possible for optimal RAM algorithms to perform very poorly when realized on actual hardware. The following sections discuss models of computation that attempt to address this issue.

2.1.2 The Input-Output Model

The first widely accepted and used model for analyzing the complexity of algorithms in external memory was the *I/O model* of Aggarwal and Vitter [21]. This model considers two levels of memory: a single processor is equipped with a slow but conceptually unlimited disk (external memory), and a fast random access memory (internal memory) capable of holding M data items. The disk is partitioned into blocks of B consecutive data items. The processor may only perform computations on items held in internal memory. In order to access other data, the processor must first make room in the internal memory by transferring data items to the external memory or discarding it if no longer needed, at which point it may then load the desired data items into internal memory. Such memory transfers are called *I/O operations* or *block transfers*. In a single block transfer, the processor may shuttle one block of data items from external memory to internal memory, or vice-versa. The complexity of an algorithm in the I/O model is the number of block transfer it performs.

The I/O model explicitly ignores the time taken by the processor to perform any required computations. This is motivated by the fact that access times to a hard-disk are about six

[1]Obviously, care must be taken not to abuse the model by exploiting the infinite storage capacity of actual real numbers.

orders of magnitude slower than a computation step [120, 121]. An algorithm that performs fewer block transfers can be expected to be faster than one performing more block transfers, assuming the amount of computation performed by the first remains within reasonable bounds. Despite this fact, most I/O model algorithms are designed using the algebraic computation model, and thus may be analyzed in terms of RAM model complexity as well. In fact, it is entirely possible (and often the case) that algorithms exist which are optimal in both the I/O and RAM models simultaneously.

The I/O model has been widely adopted because it is a conceptually simple model that captures the main bottleneck in large-scale computations. The simplicity of the model has been important to its success as it allows relatively complicated problems to be explored and solved. Moreover, it typically results in solutions that can actually be implemented and yield real-world performance improvements over internal memory models. The disadvantage of this model is that the parameters M and B are essentially hardware dependent, and implementations must be optimized for each individual machine on which the code is to be run.

2.1.3 The Cache-Oblivious Model

The I/O model of computation only consider two levels of a memory hierarchy. However, modern computers typically have many memory levels, each higher level being faster yet also smaller than the previous. An example of such a hierarchy is the following: network storage, local disk, disk cache, RAM, L2 CPU cache, L1 CPU cache. The straight-forward extension of the I/O model to a multi-level hierarchy would require the addition of a block-size parameter between each neighboring pair of levels in the hierarchy. Design and analysis of algorithms in such a model quickly becomes cumbersome. Over the years, various parameterized models have been proposed for handling multi-level cache hierarchies [20, 22], but they have proven unwieldy.

Frigo et al. proposed an elegant model avoiding this problem [64]: the *cache-oblivious* (CO) model. In their model, the algorithm is oblivious of the memory hierarchy and, thus, cannot initiate block transfers explicitly. Instead, the swapping of data between two levels of memory is the responsibility of a paging algorithm, which is assumed to be *offline optimal*. That is, it performs the minimum number of block transfers possible for the memory access sequence of the algorithm. Such a paging algorithm will elect to evict the block of memory which will be accessed the furthest in the future. Clearly, it is impossible to know this at runtime, so real-world caches typically use simpler cache eviction policies:

- *Least-Recently Used* (LRU). Evict the block which was last accessed the longest time ago.
- *First In, First Out* (FIFO). Evict the block which has been in the cache the longest.

Sleator and Tarjan [114] showed that both LRU and FIFO are within a constant factor of the optimal strategy. More precisely, they showed that

$$\text{cost}(\text{LRU or FIFO cache with size } 2M) \leq 2 \times \text{cost}(\text{optimal cache with size } M),$$

where 'cost' counts the number of incurred block transfers. For most algorithms, changing M by a constant factor incurs at most a constant factor penalty, thus we may generally assume an optimal paging strategy, justifying use of the model.

Algorithms in this model are thus designed as internal memory algorithms, but analyzed in the I/O model with respect to an arbitrary block size B. Since the memory parameters are used only in the analysis, the analysis may be applied to *any* two consecutive levels of the memory hierarchy. In particular, if the analysis shows that an algorithm is optimal with respect to two levels of memory, then it is simultaneously optimal at all levels of the memory hierarchy. As with the I/O model, it is possible to design algorithms that are optimal in both the RAM model and the CO model, and as an immediate consequence, the I/O model as well. Much as the I/O model, the CO model has seen success because it is conceptually simple enough to allow the study of hard problems.

Remark. While motivated by practical considerations, the CO model has yet to find much in the way of real-world applications; the few algorithms that have been implemented rarely perform better than their I/O-model counterparts [41]. Those algorithms that have not been implemented, including the results of Chapter 6, use complex constructions that would likely incur very large constant factors.

2.2 Space-Filling Curves and Data Locality

Space-filling curves are continuous one-to-one functions which map a compact interval to a multi-dimensional unit hypercube. Originally formulated by Giuseppe Peano in 1890 [103], the first space-filling curve was constructed to demonstrate the somewhat counter-intuitive result that the infinite number of points in a unit interval has the same cardinality as the infinite number of points in any bounded finite-dimensional set. In this thesis we are specifically interested in their application to database systems, particularly orthogonal range queries.

Since Peano introduced the first space-filling curve, numerous others have been constructed and extensively studied. Among these further developments is the family of curves generated by Hilbert [74], which to this day finds many applications. Due to the recursive geometric nature of the original construction, the Hilbert curve naturally imposes an ordering on the points in finite square grids. An ordering with *data locality* is a one-dimensional ordering of the points in a set $P \subseteq \mathbb{Z}^n$ such that points that are close in the original space tend to remain close in the one-dimensional ordering. The end result is that leaves containing a point in a query very likely contain other points belonging to the same query. Thus, queries need to visit fewer leaves in the resulting data structure, and fewer page swaps are incurred. The Hilbert curve has been shown to be particularly well suited to this task [100].

While algorithms designed in the I/O and CO models attempt to rigorously categorize cache performance, space-filling curves are used as a heuristic to improve the cache performance of simple internal memory algorithms. The very popular R-tree data structure [68] (and its many variants) was initially conceived as a spatial variant to B-trees for orthogonal range queries in n dimensions. In a standard R-tree, data points are clustered into B sets of similar size with each set being represented in the search tree by its minimum bounding box. The division is repeated recursively until leaves contain roughly B points. The end result is a standard B-tree structure over potentially overlapping bounding boxes. Axis-aligned spatial queries are easily propated through the data structure, allowing efficient queries. Variants of the R-tree typically play with the method in which points are clustered into individual boxes at each step of the recursive division, how the boxes themselves are ordered and stored, as well as the details of how insertions and deletions are performed. Space-filling curves can be used to both segment data and order individual boxes, resulting in improved cache performance [78].

2.3 Notation

When discussing I/O-efficient and cache-oblivious algorithms we will use N to denote the size of the problem (number of data items), B to denote the block size and M to denote the number of data items that fit in internal memory. The use of K will be reserved for output-sensitive algorithms, and denotes the number of data items produced in the output or the actual value of the output when it is a single scalar value.

When discussing range searching problems, we will generally use P to refer to a set of points in $\mathbb{R}^n$, with n being reserved to indicate the dimensionality of the problem. The point set P is the input to such problems, thus it follows that $N = |P|$. Similarly, we will generally use q when referring to a query over P, where q is a query of the type admissible by the range searching problem being considered. We reserve Q to represent a set of queries and $\mathcal{Q}$ to represent the set of all valid queries admitted by the problem.

When discussing complexity, ε will always refer to a small positive constant. For two values a and b, if $a \geq (1-\varepsilon)b$, then we say a is *ε-approximately greater* than b. Similarly, if $a \leq (1+\varepsilon)b$, then we say a is *ε-approximately less* than b. If both conditions hold, we say that a is ε-approximately equal to b. We will also occasionally use the notation $\mathrm{O}_\varepsilon(\cdot)$ to hide constant factors that depend on ε.

Although asymptotic notations can more properly be thought of as representing sets of functions, we will occasionally abuse the notation slightly. For example, rather than writing $4x^2 + x \in \mathrm{O}(x^2)$ we will write $4x^2 + x = \mathrm{O}(x^2)$. Similar usage will be made of $\Theta(\cdot), \Omega(\cdot)$, etc.

We will occasionally make use of the following shorthands for the I/O-complexities of sorting, permuting and scanning a list of N data items:

$$\begin{aligned}
\mathrm{sort}(N) &= \Theta\left(\frac{N}{B}\log_{\frac{M}{B}}\frac{N}{B}\right), \\
\mathrm{perm}(N) &= \Theta(\min(\mathrm{sort}(N), N)), \quad \text{and} \\
\mathrm{scan}(N) &= \Theta\left(\frac{N}{B}\right).
\end{aligned}$$

These shorthands have been introduced in the literature because they arise frequently in the analysis of I/O algorithms.

Scanning and sorting are fundamental operations in many algorithms. In particular, external memory algorithms often sort the input data appropriately, before scanning the elements and processing them one by one. Equivalent internal memory algorithms do not require such preprocessing as the elements may be accessed in a random fashion without any performance penalty.

These I/O complexities also have important and obvious relations to well-known time-complexities in internal memory. In particular, $\text{sort}(N)$ is the external memory equivalent of the $\Theta(N \log N)$ time bound for sorting N data items. As such, if a problem can be solved in $O(N \log N)$ time in internal memory, we hope to be able to solve it in $O(\text{sort}(N))$ I/Os in external memory. Similarly, the above scanning bound is the external memory equivalent of linear scanning in internal memory. Hence, we will refer to $\text{scan}(N)$ as a *linear* number of I/Os, while $O(N)$ I/Os is considered to be superlinear. This is a natural interpretation as any external memory algorithm spending $O(N)$ I/Os does not utilize the full bandwidth of the I/O system. Finally, the permutation bound is interesting as it is superlinear, while permutation in internal memory takes linear time. This is interesting in that it provides a first separation result between the internal- and external-memory models. It also implies that a superlinear external-memory lower bound can be shown for many non-trivial problems that can be solved in linear time in internal memory, simply because these problems inherently encapsulate some permutation problem.

2.4 Definitions

In this section we introduce standard definitions and conventions used in the range searching literature. In particular, we precisely define the various range searching problems that will be addressed in later chapters.

2.4.1 Range Searching Problems

Consider a set P of points in $\mathbb{R}^n$; the problem of geometric range searching is to (efficiently) preprocess P so that, for any given query range q, information regarding the points in $P \cap q$ can be efficiently reported. The exact nature of this information depends on the type of range searching problem. In *range reporting*, we are interested in efficiently enumerating all of the points $P \cap q$. In *range counting*, we are interested in reporting the size of the intersection, $|P \cap q|$. *Range emptiness* problems are decision problems where we are interested in determining if the query range q contains at least one point. Finally, *range optimization* problems are interested in finding a single best point in the query range with respect to some criterion (i.e., that with the largest weight, or with the smallest y coordinate, etc).

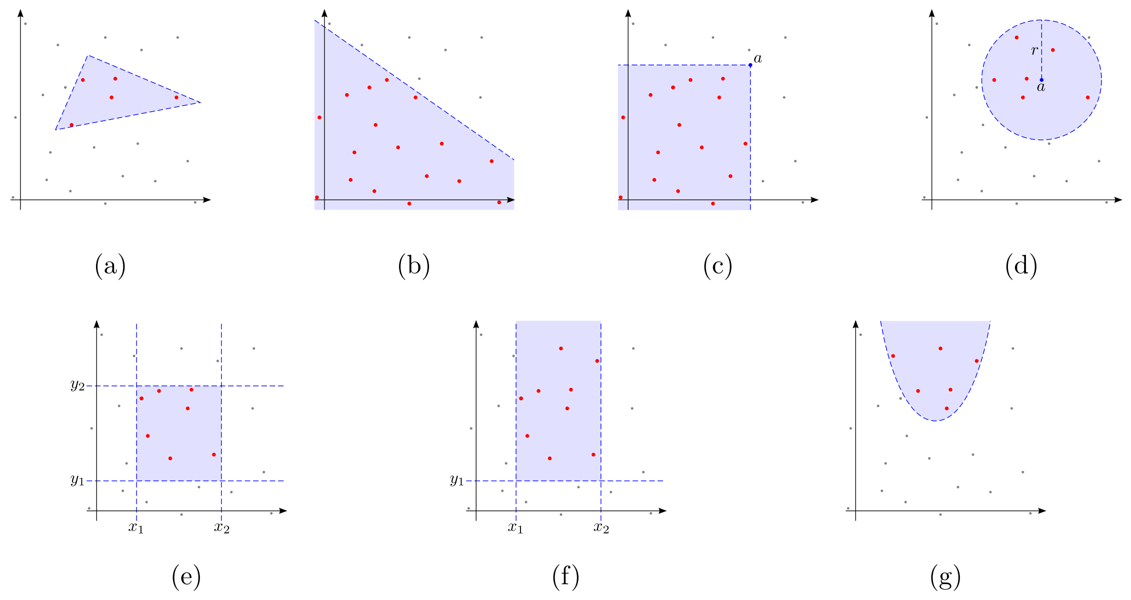


Figure 2.1: Examples of various range searching query types in two dimensions. (a) Simplex. (b) Halfspace. (c) Dominance (2-sided). (d) Circular. (e) Orthogonal (4-sided). (f) 3-sided. (g) Parabolic.

In the context of Hilbert curves, we will assume that P consists of points in $\mathbb{Z}^n$, translated such that all coordinats are positive and the minimum bounding box includes the origin. We will also make extensive reference to various bounding boxes containing P. Consider the smallest bounding box $[0, s_1 - 1] \times \cdots \times [0, s_d - 1]$ containing P and the origin. The side lengths of this bounding box are s_i, each requiring $m_i = \lceil \log_2 s_i \rceil$ bits to represent. We define $B(P) := [0, 2^{m_1}] \times \cdots \times [0, 2^{m_d}]$ to be the smallest power-of-two-sided bounding box containing P. Let $m = \max_i(m_i)$, and define $H(P) := [0, 2^m] \times \cdots \times [0, 2^m]$ as the smallest power-of-two-sided bounding hypercube containing P. As will be discussed in Chapter 4, the Hilbert curve through a set P is naturally defined over the (sometimes much) larger space $H(P)$.

It is usually the case that $\mathcal{Q}$ is restricted to a specific class of geometric objects. A few important types of geometric range searching include the following (see Figure 2.1 for example queries):

- *Simplex range searching*: In this problem, the query object q is a *simplex*, a convex hull in $\mathbb{R}^n$ formed by $n + 1$ points. Since any polyhedral object can be decomposed into simplices, the simplex case is a fundamental problem.

- *Parabolic range searching*: Here we limit ourselves to query regions consisting of the points lying on and above a parabola in $\mathbb{R}^n$.

- *Circular (spherical) range searching*: In this problem, the query regions are circles (hyperspheres) of arbitrary radius in $\mathbb{R}^2$ ($\mathbb{R}^n$). Problems of this type are regularly encountered in *GIS* (geographic information systems). As an example, consider the task of finding all restaurants within a given radius of a given location.

- *Orthogonal range searching*: Here the query objects are orthogonal boxes. Many problems outside computational geometry can be reduced to this case. The canonical example arises naturally in the context of databases. Consider the problem of finding all employees aged between x_1 and x_2 who earn a salary between y_1 and y_2. If employees are represented as points in $\mathbb{R}^2$, then this query amounts to reporting all points lying in the orthogonal range $[x_1, x_2] \times [y_1, y_2]$. In two dimensions, we often refer to orthogonal range searching as *4-sided range searching.* Similarly, the case where $y_2 = \infty$ is referred to as *3-sided range searching.*

- *Dominance range searching*: Here the query range is defined by a single point $a \in \mathbb{R}^n$, and the query region consists of the points of $\mathbb{R}^n$ that are smaller than a in *each* coordinate. In two dimensions, this is the same as 2-sided range searching. Using standard reductions, solutions to this problem can be used to construct efficient solutions to 3-sided and orthogonal range searching [4, 5]. Additionally, 3-sided range searching can be reduced to 3-d dominance searching via a simple geometric transformation (see Lemma 2.1).

- *Halfspace range searching*: In this problem the query object is a halfspace. Algorithmically, a halfspace may be considered as a simplex with one vertex placed sufficiently far away and thus this is a special case of simplex range searching. However, considered on its own it often allows more efficient specialized solutions. It is also interesting because both spherical range searching and parabolic range searching in $\mathbb{R}^n$ reduce to halfspace range searching in $\mathbb{R}^{n+1}$ via elementary geometric transformations (see Lemmas 2.2 and 2.3).

There are a variety of other range searching problems, and different ways to classify them. We limit ourselves to the above problem types, but more complete lists may be found in [12, 59, 96].

For most range reporting problems in up to 3 dimensions the optimal internal memory query bound is $O(\log N + K)$, with the $\log N$ corresponding to the cost of a search in some data structure and the K being the cost of reporting the K items that lie in the given query range. In external memory the natural cost of reporting K items is $\text{scan}(K) = K/B$. An information theoretic argument shows that the natural search cost is $\log_B N$. Consider a search tree with N possible outcomes. Specifying a single leaf in this tree requires $\lg N + O(1)$ bits of information. A comparison yields exactly 1 bit of information, yielding the $\lg N + O(1)$ lower bound on the number of comparisons necessary to navigate the tree. However, each block read reveals where the query element lies with respect to B elements in the tree, providing at most $\lg B + O(1)$ bits of information. Thus the number of block reads is at least $(\lg N + O(1))/(\lg B + O(1)) = \log_B N + O(1)$, yielding an optimal range reporting query bound of $\Omega(\log_B N + K/B)$.

As mentioned above, it is often the case that a range searching problem of one type may be transformed to a range searching problem of another type. In fact, there exist simple geometric transformations that allow 3-sided range queries to be decomposed into 3-d dominance queries, 2-d parabolic queries to be mapped to 3-d halfspace queries, and circular range searching queries to be mapped to 3-d halfspace queries. Similarly, these results can easily be lifted to higher dimensions.

Lemma 2.1 (3-sided $\subset$ 3-d dominance)**.** *Three-sided range queries can be solved by a data structure for 3-d dominance queries.*

Proof. Consider a 3-sided range query $[x_1, x_2] \times [y_1, \infty)$. We transform points using the mapping

$$p \mapsto (-p_x, p_x, -p_y),$$

and consider 3d-dominance queries of the form $q = (-x_1, x_2, -y_1)$. The transformed point is dominated by the query point if and only if

$$-x_1 \geq -p_x \quad \text{and} \quad x_2 \geq p_x \quad \text{and} \quad -y_1 \geq -p_y,$$

which is equivalent to

$$x_1 \leq p_x \leq x_2 \quad \text{and} \quad y_1 \leq p_y.$$

This is exactly the condition required for the original point to fall within the original 3-sided query. □

Lemma 2.2 (2-d parabolic $\subset$ 3-d halfspace)**.** *Parabolic range searching problems in* $\mathbb{R}^2$ *may be solved by a data structure for halfspace range searching in* $\mathbb{R}^3$*.*

Proof. A point $p = (p_x, p_y)$ is on or above a query parabola with equation $s(x - x_0)^2 + y_0$ if and only if

$$p_y \geq s(p_x - x_0)^2 + y_0.$$

Consider the halfspace defined by $\{(x, y, z) \in \mathbb{R}^3 : ax + by + cz + z_0 \leq 0\}$. We transform points with the map

$$p \mapsto (p_x, p_y, p_x^2).$$

Letting the halfspace be parameterized by the 4-tuple (a, b, c, z_0), we transform queries with the map

$$(s, x_0, y_0) \mapsto (-2sx_0, -1, s, sx_0^2 + y_0).$$

In this setting, a point (p_x, p_y, p_x^2) lies in the halfspace if and only if

$$\begin{aligned} 0 \geq&\quad -2sx_0p_x - p_y + sp_x^2 + sx_0^2 + y_0 \\ p_y \geq&\quad s(p_x^2 - 2x_0p_x + x_0^2) + y_0 \\ p_y \geq&\quad s(p_x - x_0)^2 + y_0. \end{aligned}$$

This is equivalent to the point (p_x, p_y) lying above the parabola defined by the 3-tuple (s, x_0, y_0). □

Lemma 2.3 (Circular $\subset$ 3-d halfspace)**.** *Circular range searching queries may be solved by a data structure for halfspace range searching in* $\mathbb{R}^3$*.*

Proof. Consider a circular range searching query consisting of the circle of radius r centered at the point $c = (c_x, c_y) \in \mathbb{R}^2$. A point $p = (p_x, p_y)$ lies within this query if and only if

$$(p_x - c_x)^2 + (p_y - c_y)^2 \leq r^2. \tag{2.4}$$

Consider mapping the point p to a point in $\mathbb{R}^3$ via the mapping

$$p \mapsto (p_x, p_y, p_x^2 + p_y^2),$$

and consider mapping a query circle $q = (c, r)$ to the halfspace parameterized by

$$(c, r) \mapsto (-2c_x, -2c_y, 1, c_x^2 + c_y^2 - r^2).$$

A point lies in this halfspace if and only if

$$\begin{aligned}
-2c_xp_x - 2c_yp_y + p_x^2 + p_y^2 + c_x^2 + c_y^2 - r^2 &\leq 0 \\
(p_x^2 - 2c_xp_x + c_x^2) + (p_y^2 - 2c_yp_y + c_y^2) &\leq r^2 \\
(p_x - c_x)^2 + (p_y - c_y)^2 &\leq r^2,
\end{aligned}$$

which is exactly Equation 2.4. □

Straight-forward extensions to Lemmas 2.2 and 2.3 allow the construction of similar lemmas over $\mathbb{R}^n$.

Since many fundamental range searching problems have strong connections to halfspace range searching, this problem is of particular importance. Many interesting results on halfspace range searching make use of the natural notion of geometric duality [59], as will be explored in the following section.

2.4.2 Levels

The concept of levels arises quite naturally when studying problems that deal with sets of points or arrangements of hyperplanes in $\mathbb{R}^n$. The concept was first explored by Lovász [89] and Erdős [62]. They defined a *K-set* for a set P of N points in the plane as a subset of P of size N that can be separated from the rest of P by a line (see Figure 2.2(a) for an example). Bounding the maximum number of K-sets as a function of N and K is known as the K-set problem. The concept of K-sets can be extended to higher dimensions through the use of separating hyperplanes.

Point-hyperplane duality is a well-known elementary geometric transformation that preserves the spatial relationship between points and hyperplanes [59]. We represent the dual of a geometric object s (or a set S of geometric objects) with $\bar{q}$ (or $\bar{S}$). Through this duality, a point p below a hyperplane h is mapped to a hyperplane $\bar{p}$ which passes strictly below the point $\bar{h}$. Thus, a subset of points of P below a hyperplane h corresponds to a subset of hyperplanes of $\bar{P}$ passing below[2] the point $\bar{h}$. This problem can be thought of as the *(hyperplane) aboveness reporting problem*, an example of which is shown in Figure 2.2(b).

Given an arrangement of hyperplanes A in $\mathbb{R}^n$, we define the *level* of a point p as the number of hyperplanes passing directly below p (as demonstrated in two dimensions in

[2]In this context, *below* can be more precisely defined as $\bar{P}$ passing through the half-ray extending vertically downwards from the point $\bar{h}$.

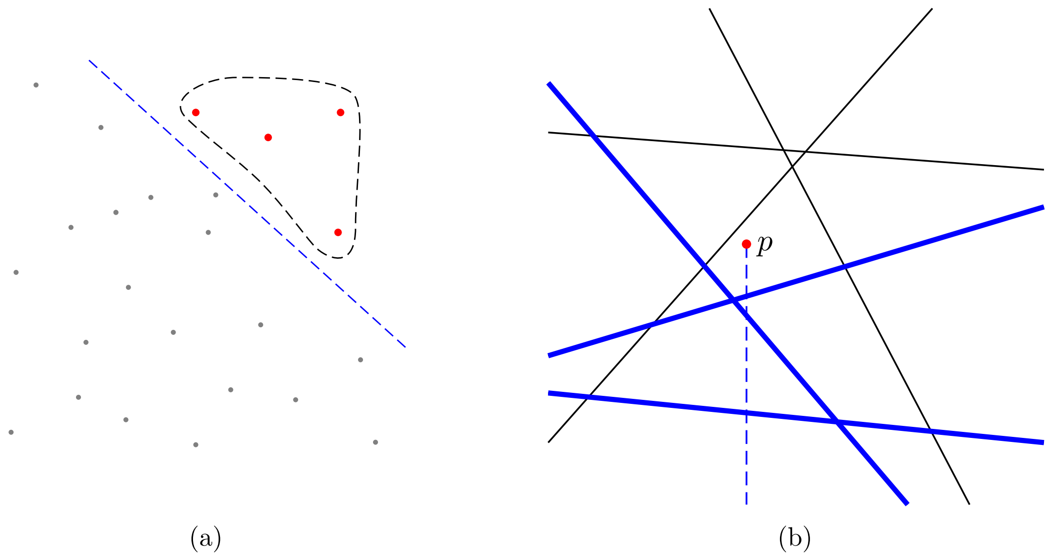


Figure 2.2: (a) An example K-set ($K = 4$) in $\mathbb{R}^2$ and a separating line. (b) An example of a 2-dimensional aboveness reporting query.

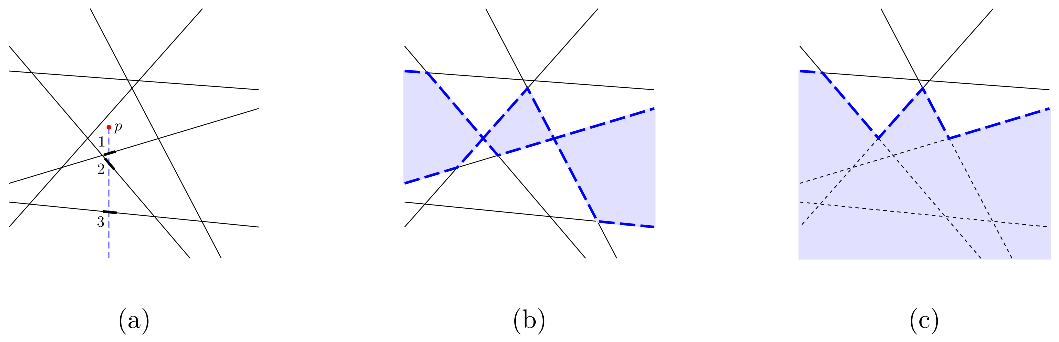


Figure 2.3: (a) A point p with a level of 3. (b) The K-level of an arrangement of lines ($K = 3$). (c) The ($\leq K$)-level of an arrangement of lines ($K = 3$).

Figure 2.3(a)). Consequently, the K-level of A is the closure of the set of all points of A with level equal to K (see Figure 2.3(b)). The size of the K-level is the number of vertices of A contained in it. In dual space, the K-set problem asymptotically translates to bounding the size of the K-level of A. This is known as the *K-level problem*.

Research into K-levels has led to many useful generalizations. Many results can be applied to the K-levels of *pseudo-lines* or *pseudo-halfplanes* as well. An arrangement of x-monotone curves is called an arrangement of pseudo-lines if every two curves intersect at most once. There are many papers dealing with such generalizations in the plane [15, 16, 18, 19, 47, 48, 50, 91, 104, 115, 116]. Similar generalizations for convex or concave hypersurfaces in higher dimensions have received some attention [49, 83, 113]. These generalizations all have associated aboveness reporting problems.

In many applications it is more convenient to talk about ($\leq K$)-levels, the closure of the set of all points of $\mathbb{R}^n$ with level at most K (see Figure 2.3(c)). The 0-level is often referred to as the *lower envelope* of an arrangement A. The term *complexity* is used interchangeably with *size* when referring to K- and ($\leq K$)-levels. It is relatively easy to construct arrangements where every K-level of A has size $\Theta(N)$. Thus, $\Omega(NK)$ is an obvious lower bound on the worst case size of the ($\leq K$)-level and a matching $\mathrm{O}(NK)$ upper bound can be proved.

Here we present a short proof based on the randomized techniques of Clarkson and Shor [57]. To use this technique we need surprisingly few elementary tools and an upper bound on the 0-level of the arrangement. In two dimensions, the worst case complexity of the 0-level is $\mathrm{O}(N)$.

Lemma 2.5 ([57]). *The complexity of the ($\leq K$)-level of an arrangement A formed by a set P of N lines in the plane is* $\mathrm{O}(NK)$.

Proof. Let S be a random p-sample of P (a subset of P where each element is chosen independently with probability p). Let L_0 be the 0-level of the arrangement formed by S. Fix $p := K^{-1}$. We have

$$\mathrm{E}[|L_0|] = \mathrm{O}(Np) = \mathrm{O}\left(\frac{N}{K}\right). \tag{2.6}$$

Let C be the set of all the vertices contained in the ($\leq k$)-level of A. We compute the probability of a vertex $v \in C$ appearing on the lower envelope of S. Let l' and l'' be the two lines incident to v, and let $l_1, \ldots, l_t$ be the lines which pass below v. Since we have assumed that v lies inside the ($\leq K$)-level of A, we must have that $t \leq K$. The probability of v appearing on L_0 is equal to the probability of l' and l'' being chosen in S times the probability that none of the lines $l_1, \ldots, l_t$ are chosen in S. Since $t \leq K$ we have

$$\Pr[v \in L_0] \geq p^2(1-p)^K = \Theta\big(p^2 e^{-Kp}\big) = \Theta\big(K^{-2}\big).$$

Thus, it follows that $\mathrm{E}[|L_0|] = \Omega(|C|K^{-2})$. Combining with Equation 2.6 we have

$$|C|K^{-2} = \mathrm{O}\left(\frac{N}{K}\right),$$

or equivalently, $|C| = \mathrm{O}(NK)$. ☐

This is a fundamental upper bound on $(\leq K)$-level complexity. In three dimensions, the worst case complexity of the lower envelope is $\mathrm{O}(N)$ and the same technique yields the following bound, stated here without proof.

Lemma 2.7. *The complexity of the* $(\leq K)$*-level of an arrangement* A *formed by a set* P *of* N *hyperplanes in* $\mathbb{R}^3$ *is* $\mathrm{O}(NK^2)$*.*

2.4.3 Cuttings and Partitions

The divide-and-conquer technique is fundamental to the design of many successful algorithms. Cuttings provide a natural way to divide a geometric problem into balanced and bounded subproblems. Let H be a set of N hyperplanes in $\mathbb{R}^n$. A $1/r$-cutting of H is a set of disjoint simplices C which cover H such that each simplex $s \in C$ intersects at most N/r hyperplanes of H. For a simplex $s \in C$, we call the subset of H intersecting s the *conflict list* of s and we denote it by Δ_s. The size of the cutting is the number of simplices in C. The main cutting theorem is the following, due to Chazelle [54, 55].

Theorem 2.8 (Cutting Theorem [54, 55])**.** *For every parameter* $0 < r < N$*, there exists a* $1/r$*-cutting of size* $\mathrm{O}(r^n)$ *for a set* H *of hyperplanes in* $\mathbb{R}^n$*.*

The bound $\mathrm{O}(r^n)$ on the size of the cutting is tight: N hyperplanes form $\Theta(N^n)$ vertices and a simplex intersecting m hyperplanes can contain at most $\mathrm{O}(m^n)$ vertices of an arrangement. So each simplex in a $1/r$-cutting contains $\mathrm{O}((N/r)^n)$ vertices and thus, there must be $\Omega(N^n/(N/r)^n) = \Omega(r^n)$ simplices in the cutting.

Consider a set P of N points in $\mathbb{R}^n$. A *simplicial partition* Π for P is a partition of P into r subsets $P_1, \ldots, P_r$ of roughly the same size together with a list of simplices $s_1, \ldots, s_r$ such that P_i lies inside s_i. The *crossing number* of any hyperplane h in this simplicial partition is defined as the number of simplices crossed by h. The maximum value of the crossing number over all hyperplanes h is called the crossing number of Π. When constructing simplicial partitions, it is desirable (from a data structure point of view) that the crossing number of the partition be minimized. Matoušek was the first to construct an optimal simplicial partition of P [93], where the crossing number is $\mathrm{O}\big(r^{1-1/n}\big)$.

Both cuttings and partitions are useful in building divide-and-conquer data structures for aboveness reporting problems [7, 95, 96]. Partitions are space efficient in that every hyperplane is found in exactly one of the subsets. However, each hyperplane has a relatively high crossing number, forcing us to look at multiple secondary data structures as we recurse down the search tree. Cuttings take the opposite approach; by oversampling the hyperplanes, super-linear storage space is required but only *one* substructure needs to be queried as we descend the search tree. Both approaches work in internal memory, but partitions require the indexing of too many secondary data structures in external memory models. As such, cuttings (and particularly shallow cuttings) have received more attention in these models.

2.4.4 Shallow Cuttings

In certain contexts, 'shallow' versions of the cutting and the partition theorem can be formulated, as originally noticed by Matoušek [94]. In the shallow version of the cutting theorem we are not partitioning the entire space. Rather, for parameters K and r a K-*shallow* $1/r$-*cutting* for a set P of hyperplanes is a set of disjoint simplices C which cover the $(\leq K)$-level of P, and where each simplex $c \in C$ intersects at most N/r hyperplanes of P. While any ordinary cutting is a shallow cutting, better size bounds are obtainable for shallow cuttings.

Lemma 2.9 (Existence of shallow cuttings, [94])**.** *For a set P of hyperplanes in $\mathbb{R}^n$ and parameters r and $K < N$, a K-shallow $1/r$-cutting of size $\mathrm{O}\big(r^n(K/N)^{\lceil n/2 \rceil}\big)$ always exists.*

The above lemma is usually most useful when $r = N/K$. Applying this and a few other modifications results in the following 3-d variant of the lemma.

Lemma 2.10 (Lemma 1.1.6 of [3])**.** *For any set of N planes in $\mathbb{R}^3$ and a parameter K, there exists a K-shallow $\mathrm{O}(K/N)$-cutting of size $\mathrm{O}(N/K)$ that covers the $(\leq K)$-level. The cells in the cutting are all vertical prisms unbounded from below (simplices with one vertex at $(0, 0, -\infty)$).*

Furthermore, we can construct these cuttings for all K of the form $\lfloor (1+\varepsilon)^i \rfloor$ simultaneously in $\mathrm{O}_\varepsilon(N \log N)$ time. The conflict lists may also be constructed in this time.

Outline of proof. The first part follows from Lemma 2.9. The construction time follows from an algorithm by Ramos [108]. The fact that vertical prisms suffice was observed by Chan, and converting to an equivalent prism shallow cutting requires computing the convex hull of the vertices in the original shallow cutting [45]. To compute the conflict lists we begin by building a halfspace range reporting data structure in $\mathrm{O}_\varepsilon(N \log N)$ expected time. For each K we issue $\mathrm{O}(N/K)$ halfspace range reporting queries (one per prism vertex), each requiring $\mathrm{O}_\varepsilon(\log N + K)$ time, for a total of $\mathrm{O}_\varepsilon(N(\log N + K)/K)$. Summing this over all K gives $\mathrm{O}_\varepsilon(N \log N)$. □

Finding simplices that are all prisms allows us to easily construct a data structure for querying the level of a point. Consider one level of the cutting. Project the prisms on the xy-plane, yielding a triangulation composed of $\mathrm{O}(N/K)$ faces. Each face of the triangulation stores the equations of the plane corresponding to the top face of its vertical prism. For any point $q \in \mathbb{R}^3$ we can easily find the face containing the point's projection on the xy-plane using a *planar point location* data structure. For a triangulation containing N faces, such a structure can be built in $\mathrm{O}(N \log N)$ time and $\mathrm{O}(N)$ space, answering queries in $\mathrm{O}(\log N)$ time. Thus, in $\mathrm{O}(\log(N/K))$ time we can test whether q lies below a given level of the cutting and if so, determine the vertical prism containing it.

Remark. It is worth noting that the proofs relating to shallow cuttings are merely existence results, and say nothing about actually building them. The actual algorithms used for constructing shallow cuttings (such as that by Ramos [108]) are directly descended from the powerful randomized techniques of Clarkson and Shor [57]. Thus, while many data structures based on shallow cuttings have deterministic run times, they only have expected bounds on the preprocessing required to actually build them.

2.4.5 Planar Point Location

Point location problems are natural and well-studied problems in computational geometry. Consider a set S of n-dimensional objects in $\mathbb{R}^n$; the point location problem consists of efficiently preprocessing S so that, for any given query point $p \in \mathbb{R}^n$, the objects $s \in S$ such that $p \in s$ can be efficiently reported. The problem may be simplified by forcing the objects to be simplices, forcing them to be disjoint, and/or forcing them to completely cover the entire space (or some defined portion of it).

Given a planar subdivision, *planar point location* consists of finding the unique cell that contains a given query point q (see Figure 2.4). As hinted at in the proof outline of Lemma 2.10, planar point location structures can be helpful in building and manipulating data structures based on shallow cuttings. As such, they are useful in the construction of data structures for solving various range searching problems, as will be seen in Chapter 6. Arge et al. developed an optimal data structure for planar point location in the CO model [29]

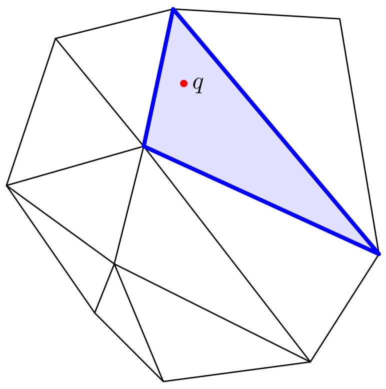


Figure 2.4: Planar point location: identifying the face of a planar subdivision containing the query point q.

which we will use in our data structures.

Chapter 3

Previous Work

3.1 Hilbert Curves

The uses of Hilbert curves are wide and varied, including mathematics [42], image processing [86, 122], image compression [99], bandwidth reduction [102], cryptology [92], algorithms [105], scientific computing [51,76], parallel computing [23,77], geographic information systems [1] and database systems [38, 78, 88].

A large variety of algorithms exist for computing the Hilbert curve [33, 39, 42, 43, 75, 87, 101, 117], each directed towards a particular application. In the context of databases and GIS, Butz's classic algorithm [43] is the most commonly used, whose standard implementation was created by Thomas [117] and later refined by Moore [101]. The algorithms are all equivalent in complexity, requiring a number of operations proportional to the number of bits output in the Hilbert index, and vary only in their details. Butz's algorithm is presented rather cryptically, introducing seven layers of subscripted variables in order to describe the algorithms in terms of fundamental bitwise boolean logic operations. The algorithm proved popular because it uses very little state, and is easily implemented non-recursively for increased performance. Moore [101] later refined Butz's algorithm to remove the dependency on lookup tables. Lawder [87] cleaned up the presentation of these algorithms and formally extended them to higher dimensions. However, all of these presentation obscure the high level operations that are actually occurring: rotations, reflections and Gray code calculations. In Chapter 4 we reproduce Butz's algorithm from a geometric point of view, casting the bit operations in terms of these high level concepts. Bartholdi [33] concentrates specifically on the rotations and translations that occur when descending through the levels of recursion, maintaining orientation information using algebraic transition functions. Their techniques generalize to other families of space-filling curves, but they are meant explicitly for forward and reverse index calculations, and do not efficiently allow enumerations of the curve. Jin and Mellor-Crumley [75] trade space for speed and precalculate all possible inputs and outputs of the state transition function. Their techniques allow for efficient traversal of arbitrary space-filling curves, as well as for efficient forward and reverse index calculations. However, they require separate lookup tables for each possible dimension, and thus do not gracefully handle arbitrary dimensions. The popularity of Moore's approach in the context of databases is due to its seamless handling of arbitrary dimensions. For domains with a fixed number of dimensions, techniques using precomputed lookup tables or state transition functions are more efficient [75].

In the context of range searching, Hilbert curves have found use as a sort order in a variety of spatial data structures [68,78,88,100,101]. Since the Hilbert curve is naturally defined over a bounding hypercube $H(P)$ that is typically much larger than the bounding box $B(P)$ of the input points P, converting the points to Hilbert indices requires an increase in space usage. To combat this, Moore [101] developed Hilbert index comparison routines that compute the Hilbert indices of two points simultaneously bit-by-bit, stopping when the relative order can be determined. This introduces an expected $\mathrm{O}(\log N)$-cost per comparison [63], resulting in a sub-optimal expected $\mathrm{O}\big(N \log^2 N\big)$-time sort algorithm. Another approach to solving this problem is to remove the inefficiency in representation of Hilbert indices. The results presented in Chapter 4 take this approach, by generating compact Hilbert indices for points sets where $B(P)$ is smaller than $H(P)$. Our results are the first to generalize Hilbert curves to 'rectangular' sets $B(P)$, and as a corollary, produce the first optimal Hilbert sorting algorithm for points in $B(P)$. The key insight necessary for this construction was made possible by revisiting Butz's algorithm from a geometric point of view.

3.2 Lower Bounds

In internal memory most range searching problems have been rather fully explored and optimal algorithms found. This includes 2- and 3-d orthogonal range reporting (and its 2- and 3-sided variants in the plane) [59, 98], 3-d dominance reporting [2, 90] and 3-d halfspace range reporting [7]. A notable exception to this general rule is for exact halfspace range counting in the plane where only an $\Omega\big(N^{1-1/n}/\log N\big)$ query lower bound is known for linear space data structures [53], but the best known data structure (believed to be optimal) requires $\mathrm{O}\big(N^{1-1/n}\big)$ query time [93]. In the plane, the lower bound sharpens to the optimal $\Omega\Big(\sqrt{N}\Big)$, but the log factor for higher dimensions is thought to be an artifact of the proof technique.

In the I/O model, Arge et al. [28] showed that $\Theta(N \log_B N / \log_B \log_B N)$ space is sufficient and necessary to obtain a query bound of $O(\log_B N + K/B)$ block transfers for 2-d orthogonal range reporting. This lower bound, when applied to blocks of size N^ε, implies that achieving the optimal query bound cache-obliviously requires $\Omega(N \log N)$ space.

Lower bound proofs for range reporting problems in the I/O model [28, 73, 85, 110] have involved the construction of a hard point set together with a set of many 'sufficiently different' queries of the *same* size. Combined with counting arguments from extremal set theory, this ensures that the point set cannot be represented in linear space while guaranteeing a certain proximity (on disk) of the points reported by each query. Such lower bound results immediately carry over into the cache-oblivious model, but they are unable to be made stronger in the CO model because they inherently discuss only a single query size. Our results are distinct in that they require arguing simultaneously over many different query sizes. This necessitates the use of new techniques as those from extremal set theory no longer apply.

There have been very few lower bound results in the cache-oblivious model, but the results that have been found show that the CO model is inherently less powerful than the I/O model. In [40], Brodal and Fagerberg established a lower bound on the amount of main memory (as a function of B) necessary for optimal cache-oblivious sorting. They showed that sorting in optimal $O(\text{sort}(N))$ time is only possible under the *tall cache assumption*, namely that $M = \Omega(B^{1+\varepsilon})$. Since sorting is such a fundamental technique underlying many more complicated data structures this assumption is often made in the CO literature. The result used an adversarial argument over two block sizes. Such a technique is doomed to failure for range searching lower bounds as we can build data structures that are optimal for any constant number of block sizes. In [36], Bender et al. proved that cache-oblivious searching has to cost a constant factor more than the search bound achieved in the I/O model using B-trees [34]. Their result applies in the limiting sense over many block sizes. Our results are unique in that they are the first to establish a gap between the two models that grows with the input size of the problem, and the first to use an explicit construction that argues simultaneously across many block sizes.

3.3 Upper Bounds

In internal memory, linear-space data structures with the optimal query bound of $\mathrm{O}(\log N + K)$ are known for 3-sided range reporting [98], 3-d dominance reporting [2,90], and 3-d halfspace range reporting [7], as well as for a number of related problems. Using the reductions of Section 2.4.1, the results for halfspace range reporting imply the same results for 2-d parabolic, circular, and 2-d K-nearest neighbour reporting.

Exactly counting the number of points in a query range seems significantly harder than reporting if the query bound is to be independent of the output size K. For 3-sided range counting Chazelle [52] obtained a linear-space data structure with query time $\mathrm{O}(\log N)$, which also immediately implies an $\mathrm{O}(N \log N)$-space data structure with query time $\mathrm{O}(\log^2 N)$ for 3-d dominance counting. For exact 3-d halfspace range counting, Matoušek [95] obtained a linear-space data structure with a query bound of $\mathrm{O}(N^{2/3})$, and this is conjectured to be the best possible. As a result, much effort has been put into obtaining *approximate* halfspace range counting structures with polylogarithmic query bounds.

Aronov and Har-Peled [31] presented a general technique that can be used to construct an approximate range counting structure from one for range emptiness queries; the obtained data structure provides a correct approximation of the number of points in the query range with high probability. The cost of this transformation is an increase of the space bound by a factor of $\log N$ and an increase of the query bound by a factor of $\log N \log \log N$.

For halfspace range searching, linear-space range emptiness structures with $\mathrm{O}(\log N)$ query time have been known for a long time (see, for example, [84]). Thus, the technique by Aronov and Har-Peled provides an $\mathrm{O}(N \log N)$-space approximate halfspace range counting structure with a query time of $\mathrm{O}(\log^2 N \log \log N)$. Kaplan and Sharir [82] improved the query time by a factor of $\log \log N$ using an interesting combinatorial lemma concerning the overlay of lower envelopes in a randomized incremental construction (see also [80,81]). Like Aronov and Har-Peled, they were able to guarantee correctness only with high probability. Later, Aronov and Har-Peled showed in an updated version of their paper [32] that an $\mathrm{O}(\log^2 N)$ query time can be obtained using $\mathrm{O}(N \log N)$ space without applying the overlay lemma. Har-Peled and Sharir [71] showed that a worst-case query time of $\mathrm{O}(\log N \log \log N)$ can be achieved using $\mathrm{O}\left(N \log^{\mathrm{O}(1)} N\right)$ space. This was improved by Afshani and Chan [6], who presented a linear-space data structure with the same worst-case query bound, as well as another linear-space data structure that uses the overlay lemma to achieve the optimal query time of $\mathrm{O}(\log(N/K))$ in the expected case.

The fact that the overlay lemma is a crucial component of Afshani and Chan's optimal data structure has a number of limiting implications: the method does not generalize to other problems, unless a similar overlay lemma is proved for each such problem; a non-trivial modification of the overlay lemma would be required to use it in models such as the I/O model or the cache-oblivious model; and, finally, it cannot be used to obtain a worst-case query bound. The other methods discussed above have similar shortcomings in that they are tailored to internal-memory models or to specific problems. For example, many of the $\log N$-factors in the above complexity bounds are the result of applying Chernoff-type inequalities (completely independent of B) and cannot easily be reduced to $\log_B N$ in the I/O model or the cache-oblivious model. Our range counting results are notable in that they are the first to achieve worst-case optimal queries in linear space (a first even in internal memory), and they generalize to a variety of problems for which no previous approximate range counting results were previously known.

In the I/O model, much work has focused on orthogonal range reporting. A number of linear-space data structures have been proposed that achieve a query bound of $\mathrm{O}\left(\sqrt{N/B} + K/B\right)$ block transfers in 2-d and $\mathrm{O}\big((N/B)^{1-1/n} + K/B\big)$ block transfers in n dimensions [27,66,67,79,106,109]. In [28], Arge et al. showed how to achieve a query bound of $\mathrm{O}(\log_B N + K/B)$ for 3-sided range reporting in the plane, using linear space. They also showed that $\Theta(N \log_B N / \log_B \log_B N)$ space is sufficient and necessary to achieve a query bound of $\mathrm{O}(\log_B N + K/B)$ block transfers for general 2-d orthogonal range reporting. The main tool used to prove the 3-sided upper bound is a linear-space I/O-efficient version of McCreight's priority search tree [98] with a query bound of $\mathrm{O}(\log_B N + K/B)$ for 3-sided queries.

For 3-d dominance reporting, Vengroff and Vitter [119] presented a data structure with a query bound of $\mathrm{O}((\log\log\log_B N)\log(N/B) + K/B)$ block transfers and using $\mathrm{O}(N\log(N/B))$ space in the I/O model. The query bound can be reduced to $\mathrm{O}(\log_B N + K/B)$ by choosing the parameters in the data structure more carefully [120]. A recent data structure by Afshani [2] achieves the optimal query bound of $\mathrm{O}(\log_B N + K/B)$ block transfers using linear space, raising the question whether this result can be achieved in the cache-oblivious model. This data structure also yields a 3-d orthogonal range reporting structure that uses $\mathrm{O}\big(N \log^3 N\big)$ space and achieves the same query bound.

Halfspace range reporting in 3-d has a longer history, in part because it generalizes various other range searching problems, as was discussed in Section 2.4.1. In internal memory, Chan described an $\mathrm{O}(N \log N)$-space data structure with an expected query time of $\mathrm{O}(\log N + K)$ [44]. Building on these ideas, Agarwal et al. [14] obtained an $\mathrm{O}(N \log N)$-space data structure with an expected query bound of $\mathrm{O}(\log_B N + K/B)$ block transfers in the I/O model. Further research led to the development of internal-memory data structures with the optimal query bound in the worst case and using $\mathrm{O}(N \log \log N)$ space [46,108]. The same improvements can be carried over to the I/O model. Recently, Afshani and Chan [7] described a linear-space data structure with the optimal query bound in internal memory and an $\mathrm{O}(N \log^* N)$-space data structure that answers queries using $\mathrm{O}(\log_B N + K/B)$ block transfers in the I/O model.

Much less is known in the cache-oblivious model. Orthogonal range reporting queries in $\mathbb{R}^n$ can be answered using $\mathrm{O}\big((N/B)^{1-1/n} + K/B\big)$ block transfers [13, 26]. Cache-oblivious range reporting structures with a query bound of $\mathrm{O}(\log_B N \log \log N + K/B)$ block transfers and using $\mathrm{O}(N \log N)$ space are easily obtained for 3-d halfspace and 3-d dominance queries using existing techniques [59]. Thus, the interesting questions are whether cache-oblivious data structures for these problems exist that achieve the optimal query bound of $\mathrm{O}(\log_B N + K/B)$ block transfers and how much space is necessary to achieve this bound.

For 3-sided queries, data structures with the optimal query bound and using $\mathrm{O}(N \log N)$ space were proposed in [13, 25, 30]. The data structure by Arge and Zeh [30] was obtained using a standard reduction to 2-d dominance queries, for which the paper presented a linear-space data structure with the optimal query bound. The data structure by Arge et al. [25] can be seen as being based on some notion of shallow cuttings for 3-sided range searching, combined with a specialized 2-d dominance counting structure. For the remaining problems, such as 3-d dominance reporting and 3-d halfspace range reporting as well as their approximate counting versions, no non-trivial results were previously known in the cache-oblivious model. Our range reporting results are distinct in that they unify similar previous approaches, and generalize to a variety of range searching problems for which no equivalent results were previously known.

Query type	Model	Space	Query bound		References
3-d halfspace	RAM	$N \log N$	$\log^2 N \log\log N$	(MC)	Aronov, Har-Peled 2005 [31]
		$N \log N$	$\log^2 N$	(MC)	Aronov, Har-Peled 2005 [31]
					Kaplan 2006 [82]
					Kaplan, Sharir 2007 [80]
					Kaplan, Sharir 2011 [81]
		$N \log^{O(1)} N$	$\log N \log\log N$	(WC)	Aronov, Har-Peled 2005 [31]
		N	$\log N \log\log N$	(WC)	Afshani, Chan 2009 [6]
		N	$\log(N/K)$	(LV)	Afshani, Chan 2009 [6]
3-d halfspace	RAM	N	$\log(N/K)$	(WC)	**new**
	I/O & CO	N	$\log_B(N/K)$	(WC)	**new**
3-d dominance	RAM	N	$\log(N/K)$	(WC)	**new**
	I/O & CO	N	$\log_B(N/K)$	(WC)	**new**

Table 3.1: A comparison of our results on approximate range counting with previous work. In the listing of query bounds, WC refers to worst-case bounds; LV to Las Vegas bounds, that is, query bounds that hold in the expected sense; and MC to Monte Carlo bounds, that is, the answer to a query is correct with high probability. For the sake of clarity, O-notation has been omitted.

Query type	Model	Space	Query bound	References
2-d 3-sided	RAM	N	$\log N + K$	McCreight 1985 [98]
	I/O	N	$\log_B N + K/B$	Arge 1999 [28]
	CO	$N \log N$	$\log_B N + K/B$	Agarwal 2003 [13]
				Arge, Brodal 2005 [25]
				Arge, Zeh 2006 [30]
				Afshani, Hamilton, Zeh 2009 [9]
3-d dominance	RAM	N	$\log N + K$	Makris 1998 [90]
				Afshani 2008 [2]
	I/O	N	$\log_B N + K/B$	Afshani 2008 [2]
	CO	$N \log N$	$\log_B N + K/B$	**new**
3-d halfspace	RAM	N	$\log N + K$	Afshani, Chan 2009 [7]
	I/O	$N \log^* N$	$\log_B N + K/B$	Afshani, Chan 2009 [7]
	CO	$N \log N$	$\log_B N + K/B$	**new**

Table 3.2: A comparison of our results on range reporting with previous work. For the sake of clarity, O-notation has been omitted.

Chapter 4

Compact Hilbert Indices

In the first part of this chapter we recreate Butz's classic algorithm [43] for Hilbert curves, but from a geometric point of view. This intuitive approach allows for a deeper level of understanding of the primitives used in Butz's algorithm, at the same time providing insight into other algorithmic approaches such as Bartholdi and Goldsman's vertex-labelling approach [33] and Jin and Mellor-Crummey's table-driven methods [75]. Consider $H(P) = [0, 2^m] \times \cdots \times [0, 2^m]$, the power-of-two-sized bounding hypercube of P. By considering the order in which the Hilbert curve visits the points in $H(P)$ we may assign an index between 0 and $2^{mn} - 1$ to each point. In other words, positions on the curve are naturally encoded by an mn-bit integer. In the context of database systems this enumeration can be used to sort the points in P while preserving data locality. This in turn translates to data structures with excellent range query performance [78, 100].

In the real world, point sets do not have such ideal distributions and consequently the bounding box $B(P) = [0, 2^{m_1}] \times \cdots \times [0, 2^{m_n}]$ may be significantly smaller than the bounding hypercube $H(P)$. Coordinates along the ith dimension require m_i bits to represent, thus points in $B(P)$ can be represented using $X = \sum_i m_i$ bits. In the second part of this chapter we explore the notion of *compact Hilbert indices*, which assign to points an index requiring exactly X bits to represent. This is done by only considering the portion of the Hilbert curve intersecting $B(P)$. This may be done trivially by calculating the Hilbert index of each point $B(P)$, sorting them based on this index, and then simply assigning indices in 0 through $2^X - 1$. However, such an approach requires an exhaustive enumeration of $B(P)$. The approach we discuss in this chapter can generate such an index for a single point of $B(P)$ in absence of any others.

The algorithms presented in this chapter were implementd in the form of a `C++` library for the computation of both regular and compact Hilbert curves and indices. In the third part of this chapter we evaluate the performance of this library and validate the utility of compact Hilbert indices.

The most important contribution of this chapter is in the development of compact Hilbert indices. The geometric reconstruction of Butz's algorithm is only necessary to gain sufficient insight into the mechanics of the algorithm, allowing us to remove the redundant computation and space requirements of ordinary Hilbert indices. No previous work attempts to generalize space-filling curves to non-hypercube lattices. The results of this chapter were published in [69, 70].

4.1 Hilbert Curves

We begin by building the necessary tools for the exploration of Hilbert curves. We take a geometric approach, yielding algorithms that are identical to those of Moore [101]. Moore's algorithms are refinements based on Thomas's implementation [117] of Butz's classic algorithm [43].

The terminology and notation used in this section is largely my own, and I have chosen to deviate from existing convention in order to highlight the geometric approach taken here, and the relationship to standard boolean operators. The need for the level of detail in this construction will become apparent in the construction of algorithms for compact Hilbert indices in Section 4.2.

We consider first the traditional recursive definition of the two dimensional Hilbert curve. For reasons that will become apparent later, we consider the Hilbert curve that starts in the bottom left corner and finishes in the upper left[1]. The curve is initially defined on a 2×2 lattice, as shown in Figure 4.1. Given the order k curve[2] defined on a $2^k \times 2^k$ lattice, we may refine it to visit all points on a $2^{k+1} \times 2^{k+1}$ lattice as follows:

- Place a copy of the original curve, rotated counter-clockwise by $90°$, in the lower left sub-grid.
- Place a copy of the original curve, rotated clockwise by $90°$, in the upper left sub-grid.
- Place a copy of the original curve in each of the right sub-grids.
- Connect these four disjoint curves in the one manner that uses only unit step sizes.

[1]In the traditional presentation, the two-dimensional Hilbert curve finishes in the bottom right corner.
[2]The *order k curve* is the curve after k levels of recursion.

This construction may be visualized in Figure 4.2, with the first four iterations of the construction shown in Figure 4.3. In a completely analogous manner one may define the Peano curve, which travels through lattices of size $3^k \times 3^k$ as shown in Figure 4.4.

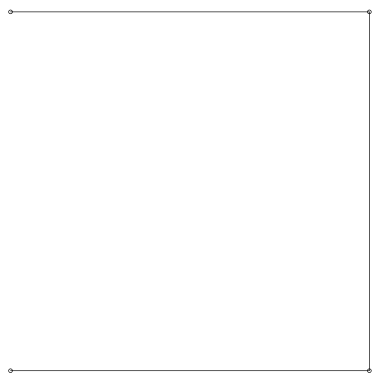


Figure 4.1: Order-1 Hilbert lattice

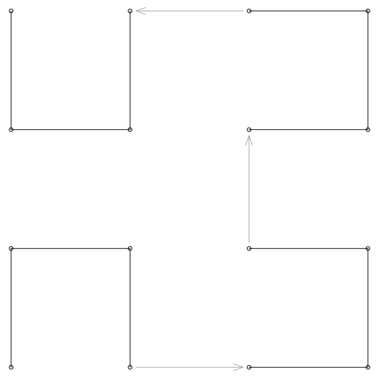


Figure 4.2: Building the order-2 Hilbert lattice

Any finite-order two-dimensional Hilbert curve allows a simple mapping from 2 dimensions into 1, by simply associating a given grid point with its index along the curve. This same concept can be extended to arbitrary space-filling curves, as well as to higher dimensions. It is worth noting the fact that the Hilbert curves always take steps of unit length: immediate neighbors on the curve are also immediate neighbors in the plane. This translates to a notion of data locality: points close to each other in the plane tend to be close to each other along the Hilbert curve.

4.1.1 Higher Dimensions

The geometric approach to the two-dimensional Hilbert curve starts by considering a 2×2 grid of points and describes the path through them. It then recurses by replacing each point with another 2×2 grid (making a $2^2 \times 2^2$ grid) and defining the curve through each of those, appropriately rotated so that the entrance and exit points to these sub-grids remain adjacent. We consider an analogous recursive approach to the multi-dimensional Hilbert curve. Consider a grid of $2 \times \cdots \times 2$ points in n dimensions, corresponding to the corners of the unit hypercube in $\mathbb{Z}^n$. The key property of the Hilbert curve is that successive points are immediate neighbors in the grid. Thus, to maintain this property we are looking for a walk through the 2^n points such that every point will be enumerated, and successive points will be neighboring corners of the hypercube.

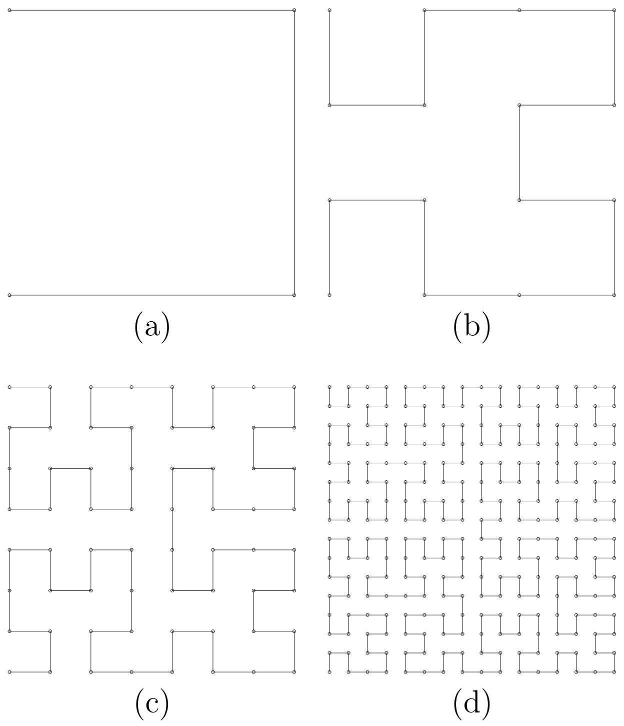


Figure 4.3: First four iterations of the Hilbert curve.

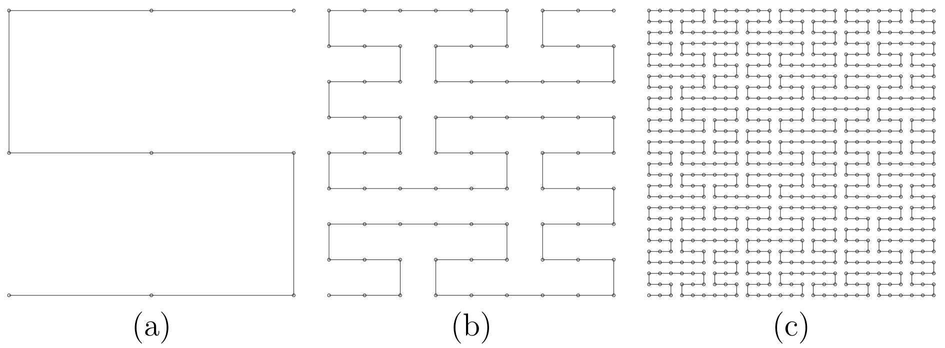


Figure 4.4: First three iterations of the Peano curve.

We let each of the 2^n vertices be labelled by an n-bit string of the form $b = [\beta_{n-1} \cdots \beta_0]_{[2]}$, where $\beta_i \in \mathbb{B} := \{0, 1\}$ represents the position of the vertex along dimension i (0 for low, 1 for high). This is easily interpreted as an n-bit non-negative integer value in $\mathbb{Z}_{2^n}$, or equivalently, $\mathbb{B}^n$. Restricting ourselves to taking steps to immediate neighbors implies that in the binary labels of successive vertices, only one bit may change. In other words, we are looking for an ordering of the 2^n distinct n-bit numbers such that any successive pair of numbers differ in exactly one bit. This corresponds exactly to the classic Gray code [65].

Gray Code

In general, a Gray code is an ordering of numbers such that adjacent numbers differ in exactly a single digit with respect to some base. More specifically, we are concerned with a binary Gray code. Perhaps the simplest form of a binary Gray code is the binary *reflected* Gray code, which is intuitively constructed in the following manner:

1. Start with the Gray code over 1-bit numbers:
$$[[0]_{[2]}, [1]_{[2]}]$$
2. Write the sequence forwards and then backwards, prepending zeroes to the first half and ones to the second half. This creates the Gray code over all 2-bit numbers:
$$[[00]_{[2]}, [01]_{[2]}, [11]_{[2]}, [10]_{[2]}]$$
3. Repeat step 2, each time growing the Gray code over k-bit numbers to one over $k+1$ bits.

Assuming the input to step 2 is itself a Gray code over all k-bit numbers, it is easy to see that the output will be a valid Gray code over all $(k + 1)$-bit numbers. The 2-bit Gray code generated in this manner coincides exactly with the ordering through the four points in a two-dimensional Hilbert curve (it generates the familiar '$\sqsupset$' shape), and it can be used as a basis to extend the concept of the Hilbert curve to higher dimensions[3]. Given this construction of the binary reflected Gray code, we may easily derive a closed form for the ith Gray code integer. The following results on Gray Codes are well known, but we have provided our own proofs for the sake of completeness.

[3]This exact agreement is due to the non-standard orientation we have chosen for the Hilbert curve. Given the standard orientation, the agreement would only be up to a rotation.

Theorem 4.1 (Closed-form Binary Reflected Gray Code)**.** *The binary reflected Gray code sequence is generated by the function*

$$\mathrm{gc}(i) = i \oplus (i \triangleright 1),$$

where $\oplus$ is the exlusive-or operator, and $\triangleright$ is the right-shift operator.

Proof. We consider the value of the jth bit of the ith Gray code, $\mathrm{bit}\,(\mathrm{gc}(i), j)$. The construction begins with the Gray code sequence over $\mathbb{B}$. After j iterations we have the Gray code defined over all $(j+1)$-bit values. In this sequence, the first half of the values are defined such that the jth bit is zero, while the second half has a one for the jth bit. In the next iteration of this construction the pattern reverses itself such that (for $0 \le i < 4(2^j)$):

$$\mathrm{bit}\,(\mathrm{gc}(i), j) = \begin{cases} 0, \text{if } 0 \le i < 2^j, \\ 1, \text{if } 2^j \le i < 2(2^j), \\ 1, \text{if } 2(2^j) \le i < 3(2^j), \\ 0, \text{if } 3(2^j) \le i < 4(2^j) \end{cases} = \begin{cases} 0, & \text{if } \lfloor \frac{i}{2^j} \rfloor = 0, \\ 1, & \text{if } \lfloor \frac{i}{2^j} \rfloor = 1, \\ 1, & \text{if } \lfloor \frac{i}{2^j} \rfloor = 2, \\ 0, & \text{if } \lfloor \frac{i}{2^j} \rfloor = 3. \end{cases}$$

In subsequent iterations of the construction this pattern will simply be repeated as it is already symmetric. Hence, it follows that for *all* $i \ge 0$

$$\mathrm{bit}\,(\mathrm{gc}(i), j) = \begin{cases} 0, & \text{if } \lfloor \frac{i}{2^j} \rfloor \bmod 4 = 0, \\ 1, & \text{if } \lfloor \frac{i}{2^j} \rfloor \bmod 4 = 1, \\ 1, & \text{if } \lfloor \frac{i}{2^j} \rfloor \bmod 4 = 2, \\ 0, & \text{if } \lfloor \frac{i}{2^j} \rfloor \bmod 4 = 3. \end{cases}$$

Since $\lfloor \frac{i}{2^{j+1}} \rfloor \bmod 2 = 1$ if and only if $\lfloor \frac{i}{2^j} \rfloor \bmod 4 \in \{2, 3\}$ we see that

$$\begin{aligned} \mathrm{bit}\,(\mathrm{gc}(i), j) &= \begin{cases} 0, & \text{if } \lfloor \frac{i}{2^j} \rfloor \bmod 4 = 0, \\ 1, & \text{if } \lfloor \frac{i}{2^j} \rfloor \bmod 4 = 1, \\ 0, & \text{if } \lfloor \frac{i}{2^j} \rfloor \bmod 4 = 2, \\ 1, & \text{if } \lfloor \frac{i}{2^j} \rfloor \bmod 4 = 3 \end{cases} + \left\lfloor \frac{i}{2^{j+1}} \right\rfloor \bmod 2 \\ &= \left\lfloor \frac{i}{2^j} \right\rfloor + \left\lfloor \frac{i}{2^{j+1}} \right\rfloor \bmod 2 \\ &= (i \triangleright j) + (i \triangleright (j+1)) \bmod 2 \\ &= \mathrm{bit}\,(i, j) + \mathrm{bit}\,(i, j+1) \bmod 2 \\ &= \mathrm{bit}\,(i, j) \oplus \mathrm{bit}\,(i \triangleright 1, j) \\ &= \mathrm{bit}\,(i \oplus (i \triangleright 1), j)\,. \end{aligned}$$

Thus it follows that $\mathrm{gc}(i) = i \oplus (i \triangleright 1)$. □

Given a non-negative integer we may wish to find at which position it lies in the Gray code sequence. In other words, we may wish to determine the inverse of the Gray code.

Theorem 4.2 (Binary Reflected Gray Code Inverse)**.** *Consider a non-negative integer i. Let m be the precision of i. That is, let $m = \lceil \log_2(i+1) \rceil$ such that i requires m bits in its binary representation. Then it follows that*

$$\mathrm{bit}\,(i,j) = \sum_{k=j}^{m-1} \mathrm{bit}\,(\mathrm{gc}(i),k) \mod 2,$$

where $\mathrm{bit}\,(i,k)$ refers to the kth bit in the binary representation of i, with 0 being the least significant bit and $m-1$ the most significant.

Proof. By Theorem 4.1 we have

$$\mathrm{bit}\,(\mathrm{gc}(i),j) = \mathrm{bit}\,(i,j) + \mathrm{bit}\,(i,j+1) \mod 2.$$

Summing over $j \le k < m$ we find that

$$\begin{aligned}
\sum_{k=j}^{m-1} \mathrm{bit}\,(\mathrm{gc}(i),k) &= \sum_{k=j}^{m-1} \big(\mathrm{bit}\,(i,k) + \mathrm{bit}\,(i,k+1)\big) \mod 2 \\
&= \left(\sum_{k=j}^{m-1} \mathrm{bit}\,(i,k) + \sum_{k=j+1}^{m} \mathrm{bit}\,(i,k) \right) \mod 2 \\
&= \mathrm{bit}\,(i,j) + \left(2 \sum_{k=j+1}^{m-1} \mathrm{bit}\,(\mathrm{gc}(i),k) \right) + \mathrm{bit}\,(i,m) \mod 2 \\
&= \mathrm{bit}\,(i,j) + \mathrm{bit}\,(i,m) \mod 2.
\end{aligned}$$

By the definition of m we see that $\mathrm{bit}\,(i,m) = 0$ and the result follows. □

We use Theorem 4.2 to construct Algorithm 1, which computes the inverse as desired.

Additionally, we are interested in knowing along which bit the Gray code will change when preceding from one term to the next. Equivalently, we are interested in knowing along which dimension we will step when proceeding from one vertex to another on the Hilbert curve. To this end, we define $\mathrm{g}(i)$ as

$$\mathrm{g}(i) = k, \text{ such that } \mathrm{gc}(i) \oplus \mathrm{gc}(i+1) = 2^k, \quad 0 \le i < 2^n - 1.$$

Algorithm 1 GRAYCODEINVERSE(g)

Given a non-negative integer g, calculates the non-negative integer i such that $\mathrm{gc}(i) = g$.

Input: A non-negative integer g.

Output: The non-negative integer i such that $\mathrm{gc}(i) = g$.

```
1: $m \leftarrow$ number of bits required to represent $g$
2: $(i, j) \leftarrow (g, 1)$
3: while $j < m$ do
4:     $i \leftarrow i \oplus (g \triangleright j)$
5:     $j \leftarrow j + 1$
6: end while
```

Lemma 4.3 (Dimension of Change in the Gray Code)**.** *The sequence* $\mathrm{g}(i)$ *is given by*

$$\mathrm{g}(i) = \mathrm{tsb}(i),$$

where tsb *is the number of* trailing set bits *in the binary representation of* i*.*

Proof. We examine the difference between two consecutive values of the Gray code:

$$\begin{aligned}\mathrm{gc}(i) \oplus \mathrm{gc}(i+1) &= i \oplus (i \triangleright 1) \oplus (i+1) \oplus ((i+1) \triangleright 1) \\ &= (i \oplus (i+1)) \oplus ((i \triangleright 1) \oplus ((i+1) \triangleright 1)) \\ &= (i \oplus (i+1)) \oplus ((i \oplus (i+1)) \triangleright 1).\end{aligned}$$

We consider first the portion $i \oplus (i+1)$. Adding 1 to i will cause a carry past the first digit if the first digit is 1. Similarly past the second digit and so on. Letting k be the number of trailing ones in the binary representation of i (or, alternatively, the index of the first zero valued bit), it follows that $i+1$ will have a one at position k, zeroes at positions 0 through $k-1$, and be identical to i elsewhere. Thus, taking the exclusive-or of these two will result in a number with $k+1$ trailing one bits. Similarly, the result of $(i \oplus (i+1)) \triangleright 1$ will be a number with k trailing one bits. Taking the exclusive-or of these two results in a number with a single non-zero bit at the kth position. Hence, between the ith and $(i+1)$th Gray code integers it is the kth bit that changes. This corresponds exactly to the definition of 'tsb' thus it follows that $\mathrm{g}(i) = \mathrm{tsb}(i)$. □

Lemma 4.4 (Symmetry of the Gray Code)**.** *Given* $n \in \mathbb{N}$ *and* $0 \leq i < 2^n$*, it follows that* $\mathrm{gc}(2^n - 1 - i) = \mathrm{gc}(i) \oplus 2^{n-1}$.

Proof. This property follows immediately from the construction algorithm for the reflected binary Gray code. The second 2^{n-1} values are simply equal to the first 2^{n-1} values in reverse, with the $(n-1)$th bit set as a 1. Thus we see that

$$\mathrm{gc}(2^n - 1 - i) = \mathrm{gc}(i) \vee 2^{n-1}, \text{ for } 0 \leq i < 2^{n-1}.$$

Replacing the 'or' operation by an 'exclusive-or' (justified in this case as exaclty one of $\mathrm{gc}(2^n - 1 - i)$ or $\mathrm{gc}(i)$ will have a zero in the $(n-1)$th bit) leads to the desired result. □

Corollary 4.5 (Symmetry of $\mathrm{g}(i)$)**.** *The sequence* $\mathrm{g}(i)$ *is symmetric such that* $\mathrm{g}(i) = \mathrm{g}(2^n - 2 - i)$ *for* $0 \leq i \leq 2^n - 2$.

Proof. Without loss of generality we consider $i \leq \frac{2^n-2}{2}$. Lemma 4.4 tells us that

$$\mathrm{gc}(2^n - 2 - i) = \mathrm{gc}(i+1) \oplus 2^{n-1}.$$

By the definition of gc we know that $\mathrm{gc}(i+1) = \mathrm{gc}(i) \oplus 2^{\mathrm{g}(i)}$ and $\mathrm{gc}(2^n - 2 - i) = \mathrm{gc}(2^n - 1 - i) \oplus 2^{\mathrm{g}(2^n-2-i)}$. Substituting these into the above equation yields

$$\mathrm{gc}(2^n - 1 - i) \oplus 2^{\mathrm{g}(2^n-2-i)} = \mathrm{gc}(i) \oplus 2^{\mathrm{g}(i)} \oplus 2^{n-1}.$$

By Lemma 4.4 this simplifies to the desired result,

$$\mathrm{g}(2^n - 2 - i) = \mathrm{g}(i).$$

□

Analogous to the Hilbert curve in two dimensions, the Gray code ordering can be used to give an ordering through the vertices of a unit hypercube in $\mathbb{Z}^n$. As in the recursive construction in two dimensions, we will recursively define the Hilbert curve by zooming in on each point in the sequence (each sub-hypercube) and iterating through the points within using a transformed/rotated version of the original curve. Like the two-dimensional case, we must determine orientations for the Hilbert curve through each of the 2^n sub-hypercubes. These orientations must be consistent in that the exit point of the curve through one sub-hypercube must be immediately adjacent to the entry point of the next sub-hypercube. Additionally, the entry and exit points of the parent hypercube must coincide with the entry point of the first sub-hypercube and the exit point of the last sub-hypercube, respectively. These constraints on entry and exit points are visualized for the two-dimensional case in Figure 4.5.

Entry Points

Using the same labelling as the vertices of the parent hypercube, we let $\mathrm{e}(i)$ and $\mathrm{f}(i)$ refer, respectively, to the entry and exit vertices of the ith sub-hypercube in a Gray code ordering of the sub-hypercubes. Since the ith and $(i+1)$th sub-hypercubes are neighbors along the $\mathrm{g}(i)$th coordinate, we must have that $\mathrm{f}(i) \oplus 2^{\mathrm{g}(i)} = \mathrm{e}(i+1)$. Like the entry and exit points of the parent hypercube, entry and exit points of a given sub-hypercube must be neighboring corners. That is, $\mathrm{e}(i)$ and $\mathrm{f}(i)$ may only differ in exactly one bit position, meaning we must have that $\mathrm{e}(i) \oplus \mathrm{f}(i) = 2^{\mathrm{d}(i)}$ for some $\mathrm{d}(i) \in \mathbb{Z}_n$. We refer to $\mathrm{d}(i)$ as the *intra* sub-hypercube direction, and $\mathrm{g}(i)$ as the *inter* sub-hypercube direction. Combining these two results shows that entry points must satisfy the relation

$$\mathrm{e}(i+1) = \mathrm{e}(i) \oplus 2^{\mathrm{d}(i)} \oplus 2^{\mathrm{g}(i)}, \quad 0 \leq i < 2^n - 1. \tag{4.6}$$

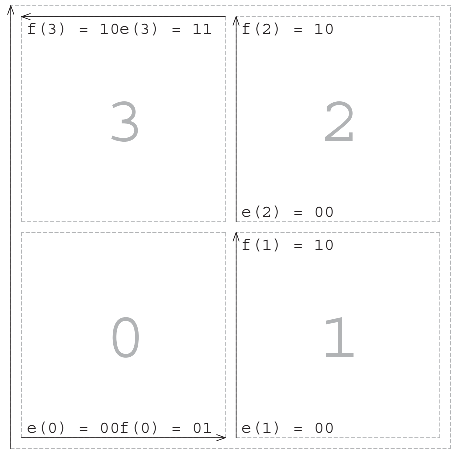
Additionally, as mentioned earlier, we must have that $\mathrm{e}(0)$ is the same as the entry point of the parent hypercube and $f(2^n - 1)$ is the same as the exit point of the parent hypercube. These constraints are displayed graphically for the two-dimensional case in Figure 4.5.

In order to fully determine closed forms for $\mathrm{e}(i)$, $\mathrm{d}(i)$ and $\mathrm{f}(i)$, we first explore various properties of these sequences. The process is guided by exploring a small two-dimensional example, as shown in Figure 4.5. From the symmetry of the Gray Code itself (an immediate consequence of its definition), we can infer similar symmetry relationships on the entry and exit corner sequences. This in turn allows us to infer the symmetry of the intra sub-hypercube direction sequence $\mathrm{d}(i)$. Aided by our small example, we can then posit a closed form that satisfies the symmetry constraint and the observed example values. The form of the hypothesis is then confirmed by verifying that it satisfies Equation 4.6.

Lemma 4.7 (Symmetry of $\mathrm{e}(i)$ and $\mathrm{f}(i)$)**.** *The sequences* $\mathrm{e}(i)$ *and* $\mathrm{f}(i)$ *are symmetric such that* $\mathrm{e}(i) = \mathrm{f}(2^n - 1 - i) \oplus 2^{n-1}$.

Proof. We consider walking through the Hilbert curve backwards, such that $\bar{e}_i = \mathrm{f}(2^n - 1 - i)$ and the ith sub-hypercube is $\overline{\mathrm{gc}}(i) = \mathrm{gc}(2^n - 1 - i)$. By Lemma 4.4 this is equivalent to $\overline{\mathrm{gc}}(i) = \mathrm{gc}(i) \oplus 2^{n-1}$. Thus, it follows that $\mathrm{f}(2^n - 1 - i) = \bar{e}_i = \mathrm{e}(i) \oplus 2^{n-1}$. □

Corollary 4.8 (Symmetry of $\mathrm{d}(i)$)**.** *The sequence* $\mathrm{d}(i)$ *is symmetric such that* $\mathrm{d}(i) = \mathrm{d}(2^n - 1 - i)$ *for* $0 \leq i \leq 2^n - 1$.



i	e(i)	f(i)	d(i)	g(i)
0	$[00]_{[2]}$	$[01]_{[2]}$	0	0
1	$[00]_{[2]}$	$[10]_{[2]}$	1	1
2	$[00]_{[2]}$	$[10]_{[2]}$	1	0
3	$[11]_{[2]}$	$[10]_{[2]}$	0	–

Figure 4.5: Entry and exit points of the 2 dimensional Hilbert curve (the x-axis corresponds to the least significant bit and the y-axis the most significant).

Proof. Lemma 4.7 tells us that $\mathrm{e}(i) = \mathrm{f}(2^n - 1 - i) \oplus 2^{n-1}$ and equivalently $\mathrm{e}(2^n - 1 - i) = \mathrm{f}(i) \oplus 2^{n-1}$. Combining these two yields

$$\mathrm{e}(i) \oplus \mathrm{f}(i) = \mathrm{e}(2^n - 1 - i) \oplus \mathrm{f}(2^n - 1 - i).$$

By the definition of $\mathrm{d}(i)$ we have that $\mathrm{e}(i) \oplus \mathrm{d}(i) = \mathrm{f}(i)$ thus we see

$$\mathrm{d}(i) = \mathrm{d}(2^n - 1 - i).$$

□

Lemma 4.9. *Suppose that*

$$\mathrm{d}(i) = \begin{cases} 0, & i = 0; \\ \mathrm{g}(i-1) \bmod n, & i = 0 \bmod 2; \\ \mathrm{g}(i) \bmod n, & i = 1 \bmod 2, \end{cases}$$

for $0 \leq i \leq 2^n - 1$. *Then* $\mathrm{d}(i)$ *is symmetric as per Corollary 4.8.*

Proof. Suppose $i = 0$. Then $\mathrm{d}(0) = 0$. Similarly, $\mathrm{d}(2^n - 1) = \mathrm{g}(2^n - 1) = \mathrm{tsb}\, 2^n - 1 = n \bmod n = 0$. Suppose $i = 0 \bmod 2$. Then $\mathrm{d}(i) = \mathrm{g}(i-1)$. Since $2^n - 1 - i = 1 \bmod 2$, we see that $\mathrm{d}(2^n - 1 - i) = \mathrm{g}(2^n - 1 - i) = \mathrm{g}(2^n - 2 - (i-1)) = \mathrm{g}(i-1)$. Suppose $i = 1 \bmod 2$. Then $\mathrm{d}(i) = \mathrm{g}(i)$. Since $2^n - 1 - i = 0 \bmod 2$, we see that $\mathrm{d}(2^n - 1 - i) = \mathrm{g}(2^n - 2 - i) = \mathrm{g}(i)$. Thus, this form for $\mathrm{d}(i)$ meets the symmetry requirement of Corollary 4.8. □

Theorem 4.10 (Intra Sub-hypercube Directions)**.** *The formula of Lemma 4.9 satisfies Equation 4.6, and hence defines the sequence of intra sub-hypercube directions,* $\mathrm{d}(i)$*.*

Proof. Let $\mathrm{d}(i, n)$ be the sequence of intra sub-hypercube directions for a fixed dimension n. By inspection (see Figure 4.5) we see that the above definition holds for the case $n = 2$. Suppose that the definition holds for $1, \ldots, n$ and consider the case $n+1$. As long as $\mathrm{g}(i) < n$, then $\mathrm{g}(i) \bmod n + 1 = \mathrm{g}(i) \bmod n$. Thus, we consider the first i such that $\mathrm{g}(i) \geq n$. By Lemma 4.3 we see that this occurs when $i = 2^n - 1$, the smallest positive integer with n trailing set bits. Hence, for $i < 2^n - 1$ we must have that $\mathrm{d}(i, n+1) = \mathrm{d}(i, n)$. Now consider $\mathrm{d}(2^n - 1, n + 1)$. Since the exit point of the ith cell must touch the face of the $(i + 1)$th cell along the $\mathrm{g}(i)$th axis, we must have that

$$\mathrm{bit}\left(\mathrm{f}(i), \mathrm{g}(i)\right) = \mathrm{bit}\left(\mathrm{gc}(i + 1), \mathrm{g}(i)\right).$$

Substituting Equation 4.6 into this we must have that

$$\mathrm{bit}\left(\bigoplus_{j=0}^{2^n-1} 2^{\mathrm{d}(j,n+1)} \oplus \bigoplus_{j=0}^{2^n-2} 2^{\mathrm{g}(j)}, \mathrm{g}(2^n - 1)\right) = \mathrm{bit}\left(\bigoplus_{j=0}^{2^n-1} 2^{\mathrm{g}(j)}, \mathrm{g}(2^n - 1)\right),$$

which simplifies to

$$\begin{aligned}
\mathrm{bit}\left(\bigoplus_{j=0}^{2^n-1} 2^{\mathrm{d}(j,n+1)}, \mathrm{g}(2^n - 1)\right) &= 1,\\
\mathrm{bit}\left(\bigoplus_{j=0}^{2^n-2} 2^{\mathrm{d}(j,n+1)} \oplus 2^{\mathrm{d}(2^n-1,n+1)}, \mathrm{g}(2^n - 1)\right) &= 1,\\
\mathrm{bit}\left(\bigoplus_{j=0}^{2^n-2} 2^{\mathrm{d}(j,n)} \oplus 2^{\mathrm{d}(2^n-1,n+1)}, \mathrm{g}(2^n - 1)\right) &= 1.
\end{aligned}$$

By the symmetry of $\mathrm{d}(i, n)$ most of the first term cancels, leaving

$$\mathrm{bit}\left(2^{\mathrm{d}(0,n)} \oplus 2^{\mathrm{d}(2^n-1,n+1)}, \mathrm{g}(2^n - 1)\right) = 1.$$

Since $\mathrm{d}(0, n) = 0$ then we must have that $\mathrm{d}(2^n - 1, n + 1) = \mathrm{g}(2^n - 1)$. We know that $\mathrm{d}(i, n+1)$ holds for $0 \leq 0 \leq 2^n - 1$, and by Lemma 4.9 we know that this holds for the other half, $2^n \leq i \leq 2^{n+1} - 1$. Hence that definition holds for the case $n + 1$ and by the inductive hypothesis it holds for all $n \geq 2$. □

With a closed form for $\mathrm{d}(i)$ we are able to determine the closed form for $\mathrm{e}(i)$.

Theorem 4.11 (Entry Points)**.** *The sequence of entry points is defined by*

$$\mathrm{e}(i) = \begin{cases} 0, & i = 0, \\ \mathrm{gc}(2\lfloor \frac{i-1}{2} \rfloor), & 0 < i \leq 2^n - 1. \end{cases}$$

Proof. By recursive application of Equation 4.6 we have that

$$\mathrm{e}(i) = \bigoplus_{j=0}^{i-1} 2^{\mathrm{d}(j)} \oplus \bigoplus_{j=0}^{i-1} 2^{\mathrm{g}(j)}.$$

By definition, for all n we have that $e(0) = 0$, thus we consider only the case $i > 0$. Simplifying the above yields

$$\begin{aligned} \mathrm{e}(i) &= 2^{\mathrm{g}(0)} \oplus \bigoplus_{j=1}^{i-1} 2^{\mathrm{d}(j)} \oplus \mathrm{gc}(i) \\ &= 2^{\mathrm{g}(0)} \oplus \underbrace{2^{\mathrm{d}(0)} \oplus 2^{\mathrm{d}(1)}} \oplus \underbrace{2^{\mathrm{d}(2)} \oplus 2^{\mathrm{d}(3)}} \oplus \ldots \oplus 2^{\mathrm{d}(i-1)} \oplus \mathrm{gc}(i). \end{aligned}$$

Suppose $i = 0 \bmod 2$. Then by Theorem 4.10 all of the $\mathrm{d}(i)$ cancel out except $\mathrm{d}(i-1)$, leaving us with $\mathrm{e}(i) = 2^{\mathrm{g}(0)} \oplus 2^{\mathrm{d}(i-1)} \oplus \mathrm{gc}(i)$. Since $\mathrm{g}(0) = \mathrm{tsb}(0) = 0 = \mathrm{tsb}(i) = \mathrm{g}(i)$ this yields $\mathrm{e}(i) = \mathrm{gc}(i) \oplus 2^{\mathrm{d}(i-1)} \oplus 2^{\mathrm{g}(i)} = \mathrm{gc}(i) \oplus 2^{\mathrm{g}(i-1)} \oplus 2^{\mathrm{g}(i)} = \mathrm{gc}(i-2)$. Thus, $\mathrm{e}(i) = \mathrm{gc}(2\lfloor \frac{i-1}{2} \rfloor)$.

Suppose now that $i = 1 \bmod 2$. All of the $\mathrm{d}(i)$ cancel, leaving $\mathrm{e}(i) = \mathrm{gc}(i) \oplus 2^{\mathrm{g}(0)}$. Since $\mathrm{g}(0) = \mathrm{tsb}(0) = 0 = \mathrm{tsb}\, i - 1 = \mathrm{g}(i-1)$ this simplifies to $\mathrm{e}(i) = \mathrm{gc}(i-1)$. For $i = 1 \bmod 2$ we have that $i - 1 = 2\lfloor \frac{i-1}{2} \rfloor$, hence $\mathrm{e}(i) = \mathrm{gc}(2\lfloor \frac{i-1}{2} \rfloor)$. □

Rotations and Reflections

As noted in Section 4.1.1 the recursive construction of the Hilbert curve requires us to construct a curve through the corners of a hypercube when provided with a particular entry and exit point. The classic Gray code explored earlier starts at $\mathrm{gc}(0) = 0$ and ends at $\mathrm{gc}(2^n - 1) = 2^{n-1}$, thus implicitly has an entry point $e = 0$, an internal direction $d = n - 1$ and an exit point $f = 2^{n-1}$. We wish to define a geometric transformation such that the Gray code ordering of sub-hypercubes in the Hilbert curve defined by e and d will map to the standard binary reflected Gray code.

To this end, let us define the *right bit rotation* operator $\circlearrowright$ as

$$b \circlearrowright i = \left[b_{(n-1+i \bmod n)} \cdots b_{(i \bmod n)} \right]_{[2]}, \text{ where } b = [b_{n-1} \cdots b_0]_{[2]}.$$

Conceptually, this function rotates the n bits of b to the right by i places. Analogously, we define the *left bit rotation* operator, $\circlearrowleft$. Trivially, both the left and right bit rotation operators are bijective over $\mathbb{Z}_2^n$ (or equivalently $\mathbb{B}^n$) for any given i. Given e and d, we may now define a transformation T as

$$T_{(e,d)}(b) = (b \oplus e) \circlearrowright (d+1).$$

Being the composition of two bijective operators, we see that the mapping is itself bijective for fixed e and d. We first explore the behaviour of the mapping on the entry and exit points.

Lemma 4.12 (Transformed Entry and Exit Points)**.** *The transform $T_{(e,d)}$ maps e and f to the first and last terms, respectively, of the binary reflected Gray code sequence over $\mathbb{B}^n$. That is,*

$$T_{(e,d)}(e) = 0, \text{ and } T_{(e,d)}(f) = 2^{n-1}.$$

Proof. Straightforward:

$$\begin{aligned}
T_{(e,d)}(e) &= (e \oplus e) \circlearrowright (d+1) = 0 \circlearrowright d+1 = 0; \text{ and,} \\
T_{(e,d)}(f) &= (f \oplus e) \circlearrowright (d+1) \\
&= (e \oplus 2^d \oplus e) \circlearrowright (d+1) \\
&= 2^d \circlearrowright (d+1) \\
&= \Big[\underbrace{0\cdots0}_{n-d-1}\, 1\, \underbrace{0\cdots0}_{d}\Big]_{[2]} \circlearrowright (d+1) \\
&= \Big[1\, \underbrace{0\cdots0}_{n-1}\Big]_{[2]} = 2^{n-1}.
\end{aligned}$$

□

Given the nature of bit-rotation and the exclusive-or operator, it is also easy to see that if neighboring elements of a sequence differ in only one bit position, then the same will hold true for the two transformed points. Hence, they will be neighbors as well. This and the fact that the mapping is bijective tells us that T preserves this critical property of a Gray code sequence. It is easy to show that the inverse of a T-transform is itself a T-transform.

Lemma 4.13 (Inverse Transform)**.** *The inverse of the transform $T_{(e,d)}$ is itself a T-transform, given by*

$$T_{(e,d)}^{-1} = T_{(e \circlearrowright (d+1), n-d-1)}.$$

Proof. It is easy to see that $(T_{(e,d)}(a) \circlearrowleft (d+1)) \oplus e = a$, simply by reversing the individual operations of $T_{(e,d)}$. Letting $b = T_{(e,d)}(a)$, this simplifies to

$$\begin{aligned}
(b \circlearrowleft (d+1)) \oplus e &= (b \circlearrowright (n-d-1)) \oplus e \\
&= \big(b \circlearrowright (n-d-1)\big) \oplus \big(e \circlearrowleft (n-d-1) \circlearrowright (n-d-1)\big) \\
&= \big(b \oplus (e \circlearrowleft (n-d-1))\big) \circlearrowright (n-d-1) \\
&= \big(b \oplus (e \circlearrowright (d+1)\big) \circlearrowright (n-d-1).
\end{aligned}$$

□

We are now ready to construct the Hilbert curve starting at e with direction d. We define $\mathrm{gc}_{(e,d)}(i) = T^{-1}_{(e,d)}(\mathrm{gc}(i))$. By our earlier discussion it follows that the sequence generated by $\mathrm{gc}_{(e,d)}$ is a Gray code sequence. Furthermore, by Lemmas 4.12 and 4.13 it follows that this Gray code sequence begins and ends on the desired points, and the mapping $T_{(e,d)}$ maps it back to the standard binary reflective Gray code. We now have the tools necessary to consistently construct Hilbert curves through hypercubes with arbitrarily defined entry points and directions. We finish this section with one last result on composed transforms which will be necessary later to deal with the recursive nature of the Hilbert curve.

Lemma 4.14 (Composed Transforms)**.** *Consider the composed transform*

$$b = T_{(e_2,d_2)}\big(T_{(e_1,d_1)}(a)\big).$$

Then it follows that

$$b = T_{(e,d)}(a)$$

where $e = e_1 \oplus (e_2 \circlearrowleft (d_1+1))$ *and* $d = d_1 + d_2 + 1$.

Proof. Straightforward:

$$\begin{aligned}
T_{(e_2,d_2)}\big(T_{(e_1,d_1)}(a)\big) &= T_{(e_2,d_2)}\big((a \oplus e_1) \circlearrowright (d_1+1)\big) \\
&= T_{(e_2,d_2)}\Big(\big(a \circlearrowright (d_1+1)\big) \oplus \big(e_1 \circlearrowright (d_1+1)\big)\Big) \\
&= \Big(\big(a \circlearrowright (d_1+1)\big) \oplus \big(e_1 \circlearrowright (d_1+1)\big) \oplus e_2\Big) \circlearrowright (d_2+1) \\
&= \Big(a \oplus \underbrace{e_1 \oplus \big(e_2 \circlearrowleft (d_1+1)\big)}_{e}\Big) \circlearrowright (\underbrace{d_1 + d_2 + 1}_{d} + 1).
\end{aligned}$$

□

It is important to note that T-transforms do in fact have the desired geometric interpretation when applied to our binary labels of the vertices of the unit hypercube. The bit rotation operator can be interpreted as a rotation operator in $\mathbb{Z}^n$, while the exclusive-or operation can be interpreted as a mirroring operation, inverting the axes i where $\mathrm{bit}\,(e, i) = 1$. Hence, the T-transform may be interpreted as a rotation and reflection operator over the space $\mathbb{Z}^n$.

Algorithms

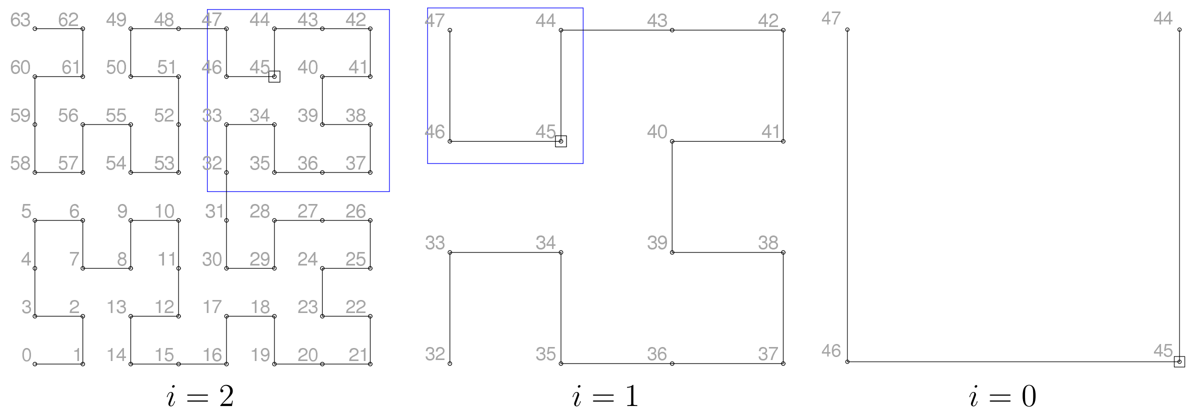
We consider a space of n-dimensional vectors where each component is an integer of precision m; that is, where each component may be represented using m bits. Given the Hilbert curve through this space $\mathbb{B}^n_m$, we wish to determine the Hilbert index, h, of a given point $\mathbf{p} = [p_0, \ldots, p_{n-1}], p_i \in \mathbb{B}^m$.

The result may be found in a series of m projections and Gray code calculations. Given $\mathbf{p}$, we may extract an n-bit number

$$l_{m-1} = [\mathrm{bit}\,(p_{n-1}, m-1) \cdots \mathrm{bit}\,(p_0, m-1)]_{[2]} \,.$$

Each bit of l tells us whether the point $\mathbf{p}$ is in the lower or upper half set of points with respect to a given axis. Thus, the point l_{m-1} locates in which sub-hypercube the point $\mathbf{p}$ may be found. Equivalently, it tells us the vertex of the Hilbert curve through the vertices of the unit hypercube to which p belongs. We wish to determine the Hilbert index of the sub-hypercube containing $\mathbf{p}$, given e and d. As discussed in Section 4.1.1, we do this in two steps: (1) rotate and reflect the space such that the Gray code ordering corresponds to the binary reflected Gray code, $\bar{l}_{m-1} = T_{(e,d)}(l_{m-1})$; and, (2) determine the index of the associated sub-hypercube, $w_{m-1} = \mathrm{gc}^{-1}(\bar{l}_{m-1})$.

We may now calculate $\mathrm{e}(w_{m-1})$ and $\mathrm{d}(w_{m-1})$ in order to determine the entry point and direction through the sub-hypercube containing the point $\mathbf{p}$. The values $\mathrm{e}(w_{m-1})$ and $\mathrm{d}(w_{m-1})$ are relative to the transformed space, thus we may compose this transformation with the existing transformation using Lemma 4.14, calculating $e = e \oplus (\mathrm{e}(w_{m-1}) \circlearrowleft (d+1))$ and $d = d + \mathrm{d}(w_{m-1}) + 1$. At this point, the parameters e and d describe the rotation and reflection necessary to map the sub-hypercube containing $\mathbf{p}$ back to the standard orientation. We then narrow our focus on the sub-hypercube containing $\mathbf{p}$. We repeat the above steps to calculate w_{m-2} and update e and d appropriately. We continue for w_{m-3} through w_0 and



$$\mathbf{p} = [5,6] = [[101]_{[2]}, [110]_{[2]}]$$

i	l	$T_{(e,d)}(l)$	w	$\mathrm{e}(w)$	$\mathrm{d}(w)$	e	d	h
-	-	-	-	-	-	0	1	0
2	$[11]_{[2]} = 3$	3	2	0	1	0	1	2
1	$[10]_{[2]} = 2$	2	3	3	0	3	0	11
0	$[01]_{[2]} = 1$	1	1	0	1	3	0	45

Figure 4.6: Running algorithm HilbertIndex with $n = 2$, $m = 3$ and $\mathbf{p} = [5, 6]$.

finally calculate the full Hilbert index as

$$h = [w_{m-1}w_{m-2}\cdots w_0]_{[2]} = \sum_{i=0}^{m-1} 2^{ni}w_i = \bigvee_{i=0}^{m-1}(w_i \lhd ni).$$

We formalize this approach in Algorithm 2. Inverting the HilbertIndex algorithm is straightforward, with the inverse given by Algorithm 3[4].

We consider an example in two dimensions where $m = 3$. Let $\mathbf{p}$ be the point at $x = 5$ and $y = 6$. Figure 4.6 displays both graphically and in tabular form the results of running Algorithm 2 on the point $\mathbf{p}$.

[4]In these algorithms we have chosen to initialize the direction d as 0, instead of $n - 1$. This corresponds to the standard orientation where the Hilbert curve starts and finishes at opposite ends of the x-axis, the orientation we had originally shunned in Section 4.1.

Algorithm 2 HilbertIndex$(n, m, \mathbf{p})$

Calculates the Hilbert index $h \in \mathbb{B}^{mn}$ of a point $\mathbf{p} \in B(P)$.

Input: $n, m \in \mathbb{Z}_+$ and a point $\mathbf{p} \in B(P)$.

Output: $h \in \mathbb{B}^X$, the Hilbert index of the point $\mathbf{p}$.

```
1: $(h, e, d) \leftarrow (0, 0, 0)$
2: for $i = m - 1$ to $0$ do
3:     $l \leftarrow [\mathrm{bit}\,(p_{n-1}, i) \cdots \mathrm{bit}\,(p_0, i)]_{[2]}$
4:     $l \leftarrow T_{(e,d)}(l)$
5:     $w = \mathrm{gc}^{-1}(l)$
6:     $e \leftarrow e \oplus (\mathrm{e}(w) \circlearrowleft (d + 1))$
7:     $d \leftarrow d + \mathrm{d}(w) + 1 \mod n$
8:     $h \leftarrow (h \lhd n) \vee w$
9: end for
```

Algorithm 3 HilbertIndexInverse(n, m, h)

Calculates the point $\mathbf{p} \in B(P)$ corresponding to a given Hilbert index $h \in \mathbb{B}^{mn}$.

Input: $n, m \in \mathbb{Z}_+$ and $h \in \mathbb{B}^{mn}$, the Hilbert index of the point $\mathbf{p}$.

Output: A point $\mathbf{p} \in B(P)$.

```
1: $(e, d) \leftarrow (0, 0)$
2: $\mathbf{p} = [p_0, \ldots, p_{n-1}] \leftarrow [0, \ldots, 0]$
3: for $i = m - 1$ to $0$ do
4:     $w \leftarrow [\mathrm{bit}\,(h, in + n - 1) \cdots \mathrm{bit}\,(h, in + 0)]_{[2]}$
5:     $l = \mathrm{gc}(w)$
6:     $l \leftarrow T^{-1}_{(e,d)}(l)$
7:     for $j = 0$ to $n - 1$ do
8:         $\mathrm{bit}\,(p_j, i) \leftarrow \mathrm{bit}\,(l, j)$
9:     end for
10:    $e \leftarrow e \oplus (\mathrm{e}(w) \circlearrowleft (d + 1))$
11:    $d \leftarrow d + \mathrm{d}(w) + 1 \mod n$
12: end for
```

4.2 Compact Hilbert Indices

Consider an n-dimensional data-set consisting of points $\mathbf{p} \in \mathbb{B}^{m_0} \times \cdots \times \mathbb{B}^{m_{n-1}} = B(P)$, where $m_i \in Z_+$ is the precision of the data in the ith dimension. Storing a point of data requires $X = \sum_i m_i$ bits. However, a Hilbert index must be calculated with respect to a hypercube $H(P)$ of precision $m = \max_i\{m_i\}$, and requires $mn \geq X$ bits of storage. As an example, we consider a customer database containing an *id*, a *province* and a *gender* of 16, 4 and 1 bits respectively. Points in their native space require $16 + 4 + 1 = 21$ bits to store, while the associated Hilbert indices will require $3 \times 16 = 48$ bits, representing a data expansion factor of $48/21 \approx 2.29$.

As the regular Hilbert walks through the hypercube $H(P)$, it repeatedly leaves and reenters the smaller bounding box of the input points $B(P)$. The points along the curve outside of $B(P)$ will never be used as indices for the points inside $B(P)$, hence the inherent inefficiency. A simple way to address this inefficiency is to imagine deleting the portions of the curve that lie outside of $B(P)$, replacing them with a single straight line from where the curve leaves $B(P)$ to where it reenters, and reindexing the point along the now shorter curve.

We wish to find an indexing scheme that preserves completely the ordering of the Hilbert indices, but requires only X bits to represent. A simple method to do this is to walk through all the points in $B(P)$, calculate their Hilbert indices and sort them based on these Hilbert indices. Then, assign to each point its rank as an index. Trivially, this index has the same ordering as the Hilbert ordering over $B(P)$, and it requires only $\sum_i m_i$ bits to represent. However, in order to generate such an index we must first enumerate the entire space, a prohibitive cost. The key to calculating this index directly, referred to as the *compact Hilbert index*, lies in a simple observation about Gray Codes.

4.2.1 Gray Code Rankings

We consider an n-bit Gray code $\text{gc}(i)$ where some subset of the bits are fixed. We let μ be a *mask* and π be a *pattern* such that $\pi \wedge \mu = 0$, where $\wedge$ is the binary-and operator. We restrict ourselves to values $\text{gc}(i)$ where $\text{bit}(\text{gc}(i), j) = \text{bit}(\pi, j)$ when $\text{bit}(\mu, j) = 0$. This is equivalent to restricting ourselves to values $\text{gc}(i)$ such that $\text{gc}(i) \wedge \neg\mu = \pi$, where $\neg$ represents the *binary-not* or *negation* operator. We let $\mathcal{I}$ be the set of integers that satisfy this condition, $\mathcal{I} = \{i |\, \mathrm{gc}(i) \wedge \neg\mu = \pi\}$. Let $\|\mu\|$ count the number of set bits of μ, or equivalently, the number of unconstrained bits in the definition of $\mathcal{I}$. It is easy to see that $|\mathcal{I}| = 2^{\|\mu\|} \leq 2^n$. We wish to determine $\|\mu\|$-bit values, the *Gray code ranks* $\mathrm{gcr}(\cdot)$, such that for all $i \neq j \in \mathcal{I}$, $i < j$ if and only if $\mathrm{gcr}(i) < \mathrm{gcr}(j)$. It is plain to see that $\mathrm{gcr}(i)$ must be equal to the rank of i with respect to all entries in $\mathcal{I}$. However, we wish to calculate the rank directly without having to enumerate over the entire set $\mathcal{I}$.

$\mathrm{gc}(i)$	8	10	12	14	20	26	28	30
i	15	12	8	11	16	19	23	20
$\mathrm{gcr}(i)$	3	2	0	1	4	5	7	6
$[\mathrm{gc}(i)]_{[2]}$	$0\underline{0}1\underline{00}0$	$0\underline{0}1\underline{01}0$	$0\underline{0}1\underline{10}0$	$0\underline{0}1\underline{11}0$	$0\underline{1}1\underline{00}0$	$0\underline{1}1\underline{01}0$	$0\underline{1}1\underline{10}0$	$0\underline{1}1\underline{11}0$
$[i]_{[2]}$	$0\underline{0}1\underline{11}1$	$0\underline{0}1\underline{10}0$	$0\underline{0}1\underline{00}0$	$0\underline{0}1\underline{01}1$	$0\underline{1}0\underline{00}0$	$0\underline{1}0\underline{01}1$	$0\underline{1}0\underline{11}1$	$0\underline{1}0\underline{10}0$
$[\mathrm{gcr}(i)]_{[2]}$	011	010	000	001	100	101	111	110

Table 4.1: Values of $\mathrm{gc}(i)$, i and $\mathrm{gcr}(i)$ for $\mu = [010110]_{[2]}$ and $\pi = [001000]_{[2]}$.

We consider an example where $n = 6$, $\|\mu\| = 3$, $\mu = [010110]_{[2]}$ and $\pi = [001000]_{[2]}$, shown in Table 4.1. The unconstrained bits are shown underlined to help in visualizing the effect of the mask and pattern. With a quick visual inspection it becomes readily apparent that the $\mathrm{gcr}(i)$ values can be constructed simply by concatenating the unconstrained bits from i. We formalize this concept with the following result.

Theorem 4.15 (Gray Code Rank)**.** *Let $\mathcal{U} = \{u_0 < \cdots < u_{\|\mu\|-1}\}$ be the indices of the unconstrained bits of a mask μ, such that* $\mathrm{bit}(\mu, u_k) = 1$ *for all* $0 \leq k < \|\mu\|$*, and let π be a pattern with respect to μ. Consider $i \neq j \in \mathcal{I}$, and define*

$$\bar{\imath} = \left[\mathrm{bit}\left(i, u_{\|\mu\|-1}\right) \cdots \mathrm{bit}\left(i, u_0\right)\right]_{[2]}.$$

Then $i < j$ if and only if $\bar{\imath} < \bar{\jmath}$. That is, the Gray code rank is given by $\mathrm{gcr}(i) = \bar{\imath}$*.*

Proof. It is obvious that two values $i \neq j$ may only differ at bit positions $u \in \mathcal{U}$. In other words, the only bits necessary to compare the relative order of i and j are precisely the bits of index $u \in \mathcal{U}$. If we remove the constrained bits from i, and keep the unconstrained bits in the same relative order, we are left with $\bar{\imath}$. Thus, it follows that $\bar{\imath}$ and $\bar{\jmath}$ will always have the same relative ordering as i and j. Since $\bar{\imath}$ is a $\|\mu\|$ digit binary number, it follows by the definition of gcr that $\mathrm{gcr}(i) = \bar{\imath}$. □

Algorithm 4 GRAYCODERANK(n, μ, π, i)

Given μ, π and n as per Lemma 4.15 and a value $i \in \mathcal{I}$, calculates $r \in \mathbb{B}^{\|\mu\|}$ such that $r = \mathrm{gcr}(i)$.

Input: $n \in \mathbb{Z}_+$, $\mu \in \mathbb{B}^n$ and $i \in \mathcal{I}$.

Output: $r \in \mathbb{B}^{\|\mu\|}$ such that $r = \mathrm{gcr}(i)$.

```
1: $r \leftarrow 0$
2: for $k = n - 1$ to $0$ do
3:     if $\mathrm{bit}\,(\mu, k) = 1$ then
4:         $r \leftarrow (r \lhd 1) \vee \mathrm{bit}\,(i, k)$
5:     end if
6: end for
```

As per Theorem 4.15, Algorithm 4 computes $\mathrm{gcr}(i)$ given n, μ, π and i. Given $\mathrm{gcr}(i)$ it is natural to want to reconstruct one or both of i and $\mathrm{gc}(i)$. We work in parallel to reconstruct the values of $\mathrm{gc}(i)$ and i given $\mathrm{gcr}(i)$. Since $i \in \mathcal{I}$ it follows that $\mathrm{bit}\,(\mathrm{gc}(i), k) = \mathrm{bit}\,(\pi, k)$ for $k \notin \mathcal{U}$. Additionally, when $k \in \mathcal{U}$ it follows that $\mathrm{bit}\,(i, k) = \mathrm{bit}\,(\mathrm{gcr}(i), j)$ where $k = u_j$. Given any k, exactly one of $\mathrm{bit}\,(i, k)$ or $\mathrm{bit}\,(\mathrm{gc}(i), k)$ is known. Theorem 4.1 lets us fill in the blanks as $\mathrm{bit}\,(\mathrm{gc}(i), k) = \mathrm{bit}\,(i, k) + \mathrm{bit}\,(i, k + 1)$. If we work from the most significant bit to the least significant bit, $\mathrm{bit}\,(i, k + 1)$ will be known at step k, allowing us to solve for the unknown bit. We formalize this procedure in Algorithm 5.

4.2.2 Algorithms

When calculating the Hilbert index, we determine in which side of the half-plane the coordinate $\mathbf{p}$ lies in with respect to each of the axes. The integer l is calculated at each iteration i of the algorithm as

$$\begin{aligned} l &= T_{(e,d)}([\mathrm{bit}\,(p_{n-1}, i) \cdots \mathrm{bit}\,(p_0, i)]_{[2]}) \\ &= \big([\mathrm{bit}\,(p_{n-1}, i) \cdots \mathrm{bit}\,(p_0, i)]_{[2]} \circlearrowleft (d+1)\big) \oplus \big(e \circlearrowleft (d+1)\big). \end{aligned}$$

We consider the case where axis j has precision m_j instead of all axes having precision m. Regardless of $\mathbf{p}$ it follows that $\mathrm{bit}\,(p_j, i) = 0$ when $i \geq m_j$. At iteration i, we define

$$\mu = [\alpha_{n-1} \cdots \alpha_0]_{[2]} \circlearrowleft (d+1), \text{ where } \alpha_j = \begin{cases} 1, & \text{if } m_j > i, \\ 0, & \text{otherwise;} \end{cases}$$

Algorithm 5 GRAYCODERANKINVERSE(n, μ, π, r)

Given μ, π and n as per Lemma 4.15 and a value $r \in \mathbb{B}^{\|\mu\|}$, calculates $i \in \mathcal{I}$, and $\mathrm{gc}(i) \in \mathbb{B}^n$ such that $r = \mathrm{gcr}(i)$.

Input: $n \in \mathbb{Z}_+$, $\mu, \pi \in \mathbb{B}^n$ and $r \in \mathbb{B}^{\|\mu\|}$.

Output: $i \in \mathcal{I}$ such that $r = \mathrm{gcr}(i)$; and $g = \mathrm{gc}(i) \in \mathbb{B}^n$.

```
1:  $(i, g, j) \leftarrow (0, 0, \|\mu\| - 1)$
2:  **for** $k$ from $n - 1$ to 0 **do**
3:      **if** $\mathrm{bit}\,(\mu, k) = 1$ **then**
4:          $\mathrm{bit}\,(i, k) \leftarrow \mathrm{bit}\,(r, j)$
5:          $\mathrm{bit}\,(g, k) \leftarrow \mathrm{bit}\,(i, k) + \mathrm{bit}\,(i, k + 1) \mod 2$
6:          $j \leftarrow j - 1$
7:      **else**
8:          $\mathrm{bit}\,(g, k) \leftarrow \mathrm{bit}\,(\pi, k)$
9:          $\mathrm{bit}\,(i, k) \leftarrow \mathrm{bit}\,(g, k) + \mathrm{bit}\,(i, k + 1) \mod 2$
10:     **end if**
11: **end for**
```

and $\pi = \big(e \circlearrowright (d+1)\big) \wedge \neg\mu$. It can be seen that $l \wedge \neg\mu = \pi$, thus we may apply Theorem 4.15 to $\mathrm{gc}^{-1}(l)$ to calculate a $\|\mu\|$-bit rank that maintains the same relative ordering as $\mathrm{gc}^{-1}(l)$. Thus, at each iteration i, instead of appending the n-bit value $\mathrm{gc}^{-1}(l)$ to h, we may append the $\|\mu\|$-bit value $\mathrm{gcr}(\mathrm{gc}^{-1}(l))$. Each dimension j will contribute a 1-bit to μ for iterations $0 \le i < m_j$, each time contributing a single bit to h. Thus, each dimension j will contribute *exactly* m_j bits to h, yielding a final index $X = \sum_j m_j$ bits in length. As desired, the constructed compact Hilbert code will have the same precision as the original point $\mathbf{p}$. We formalize this approach with Algorithms 6 and 7. The inverse procedure is equally straight-forward and is shown in Algorithm 8.

Given the tools presented in this chapter, it is relatively straight-forward to construct algorithms for efficiently iterating through all points on a regular or compact Hilbert curve, as well as for calculating various other quantities as per Moore [101].

Remark. The approach discussed here simply deletes the portions of the full Hilbert curve that lie outside of $H(P)$. This approach maintains completely the ordering imposed by the full Hilbert curve, but the compact curve no longer has a guarantee that successive points on

Algorithm 6 ExtractMask$(n, m_0, \ldots, m_{n-1}, i)$

Extracts a mask μ indicating which axes are active at a given iteration i of the CompactHilbertIndex algorithm.

Input: $n, m_0, \ldots, m_{n-1} \in \mathbb{Z}_+$ and $i \in \mathbb{Z}_n$.

Output: The mask μ of active dimensions at iteration i.

```
1: $\mu \leftarrow 0$
2: for $j = n-1$ to $0$ do
3:    $\mu \leftarrow \mu \triangleleft 1$
4:    if $m_j > i$ then
5:       $\mu \leftarrow \mu \vee 1$
6:    end if
7: end for
```

Algorithm 7 CompactHilbertIndex$(n, m_0, \ldots, m_{n-1}, \mathbf{p})$

Calculates the compact Hilbert index $h \in \mathbb{B}^X$ of a point $\mathbf{p} \in B(P)$.

Input: $n, m_0, \ldots, m_{n-1} \in \mathbb{Z}_+$ and a point $\mathbf{p} \in B(P)$.

Output: $h \in \mathbb{B}^H$, the compact Hilbert index of the point $\mathbf{p} \in B(P)$.

```
1: $(h, e, d) \leftarrow (0, 0, 0)$
2: $m \leftarrow \max_i\{m_i\}$
3: for $i = m-1$ to $0$ do
4:    $\mu \leftarrow$ ExtractMask$(n, m_0, \ldots, m_{n-1}, i)$
5:    $\mu \leftarrow \mu \circlearrowright (d+1)$
6:    $\pi \leftarrow (e \circlearrowright (d+1)) \wedge \neg\mu$
7:    $l \leftarrow [\mathrm{bit}\,(p_{n-1}, i) \cdots \mathrm{bit}\,(p_0, i)]_{[2]}$
8:    $l \leftarrow T_{(e,d)}(l)$
9:    $w = \mathrm{gc}^{-1}(l)$
10:   $r =$ GrayCodeRank$(n, \mu, \pi, w)$
11:   $e \leftarrow e \oplus (\mathrm{e}(w) \circlearrowleft (d+1))$
12:   $d \leftarrow d + \mathrm{d}(w) + 1 \bmod n$
13:   $h \leftarrow (h \triangleleft \|\mu\|) \vee r$
14: end for
```

Algorithm 8 COMPACTHILBERTINDEXINVERSE$(n, m_0, \ldots, m_{n-1}, h)$

Calculates the point $\mathbf{p} \in B(P)$ corresponding to a given compact Hilbert index $h \in \mathbb{B}^X$.

Input: $n, m_0, \ldots, m_{n-1} \in \mathbb{Z}_+$ and $h \in \mathbb{B}^X$, the compact Hilbert index of the point $\mathbf{p}$.

Output: A point $\mathbf{p} \in B(P)$.

```
$(e, d, k) \leftarrow (0, 0, 0)$
$\mathbf{p} = [p_0, \ldots, p_{n-1}] \leftarrow [0, \ldots, 0]$
$m \leftarrow \max_i \{m_i\}$
$X \leftarrow \sum_i \{m_i\}$
for $i = m - 1$ to $0$ do
    $\mu \leftarrow$ EXTRACTMASK$(n, m_0, \ldots, m_{n-1}, i)$
    $\mu \leftarrow \mu \circlearrowright (d + 1)$
    $\pi \leftarrow (e \circlearrowright (d + 1)) \wedge \neg\mu$
    $r \leftarrow [\mathrm{bit}\,(h, X - k - 1) \cdots \mathrm{bit}\,(j, X - k - \|\mu\|)]_{[2]}$
    $k \leftarrow k + \|\mu\|$
    $w \leftarrow$ GRAYCODERANKINVERSE$(n, \mu, \pi, r)$
    $l = \mathrm{gc}(w)$
    $l \leftarrow T^{-1}_{(e,d)}(l)$
    for $j = 0$ to $n - 1$ do
        $\mathrm{bit}\,(p_j, i) \leftarrow \mathrm{bit}\,(l, j)$
    end for
    $e \leftarrow e \oplus (\mathrm{e}(w) \circlearrowleft (d + 1))$
    $d \leftarrow d + \mathrm{d}(w) + 1 \bmod n$
end for
```

the curve are seperating by a unit distance. A complementary approach to creating compact Hilbert curves would consider creating a Hilbert-like curve that walks through $B(P)$ while respecting the property that adjacent points along the curve must always be seperated by a unit distance.

4.3 Experimental Results

To explore the performance of compact Hilbert indices we performed a series of experiments with both synthetic and real multi-dimensional data. In both cases, in addition to significant space savings, the use of compact Hilbert curves reduced the time required to sort data in Hilbert order. For example, for a 4-dimensional data-set extracted from a large Apache web log, compact Hilbert indices achieved a data size reduction of 2.2 and sorting based on these indices was 4.3 times faster than the dynamic comparison routine implemented in Moore's widely used library [101].

We implemented routines for mapping to and from both regular and compact Hilbert indices using the algorithms developed here. The algorithms are written in `C++` and seamlessly handle arbitrary precision data[5]. Our Hilbert curve algorithms were then compared to Moore's [101] implementation of Butz's [43] algorithms for various precisions and dimensions (up to $nm \leq 64$, the maximum supported by Moore's code) on both artificial and real data. The running times of our compact Hilbert indices were then compared to those of regular Hilbert indices over these and other data-sets. Finally, we examined the effect of using compact Hilbert indices in applications where regular Hilbert indices are currently used. All experiments were performed on a commodity Dual Intel Xeon 3.06GHz based computer with 2GB of main memory. All quoted times are wall time.

A variety of data-sets were used in the testing. The `WEBLOG` data-set consists of the log files of an Apache web server, taken over a 139 day period from August to September of 2004. A 4-dimensional data-set of $\sim$ 7.7 million points was extracted from the over 154 million rows of log data. The four dimensions recorded the IP address, day of access, hour of access and HTTP return code for each log entry. They had cardinalities of 834406, 139, 24 and 16, respectively, with bit sizes of 20, 8, 5 and 4. A regular Hilbert index requires 80 bits to represent while a compact Hilbert index requires only 37, a savings of over 2.16 times.

[5]See `http://web.cs.dal.ca/~chamilto/hilbert/`.

Further data-sets were taken from the `pgFoundry` sample database project[6], summarized in Table 4.2.

4.3.1 Space Savings

We evaluated the size of each column of each data-set and used this information to determine the bounding hypercube $H(P)$ size m, and the boundings box $B(P)$ sizes m_i. These values were then used to calculate the space savings factor that would be achieved by using compact Hilbert indices. The sizes of text columns were determined using the unique number of entries in the columns. Floating point columns were mapped to fixed-precision floating point columns, and treated as integers (effectively ignoring the decimal point). The results of this evaluation can be found in Table 4.2. Savings factors of up to 3.33 were seen, but the aggregate savings factor across all tested data-sets was 1.97. Thus a savings factors value of 2 was used as a baseline for the performance testing of the following section.

4.3.2 Performance

In order to characterize the performance of our algorithms we compared them to Moore's code over randomly generated data-sets for varying choices of N, m, n and X. For the purposes of compact Hilbert indices, precisions m_i were chosen in a monotonically decreasing fashion such that $X = nm/2$. Figure 4.7 shows the basic results. The jump visible at $n = 32$ in Figure 4.7(a) arises from the code switching to multiple precision representations of n-bit intermediate variables. In general, our regular Hilbert curve implementation slightly outperforms Moore's implementation. When $n \leq 32$ the overhead associated with compact Hilbert indices is as much as 2.5 times. However, as both n and m increase this reduces to a more reasonable ratio of 1.4. Although the compact Hilbert indices take slightly longer to compute, they are smaller than full Hilbert indices allowing data points to be replaced with compact Hilbert indices in-place. The results for Moore's implementation are not displayed as the total precision mn of these algorithms is limited to machine precision, 64 bits.

4.3.3 Sorting

As discussed earlier, a common application of Hilbert curves is to use them as a sort order for multi-dimensional data. There are typically two approaches used for doing this:

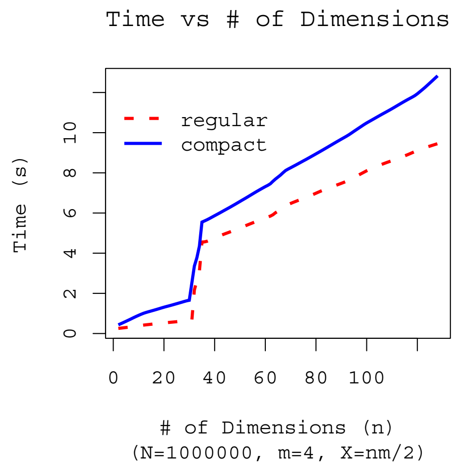
[6]See `http://pgfoundry.org/projects/dbsamples/`.

Dataset	**Table**	n	m	nm	X	nm/X
WEBLOG		4	20	80	37	2.16
booktown	alternate_stock	4	7	28	21	1.33
booktown	authors	3	15	45	25	1.80
booktown	books	4	16	64	39	1.64
booktown	customers	3	11	33	21	1.57
booktown	daily_inventory	2	3	6	4	1.50
booktown	distinguished_authors	4	15	60	18	**3.33**
booktown	editions	6	32	192	63	3.05
booktown	employees	3	10	30	16	1.88
booktown	recent_shipments	3	32	96	39	2.46
booktown	shipments	4	32	128	57	2.25
booktown	states	3	4	12	6	2.00
booktown	stock	4	7	28	21	1.33
booktown	stock_view	3	7	21	16	1.31
booktown	subjects	3	4	12	11	1.09
dellstore	cust_hist	3	15	45	43	1.05
dellstore	customers	20	17	340	199	1.71
dellstore	inventory	3	14	42	32	1.31
dellstore	orderlines	5	32	160	66	2.42
dellstore	orders	6	32	192	85	2.26
dellstore	products	7	14	98	66	1.48
dellstore	reorder	6	32	192	99	1.94
world	city	5	24	120	67	1.79
world	country	15	32	480	220	2.18
world	countrylanguage	4	32	128	50	2.56

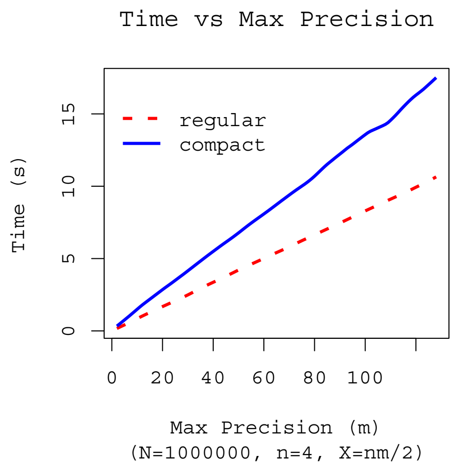
Table 4.2: Sample datasets and their dimensions, sizes and space-savings factors.

- **Precomputed.** In this approach, the Hilbert index of each point is pre-calculated, and either stored alongside the original coordinate or in place of the original coordinate. The data is then sorted based on this key, and the Hilbert index discarded. This approach is efficient in that Hilbert indices are only calculated once, but it may incur a space penalty as the Hilbert indices require $mn \geq X$ bits to represent. In fact, in the worst case the space-waste factor may be as high as $\mathrm{O}(nm/(n+m))$.

- **Online.** In this approach, the Hilbert index is never explicitly stored. Instead, a sort algorithm is run directly on the underlying coordinates and Hilbert indices are calculated on-the-fly as two coordinates are compared. A small optimization allows for simultaneous computation of the two Hilbert coordinates, one bit at a time, until they can be distinguished and ordered. When comparing values over a uniformly distributed random set of N values with precisions of $\Omega(\log N)$, it is expected that a



(a)



(b)

Figure 4.7: Comparing performance over random data-sets. (a) Time to calculate N indices with $m = 4$ as n varies. (b) Time to calculate N indices with $n = 4$ as m varies.

> single comparison will need to look at $\mathrm{O}(\log N)$ bits before distinguishing between the two numbers [63]. Since each bit of the Hilbert index requires $\mathrm{O}(1)$ time to compute, a sort of this type will have an expected complexity of $\mathrm{O}\big(N \log^2 N\big)$. This result assumes a quick-sort type algorithm is being used, but it is thought to hold for the general problem of sorting[7]. Since Algorithm 1 computes bits one at a time and is linear in the number of bits computed, the expected cost of an online comparison of compact Hilbert indices is proportional to the number of bits required to resolve the comparison, namely $\mathrm{O}(\log N)$.

Thus, it can be seen that sorting based on Hilbert indices introduces either a space inefficiency or a computation-time inefficiency. Compact Hilbert indices allow us to take the precomputation approach, but without paying a space penalty. This allows us to perform an in-place optimal-time sort of the original data, something which was not previously possible.

Figure 4.8a shows the results of sorting the `WEBLOG` data-set using both dynamic Hilbert indices and compact Hilbert indices. As predicted, for this and all other data-sets tested, the compact Hilbert sorting proved to be much more efficient. As shown in Figure 4.8b, for as little as 100K data items a speedup of 2 was observed. By 1M data items that speedup had

[7]Under the constraint that in order to compare a bit, we must first have compared all bits more significant than it; if we have random bit access a radix sort can generally do better. However, Hilbert indices are calculated incrementally precluding random bit access.

grown to a factor of 3.4. Speedup continued to increase beyond this point until it reached a factor of over 4.3 on the whole data-set. The shape of Figure 4.8 strongly validates the expected $O(\log N)$-factor improvement. Note that in distributed applications that order and partition data using Hilbert curves, such as [60, 111, 112], the benefits of using compact Hilbert curves would be even more pronounced. The use of compact Hilbert curves would result in the memory and time savings illustrated in Figure 4.8 as well as a corresponding reduction in the overall comunication volume and time.

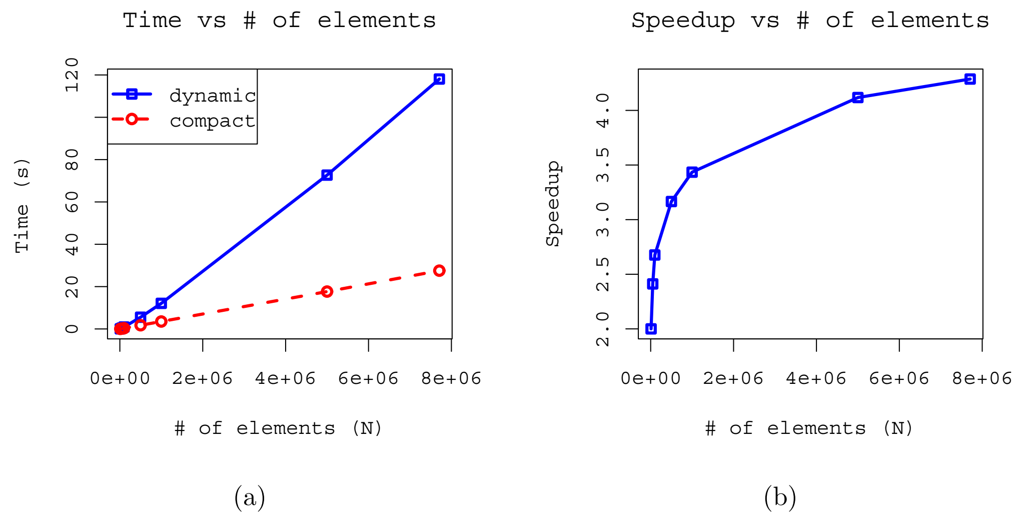


Figure 4.8: A comparison of dynamic Hilbert sorting and compact Hilbert sorting using the `WEBLOG` data-set. The `compact` curve includes the cost of converting both to and from compact Hilbert indices. (a) Wall times. (b) Relative speed-up.

Chapter 5

Lower Bounds

In this chapter, we study 3-sided range reporting, 3-d dominance reporting, and 3-d halfspace range reporting in the cache-oblivious model. We prove that any cache-oblivious data structure for these problems that achieves the optimal (or even a much weaker) query bound has to use asymptotically more space than a data structure with the same query bound in the I/O model.

As discussed in Chapter 3, there exist linear- or $\mathrm{O}(N \log^* N)$-space data structures that achieve the optimal query bound of $\mathrm{O}(\log_B N + K/B)$ block transfers for 3-sided range reporting, 3-d dominance reporting, and 3-d halfspace range reporting in the I/O model. In contrast, the best known data structures achieving the same query bound in the cache-oblivious model use $\mathrm{O}(N \log N)$ space. This raises the question whether linear-space cache-oblivious data structures with the optimal query bound exist for these problems. In this chapter, we give a negative answer to this question. We prove that any cache-oblivious data structure for 3-sided range reporting, 3-d dominance reporting or 3-d halfspace range reporting that achieves a query bound of $f(\log_B N, K/B)$, for any monotonically increasing function $f(\cdot,\cdot)$, has to use $\Omega(N(\log\log N)^\varepsilon)$ space.[1] This lower bound holds also for the expected size of Las-Vegas-type data structures, that is, data structures with randomized construction and query algorithms that guarantee correct query answers but achieve the desired query bound only in the expected case. The exponent ε depends on the function $f(\cdot,\cdot)$ and on the range of permissible block sizes. Our results are shown in a multi-level extension of the *indexability model* of [73], which is discussed in more detail in Section 5.1. In particular, they are independent of the set of supported internal-memory operations (comparisons, algebraic operations, etc.).

As a consequence of our lower bound for 3-sided range reporting, it follows that there is no linear-space cache-oblivious persistent B-tree that achieves the optimal 1-d range reporting bound of $\mathrm{O}(\log_B N + K/B)$ block transfers in the worst or expected case, while such a data

[1]We call a function $f(\cdot,\cdot)$ *monotonically increasing* if $x \geq x'$ and $y \geq y'$ imply $f(x,y) \geq f(x',y')$.

structure exists in the I/O model [35].

As discussed in Section 3.2, there have been previous results showing that the cache-oblivious model is less powerful than the I/O model. Brodal and Fagerberg [40] established a lower bound on the amount of main memory (as a function of B) necessary for optimal cache-oblivious sorting, while Bender et al. [36] proved that cache-oblivious searching has to cost a constant factor more than the search bound achieved in the I/O model using B-trees [34]. Our result is the first to establish an asymptotically growing gap between the *space* used by cache-oblivious and I/O-efficient data structures.

The key to obtaining our result is the construction of a hard point set and of a set of hard queries over this point set in combination with techniques to explicitly use the multi-level structure of the cache-oblivious model. Previous lower bound proofs for range reporting problems in the I/O model [28, 73, 85, 110] involved the construction of a hard point set together with a set of many "sufficiently different" queries of the *same* size. Combined with counting arguments, this ensured that the point set cannot be represented in linear space while guaranteeing a certain proximity (on disk) of the points reported by each query. The problems we study in this chapter allow linear-space or $O(N \log^* N)$-space solutions in the I/O model [2, 7, 28], as well as linear-space cache-oblivious solutions for queries of any fixed output size [9, 13, 25]. This means the previous techniques are ineffective for our purposes. In order to force a given point set to be hard for the problems we study, we construct many queries of *different* sizes. Combined with the multi-level nature of the cache-oblivious model, this allows us to create many incompatible proximity requirements for subsets of the point set and, thus, force duplication. It should be noted here that our construction of a hard point set (as well as the proof of Lemma 5.5) is inspired by a similar construction used by Afshani and Chan to prove a lower bound on the shallow partition theorem [7].

The results of this chapter have been published in [8].

5.1 A Multi-Level Indexability Model

The results in this chapter are proved in a multi-level extension of the indexability model of [73]. In particular, apart from a change of notation, the model of [73] is the same as our model restricted to one block size. This model allows us to prove lower bounds for (cache-oblivious and cache-aware) data structures for multi-level memory hierarchies without any restrictions of the set of supported internal-memory operations. The two main concepts of the model are those of *workload*, which captures a particular combination of problem instance and memory hierarchy, and *indexing scheme*, which provides an abstraction of a data structure. Next we define these two concepts, as well as the *size* and *efficiency* of an indexing scheme, and discuss their relationships to the data structure concepts they capture.

A *workload* is a triple $\mathcal{W} = (S, Q, \mathcal{B})$, where S is a set, Q is a set of subsets of S, and $\mathcal{B}$ is a set of integers. We call S the *base set*, Q the *query set*, and the elements of $\mathcal{B}$ *block sizes*. Intuitively, S is the set of elements to be stored in a data structure, every element of Q contains the set of elements in S to be reported by a particular query, and the elements of $\mathcal{B}$ are the cache block sizes of a particular memory hierarchy. We call a workload *realizable* as a range reporting problem of a given type if the elements in S can be identified with points in $\mathbb{R}^n$ so that, for every set $q \in Q$, there exists a query range of the given type containing exactly those points in S that correspond to the elements of q.

A *block cover* of S with block size B (short: *B-cover*) is a set of subsets (*blocks*) of S, each of size at most B and such that their union is S. An *indexing scheme* for the workload $\mathcal{W} = (S, Q, \mathcal{B})$ is a set of block covers of S, one for each block size in $\mathcal{B}$. Intuitively, an indexing scheme provides an abstraction of a data structure, as every layout of a concrete data structure $\mathcal{D}$ in external memory can place only B_i elements of S into each cache block of size $B_i \in \mathcal{B}$. We refer to the indexing scheme representing this placement of elements into cache blocks as $\mathcal{I}_{\mathcal{D}}$.

For a workload $\mathcal{W} = (S, Q, \mathcal{B})$ and an indexing scheme $\mathcal{I}$ for $\mathcal{W}$, we say that a B_i-cover $\mathcal{C}_i$ in $\mathcal{I}$ is *f-efficient*, for a function $f(\cdot, \cdot)$, if every query $q \in Q$ can be covered with at most $f(\log_{B_i} N, K/B_i)$ blocks in $\mathcal{C}_i$, where $N := |S|$ and $K := |q|$. The indexing scheme $\mathcal{I}$ is f-efficient if all its block covers are f-efficient. Applied to the indexing scheme $\mathcal{I}_{\mathcal{D}}$ corresponding to a data structure $\mathcal{D}$, this captures the number of cache blocks at each level of the memory hierarchy that need to be retrieved from $\mathcal{D}$ to answer any query $q \in Q$. As in [73], the cost of locating these blocks is ignored. In particular, a lower bound proved in this model is independent of the set of supported internal-memory operations.

For a block cover $\mathcal{C}$ of S, the *multiplicity* $\mu_{\mathcal{C}}(x)$ of an element $x \in S$ is the number of blocks in $\mathcal{C}$ that contain x. For an indexing scheme $\mathcal{I}$ for a workload $\mathcal{W} = (S, Q, \mathcal{B})$, the multiplicity of x is defined as $\mu_{\mathcal{I}}(x) = \max_{\mathcal{C}_i} \mu_{\mathcal{C}_i}(x)$, where $\mathcal{C}_i$ ranges over all block covers in $\mathcal{I}$. The *size* of $\mathcal{I}$ is $\sum_{x \in S} \mu_{\mathcal{I}}(x)$. For the indexing scheme $\mathcal{I}_{\mathcal{D}}$ representing a data structure $\mathcal{D}$ and a B_i-cover $\mathcal{C}_i$ in $\mathcal{I}_{\mathcal{D}}$, every element $x \in S$ is stored in $\mu_{\mathcal{C}_i}(x)$ cache blocks of size B_i

in $\mathcal{D}$. Hence, the multiplicity $\mu_{\mathcal{C}_i}(x)$ of an element $x \in S$ is a lower bound on the number of times x is stored in $\mathcal{D}$. Since this is true for every block cover in $\mathcal{I}_\mathcal{D}$, $\mu_{\mathcal{I}_\mathcal{D}}(x)$ is also a lower bound on the number of times x is stored in $\mathcal{D}$, and the size of $\mathcal{I}_\mathcal{D}$ is a lower bound on the size of $\mathcal{D}$. Thus, given a workload $\mathcal{W} = (S, Q, \mathcal{B})$, it suffices to prove a lower bound on the size of any f-efficient indexing scheme for $\mathcal{W}$ in order to obtain a lower bound on the size of any data structure that achieves a query cost of $f(\log_B N, K/B)$ block transfers for all queries in Q and all block sizes $B \in \mathcal{B}$.

The workloads $\mathcal{W} = (S, Q, \mathcal{B})$ we construct in this chapter have a large number of block sizes in $\mathcal{B}$. Thus, our lower bounds apply to *cache-aware* data structures with query cost bounded by $f(\log_B N, K/B)$ block transfers only under the assumption of a very deep memory hierarchy. A cache-oblivious data structure, on the other hand, is independent of the specific memory hierarchy. If it achieves a query bound of $f(\log_B N, K/B)$ block transfers for an arbitrary two-level hierarchy with block size B, it achieves this query bound at all levels of a multi-level hierarchy with the block sizes in $\mathcal{B}$. Thus, any lower bound we prove on the size of an f-efficient indexing scheme for $\mathcal{W}$ applies to any cache-oblivious data structure with a query cost of $f(\log_B N, K/B)$ block transfers.

5.2 A Lower Bound for Three-Sided Range Reporting

In this section, we present the main result of our chapter: a lower bound on the space used by any deterministic cache-oblivious data structure that supports 3-sided range reporting queries using at most $f(\log_B N, K/B)$ block transfers in the worst case, as stated in Theorem 5.1. The lower bounds for 3-d dominance reporting and 3-d halfspace range reporting are obtained from this result using reductions and are discussed in Section 5.4. The same section discusses how to extend the lower bound to Las-Vegas-type randomized data structures.

Theorem 5.1. *Let $f(\cdot,\cdot)$ be a monotonically increasing function, and $0 < \delta \leq 1/2$ a constant. Any cache-oblivious data structure capable of answering 3-sided range reporting queries using at most $f(\log_B N, K/B)$ block transfers in the worst case, for every block size $B \leq N^{2\delta}$, must use $\Omega(N(\log \log N)^\varepsilon)$ space, where $\varepsilon := 1/f(\delta^{-1}, 1)$.*

We prove Theorem 5.1 by constructing a workload $\mathcal{W}$ realizable as a 3-sided range reporting problem such that any f-efficient indexing scheme for $\mathcal{W}$ has size $\Omega(N(\log \log N)^\varepsilon)$; see Lemma 5.2 below. As discussed in Section 5.1, this implies Theorem 5.1. However, while an indexing scheme and its efficiency are defined without reference to cache sizes, a data structure can take advantage of the amount of available cache to speed up its queries. Our proof of Lemma 5.2 considers block sizes between N^{δ} and $N^{2\delta}$. Thus, a sufficiently strong tall cache assumption ($M = \omega\big(B^{1/\delta}\big)$, for all $0 < \delta \leq 1/2$) implies that the entire point set fits in cache, and the queries in Q can be answered without any block transfers after loading S into cache. This means that Theorem 5.1 holds only under the assumption that M is polynomial in B. In practice, this is a reasonable assumption. In the remainder of this section, we prove the following lemma.

Lemma 5.2. *Let $f(\cdot,\cdot)$ be a monotonically increasing function, and $0 < \delta \leq 1/2$ a constant. For every integer $N > 0$, there exists a workload $\mathcal{W} = (S, Q, \mathcal{B})$ realizable as a 3-sided range reporting problem, with $|S| = N$ and $B_i \leq N^{2\delta}$, for all $B_i \in \mathcal{B}$, and such that any f-efficient indexing scheme for $\mathcal{W}$ has size $\Omega(N(\log\log N)^{\varepsilon})$, where $\varepsilon := 1/f(\delta^{-1}, 1)$.*

First we show that, to prove Lemma 5.2, it suffices to focus on the case $\delta = 1/2$, that is, to allow arbitrarily large block sizes $B_i \leq N$ in $\mathcal{B}$.

Lemma 5.3. *If Lemma 5.2 holds for $\delta = 1/2$, it holds for any $0 < \delta \leq 1/2$.*

Proof. Consider a particular choice of N, f, and δ in Lemma 5.2, and let $N' := N^{2\delta}$ and $f'(x, y) := f(x/(2\delta), y)$. Since we assume that Lemma 5.2 holds for $\delta = 1/2$, there exist a workload $\mathcal{W}' = (S', Q', \mathcal{B})$ realizable as a 3-sided range reporting problem, with $|S'| = N'$ and $B_i \leq N'$, for all $B_i \in \mathcal{B}$, and such that any f'-efficient indexing scheme for $\mathcal{W}'$ has size $\Omega(N'(\log\log N')^{\varepsilon})$, where $\varepsilon := 1/f'(2, 1) = 1/f(\delta^{-1}, 1)$.

Now we construct a workload $\mathcal{W} = (S, Q, \mathcal{B})$ with $|S| = N$ as follows. Let $m := N/N'$. For $1 \leq i \leq m$, we define sets $S_i := \{(x, i) \mid x \in S'\}$ and $Q_i := \{\{(x, i) \mid x \in q\} \mid q \in Q'\}$. The sets S and Q are defined as $S := S_1 \cup S_2 \cup \cdots \cup S_m$ and $Q := Q_1 \cup Q_2 \cup \cdots \cup Q_m$. Since $\mathcal{W}'$ is realizable as a 3-sided range reporting problem, so is $\mathcal{W}$: it suffices to place m copies of the point set representing S' side by side.

The workload $\mathcal{W}_i = (S_i, Q_i, \mathcal{B})$ is the same as $\mathcal{W}'$ after renaming every element $x \in S'$ to (x, i). Thus, every f'-efficient indexing scheme for $\mathcal{W}_i$ has size $\Omega(N'(\log\log N')^{\varepsilon})$. Now observe that $f'(\log_B N', K/B) = f((\log_B(N^{2\delta}))/(2\delta), K/B) = f(\log_B N, K/B)$, that is, the restriction of every f-efficient indexing scheme for $\mathcal{W}$ to the elements of S_i is an f'-efficient indexing scheme for $\mathcal{W}_i$. This implies that every f-efficient indexing scheme for $\mathcal{W}$ has size $\Omega(mN'(\log\log N')^{\varepsilon}) = \Omega(N(\log\log N)^{\varepsilon})$.

□

By Lemma 5.3, it suffices to prove Lemma 5.2 for arbitrarily large block sizes. For a given size N of the set S, we construct the sets S, Q, and $\mathcal{B}$ in the workload $\mathcal{W} = (S, Q, \mathcal{B})$ recursively. Since the workload is to be realizable as a 3-sided range reporting problem, we do not distinguish between the elements in S and the points in $\mathbb{R}^2$ representing them, nor between the queries in Q and their corresponding 3-sided query ranges.

The recursive construction of the point set S is illustrated in Figure 5.1(a). At the first level of recursion, we divide the plane into a $t \times 2^{t-1}$-grid T, for a parameter t to be chosen later, and place different numbers of points into its cells. The points within each cell are arranged by dividing the cell into a $t \times 2^{t-1}$-grid of subcells and distributing the points over those subcells. This process continues recursively as long as each cell of the current subgrid contains more than $\sqrt{N}$ points.

The construction of the query set Q follows the recursive construction of the point set S. Each query at the top level comprises a union of cells of T chosen so that each top-level query outputs roughly the same number of points. For each grid cell, we construct a set of queries over the subgrid in this cell in a similar fashion. We repeat this construction recursively until we reach the last level in the recursive construction of the point set S.

The main idea now is to prove that, by choosing the top-level queries in Q appropriately and including an appropriate block size in $\mathcal{B}$, we can ensure that, for every f-efficient indexing scheme $\mathcal{I}$ for $\mathcal{W}$, there exists a grid cell in T at least half of whose points have multiplicity $\Omega(t^\varepsilon)$ in $\mathcal{I}$. For each subgrid at the next level of recursion, we add another block size to $\mathcal{B}$ that ensures that this subgrid has a cell at least half of whose points have multiplicity $\Omega(t^\varepsilon)$, and so on for every level of recursion. By making the recursion sufficiently deep, we achieve that at least a constant fraction of the points have multiplicity $\Omega(t^\varepsilon)$. The required recursion depth depends on t, and the largest value of t that allows for a sufficient recursion depth is $t \approx \log\log N$ (see remark at the end of Section 5.2.1). This gives the lower bound stated in Lemma 5.2.

We divide the details of our proof into two parts. In Section 5.2.1, we discuss the construction of the point set S and of the query set Q more precisely and prove that Lemma 5.2 follows if we can force at least half the points in one cell of each of the subgrids in the recursive construction to have the claimed multiplicity. In Section 5.2.2, we discuss how to achieve this duplication of points for each subgrid by including appropriate block sizes in $\mathcal{B}$.

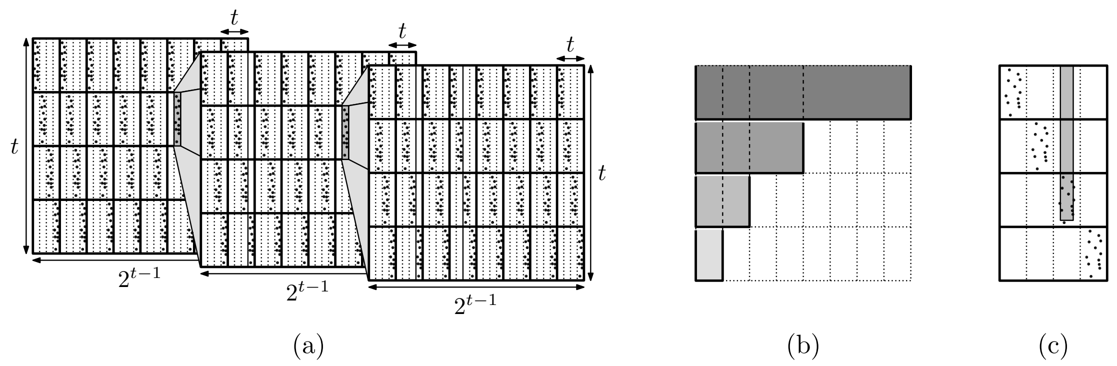


Figure 5.1: (a) The recursive construction of the point set. Fat solid lines bound grid cells, dotted lines separate subcolumns. (b) The set of queries in Q_T. Only one query is shown for each level of Q_T. (c) Queries at recursive levels output only points from their subgrids.

5.2.1 The Point Set and Query Set

To define the point set S, we construct a $t \times 2^{t-1}$-grid T, for a parameter $t := (\log\log N)/4$. This parameter remains fixed throughout the recursive construction. We refer to the grid cell in row i and column j as T_{ij}, for $1 \leq i \leq t$ and $1 \leq j \leq 2^{t-1}$. Every column of T is divided into t subcolumns, which also splits each cell T_{ij} into t subcells T_{ijk}, for $1 \leq k \leq t$. We now place $2^{i-1}N_1$ points into each cell in row i, where $N_1 := N/(2^{2t-1} - 2^{t-1})$. The points in cell T_{ij} are placed into subcell T_{iji}. This is illustrated in Figure 5.1(a). Observe that this ensures that each column of the grid receives $(2^t - 1)N_1$ points. Since there are 2^{t-1} columns, the total number of points in the grid is $(2^{2t-1} - 2^{t-1})N_1 = N$. The layout of the points within each cell is now obtained by applying the same procedure recursively to the set of points assigned to each cell. The recursion stops when the smallest cell in the current subgrid receives at most $\sqrt{N}$ points.

The query set Q is constructed by following the recursive construction of S. For the top-level grid T, we construct a set Q_T of queries consisting of t *levels*. For $1 \leq i \leq t$, level i contains 2^{i-1} queries, the kth of which is the union of all grid cells $T_{i'j'}$ satisfying $0 < i' \leq i$ and $(k-1)2^{t-i} < j' \leq k2^{t-i}$; see Figure 5.1(b). It is easily verified that every query in Q_T contains between $2^{t-1}N_1$ and $(2^t - 1)N_1$ points. Note that, even though we specify these queries as unions of grid cells, that is, effectively, as 4-sided queries, we can move their top boundaries to infinity without changing the set of points they report. To complete the construction of the query set Q, we apply the same construction recursively to each cell, adding a query set $Q_{T'}$ to Q, for each subgrid T' in the recursive construction of

S. Again, we can move the top boundary of each query in $Q_{T'}$ to infinity to make it 3-sided without changing the set of points it reports. Indeed, this clearly does not change the set of points from T' reported by the query, and the staggered layout of the points in each grid column into x-disjoint subcolumns ensures that there are no points in S that belong to the x-range of T' but are outside its y-range. This is illustrated in Figure 5.1(c).

Now let $\mathcal{W} = (S, Q, \mathcal{B})$ be a workload with S and Q as just defined. The following lemma provides the framework we use to prove Lemma 5.2.

Lemma 5.4. *Let $\mathcal{I}$ be an arbitrary indexing scheme for $\mathcal{W}$ and assume that every subgrid T' in the construction of S has a cell at least half of whose points have multiplicity $\Omega(m)$ in $\mathcal{I}$. Then the size of $\mathcal{I}$ is $\Omega(mN)$.*

Proof. We construct a set of disjoint cells such that at least half of the points in each cell have multiplicity $\Omega(m)$ and the total number of points in these cells is $\Omega(N)$. This proves that $\mathcal{I}$ has size $\Omega(mN)$.

To construct this set of cells, we apply the following recursive selection process, starting with $T' = T$. By the assumption of the lemma, the grid T' has a cell C' at least half of whose points have multiplicity $\Omega(m)$. We add C' to the set of selected cells and recurse on the subgrids in each of the remaining cells of T' unless the current grid T' is already at the lowest level of recursion in the construction of the point set S.

The set of cells selected in this fashion is easily seen to be disjoint, since we recursively select cells only from subgrids in cells not selected at the current level. We call a point *selected* if it is contained in one of the selected cells. We have to show that there are $\Omega(N)$ selected points in T.

For a grid T' containing N' points, the construction of S implies that each cell contains at least $N'/(2^{2t-1} - 2^{t-1}) > N'/4^t$ points. This allows us to show that at most $N'(1 - 4^{-t})^{r'+1}$ points from T' are not selected by the recursive selection process, where r' denotes the minimum recursion depth inside the cells of T'. Indeed, if $r' = 0$, exactly the points in C' are selected, leaving at most $N'' = N'(1 - 4^{-t})$ points in T' unselected. If $r' > 0$, we observe that, by induction, at most $N''(1 - 4^{-t})^{r'} = N'(1 - 4^{-t})^{r'+1}$ points in the cells of T' other than C' are not selected, while all points in C' are selected.

Since the recursion stops when the smallest cell of the current subgrid contains at most $\sqrt{N}$ points and, as just argued, every cell of a grid T' containing N' points contains at least

$N'/4^t$ points, the recursion depth r inside each cell of the top-level grid T is at least

$$\log_{4^t} \frac{N}{\sqrt{N}} = \frac{(\log N)/2}{2t} = \frac{\log N}{\log\log N} > \sqrt{\log N},$$

for N sufficiently large. Hence, the number of points in T that are not selected is at most

$$N(1 - 4^{-t})^{r+1} \leq Ne^{-(r+1)/4^t} \leq N/e$$

because $t = (\log\log N)/4$. Therefore, at least $(1 - 1/e)N = \Omega(N)$ points in T are selected. This completes the proof.

□

The proof of Lemma 5.4 is the reason we cannot choose a value of $t = \omega(\log\log N)$. To bound the number of unselected points by N/e, we need that $r + 1 \geq 4^t$. Since the recursion depth r is at most logarithmic in N, we cannot have $t = \omega(\log\log N)$.

5.2.2 Forcing Duplication in at Least One Cell

In this section, we prove that there exists a block size $B_{T'}$, for every subgrid T' in the recursive construction of S, such that every f-efficient $B_{T'}$-covering $\mathcal{C}$ of S has the property that there exists a cell of T' at least half of whose points have multiplicity $\Omega(t^\varepsilon)$ in $\mathcal{C}$. By including the block size $B_{T'}$ in $\mathcal{B}$, for every subgrid T' in the recursive construction of S, we obtain that every f-efficient indexing scheme $\mathcal{I}$ for the workload $\mathcal{W} = (S, Q, \mathcal{B})$ satisfies Lemma 5.4 with $m = t^\varepsilon$. Lemma 5.2 follows.

The argument is the same for all subgrids T'. Therefore, we focus on the top-level grid T and its corresponding query set Q_T in the remainder of this section. Recall that every query in Q_T contains between $N_1 2^{t-1}$ and $N_1(2^t-1)$ points. We choose a block size $B_T := (2^t-1)N_1$. Then every query $q \in Q_T$ satisfies $|q| \leq B_T$. Furthermore, $B_T \geq N_1 \geq \sqrt{N}$ and, therefore, $\log_{B_T} N \leq 2$. Thus, by the monotonicity of $f(\cdot,\cdot)$, every f-efficient B_T-covering $\mathcal{C}$ of S covers every query in Q_T with at most $\alpha := f(2, 1)$ blocks.

In the remainder of this section, we fix an arbitrary f-efficient B_T-covering $\mathcal{C}$ of S. We can represent every block in $\mathcal{C}$ by a unique colour and assign corresponding colour sets to the points in S and queries in Q_T. The colour set $C(p)$ of a point p comprises the colours of those blocks in $\mathcal{C}$ that contain p. Every query $q \in Q_T$ can be covered using at most α blocks in $\mathcal{C}$; we fix such a set of α blocks and define $C(q)$ to be the set of their colours. Our goal can now be rephrased as showing that there exists a cell in T at least half of whose points have $\Omega(t^\varepsilon)$ colours, for $\varepsilon := 1/\alpha$. We do this in two steps. In Lemma 5.5, we show that there exists a set of $s := \Omega(t/\log \alpha)$ cells in the same column of T such that half of the points in each cell are "exposed" in a sense defined below. Then we show that there exists at least one cell among these s cells such that every exposed point in this cell has at least $s^\varepsilon = \Omega(t^\varepsilon)$ colours.[2]

Consider a subset Z of cells of T. Each cell $c \in Z$ has a corresponding query $q_c \in Q_T$ that contains the points in c but not the points in the cell of T immediately below c. Let Q_Z be the set of queries corresponding to the cells in Z. We say a query q *covers* a point p if $C(p) \cap C(q) \neq \emptyset$, that is, p is contained in one of the α blocks of $\mathcal{C}$ chosen to cover q. A point p in the ith row of T is *Z-exposed* if no query $q_c \in Q_Z$ that belongs to a level $j < i$ in Q_T covers p. A cell of T is Z-exposed if at least half its points are Z-exposed. An observation we use in the proof of the following lemma is that a Z-exposed point or cell is also Z'-exposed, for every subset $Z' \subseteq Z$.

Lemma 5.5. *There exists a column in T containing a set Z of $\Omega(t/\log \alpha)$ Z-exposed cells.*

Proof. Let h be a parameter to be chosen later, and consider the subgrid T_h of T consisting of the rows with numbers $1, h+1, 2h+1, \ldots$. As each cell in row $ih+1$ contains $N_1 2^{ih}$ points, and there are 2^{t-1} cells in each row, the total number of points in row $ih+1$ is $X_i := N_1 2^{t+ih-1}$.

Next we bound the number of points in row $ih+1$ covered by queries at levels $1, h+1, \ldots, (i-1)h+1$ of the query set Q_T. Level $jh+1$ contains 2^{jh} queries. Hence, the total number of queries at levels $1, h+1, \ldots, (i-1)h+1$ is $\sum_{j=0}^{i-1} 2^{jh} < 2^{(i-1)h+1}$. The colour set of each such query contains at most α colours, and there are at most $B = N_1(2^t - 1)$ points with the same colour. Hence, at most $Y_i := \alpha N_1 2^{t+(i-1)h+1} = (4\alpha/2^h)X_i$ points have a colour belonging to the colour set of at least one of these queries and, thus, can be covered by these queries.

Now we choose $h = \lceil \log(16\alpha) \rceil$. Then $Y_i \leq X_i/4$. This implies that at least half of the cells in row $ih+1$ are T_h-exposed. Since this applies to all rows in T_h, there exists a column in T_h at least half of whose cells are T_h-exposed. Let Z be the set of T_h-exposed cells in this column. Since T_h has $\lceil t/h \rceil = \Omega(t/\log \alpha)$ rows, Z contains $\Omega(t/\log \alpha)$ cells, all of which are Z-exposed because $Z \subseteq T_h$.

[2]Since $\alpha = 1/\varepsilon$, we have $s^\varepsilon = \Omega((t/\log(1/\varepsilon))^\varepsilon)$, and it is not hard to show that $(\log(1/\varepsilon))^{-\varepsilon} > 1/2$, for all $0 < \varepsilon < 1$.

□

By Lemma 5.5, there exists a sequence $Z = \langle c_1, c_2, \ldots, c_s \rangle$ of $s = \Omega(t/\log \alpha)$ Z-exposed cells in some column of T. Now, for all $1 \le i \le s$, we choose p_i to be a Z-exposed point in cell c_i with the minimum number of colours. It suffices to show that there exists an index i such that p_i has at least s^ε colours, as this implies that all Z-exposed points in c_i have at least this many colours and at least half the points in c_i are Z-exposed. To prove this, we consider the point-query sequence $\langle (p_1, q_1), (p_2, q_2), \ldots, (p_s, q_s) \rangle$, where p_i is chosen from the set of Z-exposed points in cell c_i as just discussed, and $q_i := q_{c_i}$, for all $1 \le i \le s$. The sequence $\langle (p_1, q_1), (p_2, q_2), \ldots, (p_s, q_s) \rangle$ has the following three properties.

(i) For $1 \le i \le j \le s$, $C(p_i) \cap C(q_j) \ne \emptyset$ (because $p_i \in q_j$).

(ii) For $1 \le j < i \le s$, $C(p_i) \cap C(q_j) = \emptyset$ (because p_i is exposed).

(iii) For $1 \le j \le s$, $|C(q_j)| \le \alpha$.

We prove that any assignment of colour sets to the points and queries in $\langle (p_1, q_1), (p_2, q_2), \ldots, (p_s, q_s) \rangle$ that satisfies these three properties also has the property that $|C(p_i)| \ge s^{1/\alpha} = s^\varepsilon$, for some $1 \le i \le s$. We use the following terminology and notation.

We call an assignment of colour sets to the points and queries in the sequence $\langle (p_1, q_1), (p_2, q_2), \ldots, (p_s, q_s) \rangle$ a *colouring*. The colouring is *proper* if it satisfies conditions (i) and (ii). The colouring is an (α, β)-*colouring* if each query is assigned at most α colours (condition (iii)) and each point is assigned at most β colours. We say that a point-query sequence $\langle (p_1, q_1), (p_2, q_2), \ldots, (p_s, q_s) \rangle$ is (α, β)-*colourable* if it has a proper (α, β)-colouring. Finally, let $L(\alpha, \beta)$ be the maximum length s such that the sequence $\langle (p_1, q_1), (p_2, q_2), \ldots, (p_s, q_s) \rangle$ is (α, β)-colourable, and $\ell(\alpha, \beta) := L(\alpha, \beta - 1) + 1$ the minimum length s such that the sequence $\langle (p_1, q_1), (p_2, q_2), \ldots, (p_s, q_s) \rangle$ is not $(\alpha, \beta - 1)$-colourable. With this notation, our goal is to show that $\ell(\alpha, \beta) \le \beta^\alpha$ because it implies that $\ell(\alpha, s^{1/\alpha}) \le s$, that is, a colouring of a point-query sequence $\langle (p_1, q_1), (p_2, q_2), \ldots, (p_s, q_s) \rangle$ of length s that satisfies conditions (i)–(iii) assigns at least $s^{1/\alpha}$ points to at least one point p_i.

Lemma 5.6. *For $\alpha \ge 0$ and $\beta \ge 1$, $\ell(\alpha, \beta) = \binom{\alpha+\beta-1}{\alpha} \le \beta^\alpha$.*

Proof. First we prove that $\ell(\alpha, \beta)$ satisfies the recurrence relation

$$\ell(\alpha, \beta) = \begin{cases} 1 & \alpha = 0 \text{ or } \beta = 1 \\ \ell(\alpha, \beta - 1) + \ell(\alpha - 1, \beta) & \alpha > 0 \text{ and } \beta > 1. \end{cases} \tag{5.7}$$

The base case ($\alpha = 0$ or $\beta = 1$) is fairly obvious: a point-query sequence of length 1 is neither $(0, \beta)$-colourable, for any β, nor $(\alpha, 0)$-colourable, for any α, while a point-query sequence of length 0 is (α, β)-colourable, for any α and β.

For the inductive step ($\alpha > 0$ and $\beta > 1$), we prove that $L(\alpha, \beta) = \ell(\alpha, \beta) + L(\alpha - 1, \beta)$. This implies that

$$\begin{aligned}\ell(\alpha, \beta) &= L(\alpha, \beta - 1) + 1\\ &= \ell(\alpha, \beta - 1) + L(\alpha - 1, \beta - 1) + 1\\ &= \ell(\alpha, \beta - 1) + (\ell(\alpha - 1, \beta) - 1) + 1\\ &= \ell(\alpha, \beta - 1) + \ell(\alpha - 1, \beta).\end{aligned}$$

First we prove that $L(\alpha, \beta) \leq \ell(\alpha, \beta) + L(\alpha - 1, \beta)$. To this end, we consider a point-query sequence $\langle (p_1, q_1), (p_2, q_2), \ldots, (p_s, q_s) \rangle$, where $s := L(\alpha, \beta)$. Since $L(\alpha - 1, \beta) \geq 0$, for all α and β, $s \leq \ell(\alpha, \beta)$ would immediately imply $s \leq \ell(\alpha, \beta) + L(\alpha - 1, \beta)$. Thus, we can assume that $s > \ell(\alpha, \beta)$. Let $s' := \ell(\alpha, \beta)$.

Now let C be a proper (α, β)-colouring of $\langle (p_1, q_1), (p_2, q_2), \ldots, (p_s, q_s) \rangle$. Since the restriction of the C to the point-query sequence $\langle (p_1, q_1), (p_2, q_2), \ldots, (p_{s'}, q_{s'}) \rangle$ is a proper (α, β)-colouring of this sequence, the definition of $\ell(\alpha, \beta)$ implies that there exists a point p_k, $1 \leq k \leq s'$, such that $|C(p_k)| = \beta$ and $C(p_k) \subseteq \bigcup_{j=1}^{s'} C(q_j)$. We use this to construct a proper $(\alpha - 1, \beta)$-colouring C' of the subsequence $\langle (p_{s'+1}, q_{s'+1}), (p_{s'+2}, q_{s'+2}), \ldots, (p_s, q_s) \rangle$, thereby showing that its length $s - s'$ is at most $L(\alpha - 1, \beta)$, which implies that $L(\alpha, \beta) = s = s' + (s - s') \leq \ell(\alpha, \beta) + L(\alpha - 1, \beta)$.

To obtain such a colouring C', we define $C'(p_i) = C(p_i)$ and $C'(q_i) = C(q_i) \setminus C(p_k)$, for all $s' < i \leq s$. By property (i) of C, $C(q_i) \cap C(p_k) \neq \emptyset$, for $s' < i \leq s$ and, hence, $|C'(q_i)| < |C(q_i)| \leq \alpha$, while $|C'(p_i)| = |C(p_i)| \leq \beta$. Thus, C' is an $(\alpha - 1, \beta)$-colouring of the sequence $\langle (p_{s'+1}, q_{s'+1}), (p_{s'+2}, q_{s'+2}), \ldots, (p_s, q_s) \rangle$. Next we show that C' is proper.

First observe that, for $s' < j < i \leq s$, $C'(p_i) \cap C'(q_j) \subseteq C(p_i) \cap C(q_j) = \emptyset$, by property (ii) of C. Hence, C' satisfies property (ii).

For $s' < i \leq j \leq s$, we have $C(p_i) \cap C(p_k) \subseteq C(p_i) \cap \bigcup_{h=1}^{s'} C(q_h) = \emptyset$, by property (ii) of C. Hence, $C'(p_i) \cap C'(q_j) = C(p_i) \cap (C(q_j) \setminus C(p_k)) = C(p_i) \cap C(q_j) \neq \emptyset$, by property (i) of C. Thus, C' satisfies property (i). This concludes the proof of the inequality $L(\alpha, \beta) \leq \ell(\alpha, \beta) + L(\alpha - 1, \beta)$.

Next we prove that a point-query sequence $\langle (p_1, q_1), (p_2, q_2), \ldots, (p_s, q_s) \rangle$ of length $s :=$

$\ell(\alpha,\beta) + L(\alpha-1,\beta) = L(\alpha,\beta-1) + L(\alpha-1,\beta) + 1$ is (α,β)-colourable. This proves that $L(\alpha,\beta) \geq \ell(\alpha,\beta) + L(\alpha-1,\beta)$.

We divide the sequence into three subsequences

$$\begin{gathered} \langle (p_1,q_1),(p_2,q_2),\ldots,(p_{s_1},q_{s_1})\rangle\,, \\ \langle (p_{s_1+1},q_{s_1+1})\rangle\,,\ \text{and} \\ \langle (p_{s_1+2},q_{s_1+2}),(p_{s_1+3},q_{s_1+3}),\ldots,(p_s,q_s)\rangle\,, \end{gathered}$$

where $s_1 := L(\alpha,\beta-1)$ and $s_3 := s - s_1 - 1 = L(\alpha-1,\beta)$. By the choice of s_1 and s_3, the sequence $\langle (p_1,q_1),(p_2,q_2),\ldots,(p_{s_1},q_{s_1})\rangle$ has a proper $(\alpha,\beta-1)$-colouring C_1, and the sequence $\langle (p_{s_1+2},q_{s_1+2}),(p_{s_1+3},q_{s_1+3}),\ldots,(p_s,q_s)\rangle$ has a proper $(\alpha-1,\beta)$-colouring C_3. We can assume that the two colourings use different colours. We define a colouring C of the sequence $\langle (p_1,q_1),(p_2,q_2),\ldots,(p_s,q_s)\rangle$ as

$$C(p_i) = \begin{cases} C_1(p_i) \cup \{\gamma\} & 1 \leq i \leq s_1 \\ \{\gamma\} & i = s_1+1 \\ C_3(p_i) & s_1+2 \leq i \leq s \end{cases}$$

and

$$C(q_i) = \begin{cases} C_1(q_i) & 1 \leq i \leq s_1 \\ \{\gamma\} & i = s_1+1 \\ C_3(q_i) \cup \{\gamma\} & s_1+2 \leq i \leq s \end{cases}\,,$$

where γ is a new colour not used by either C_1 or C_3. Since C_1 is an $(\alpha,\beta-1)$-colouring of the sequence

$$\langle (p_1,q_1),(p_2,q_2),\ldots,(p_{s_1},q_{s_1})\rangle$$

and C_3 is an $(\alpha-1,\beta)$-colouring of

$$\langle (p_{s_1+2},q_{s_1+2}),(p_{s_1+3},q_{s_1+3}),\ldots,(p_s,q_s)\rangle\,,$$

C is an (α,β)-colouring of

$$\langle (p_1,q_1),(p_2,q_2),\ldots,(p_s,q_s)\rangle\,.$$

Next we show that C is proper.

First consider property (i). For a point p_i and a query q_j with $1 \le i \le j \le s$, we distinguish three cases. If $j \le s_1$, then $C(p_i) \supset C_1(p_i)$ and $C(q_j) = C_1(q_j)$. Hence, since $C_1(p_i) \cap C_1(q_j) \neq \emptyset$ (by property (i) of the colouring C_1), we have $C(p_i) \cap C(q_j) \neq \emptyset$. If $i \le s_1 + 1 \le j$, we have $\gamma \in C(p_i) \cap C(q_j)$. If $s_1 + 1 < i$, then $C(p_i) = C_3(p_i)$ and $C(q_j) \supset C_3(q_j)$. Hence, since $C_3(p_i) \cap C_3(q_j) \neq \emptyset$ (by property (i) of the colouring C_3), we have $C(p_i) \cap C(q_j) \neq \emptyset$.

Next we verify property (ii). Consider a point p_i and a query q_j with $1 \le j < i \le s$. If $i \le s_1$, we have $C(p_i) \cap C(q_j) = \emptyset$ because $C_1(p_i) \cap C(q_j) = C_1(p_i) \cap C_1(q_j) = \emptyset$ (by property (ii) of the colouring C_1) and $\gamma \notin C(q_j)$. If $j \le s_1 < i$, we have $C(q_j) \cap C(p_i) = \emptyset$ because $\gamma \notin C(q_j)$ and colourings C_1 and C_3 use different colours. Finally, if $s_1 + 1 \le j$, then $C(p_i) \cap C(q_j) = \emptyset$ because $C(p_i) \cap C_3(q_j) = C_3(p_i) \cap C_3(q_j) = \emptyset$ (by property (ii) of the colouring C_3) and $\gamma \notin C(p_i)$.

This shows that C is a proper (α, β)-colouring of the sequence $\langle (p_1, q_1), (p_2, q_2), \ldots, (p_s, q_s) \rangle$ and, hence, that $\ell(\alpha, \beta) + L(\alpha - 1, \beta) = s \le L(\alpha, \beta)$.

Having established the correctness of (5.7), it remains to derive a closed form for $\ell(\alpha, \beta)$. We do this using induction. For the base case, we have $\ell(0, \beta) = 1 = \binom{\beta-1}{0}$ and $\ell(\alpha, 1) = 1 = \binom{\alpha}{\alpha}$. For the inductive step, we obtain

$$\begin{aligned}
\ell(\alpha, \beta) &= \ell(\alpha, \beta - 1) + \ell(\alpha - 1, \beta) \\
&= \binom{\alpha + \beta - 2}{\alpha} + \binom{\alpha + \beta - 2}{\alpha - 1} \\
&= \binom{\alpha + \beta - 1}{\alpha}.
\end{aligned}$$

This finishes the proof.

□

To summarize, Lemma 5.2 is established by invoking Lemma 5.5 to prove that there exists a set Z of $s = \Omega(t/\log \alpha)$ Z-exposed cells in T. Using Lemma 5.6, we show that the Z-exposed point with the minimum number of colours in at least one of these cells has at least $s^{\varepsilon} = \Omega(t^{\varepsilon})$ colours, for $\varepsilon = 1/\alpha$ and $\alpha = f(2, 1)$. Thus, every Z-exposed point in this cell has at least $\Omega(t^{\varepsilon})$ colours. Since at least half the points in a Z-exposed cell are Z-exposed, at least half the points in this cell have $\Omega(t^{\varepsilon})$ colours. The same argument can be applied to every subgrid in the recursive construction of the point set S, which implies that every

f-efficient indexing scheme for the workload $\mathcal{W}$ satisfies Lemma 5.4 with $m = \Omega(t^\varepsilon)$, which proves Lemma 5.2 for $\delta = 1/2$. By Lemma 5.3, this implies Lemma 5.2 for any $0 < \delta \leq 1/2$.

5.3 Tightness of the Lower Bound

In this section, we show that the lower bound of Lemma 5.2 is tight for the workload $\mathcal{W} = (S, Q, \mathcal{B})$ we constructed in the previous section, up to the dependence of ε on $f(\cdot,\cdot)$.

Theorem 5.8. *Let $f(\cdot,\cdot)$ be a monotonically increasing function such that $f(x, y) \geq \lceil y \rceil$, and let $\alpha := f(2, 1)$. There exists a g-efficient indexing scheme $\mathcal{I}$ for $\mathcal{W}$ of size $\mathrm{O}\big(N(\log\log N)^{1/\alpha}\big)$, for some function $g(\cdot,\cdot)$ that satisfies $g(x, y) = \mathrm{O}(f(x, y))$.*

The condition that $f(x, y) \geq \lceil y \rceil$ is not a restriction because every function $f(\cdot,\cdot)$ such that there exists an f-efficient indexing scheme for $\mathcal{W}$ satisfies this condition. To prove Theorem 5.8, we define a linear layout of the points in S that stores each point $\mathrm{O}\big(t^{1/\alpha}\big)$ times and such that every query $q \in Q$ can be decomposed into $\mathrm{O}(\alpha + 1)$ contiguous subsequences of this linear layout. By partitioning the layout into blocks of size B, for any B, we obtain a B-cover of S that can cover every query in Q with $\mathrm{O}(\alpha + 1 + K/B) = \mathrm{O}(f(\log_B N, K/B))$ blocks because $\alpha = f(2, 1) = \mathrm{O}(1)$ and $f(\log_B N, K/B) \geq \lceil K/B \rceil$. In particular, the B_i-covers obtained in this fashion, for all $B_i \in \mathcal{B}$, form a g-efficient indexing scheme of size $\mathrm{O}\big(Nt^{1/\alpha}\big) = \mathrm{O}\big(N(\log\log N)^{1/\alpha}\big)$ for $\mathcal{W}$, where $g(x, y) = \mathrm{O}(f(x, y))$.

The layout. To construct the layout of the points in S, let β be the smallest integer such that $\ell(\alpha, \beta + 1) \geq t$; that is, $\beta = \mathrm{O}\big(t^{1/\alpha}\big) = \mathrm{O}\big((\log\log N)^{1/\alpha}\big)$. To simplify the argument, we can assume that $\ell(\alpha, \beta + 1) = t$, which we can achieve by padding each grid with empty rows at the bottom. Our layout consists of β copies of S, which immediately implies that the layout uses $N\beta = \mathrm{O}\big(N(\log\log N)^{1/\alpha}\big)$ space.

In the first copy, we divide the rows of the top-level grid T into $\alpha+1$ groups $G_0, G_1, \ldots, G_\alpha$, ordered from top to bottom. The size of group G_i is $\ell(\alpha - i, \beta)$. The cells in each group are laid out in column-major order. This is illustrated in Figure 5.2(a). Note that the groups $G_0, G_1, \ldots, G_\alpha$ cover all the rows of T because $t = \ell(\alpha, \beta + 1) = \sum_{i=0}^{\alpha} \ell(i, \beta)$; the second equality follows from the classic equality $\binom{n+1}{m+1} = \sum_{i=0}^{n} \binom{i}{m}$. To complete the layout of the points in the first copy of S, we have to determine the order in which to arrange the points in each cell of T. Since each cell of T is divided recursively into $t \times 2^{t-1}$ subgrids,

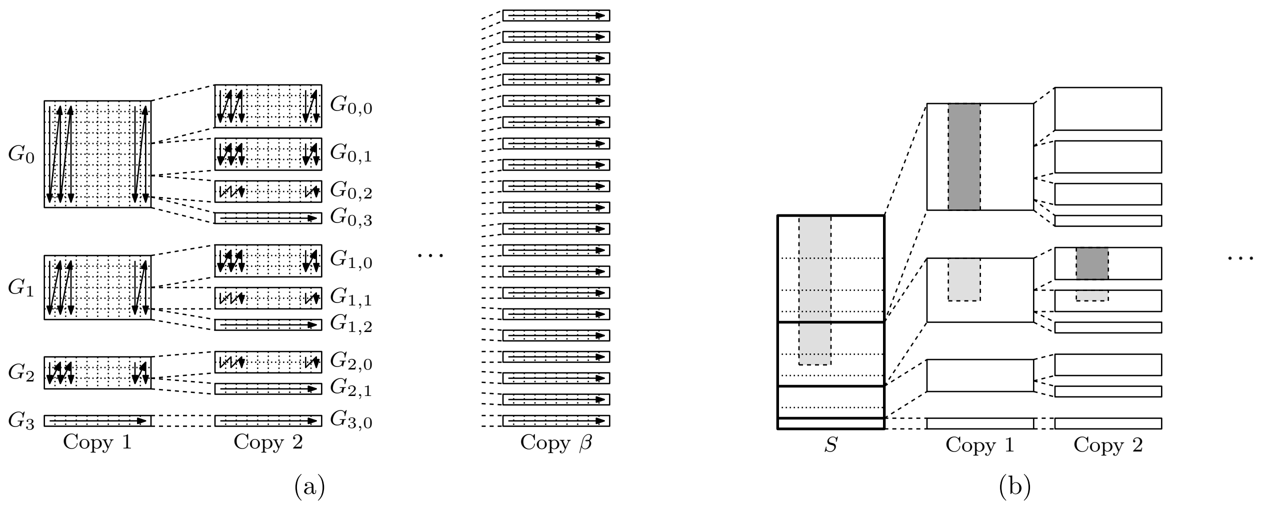


Figure 5.2: (a) An $\mathrm{O}\big(N(\log\log N)^{1/\alpha}\big)$-space layout for the point set S (layout shown for $\alpha = 3$). (b) Answering a query on the first two levels of the layout. The dark portion of the query in each copy of S forms a contiguous subsequence of the column-major layout of the respective group. The light portion is answered using subsequent copies of S.

we can divide the rows of each such subgrid T' into groups $G'_0, G'_1, \ldots, G'_\alpha$ in the same fashion as just described and lay out the cells in each group in column-major order. This continues recursively until we reach subgrids T' whose cells have not been divided further in the construction of the point set S. For each such subgrid, we arrange the points in each cell in an arbitrary order.

For $1 < k \leq \beta$, the layout of the points in the kth copy of S is obtained by dividing the groups used to define the $(k-1)$st copy into subgroups and laying out the cells in each subgroup in column-major order. In particular, a group G in the $(k-1)$st copy consisting of $\ell(\alpha', \beta - k + 2)$ rows is divided into $\alpha' + 1$ subgroups $G_0, G_1, \ldots, G_{\alpha'}$, with G_i covering $\ell(i, \beta - k + 1)$ rows. This is illustrated in Figure 5.2(a) for the second copy of S. Similar to the argument for the first copy of S, we have $\ell(\alpha', \beta - k + 2) = \sum_{i=0}^{\alpha'} \ell(i, \beta - k + 1)$, that is, the subgroups cover the rows of G exactly. To determine the order in which to lay out the points in each cell of T, we apply the same refinement process to the groups in each subgrid T' in the recursive construction of S.

By continuing in this fashion until $k = \beta$, we obtain that every group in the βth copy of S consists of $\ell(\alpha', 1) = 1$ rows, for some α'. Hence, the βth copy of S stores the cells in each row of T (and of any subgrid T') in x-sorted order. This is illustrated on the right of Figure 5.2(a).

Answering queries. Since the layout arranges the cells of each subgrid T' in the recursive construction of S in the same fashion as the cells of T, it suffices to prove that any query $q \in Q_T$ can be partitioned into at most $\alpha + 1$ contiguous subsequences in the layout.

Assume the query q is at the ith level of Q_T, that is, its bottom boundary coincides with the bottom boundary of the ith row of T. If this row is the bottom-most row of some group G_{j_1} in the first copy of S, observe that the points in q that belong to any group G_j, $0 \le j \le j_1$, are stored contiguously in the first copy of S. Hence, the points in q are divided into $j_1 + 1$ contiguous subsequences.

If row i belongs to group G_{j_1} but is not its bottom row, we use the first copy of S to report all points of q that belong to groups $G_0, G_1, \ldots, G_{j_1-1}$ and use subsequent copies of S to report all points in q that belong to G_{j_1}. We say that group G_{j_1} is the *partial group of* q in the first copy of S. By the arguments in the previous paragraph, the points of q that belong to groups $G_0, G_1, \ldots, G_{j_1-1}$ form j_1 contiguous subsequences in the first copy of S.

To see how subsequent copies of S are used to report the remaining points of q, consider the kth copy and let G be the partial group of q in the $(k-1)$st copy. This group has $\ell(\alpha', \beta - k + 2)$ rows, for some α'. Then we consider the subgroup G_{j_k} of G containing row i. As for the first copy of S, if row i is the bottom-most row of group G_{j_k}, we use groups $G_0, G_1, \ldots, G_{j_k}$ to report all points in G that belong to q. These points are stored in $j_k + 1$ contiguous subsequences of the kth copy of S. If row i is not the bottom-most row of group G_{j_k}, then G_{j_k} is once again a partial group of q, and we report only those points in q that belong to $G_0, G_1, \ldots, G_{j_k-1}$ using the kth copy of S, and use subsequent copies of S to report the points in q that belong to G_{j_k}. Figure 5.2(b) illustrates this recursive partitioning of query q for the first two copies of S. Note that this procedure terminates at the latest when reaching the last copy of S because this copy stores all rows in x-sorted order, that is, the group G_{j_β} cannot be partial.

It remains to bound the number of contiguous subsequences into which the points of q are divided by this procedure. If the procedure terminates after inspecting the first s copies of S, the points in q are divided into $1 + \sum_{k=1}^{s} j_k$ contiguous subsequences, j_k in the kth copy of S, for $1 \le k < s$, and $j_s + 1$ in the sth copy. For all $1 < k \le s$, if the partial group G of q in the $(k-1)$st copy of S has $\ell(\alpha', \beta - i + 2)$ rows, then $j_k \le \alpha'$ and G_{j_k} has $\ell(\alpha' - j_k, \beta - i + 1)$ rows. Since $j_1 \le \alpha$, this implies that $1 + \sum_{k=1}^{s} j_k \le \alpha + 1$ using a simple inductive argument, that is, each query $q \in Q$ is partitioned into at most $\alpha + 1$ contiguous subsequences of the layout.

5.4 Further Lower Bounds

In this section, we show that the proof technique from Section 5.2 can be used to obtain the same lower bound as in Theorem 5.1 for other range searching problems and to obtain a lower bound on the space consumption of cache-oblivious persistent B-trees with optimal 1-d range queries. The lower bounds in Section 5.4.1, for 3-d dominance reporting and persistent B-trees, are obtained using direct reductions from 3-sided range reporting. In Section 5.4.2, we prove the same lower bound for 3-d halfspace range reporting, but this requires more care, as there is no direct reduction from 3-sided range reporting to 3-d halfspace range reporting. Finally, in Section 5.4.3, we show that all lower bounds proved in Sections 5.2, 5.4.1, and 5.4.2 also hold for the expected size of Las-Vegas-type data structures that achieve an expected query bound of at most $f(\log_B N, K/B)$ block transfers.

5.4.1 3-D Dominance Reporting and Persistent B-Trees

The first result in this section is a lower bound on the space consumption of any cache-oblivious data structure for 3-d dominance reporting, as summarized in the following theorem.

Theorem 5.9. *Let $f(\cdot, \cdot)$ be a monotonically increasing function, and $0 < \delta \leq 1/2$ a constant. Any cache-oblivious data structure capable of answering 3-d dominance reporting queries using at most $f(\log_B N, K/B)$ block transfers in the worst case, for every block size $B \leq N^{2\delta}$, must use $\Omega(N(\log \log N)^{\varepsilon})$ space, where $\varepsilon = 1/f(\delta^{-1}, 1)$.*

Proof. Just as Theorem 5.1, this result follows if we can show that the workload $\mathcal{W} = (S, Q, \mathcal{B})$ constructed in Section 5.2 to prove Lemma 5.2 is realizable as a 3-d dominance reporting problem. We do this using a simple geometric transformation. We map each input point $p = (x_p, y_p)$ in S to the point $\phi(p) = (-x_p, x_p, -y_p)$ in $\mathbb{R}^3$, and every 3-sided query range $q = [l, r] \times [b, +\infty)$ in Q to the query range $\phi(q) = (-\infty, -l] \times (-\infty, r] \times (-\infty, -b]$. A point $p \in S$ belongs to a 3-sided query range $q \in Q$ if and only if $\phi(p)$ belongs to $\phi(q)$, that is, the mapping provides a realization of $\mathcal{W}$ as a 3-d dominance reporting problem. □

A similar reduction shows the following result.

Theorem 5.10. *Let $f(\cdot,\cdot)$ be a monotonically increasing function, and $0 < \delta \leq 1/2$ a constant. Any (partially) persistent cache-oblivious B-tree capable of answering 1-d range reporting queries on any previous version of the tree using at most $f(\log_B N, K/B)$ block transfers in the worst case, for every block size $B \leq N^{2\delta}$, must use $\Omega(N(\log\log N)^{\varepsilon})$ space to represent a sequence of N update operations, where $\varepsilon = 1/f(\delta^{-1}, 1)$.*

Proof. Again, we need to show that the workload $\mathcal{W} = (S, Q, \mathcal{B})$ from Section 5.2 is realizable in the framework of 1-d range reporting queries on persistent B-trees. We say that $\mathcal{W}$ is realizable as a persistent 1-d range reporting problem if there exists a sequence of insertions and deletions of the elements in S and a mapping of the elements in S to points on the real line such that, for every query $q \in Q$, there exists an interval $[l, r]$ and an integer t with the property that a 1-d range reporting query with interval $[l, r]$ after the tth operation in the update sequence reports exactly the points in q.

To obtain such a realization of the workload $\mathcal{W}$, we map each point $p = (x_p, y_p)$ in S to the point x_p on the real line and define an update sequence that inserts the points in S by decreasing y-coordinates. For a query $q = [l, r] \times [b, +\infty)$ in Q, let t be the number of points $p \in S$ with $y_p \geq b$. Then a 1-d range query with interval $[l, r]$ after the tth insertion reports exactly the points in q. Thus, $\mathcal{W}$ is realizable as a persistent 1-d range reporting problem. □

5.4.2 Halfspace Range Reporting in Three Dimensions

In this section, we prove the same lower bound as in Theorem 5.1 for 3-d halfspace range reporting.

Theorem 5.11. *Let $f(\cdot,\cdot)$ be a monotonically increasing function, and $0 < \delta \leq 1/2$ a constant. Any cache-oblivious data structure capable of answering 3-d halfspace range reporting queries using at most $f(\log_B N, K/B)$ block transfers in the worst case, for every block size $B \leq N^{2\delta}$, must use $\Omega(N(\log\log N)^{\varepsilon})$ space, where $\varepsilon = 1/f(\delta^{-1}, 1)$.*

As in the proofs of Theorems 5.9 and 5.10, we need to show that the workload $\mathcal{W} = (S, Q, \mathcal{B})$ from Section 5.2 is realizable as a 3-d halfspace range reporting problem. This requires more care than for 3-d dominance reporting and persistent 1-d range reporting. We begin with a reduction from 2-d parabolic range reporting to 3-d halfspace range reporting (see, e.g., [7]). A parabolic range reporting query is defined by a parabola $y = a(x - b)^2 + c$, for $a \geq 0$, and asks to report all points $p = (x_p, y_p)$ in S that satisfy $y_p \geq a(x_p - b)^2 + c$. The reduction from [7] maps each point p in $\mathbb{R}^2$ to a point $\psi(p)$ in $\mathbb{R}^3$ and each parabola q to a halfspace $\psi(q)$ such that $p \in q$ if and only if $\psi(p) \in \psi(q)$. Thus, every workload realizable as a 2-d parabolic range reporting problem is also realizable as a 3-d halfspace range reporting problem. In particular, it suffices to show that $\mathcal{W}$ is realizable as a 2-d parabolic range reporting problem. We do this by distorting the point set S so that there exists a parabolic query range $\phi(q)$ for every 3-sided query $q \in Q$ with the property that q and $\phi(q)$ contain the same points in S.

We follow the recursive construction of S and Q from Section 5.2 and, for every subgrid T' in the construction, replace the 3-sided queries in $Q_{T'}$ with parabolic ones as shown in Figure 5.3(a). If we embed the subgrids of T' inside the white squares in the figure, each query outputs the same set of points in T' as its corresponding 3-sided query. However, we also have to ensure that the queries in $Q_{T'}$ output only points from T'. For 3-sided queries, we achieved this by ensuring that there are no points in the x-range of T' but outside its y-range. As shown in Figure 5.3(b), this is not sufficient for parabolic queries, as parabolic queries over T' cannot be confined to the x-range of T'. Instead, we use the following construction, which extends a construction from [7].

Given a grid cell and a subgrid T' to be embedded inside this cell, we use G to denote the portion of the cell where the points in T' are to be placed (the white squares in Figure 5.3(a)). We call G the *grid box* of T'. Every parabolic query over T' is guaranteed to output only points from T' if it leaves the x-range of G only above the top boundary of the top-level grid T. We represent this using a *column box* C that shares its left, right, and bottom boundaries with G and whose top boundary is the top boundary of T; see Figure 5.3(c). The parabolic queries over T' must intersect only the top boundary edge of C. Once we have obtained an embedding of T' inside G and a set of parabolic queries over T' that output the same set of points as the 3-sided queries in $Q_{T'}$ and intersect only the top boundary of C, we can define grid and column boxes for the subgrids of T' as above and apply this construction recursively. Thus, to prove that $\mathcal{W}$ is realizable as a 2-d parabolic range reporting problem, it suffices to prove that, given *any* grid box G and *any* column box $C \supseteq G$, we can embed T' inside G and construct appropriate parabolic queries over T' that intersect only the top boundary of C.

Lemma 5.12. *Given a column box C and a grid box $G \subseteq C$, a subgrid T' can be embedded*

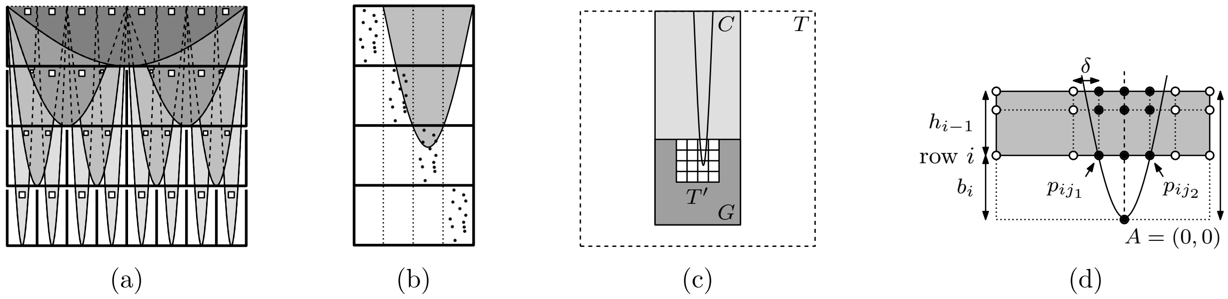


Figure 5.3: (a) Replacing 3-sided queries with parabolic ones. The white squares are the areas where the subgrids in each grid cell are to be placed. (b) A naively constructed query in the subgrid in cell $T_{3,6}$ also reports points in other cells (e.g., $T_{2,6}$). (c) Placement of a subgrid within a grid box G that is nested inside a column box C. (d) Incremental embedding of a subgrid T' inside a grid box.

inside G so that, for every 3-sided range reporting query $q \in Q_{T'}$, there exists a parabolic range reporting query q' that reports the same set of points in T' as q and intersects only the top boundary of C.

Proof. For the sake of this proof, we can assume that each grid cell of T' contains exactly one point, as we can embed the subgrid represented by each such point in a sufficiently small neighbourhood of the point without altering the properties of the construction. We can further assume that the bottom boundaries of C and G coincide, that C is twice as wide as G, and that G is horizontally centred inside C, as this can be enforced by shrinking G and C appropriately without relaxing the constraints placed by these two boxes on the embedding of T' and on the queries constructed over T'. We denote the widths and heights of G and C by w_G, w_C, h_G, and h_C, respectively.

We construct T' row by row, placing the points in each row at the same y-coordinate and spacing them evenly in x-direction. The top row of T' coincides with the top boundary of G, and we centre T' horizontally in G. The points in each row have distance $\delta > 0$ between every pair of consecutive points. After placing the points in the ith row, we construct parabolic queries for all queries at level i in $Q_{T'}$ that output exactly those points in rows 1 through i contained in their 3-sided counterparts in $Q_{T'}$. The next row of points is then placed infinitesimally below the horizontal line through the apexes of these level-i queries. This is illustrated in Figure 5.3(d).

Once we have placed the points of all rows of T' in this fashion, we obtain an embedding of T' inside a box of width $\delta(2^{t-1} - 1)$ and height h_t. We derive a bound on h_t as a function of δ. By choosing δ small enough, we can ensure that the constructed box (and, hence, T') is completely contained in G.

To discuss this construction in detail, consider the construction of the ith row. Let B_{i-1} be the smallest box containing all points already placed in rows 1 and $i-1$ and such that its bottom boundary passes through the apexes of all level-$(i-1)$ queries we have just constructed. The width of B_{i-1} is $\delta(2^{t-1}-1)$, and we denote its height by h_{i-1}. For $i=1$, we assume that $h_0 = 0$, that is, that B_0 is a line segment contained in the top boundary of G. To construct the ith row of T', we evenly distribute 2^{t-1} points along a horizontal line segment of the same width as B_{i-1} and infinitesimally below the bottom boundary of B_{i-1}. This ensures that no point in the ith row is contained in any query at a level less than i. To construct the queries at the ith level, we observe that each such query q' has to output points $p_{i'j'}$ with $1 \le i' \le i$ and $(k'-1)2^{t-i} < j' \le k'2^{t-i}$, for some $0 < k' \le 2^{i-1}$, where p_{ij} denotes the jth point in the ith row. For ease of notation, let $j_1 = (k'-1)2^{t-i}+1$ and $j_2 = k'2^{t-i}$. We construct the parabola q' so that it passes through points p_{ij_1} and p_{ij_2} and such that points p_{1,j_1-1} and p_{1,j_2+1}, as well as the two top corners of C, lie below q'. This ensures that q' outputs exactly the desired set of points and intersects only the top boundary edge of C.

To obtain a bound on the height h_t of the final box B_t, we first derive a bound on the distance b_i between the apex A of q' and the ith row of points, for all $1 \le i \le t$, and then use the recurrence $h_i > h_{i-1} + b_i$. To ease the exposition, we assume that the apex A of q' is at the origin, so that q' is given by an equation of the form $y = a_i x^2$. Let $d_i = \delta(2^{t-i}-1)/2$ be half the distance between points p_{ij_1} and p_{ij_2}. Since A has distance b_i from the ith row of points and q' passes through points p_{ij_1} and p_{ij_2}, q' must satisfy

$$b_i = a_i d_i^2. \tag{5.13}$$

In order for points p_{1,j_1-1} and p_{1,j_2+1} to lie below q', q' must satisfy

$$h_{i-1} + b_i < a_i(d_i + \delta)^2, \tag{5.14}$$

as the distance of A from the first row is infinitesimally greater than $h_{i-1}+b_i$ and the distance between two neighbouring columns of T' is δ. Finally, if we can satisfy (5.13) and (5.14) while placing A inside G, then q' intersects only the top boundary edge of C if

$$h_C < a_i \frac{w_G^2}{4}, \tag{5.15}$$

as the height of C is h_C and the distance between G and either of the two vertical boundary edges of C is $w_G/2$.

By substituting (5.13) into (5.14) and rearranging the result, we obtain that (5.14) holds if

$$a_i > \frac{h_{i-1}}{2d_i\delta + \delta^2}. \tag{5.16}$$

Inequalities (5.15) and (5.16) are both satisfied if

$$a_i > \frac{h_{i-1}}{2d_i\delta + \delta^2} + \frac{4h_C}{w_G^2}, \tag{5.17}$$

which gives

$$b_i > \left(\frac{h_{i-1}}{2d_i\delta + \delta^2} + \frac{4h_C}{w_G^2}\right) d_i^2 \tag{5.18}$$

by substituting (5.17) into (5.13). By using the equation $d_i = \delta(2^{t-i} - 1)/2$ and simplifying appropriately, we obtain that (5.18) holds if

$$b_i > 2^{t-i-2}h_{i-1} + \frac{4^{t-i}\delta^2 h_C}{w_G^2}. \tag{5.19}$$

By substituting this into the recurrence $h_i > h_{i-1} + b_i$, we obtain that we can satisfy (5.19) as long as

$$h_i > h_{i-1}(1 + 2^{t-i-2}) + \frac{4^{t-i}\delta^2 h_C}{w_G^2},$$

which holds if we set $h_0 = 0$ and $h_i = 4^{it}\delta^2 h_C/w_G^2$ and as long as $t \geq 2$. Thus, by choosing δ no greater than $\min\left(w_G/2^t, \sqrt{h_G w_G^2/(4^{t^2}h_C)}\right)$, we can ensure that all points of T' lie inside G, and that all parabolas in $Q_{T'}$ have their apexes inside G, output the same set of points as their corresponding 3-sided queries, and intersect only the top boundary edge of C. This completes the proof. $\square$

5.4.3 Las-Vegas-Type Data Structures

The lower bounds we have proved so far apply to data structures with deterministic construction algorithms and worst-case query bounds. In many cases, however, it is significantly easier to obtain efficient randomized data structures of the Las Vegas kind, that is, with expected query bounds resulting from randomness in their construction and in the query procedure (see, e.g., [46]). Therefore, we obtain a much stronger statement of the difficulty of cache-oblivious range reporting if we can show that the lower bounds shown in the previous sections apply also to this type of data structure.

Formally, we consider data structures whose construction algorithm and query procedure both have access to a sequence of random bits. The random bits used during the construction influence the shape of the data structure, while the random bits used by the query procedure influence which blocks are read to answer a given query. In a Las-Vegas-type data structure, the random bits may influence the costs of individual queries, but the answer provided by a query must always be correct. The following theorem extends the lower bounds we have proved for deterministic data structures with worst-case query bounds to randomized data structures of the Las Vegas kind.

Theorem 5.20. *Let $f(\cdot,\cdot)$ be a monotonically increasing function, and $0 < \delta \leq 1/2$ a constant. Any cache-oblivious data structure for 3-sided range reporting, 3-d dominance reporting or 3-d halfspace range reporting constructed by a randomized algorithm and capable of answering queries using at most $f(\log_B N, K/B)$ block transfers in the expected sense, for every block size $B \leq N^{2\delta}$, must use expected $\Omega(N(\log\log N)^{\varepsilon})$ space, where $\varepsilon = 1/(4f(\delta^{-1},1))$.*

A cache-oblivious persistent B-tree that supports 1-d range reporting queries on any previous version of the tree using at most $f(\log_B N, K/B)$ block transfers in the expected sense, for every block size $B \leq N^{2\delta}$, must use expected $\Omega(N(\log\log N)^{\varepsilon})$ space to represent a sequence of N update operations.

To prove Theorem 5.20, it suffices to prove it for 3-sided range reporting and $\delta = 1/2$. As before, Lemma 5.3 then extends the result to smaller values of δ, and the reductions in Sections 5.4.1 and 5.4.2 extend the lower bound to 3-d dominance reporting, 3-d halfspace range reporting, and cache-oblivious persistent B-trees.

Again, we formulate our proof in the multi-level indexability model from Section 5.1. To do so, we need to extend the model to allow for randomness in the construction of the indexing scheme. The randomness in the query procedure can be eliminated using the following argument. A randomized query procedure for an indexing scheme $\mathcal{I}$ would make random choices in selecting the blocks to cover a given query q. The efficiency of an indexing scheme, however, is defined based on the *existence* of a small set of blocks to cover each query $q \in Q$. This is equivalent to a query procedure that deterministically selects the minimum number of blocks to cover each query in Q, and randomness cannot improve on this query cost.

As a model of randomness in the construction of an indexing scheme for a workload $\mathcal{W} = (S, Q, \mathcal{B})$, we define the concepts of a randomized block cover and of a randomized indexing scheme. If the construction of an indexing scheme uses b random bits, it is capable of constructing $n := 2^b$ different indexing schemes $\mathcal{I}^1, \mathcal{I}^2, \ldots, \mathcal{I}^n$, depending on the value of these bits. For a block size $B_i \in \mathcal{B}$, let $\mathcal{C}_i^k$ be the B_i-cover in $\mathcal{I}^k$. Then we call the set $\mathcal{C}_i = \{\mathcal{C}_i^1, \mathcal{C}_i^2, \ldots, \mathcal{C}_i^n\}$ a *randomized B_i-cover* and the set $\mathcal{I} = \{\mathcal{I}^1, \mathcal{I}^2, \ldots, \mathcal{I}^n\}$ a *randomized indexing scheme* for $\mathcal{W}$. The expected size of $\mathcal{I}$ is the average size of $\mathcal{I}^1, \mathcal{I}^2, \ldots, \mathcal{I}^n$. We say the randomized B_i-cover $\mathcal{C}_i$ in $\mathcal{I}$ is f-efficient if the average number of blocks needed to cover each query in Q is at most $f(\log_{B_i} N, K/B_i)$, where the average is taken over all block covers $\mathcal{C}_i^1, \mathcal{C}_i^2, \ldots, \mathcal{C}_i^n$ in $\mathcal{C}_i$. The randomized indexing scheme $\mathcal{I}$ is f-efficient if all its randomized block covers are f-efficient. As such, the efficiency of a randomized indexing scheme and its expected size capture the expected query cost and the expected size of a cache-oblivious data structure, assuming that each of the n data structures that can be constructed using b random bits is equally likely.

To prove Theorem 5.20, we show now that an f-efficient randomized indexing scheme $\mathcal{I} = \{\mathcal{I}^1, \mathcal{I}^2, \ldots, \mathcal{I}^n\}$ for the workload $\mathcal{W} = (S, Q, \mathcal{B})$ from Section 5.2 has expected size $\Omega(N(\log\log N)^\varepsilon)$. To this end, we represent the construction of the point set S using a tree $\mathcal{T}$. The root of $\mathcal{T}$ is the top-level grid T, and its internal nodes represent the subgrids constructed recursively. A grid T'' is a child of a grid T' in $\mathcal{T}$ if T'' is the subgrid constructed in one of the cells of T'. Now we make n copies $\mathcal{T}^2, \mathcal{T}^2, \ldots, \mathcal{T}^n$ of $\mathcal{T}$ and associate the copy $\mathcal{T}^k$ with the indexing scheme $\mathcal{I}^k$, for all $1 \le k \le n$. We refer to the copy of a node $T' \in \mathcal{T}$ in $\mathcal{T}^k$ as T'_k, and we say that a point p in T'_k has multiplicity m if its multiplicity in $\mathcal{I}^k$ is m.

To prove the desired lower bound on the expected size of $\mathcal{I}$, we mimic the proof of Lemma 5.2. To prove Lemma 5.2, we first showed that every grid $T' \in \mathcal{T}$ has a cell at least half of whose points have multiplicity $\Omega(t^\varepsilon)$ in an f-efficient (deterministic) indexing scheme $\mathcal{I}$. By Lemma 5.4, this implied Lemma 5.2. This proof was based on the fact that every query $q \in Q_{T'}$ can be covered with at most α blocks in the $B_{T'}$-cover in $\mathcal{I}$. In a randomized indexing scheme $\{\mathcal{I}^1, \mathcal{I}^2, \ldots, \mathcal{I}^n\}$, there may be no indexing scheme $\mathcal{I}^k$ whose $B_{T'}$-cover can cover every query in $Q_{T'}$ with at most α blocks. Nevertheless, we can prove that at least $n/4$ of the copies $T'_1, T'_2, \ldots, T'_n$ of each grid $T' \in \mathcal{T}$ have a cell at least half of whose points have multiplicity $\Omega(t^\varepsilon)$ (Lemma 5.21 below). This suffices to show that the total size of the indexing schemes $\mathcal{I}^1, \mathcal{I}^2, \ldots, \mathcal{I}^n$ is $\Omega(nN(\log\log N)^\varepsilon)$, that is, their average size is $\Omega(N(\log\log N)^\varepsilon)$.

Lemma 5.21. *Every node $T' \in \mathcal{T}$ has at least $n/4$ copies among $T'_1, T'_2, \ldots, T'_n$ such that* *each has a cell at least half of whose points have multiplicity $\Omega(t^\varepsilon)$.*

Proof. Consider the B_i-covers $\mathcal{C}_i^1, \mathcal{C}_i^2, \ldots, \mathcal{C}_i^n$, where $B_i = B_{T'}$ is the block size in $\mathcal{B}$ corresponding to the grid T'. We consider a query $q \in Q_{T'}$ *cheap* for $\mathcal{C}_i^k$ if q can be covered using at most $4f(2,1)$ blocks in $\mathcal{C}_i^k$, and *expensive* otherwise. Since the randomized B_i-cover $\mathcal{C}_i$ is f-efficient, q must be cheap for at least $3n/4$ of the B_i-covers $\mathcal{C}_i^1, \mathcal{C}_i^2, \ldots, \mathcal{C}_i^n$. The jth row of T'_k is *cheap* if at least half of the level-j queries in $Q_{T'}$ are cheap for $\mathcal{C}_i^k$; otherwise the row is *expensive*. As each query is cheap for at least $3n/4$ of the B_i-covers $\mathcal{C}_i^1, \mathcal{C}_i^2, \ldots, \mathcal{C}_i^n$, every row of T' is cheap in at least $n/2$ of the copies $T'_1, T'_2, \ldots, T'_n$. Thus, the total number of cheap rows over all copies of T' is at least $tn/2$, which implies that there are at least $n/4$ copies of T' that have at least $t/3$ cheap rows each. We show that each such copy T'_k has a cell at least half of whose points have multiplicity $\Omega(t^\varepsilon)$.

The block cover $\mathcal{C}_i^k$ defines a colouring of the points in T'_k and of the queries in $Q_{T'}$ as in Section 5.2.2. For a subset Z of cells of T'_k, we call a point p in a cell c in row j of T'_k *weakly Z-exposed* if no cheap query $q \in Q_Z$ that belongs to a level less than j in $Q_{T'_k}$ covers p, that is, if it is Z-exposed in the sense defined in Section 5.2.2. Point p is *strongly Z-exposed* if it is weakly Z-exposed and the query q_c is cheap for $\mathcal{C}_i^k$. As in Section 5.2.2, we call a cell of T'_k weakly or strongly Z-exposed if at least half its points are weakly or strongly Z-exposed. Now it suffices to show that T'_k contains a set $Z := \{c_1, c_2, \ldots, c_s\}$ of cells in the same column whose length is $s = \Omega(t/\log\alpha)$ and all of whose cells are strongly Z-exposed, for $\alpha := 4f(2,1)$. Since each query q_{c_i} is α-coloured in this case, Lemma 5.6 then shows that the Z-exposed points in at least one of these cells have at least $s^\varepsilon = \Omega(t^\varepsilon)$ colours. This finishes the proof because at least half the points in a Z-exposed cell are Z-exposed.

Let T^c be the subgrid of T'_k consisting of only the cheap rows of T'_k, and, similarly to the proof of Lemma 5.5, let T^c_h be the subgrid of T^c consisting of rows $1, h+1, 2h+1, \ldots$ of T^c. Using the same arguments as in the proof of Lemma 5.5, at most a $(4\alpha/2^h)$-fraction of the points in each row of T^c_h can be covered by cheap queries at higher levels of T^c_h. Thus, for $h = \lceil \log(32\alpha) \rceil$, at least a $7/8$-fraction of the points in each row of T^c_h are weakly T^c_h-exposed. Since a cell is weakly T^c_h-exposed if at least half of its points are weakly T^c_h-exposed, this implies that at least a $3/4$-fraction of the cells in each row of T^c_h are weakly T^c_h-exposed. It follows that at least a $1/4$-fraction of the cells in each row are strongly T^c_h-exposed because every row of T^c_h is cheap. This, however, implies that there exists a column of T^c_h such that at least a $1/4$-fraction of its cells are strongly T^c_h-exposed. Since the height of T^c_h is at least

$t/(3h) = \Omega(t/\log\alpha)$, this shows that T'_k contains a set Z of $\Omega(t/\log\alpha)$ strongly T^c_h-exposed cells in the same column. These cells are also strongly Z-exposed because $Z \subseteq T^c_h$. $\square$

Using Lemma 5.21, we can now prove that the average size of the indexing schemes $\mathcal{I}^1, \mathcal{I}^2, \ldots, \mathcal{I}^n$—that is, the expected size of $\mathcal{I}$—is $\Omega(N(\log\log N)^\varepsilon)$. To this end, we call a node $T'_k \in \mathcal{T}^k$ *accounted for* if there exists an ancestor T''_k of T'_k in $\mathcal{T}^k$ for which we can guarantee that at least half the points in T''_k have multiplicity $\Omega(t^\varepsilon)$. We call a point accounted for if it is contained in a grid T'_k that is accounted for. Our goal now is to show that there exists a level i in $\mathcal{T}$ such that at least a constant fraction of the points in all level-i nodes of trees $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^n$ are accounted for. Since at least half of the accounted-for points have multiplicity $\Omega(t^\varepsilon)$, this shows that the total size of the indexing schemes $\mathcal{I}^1, \mathcal{I}^2, \ldots, \mathcal{I}^n$ is $\Omega(nNt^\varepsilon)$, and their average size is $\Omega(Nt^\varepsilon) = \Omega(N(\log\log N)^\varepsilon)$, as claimed.

For a given level i in $\mathcal{T}$, we consider all nodes at level i in $\mathcal{T}$ that have more than $7n/8$ unaccounted-for copies in trees $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^n$. Let N_i be the total number of points in these nodes of $\mathcal{T}$. By Lemma 5.21, each such node T' has at least $n/8$ copies in trees $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^n$ that are unaccounted for and such that each has a child that is accounted for at level $i+1$. If T' contains N' points, then any child of T' contains at least $N'/4^t$ points, as argued in Section 5.2.1. Hence, at least $(n/8)\cdot(N_i/4^t)$ new points are accounted for at level $i+1$.

For $N_i > N/2$, this shows that at least $nN/(16\cdot 4^t) = nN/(16\sqrt{\log N})$ new points are accounted for at depth $i+1$. Since there are only nN points in total, this implies that there can be at most $16\sqrt{\log N}$ levels in $\mathcal{T}$ that satisfy $N_i > N/2$. However, we have shown in Section 5.2.1 that the height of $\mathcal{T}$ is at least $\log N/\log\log N$, which is greater than $16\sqrt{\log N}$, for N sufficiently large. Hence, there exists a level i in $\mathcal{T}$ with $N_i \le N/2$, and at least $N/2$ of the points in S are accounted for in at least $n/8$ of the trees $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^n$. Since at least half of the accounted for points have multiplicity $\Omega(t^\varepsilon)$, the total size of the indexing schemes $\mathcal{I}^1, \mathcal{I}^2, \ldots, \mathcal{I}^n$ is thus $\Omega(nNt^\varepsilon)$, as claimed. This concludes the proof of Theorem 5.20.

Chapter 6

Upper Bounds

In this chapter we develop a general framework for approximate range counting and range reporting in the cache-oblivious model. The framework is sufficiently general to be applicable to a wide variety of range searching problems, including 3-sided range reporting, 3-d halfspace range reporting, 3-d dominance reporting, circular range reporting, 2-d K-nearest neighbour searching, and 3-d orthogonal range reporting. Section 6.1 discusses the setting in which our framework is constructed and describes the family of admissible range searching problems.

Section 6.2 constructs a general cache-oblivious data structure for approximate range counting. This data structure use linear space and provides guaranteed $(1+\varepsilon)$-approximate answers using $\mathrm{O}(\log_B(N/K))$ block transfers in the worst case (where K is the reported count), which is optimal. This is in contrast to previous results even in internal memory, where the optimal query bound was not achieved in the worst case before, even using super-linear space. The only previous data structure with the optimal query bound, by Afshani and Chan [6], achieves this bound only in the expected case. Thus, our construction also provides new worst-case optimal data structures for approximate 3-d halfspace range counting and approximate 3-d dominance counting in the pointer machine model.[1]

Section 6.3 describes the first cache-oblivious data structures with the optimal query bound of $\mathrm{O}(\log_B N + K/B)$ block transfers for 3-d halfspace range reporting, 3-d dominance reporting and, as a consequence, for circular range reporting, 2-d K-nearest neighbour searching, and 3-d orthogonal range reporting. All our data structures, except the 3-d orthogonal range reporting structure, use $\mathrm{O}(N \log N)$ space. Using a standard transformation, our 3-d dominance reporting structure also provides a new $\mathrm{O}(N \log N)$-space data structure with the optimal query bound for 3-sided range reporting, thereby matching the previous results obtained in [13, 25, 30]. These results are fairly easy to obtain using standard constructions based on shallow cuttings once the output size of a query can be efficiently determined or at least approximated (using the results of Section 6.2). A full list of new results contrasted

[1] It is easy to verify that we use only operations available on a pointer machine equipped with the necessary algebraic operations to compute intersections of curves and determine the side of a curve a point is on.

with existing results is shown in Tables 3.1 and 3.2.

The main tool used in our data structures is that of shallow cuttings, which can be obtained for a general class of problems, albeit using a randomized construction [17]. The use of shallow cuttings in problems related to range searching is by now fairly standard; the novelty of our approach lies in the manner in which we combine them with other equally standard techniques to obtain the above series of new results.

All data structures presented in this chapter are static, and no efficient construction methods for these structures are known in the I/O or cache-oblivious model. The main obstacle is the lack of an I/O-efficient or cache-oblivious construction procedure for shallow cuttings. Even in internal memory, the running time of the general construction procedure of [17] is analyzed only for shallow cuttings of constant size. For 3-d halfspace arrangements, Ramos [108] gave a fairly complicated algorithm that constructs a shallow cutting in $\mathrm{O}(N \log N)$ time, but it is not clear whether an efficient cache-oblivious construction procedure can be obtained using the same ideas.

The results of this chapter have been published in [9, 10].

6.1 Family of Admissible Problems

The framework presented in this chapter is applicable to any range searching problem that, through application of duality or other techniques, can be translated into the following type of 'aboveness reporting problem' and satisfies a number of additional properties discussed below. Let $\mathcal{F}$ be a collection of continuous and totally defined algebraic functions $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ of constant degree. Each such function defines a continuous surface in $\mathbb{R}^3$ consisting of the points $(x, y, f(x, y))$, and we do not distinguish between a function and the surface it defines. We say that a function f *passes below* a point $q = (x_q, y_q, z_q)$ if $f(x_q, y_q) < z_q$. Our goal is to preprocess $\mathcal{F}$ so that, given any query point q, we can efficiently report or (approximately) count the functions in $\mathcal{F}$ passing below q. Since we assume that such an aboveness query is an alternative representation of a range query, we refer to reporting or counting the functions that pass below a query point as range reporting or range counting throughout this chapter.

The *arrangement* of a collection $\mathcal{F}$ of functions is a subdivision of $\mathbb{R}^3$ into *cells*; see Figure 6.1(a).[2] Each such cell is a maximal connected set of points that are contained in a

[2]For the sake of keeping the figures simple, all figures in this chapter illustrate 2-dimensional versions of the discussed 3-dimensional structures.

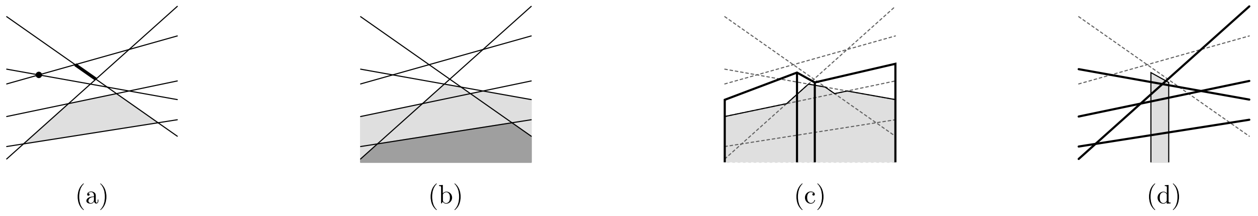


Figure 6.1: (a) An arrangement of lines. The shaded region, excluding its boundary is a 2-dimensional cell. The fat line segment excluding its endpoints is a 1-dimensional cell. The highlighted vertex is a 0-dimensional cell. (b) The lower envelope of the arrangement is shaded dark gray. The (≤ 2)-level includes all light gray and all dark gray faces. (c) The regions bounded by fat lines are a shallow cutting for the (≤ 2)-level of the arrangement. (d) The fat lines belong to the conflict list of the shaded cell.

subset $\mathcal{F}' \subseteq \mathcal{F}$ of functions and in no other functions in $\mathcal{F}$. We define the *level* of a point q to be the number of functions in $\mathcal{F}$ that pass below q. The *k-level* or *($\leq k$)-level* of $\mathcal{F}$ is the closure of the set of points in $\mathbb{R}^3$ at level k or at most k, respectively; see Figure 6.1(b). The 0-level of $\mathcal{F}$ is also known as the *lower envelope* of $\mathcal{F}$. Any k- or ($\leq k$)-level is a collection $\mathcal{C}$ of cells. Its *complexity* $|\mathcal{C}|$ is defined as the number of cells in $\mathcal{C}$.

A *shallow cutting for the ($\leq k$)-level* of $\mathcal{F}$ is a collection $\mathcal{C}$ of disjoint cells that cover the ($\leq k$)-level of $\mathcal{F}$ and have the property that every cell $C \in \mathcal{C}$ intersects $\mathrm{O}(k)$ functions in $\mathcal{F}$; see Figure 6.1(c). Without loss of generality, we can assume that every cell in $\mathcal{C}$ intersects the ($\leq k$)-level of $\mathcal{F}$ (otherwise, we can obtain a smaller shallow cutting for the ($\leq k$)-level by removing all cells from $\mathcal{C}$ that do not intersect this level). The *conflict list* Δ_C of a cell C is the set of functions in $\mathcal{F}$ that intersect C or pass below it; see Figure 6.1(d). By the above assumption, we have $|\Delta_C| = \mathrm{O}(k)$ and, for every point $p \in C$, all functions in $\mathcal{F}$ that pass below p are included in Δ_C.

For our approximate range counting framework to be applicable, the collection, $\mathcal{F}$, of functions has to satisfy the following properties:

(i) For every k, there exists a shallow cutting for the ($\leq k$)-level of $\mathcal{F}$ consisting of $\mathrm{O}(|\mathcal{F}|/k)$ cells, each bounded by a constant number of algebraic curves of constant degree.

(ii) For every shallow cutting $\mathcal{C}$ as in (i), there exists a cache-oblivious *point location structure* $\mathcal{L}(\mathcal{C})$. For any query point q, this data structure finds the cell C of $\mathcal{C}$ that contains q, or reports that no such cell exists. We require that this data structure uses

$O(|\mathcal{C}|)$ space and supports queries using $O(\log_B |\mathcal{C}|)$ block transfers.

(iii) Consider the 2-d arrangement $\mathcal{A}$ formed by projecting a number of shallow cuttings as in (i) into the xy-plane. Then there exists a cache-oblivious point location structure for $\mathcal{A}$ with a query bound of $O(\log_B |\mathcal{A}|)$ block transfers and with space complexity polynomial in $|\mathcal{A}|$.

By using results by Agarwal et al. [17], it is possible to replace (i) with a weaker condition that implies (i):

(i′) The lower envelope of every subset $\mathcal{F}' \subset \mathcal{F}$ has complexity $O(|\mathcal{F}'|)$.

For the problems we are interested in—3-d halfspace range searching and 3-d dominance searching—shallow cuttings satisfying condition (i) exist [2,94]. In addition, each cell of these shallow cuttings is a vertical prism bounded by $O(1)$ linear functions. Hence, the projection of such a shallow cutting $\mathcal{C}$ into the xy-plane is a planar straight-line subdivision of size $O(|\mathcal{C}|)$, and conditions (ii) and (iii) can both be satisfied using the cache-oblivious planar point location structure by Bender et al. [37]. We discuss this in more detail in Section 6.2.5.

In general, condition (i) does not hold as soon as we consider dimensions higher than three. Even in three dimensions, one can easily construct simple examples where this condition fails. For example, consider a collection of functions $\mathcal{F} = \{f_1, f_2, \ldots, f_N\}$, where $f_i(x, y) = (x - i)^2$ and $f_{i+N/2}(x, y) = (y - i)^2$, for all $1 \leq i \leq N/2$. These functions are algebraic of constant degree, but their lower envelope has quadratic size.

Condition (ii) requires a linear-space data structure that supports point location in a 3-d arrangement (the shallow cutting) in logarithmic time. In general, it is not known whether such a structure exists. It is fortunate that, for 3-d halfspace range searching and 3-d dominance searching, the shallow cuttings have a structure that allow us to reduce this problem to 2-d point location.

Condition (iii) is easy to satisfy for any set of algebraic functions of constant degree because a point location structure of size $O(N^3)$ and with the required query bound can be obtained for the arrangement of any set of N algebraic functions of constant degree in the plane using the slab method.

6.2 Approximate Range Counting

This section presents the main result of this chapter: a general framework for constructing a cache-oblivious approximate range counting structure for any range searching problem that satisfies the conditions discussed in Section 6.1. The following theorem states this precisely.

Theorem 6.1. *For a set $\mathcal{F}$ of N functions satisfying conditions (i)–(iii), there exists a cache-oblivious data structure that uses $\mathrm{O}(N(1+\varepsilon^{-2}\log\varepsilon^{-1}))$ space and supports $(1+\varepsilon)$-approximate range counting queries using $\mathrm{O}(\log_B(N/K)+(\varepsilon^{-2}/B)\log\varepsilon^{-1})$ block transfers in the worst case, where K is the actual value of the count.*

Throughout this chapter, we use q to refer to a particular query point, and K to denote the number of functions in $\mathcal{F}$ that pass below q. Our goal is to compute a number K' that satisfies $K \leq K' \leq (1+\varepsilon)K$. The first step towards proving Theorem 6.1 is to show that the difficult part of the problem is to obtain *any* constant-factor approximation of K.

Lemma 6.2. *Consider a set $\mathcal{F}$ of N functions satisfying conditions (i)–(iii) and assume there exists a linear-space cache-oblivious data structure $\mathcal{D}$ that supports c-approximate range counting queries over $\mathcal{F}$ using $\mathrm{O}(\log_B(N/K))$ block transfers, where c is an arbitrary constant and K is the actual value of the count. Then there exists a cache-oblivious data structure that uses $\mathrm{O}(N(1+\varepsilon^{-2}\log\varepsilon^{-1}))$ space and supports $(1+\varepsilon)$-approximate range counting queries using $\mathrm{O}(\log_B(N/K)+(\varepsilon^{-2}/B)\log\varepsilon^{-1})$ block transfers.*

Proof. The $(1+\varepsilon)$-approximate range counting structure we construct for $\mathcal{F}$ consists of $\mathcal{D}$ and data structures representing shallow cuttings $\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_n$, where $n = \lceil \log N \rceil$ and $\mathcal{C}_i$ is a shallow cutting for the $(\leq 2^i)$-level of $\mathcal{F}$. The data structure representing each shallow cutting $\mathcal{C}_i$ consists of the point location structure $\mathcal{L}(\mathcal{C}_i)$ of $\mathcal{C}_i$ and a collection of δ-*approximate conflict lists* for the cells of $\mathcal{C}_i$, where $\delta > 0$ is a constant defined below. The δ-approximate conflict list $\tilde{\Delta}_C$ of a cell $C \in \mathcal{C}_i$ is a sublist of Δ_C such that the level of a query point $q \in C$ can be approximated to within an additive error of $\delta\,|\Delta_C|$ using only the level of q in $\tilde{\Delta}_C$. Section 6.4 discusses how to apply results in VC-dimension to obtain such a sublist $\tilde{\Delta}_C$ of Δ_C of size $\mathrm{O}(\delta^{-2}\log\delta^{-1})$ for each cell $C \in \mathcal{C}_i$.

Now consider a query point q. The query procedure is illustrated in Figure 6.2. We use $\mathcal{D}$ to obtain a c-approximation K' of K. The shallow cutting $\mathcal{C}_i$ with $i = \lceil \log K' \rceil$ contains q because $K' \geq K$. We use $\mathcal{L}(\mathcal{C}_i)$ to find the cell $C \in \mathcal{C}_i$ that contains q. Next we

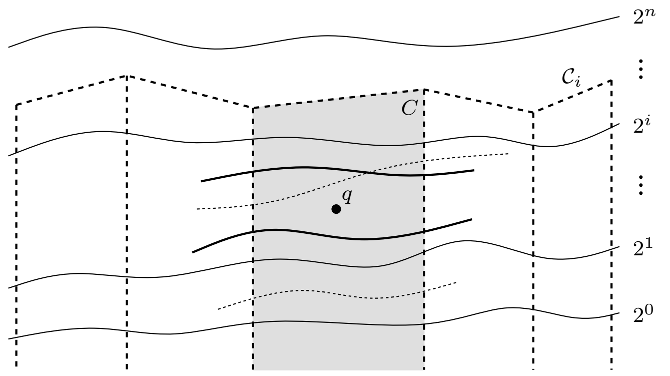


Figure 6.2: Illustration of the query procedure in the proof of Lemma 6.2. Thin solid lines show the top boundaries of some ($\leq 2^j$)-levels. The shallow cutting $\mathcal{C}_i$ with $i = \lceil \log K' \rceil$ is shown using fat dashed lines. The approximate conflict list $\tilde{\Delta}_C$ of the cell $C \in \mathcal{C}_i$ containing the query point q contains the fat solid curves crossing C but not the thin ones, solid or dashed.

compute the level of q in $\tilde{\Delta}_C$ and use it to compute an approximation K'' of K that satisfies $K \leq K'' \leq K + \delta\,|\Delta_C|$. Since $K' \leq cK$ and $|\Delta_C| \leq c'K'$, for some constant c', K'' is a $(1+\varepsilon)$-approximation of K if we choose $\delta = \varepsilon/(cc') = \Theta(\varepsilon)$.

The size of the data structure is $\mathrm{O}(N(1 + \delta^{-2}\log\delta^{-1})) = \mathrm{O}(N(1 + \varepsilon^{-2}\log\varepsilon^{-1}))$. By the assumptions in the lemma, the c-approximate counting structure $\mathcal{D}$ uses linear space. By condition (i), each shallow cutting $\mathcal{C}_i$ has size $\mathrm{O}(N/2^i)$; that is, the total size of all shallow cuttings $\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_n$ is $\mathrm{O}(N)$. By condition (ii), this implies that the total size of the point location structures $\mathcal{L}(\mathcal{C}_0), \mathcal{L}(\mathcal{C}_1), \ldots, \mathcal{L}(\mathcal{C}_n)$ is $\mathrm{O}(N)$. Each cell in a shallow cutting $\mathcal{C}_i$ has an associated approximate conflict list of size $\mathrm{O}(\delta^{-2}\log\delta^{-1})$. As the total number of cells in $\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_n$ is $\mathrm{O}(N)$, the total space used to store all approximate conflict lists is $\mathrm{O}(N\delta^{-2}\log\delta^{-1})$. By summing the sizes of the different parts of the data structure, we obtain the claimed space bound.

The query cost is $\mathrm{O}(\log_B(N/K) + (\delta^{-2}/B)\log\delta^{-1}) = \mathrm{O}(\log_B(N/K) + (\varepsilon^{-2}/B)\log\varepsilon^{-1})$ block transfers: querying $\mathcal{D}$ takes $\mathrm{O}(\log_B(N/K))$ block transfers, as does querying $\mathcal{L}(\mathcal{C}_i)$, by condition (ii) and because $|\mathcal{C}_i| = \mathrm{O}(N/K)$; given the cell $C \in \mathcal{C}_i$ that contains q, scanning $\tilde{\Delta}_C$ to compute the level of q in $\tilde{\Delta}_C$ takes $\mathrm{O}((\delta^{-2}/B)\log\delta^{-1})$ block transfers because $\left|\tilde{\Delta}_C\right| = \mathrm{O}(\delta^{-2}\log\delta^{-1})$.

□

By Lemma 6.2, it suffices to construct a range counting structure for $\mathcal{F}$ that approximates K to within any constant factor. We split the construction of such a structure into three parts. We say that a query q is *polynomial* or *polylogarithmic* if $K \geq N^{\alpha}$ or $K \geq \log^{\tau} N$, respectively, for appropriate constants $\alpha > 0$ and $\tau > 0$. The first step is to obtain a data structure for polynomial queries that uses *sublinear* space and achieves the query bound stated in Lemma 6.2. Using this structure, we can obtain a data structure for polylogarithmic queries. By applying this construction a second time, the structure for polylogarithmic queries can be made to support arbitrary approximate range counting queries and, thus, can be used as the data structure $\mathcal{D}$ in Lemma 6.2.

6.2.1 Overview

Shallow cuttings provide a natural framework for approximate range counting. In order to determine a relative 2-approximation to the size of a query we need only find i such that the query q is a member of the $(\leq 2^i)$-level, but not a member of the $(\leq 2^{i-1})$-level. Then by definition, we have that $2^{i-1} < K \leq 2^i$ and $K' = 2^i$ is the 2-approximation we desire. We do not even require the full conflict lists of each cell in the shallow cutting, rather we need only maintain a description of the top surface of each of the $O(\log N)$ $(\leq 2^i)$-levels, for $0 \leq i \leq \lceil \log N \rceil$. Such a structure will be linear in size, but a binary search through the $O(\log N)$ levels will require $O(\log \log N)$ tests, with a test at level i costing $O(\log(N/2^i))$ comparisons. In order to improve on this simple bound we need to find ways to test against multiple levels simultaneously.

Previous approaches have used a variety of complicated overlay lemmas that are specific to individual problem types [6,80,81]. The fact that the overlay lemma is a crucial component of these approaches has a number of limiting implications: the method does not generalize to other problems, unless a similar overlay lemma is proved for each such problem; a non-trivial modification of the overlay lemma would be required to use it in models such as the I/O model or the cache-oblivious model; and, finally, it is inherently non-deterministic and cannot be used to obtain a worst-case query bound.

Our approach is unique in that it considers the problem at various independent output sizes: polynomial, polylogarithmic, and smaller than polylogarithmic. Specifically, by considering polynomial-sized queries we are able to find a *sublinear*-space data structure using simple overlay techniques. This data structure can then be used to bootstrap data structures for the remaining query sizes, a technique that had not been tried before.

Our structure for polynomial queries only considers queries such that $K \geq N^{1-\delta}/2^c$. By doing this, we are able to ignore the bottom $O((1-\delta)\log N)$ shallow cuttings, an important space savings. Next we use a simple overlay technique to combine the top $O(2^i)$ levels. This allows us to test against $O(2^i)$ levels at once using a single point location in the overlay, followed by a linear scan. If the procedure fails for the overlay of size $O(2^i)$, we continue on to the more detailed and larger overlay of size $O(2^{i+1})$ and continue the search. The nature of the search is such that it terminates sooner for larger K, leading to a natural query bound of $O(\log(N/K))$. Judiciously choosing the parameter δ allows us to limit the entire data structure to sublinear space.

The sublinear-space data structure is then used as the basic building block in a recursive data structure that can handle polylogarithmic queries, where $K > \log^\tau N$. Carefully choosing τ limits the recursion depth and keeps the data structure to linear space.

Finally, queries that fail on the polylogarithmic structure are sufficiently small that we can treat them using a brute force approach with another linear sized data structure. The next three subsections discuss these three parts of our construction in detail.

6.2.2 A Structure for Polynomial Queries

In this section, we prove that we can achieve the desired query bound for polynomial queries using *sublinear* space, as stated in the next lemma. The sublinear space bound is crucial to ensure that our data structure for polylogarithmic queries, discussed in Section 6.2.3, uses linear space.

Throughout the next two sections, we fix c to be an integer constant such that the conflict list of any cell C in a shallow cutting for the $(\leq k)$-level of $\mathcal{F}$ has size less than $2^c k$. By the definition of a shallow cutting, such a constant exists.

Lemma 6.3. *For a set $\mathcal{F}$ of N functions satisfying conditions (i)–(iii), there exists a cache-oblivious data structure that uses $O\left(\sqrt{N}\right)$ space and supports 2^{c+1}-approximate range counting queries using $O(\log_B(N/K))$ block transfers in the worst case, as long as the count K satisfies $K \geq N^{1-\delta}/2^c$, for a sufficiently small constant $\delta > 0$.*

Data structure To obtain a data structure as in Lemma 6.3, we choose a constant $\delta > 0$ to be defined later and construct a shallow cutting $\mathcal{C}_j$ for the $(\leq N/2^j)$-level of $\mathcal{F}$, for each $0 \leq j \leq 2(c + \delta \log N)$. Let $\mathcal{A}_j$ be the 2-d arrangement obtained by projecting the cells of $\mathcal{C}_j$ into the xy-plane. Next we construct arrangements $\mathcal{A}_0^*, \mathcal{A}_1^*, \ldots, \mathcal{A}_n^*$, where

$n = \lfloor \log(2\delta \log N) \rfloor$ and $\mathcal{A}_i^*$ is obtained by overlaying arrangements $\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_{2c+2^i}$. We represent each arrangement $\mathcal{A}_i^*$ using a point location structure as in condition (iii) and store for each face $f \in \mathcal{A}_i^*$, the list of cells of the shallow cuttings $\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_{2c+2^i}$ that project onto f; if more than one cell of a shallow cutting $\mathcal{C}_j$ projects onto f, we store only the highest one. These representations of arrangements $\mathcal{A}_0^*, \mathcal{A}_1^*, \ldots, \mathcal{A}_n^*$ are stored consecutively in memory, in order of increasing indices.

Space bound To bound the space used by this data structure, we observe that $|\mathcal{C}_j| = \mathrm{O}(2^j)$ and, hence, that the total number of cells in the shallow cuttings $\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_{2c+2^i}$ is $\mathrm{O}\left(2^{2^i}\right)$. This implies that $\mathcal{A}_i^*$ has size polynomial in 2^{2^i} because, by condition (i), each cell in each shallow cutting $\mathcal{C}_j$ is bounded by a constant number of algebraic curves of constant degree. In particular, the size of the largest arrangement $\mathcal{A}_n^*$—and, thus, the total size of all arrangements $\mathcal{A}_0^*, \mathcal{A}_1^*, \ldots, \mathcal{A}_n^*$— is polynomial in $2^{2^n} \leq N^{2\delta}$. Since each face of an arrangement $\mathcal{A}_i^*$ stores a list of $2c + 1 + 2^i = \mathrm{O}(\log N)$ cells and, by condition (iii), the size of the point location structure for each arrangement $\mathcal{A}_i^*$ is polynomial in $|\mathcal{A}_i^*|$, this implies that the entire data structure uses space polynomial in $N^{2\delta}$. Thus, by choosing a sufficiently small $\delta > 0$, we can ensure that the data structure uses $\mathrm{O}\left(\sqrt{N}\right)$ space.

Query procedure To answer a query q, note that it suffices to find an index j such that $\mathcal{C}_j$ contains q but $\mathcal{C}_{j+1}$ does not: the former condition implies that $K \leq 2^c N/2^j$, because q is contained in a cell of $\mathcal{C}_j$; the latter condition implies that $K > N/2^{j+1}$, because q does not belong to the $(\leq N/2^{j+1})$-level of $\mathcal{F}$; thus, $K' = 2^c N/2^j$ is a 2^{c+1}-approximation of K.

To find such an index j, we start from $i = 0$ and decide for each of the shallow cuttings $\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_{2c+2^i}$ whether it contains q. If we find an index $0 \leq j < 2c + 2^i$ such that $q \in \mathcal{C}_j$ but $q \notin \mathcal{C}_{j+1}$, we report $K' = 2^c N/2^j$. Otherwise, if $i < n$, we increment i and repeat this procedure or, if $i = n$, report that K is too small, and the query fails.

To determine in iteration i which of the shallow cuttings $\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_{2c+2^i}$ contain the query point q, note that $q \in \mathcal{C}_j$, for some $0 \leq j \leq 2c + 2^i$ if and only if q lies in or below the cell C of $\mathcal{C}_j$ stored with the face f of $\mathcal{A}_i^*$ that contains the projection of q into the xy-plane. Hence, we implement the ith iteration of our query procedure by finding this face f, using the point location structure of $\mathcal{A}_i^*$, and then scanning the list of cells of the shallow cuttings $\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_{2c+2^i}$ stored with f.

Correctness We have already argued that the query procedure outputs a 2^{c+1}-approximation of K unless it fails. Thus, it suffices to show that the query procedure does not fail for any $K \geq N^{1-\delta}/2^c$. For any such K, we have $K \geq N^{1-\delta}/2^c \geq N/2^{2^{\lfloor \log(2\delta \log N) \rfloor}+c} = N/2^{2^n+c}$, and the choice of the constant c implies that $q \notin \mathcal{C}_{2c+2^n}$, that is, the query does not fail.

Query bound To bound the cost of a successful query, we observe that $2c + 2^{i-1} \leq j < 2c + 2^i$ when the query terminates. Since the size of the representation of each arrangement $\mathcal{A}_i^*$ is polynomial in 2^{2^i}, the combined size of the representations of the first $i_0 := \log\log B - a$ arrangements $\mathcal{A}_0^*, \mathcal{A}_1^*, \ldots, \mathcal{A}_{i_0}^*$ is $\mathrm{O}(B)$, for an appropriate constant a. Thus, these structures fit in $\mathrm{O}(1)$ blocks and can be queried using $\mathrm{O}(1)$ block transfers; that is, the query cost is $\mathrm{O}(1)$ when $i \leq i_0$. For $i' > i_0$, the cost of the (i')th iteration, is $\mathrm{O}\left(\log_B 2^{2^{i'}} + 2^{i'}/B\right)$. The first term is the cost of querying the point location structure for the arrangement $\mathcal{A}_{i'}^*$. The second term is the cost of scanning the cell list associated with a face of $\mathcal{A}_{i'}^*$, as this list has size $\mathrm{O}\left(2^{i'}\right)$. The query cost in the case $i > i_0$ is therefore bounded by

$$\mathrm{O}(1) + \sum_{i'=i_0+1}^{i} \mathrm{O}\left(\log_B 2^{2^{i'}} + 2^{i'}/B\right) = \mathrm{O}\left(\log_B 2^{2^i}\right).$$

Observe, however, that $K = \Theta(N/2^j)$ and $2^{2c+2^{i-1}} \leq 2^j < 2^{2c+2^i}$. This implies that $2^{2^i} = \mathrm{O}((N/K)^2)$, and the query bound is $\mathrm{O}(\log_B(N/K))$.

For an unsuccessful query, the query terminates when $i = n$. By substituting this into the previous summation, we obtain a query bound of $\mathrm{O}\left(\log_B N^\delta\right)$ block transfers, which is $\mathrm{O}(\log_B(N/K))$, as $K \leq N^{1-\delta}/2^c$ in this case.

6.2.3 A Structure for Polylogarithmic Queries

In this section, we present the second building block of our data structure, a linear-space data structure for answering polylogarithmic approximate range counting queries, as summarized in the following lemma. Note that the query bound is $\mathrm{O}(\log_B N)$, not $\mathrm{O}(\log_B(N/K))$. This is sufficient for the purposes of our final data structure discussed in Section 6.2.4.

Lemma 6.4. *For a set $\mathcal{F}$ of N functions that satisfy conditions (i)–(iii), there exists a cache-oblivious data structure that uses $\mathrm{O}(N)$ space and supports 2^{c+1}-approximate range counting queries using $\mathrm{O}(\log_B N)$ block transfers in the worst case, as long as the count K satisfies $K > \log^\tau N$, for a sufficiently large constant $\tau > 0$.*

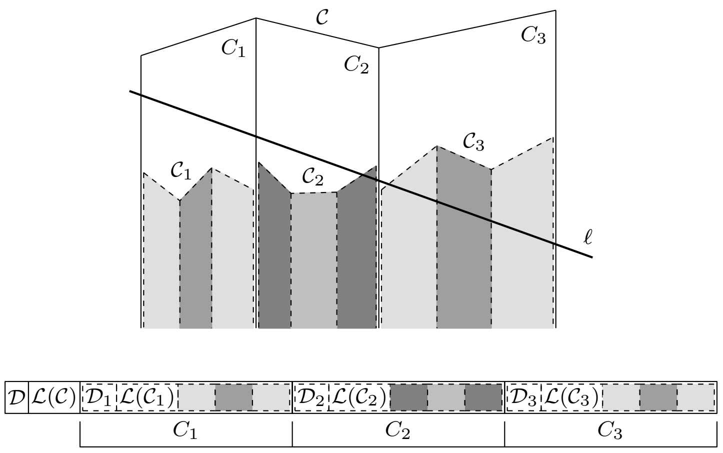


Figure 6.3: Two levels of the recursive structure for polylogarithmic queries. The entire structure consists of of an approximate counting structure $\mathcal{D}$ for polynomial queries over $\mathcal{F}$, a point location structure $\mathcal{L}(\mathcal{C})$ for the shallow cutting $\mathcal{C}$ of the ($\leq N^{1-\delta}/2^c$)-level of $\mathcal{F}$, as well as structures constructed recursively for the cells C_1, C_2, C_3 of $\mathcal{C}$. The representation of each cell C_i consists of an approximate counting structure $\mathcal{D}_i$ for polynomial queries over Δ_{C_i}, a point location structure $\mathcal{L}(\mathcal{C}_i)$ for a shallow cutting $\mathcal{C}_i$ of Δ_{C_i}, as well as structures constructed recursively for the cells of $\mathcal{C}_i$, indicated by the different shading of cells. Note that, even though the shallow cuttings define a recursive partition of space, this does not necessarily produce a partition of $\mathcal{F}$. For instance, if the structure represents an arrangements of lines, the line ℓ would be represented in the data structures of all three cells C_1, C_2, and C_3 and of some sub-cells of C_2 and C_3.

Data structure To obtain a data structure as in Lemma 6.4, we apply Lemma 6.3 recursively; see Figure 6.3. Let $\mathcal{D}$ be the data structure for $\mathcal{F}$ provided by Lemma 6.3, and let $\mathcal{C}$ be a shallow cutting for the ($\leq N^{1-\delta}/2^c$)-level of $\mathcal{F}$, for the same constants c and δ as in Section 6.2.2. In the memory layout, we represent $\mathcal{F}$ using the point location structure $\mathcal{L}(\mathcal{C})$ for $\mathcal{C}$, the data structure $\mathcal{D}$, and data structures representing the cells of $\mathcal{C}$, arranged in this order. The data structure representing each cell $C \in \mathcal{C}$ is constructed by recursively applying the construction just described to Δ_C. The recursion stops as soon as we obtain conflict lists of size at most $\log^\tau N$, for a constant τ to be chosen below.

Space bound Let $S(N)$ be the space complexity of our data structure for a set of N functions. The data structure $\mathcal{D}$ for polynomial queries has size $\mathrm{O}\left(\sqrt{N}\right)$, by Lemma 6.3, and the point location structure for $\mathcal{C}$ has size $\mathrm{O}\left(N^\delta\right) = \mathrm{O}\left(\sqrt{N}\right)$, for sufficiently small δ, by condition (ii) and because $\mathcal{C}$ is a shallow cutting for the ($\leq N^{1-\delta}/2^c$)-level of $\mathcal{F}$ and, thus, has $\mathrm{O}\left(N^\delta\right)$ cells. Since each cell of $\mathcal{C}$ has a conflict list of size at most $N^{1-\delta}$, and we do not recurse on conflict lists of size $\log^\tau N$, this implies that $S(N)$ is bounded by the recurrence

$$S(x) \le \begin{cases} ax^\delta S(x^{1-\delta}) + \mathrm{O}(\sqrt{x}) & x > \log^\tau N \\ \mathrm{O}(1) & x \le \log^\tau N \end{cases},$$

for an appropriate constant $a > 0$. After i steps of recursion applied to $S(N)$, we have

$$S(N) \le a^i N^{1-(1-\delta)^i} S(N^{(1-\delta)^i}) + \mathrm{O}\Big(a^i N^{1-(1-\delta)^i/2}\Big).$$

For $i = \log_{(1-\delta)}(\log\log^\tau N / \log N)$, we obtain

$$S(N) = \mathrm{O}\left(a^{(\log\log N)/\delta} \frac{N}{\log^\tau N} S(\log^\tau N) + a^{(\log\log N)/\delta} \frac{N}{\log^{\tau/2} N}\right),$$

as $\log_{(1-\delta)}(\log\log^\tau N/\log N) \le (\log\log N)/\delta$, for all $\tau \ge 1$ and $N \ge 4$. By choosing τ large enough, we thus obtain $S(N) = \mathrm{O}(N)$ because $S(\log^\tau N) = \mathrm{O}(1)$.

Query procedure To answer a query with a query point q, we use $\mathcal{L}(\mathcal{C})$ to decide whether q is contained in $\mathcal{C}$ and, if so, determine the cell $C \in \mathcal{C}$ that contains q. If $q \in \mathcal{C}$, we recurse on the data structure representing Δ_C or report a failure if we are already at the last level of recursion in the structure. If $q \notin \mathcal{C}$, we use $\mathcal{D}$ to answer the query.

Correctness First assume that the query does not fail. If $q \notin \mathcal{C}$, then $K \ge N^{1-\delta}/2^c$, and Lemma 6.3 shows that $\mathcal{D}$ provides a 2^{c+1}-approximation of K in this case. If q belongs to a cell C of $\mathcal{C}$, then Δ_C contains all functions in $\mathcal{F}$ that pass below q, and an inductive argument shows that recursing on Δ_C produces a 2^{c+1}-approximation of K. Thus, in both cases, we obtain a correct approximation of K.

If the query fails, on the other hand, then q is contained in a shallow cutting used at the last level of recursion. Since each cell of this cutting has a conflict list of size at most $\log^\tau N$, this means that $K \le \log^\tau N$. Thus, for $K > \log^\tau N$, the query procedure does not fail.

Query bound The query bound obeys the recurrence

$$Q(N) = \begin{cases} Q(N^{1-\delta}) + \mathrm{O}(\log_B N) & N > B \\ \mathrm{O}(1) & N \le B \end{cases}.$$

The bound for $N \le B$ follows because $S(B) = \mathrm{O}(B)$ and, hence, the entire recursive structure for a conflict list of size B fits in $\mathrm{O}(1)$ blocks. The bound for $N > B$ follows because querying $\mathcal{L}(\mathcal{C})$ and $\mathcal{D}$ requires $\mathrm{O}(\log_B N)$ block transfers, and querying the structure for the conflict list Δ_C in the case when $q \in \mathcal{C}$ takes $Q(|\Delta_C|) \le Q(N^{1-\delta})$ block transfers. This recurrence is easily seen to yield $Q(N) = \mathrm{O}(\log_B N)$, that is, the constructed data structure achieves the query bound claimed in Lemma 6.4.

6.2.4 The Final Structure

In this section, we combine Lemmas 6.3 and 6.4 to obtain a linear-space data structure that supports constant-factor approximate range queries with the optimal query bound, for any query range. By Lemma 6.2, this implies Theorem 6.1.

Lemma 6.5. *For a set $\mathcal{F}$ of N functions satisfying conditions (i)–(iii), there exists a cache-oblivious data structure that uses linear space and supports 2^{c+1}-approximate range counting queries using $\mathrm{O}(\log_B(N/K))$ block transfers in the worst case.*

Data structure Our final data structure consists of the following components; see Figure 6.4. Let $\mathcal{C}_s$ and $\mathcal{C}_b$ be shallow cuttings for the $(\le \log N)$-level and for the $(\le \log^\tau N)$-level of $\mathcal{F}$, respectively. We represent $\mathcal{F}$ using four data structures: a structure $\mathcal{D}_b$ for polynomial queries, a structure $\mathcal{D}_s$ for polylogarithmic queries, and the two point location structures $\mathcal{L}(\mathcal{C}_s)$ and $\mathcal{L}(\mathcal{C}_b)$ for the two shallow cuttings $\mathcal{C}_s$ and $\mathcal{C}_b$. In addition, we store for each cell $C \in \mathcal{C}_b$, a data structure $\mathcal{D}_C$ for polylogarithmic queries over Δ_C, and for each cell $C' \in \mathcal{C}_s$, the list of functions in $\Delta_{C'}$.

Space bound By Lemmas 6.3 and 6.4, both, $\mathcal{D}_b$ and $\mathcal{D}_s$, use linear space. The data structure representing each cell $C \in \mathcal{C}_b$ has size $\mathrm{O}(|\Delta_C|) = \mathrm{O}(\log^\tau N)$, and the number of cells in $\mathcal{C}_b$ is $\mathrm{O}(N/\log^\tau N)$. Thus, the representation of $\mathcal{C}_b$ uses linear space. Similarly, the conflict list of each cell $C' \in \mathcal{C}_s$ can be stored in $\mathrm{O}(\log N)$ space, and there are $\mathrm{O}(N/\log N)$ such cells, that is, the representation of $\mathcal{C}_s$ uses linear space. By summing these space bounds, we obtain that the entire data structure uses linear space.

Query procedure Given a query point q, we first query $\mathcal{D}_b$ to see whether q is polynomial ($K \ge N^{1-\delta}/2^c$) and, if so, obtain the desired approximation of K. If this fails, we try $\mathcal{D}_s$. If this also fails, that is, $K \le \log^\tau N$, then $q \in \mathcal{C}_b$. In this case, we query $\mathcal{L}(\mathcal{C}_s)$ to decide whether $q \in \mathcal{C}_s$ and, if so, determine the cell $C' \in \mathcal{C}_s$ that contains q. If $q \in \mathcal{C}_s$, we scan $\Delta_{C'}$

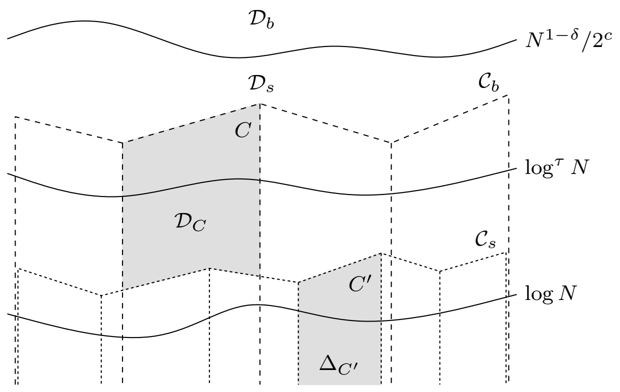


Figure 6.4: For queries above the $(N^{1-\delta}/2^c)$-level, $\mathcal{D}_b$ is used to answer the query. For queries between the $(\log^\tau N)$-level and the $(N^{1-\delta}/2^c)$-level, $\mathcal{D}_s$ is used. For queries between $\mathcal{C}_s$ and $\mathcal{C}_b$, $\mathcal{L}(\mathcal{C}_b)$ is used to locate the cell $C \in \mathcal{C}_b$ containing the query point, and the final answer is obtained using $\mathcal{D}_C$. For queries contained in $\mathcal{C}_s$, $\mathcal{L}(\mathcal{C}_s)$ is used to locate the cell $C' \in \mathcal{C}_s$ containing the query point, and the final answer is obtained by scanning $\Delta_{C'}$.

to count the number of functions passing below q. If $q \notin \mathcal{C}_s$, we query $\mathcal{L}(\mathcal{C}_b)$ to find the cell $C \in \mathcal{C}_b$ that contains q and then use $\mathcal{D}_C$ to obtain the desired approximation of K. (As we argue next, the query on $\mathcal{D}_C$ does not fail.)

Correctness By Lemmas 6.3 and 6.4, if one of the queries on $\mathcal{D}_b$ and $\mathcal{D}_s$ succeeds, it reports a 2^{c+1}-approximation of K. As already observed, if both queries fail, then $q \in \mathcal{C}_b$. If $q \in \mathcal{C}_s$, then all functions passing below q belong to $\Delta_{C'}$, where C' is the cell of $\mathcal{C}_s$ that contains q; thus, scanning $\Delta_{C'}$ in this case allows us to determine K exactly. If $q \notin \mathcal{C}_s$, then $K \geq \log N$. This implies in particular that q is a polylogarithmic query for the conflict list Δ_C of the cell C of $\mathcal{C}_b$ that contains q. Hence, the query on $\mathcal{D}_C$ does not fail in this case and reports a 2^{c+1}-approximation of K.

Query bound We divide the analysis of the query cost into polynomial queries and sub-polynomial queries. If $K \geq N^{1-\delta}/2^c$, the query on $\mathcal{D}_b$ succeeds and takes $O(\log_B(N/K))$ block transfers, by Lemma 6.3. Since no further queries are performed after a successful query on $\mathcal{D}_b$, this shows that polynomial queries can be answered using $O(\log_B(N/K))$ block transfers.

If $K < N^{1-\delta}/2^c$, the cost of the query procedure is bounded by the cost of one query on each of $\mathcal{D}_b$, $\mathcal{D}_s$, $\mathcal{L}(\mathcal{C}_b)$, and $\mathcal{L}(\mathcal{C}_s)$, plus the cost of querying the approximate counting structure $\mathcal{D}_C$ associated with a cell C of $\mathcal{C}_b$ or scanning the conflict list $\Delta_{C'}$ of a cell

C' of $\mathcal{C}_s$. Querying any of the point location or approximate counting structures takes $\mathrm{O}(\log_B N)$ block transfers, by Lemmas 6.3 and 6.4 and by condition (ii). Scanning $\Delta_{C'}$ takes $\mathrm{O}((\log N)/B) = \mathrm{O}(\log_B N)$ block transfers, as $|\Delta_{C'}| = \mathrm{O}(\log N)$. Hence, the total query cost for sub-polynomial queries is $\mathrm{O}(\log_B N)$ block transfers. Since $K < N^{1-\delta}/2^c$, however, we have $\mathrm{O}(\log_B N) = \mathrm{O}\big(\log_B(N^\delta)\big) = \mathrm{O}(\log_B(N/K))$; that is, the query procedure achieves the query bound claimed in Lemma 6.5 in this case as well.

6.2.5 Applications

By verifying that halfspace range counting and dominance counting satisfy conditions (i)–(iii), we obtain the following result as an immediate consequence of Theorem 6.1.

Theorem 6.6. *There exist cache-oblivious data structures that use* $\mathrm{O}(N(1+\varepsilon^{-2}\log\varepsilon^{-1}))$ *space and respectively support* $(1+\varepsilon)$*-approximate 3-d halfspace range counting queries and approximate 3-d dominance counting queries using* $\mathrm{O}(\log_B(N/K) + (\varepsilon^{-2}/B)\log\varepsilon^{-1})$ *block transfers and* $\mathrm{O}(\log(N/K) + \varepsilon^{-2}\log\varepsilon^{-1})$ *time in the worst case.*

Proof. The dual problem to 3-d halfspace range counting is to count all the planes in a set $\mathcal{F}$ that pass below a query point q [59]. It is well known that the lower envelope of a set of N linear functions corresponds to the convex hull of the points dual to the functions and, thus, has worst-case complexity $\mathrm{O}(N)$ [59]. Therefore, the set $\mathcal{F}$ of planes satisfies condition (i′) and, hence, condition (i). In fact, halfspace range searching was the problem used by Matoušek to introduce the notion of shallow cuttings [94]. A structure satisfying condition (ii) can be obtained by projecting the cells of the given shallow cutting into the plane and preprocessing the resulting planar straight-line subdivision for point location queries (see [7, 45]). Using the linear-space cache-oblivious planar point location structure by Bender et al. [37], point location queries on this arrangement can be answered using $\mathrm{O}(\log_B N)$ block transfers. The same data structure can be used to satisfy condition (iii).

For 3-d dominance counting, we can represent every input point p using a range $\overline{p}$ containing all points that dominate p; see Figure 6.5(b). The boundary of this range is not a totally defined function. However, a small perturbation turns the boundary of $\overline{p}$ into a totally defined function composed of three linear functions; see Figure 6.5(c). This allows us to phrase a dominance query with a query point q as identifying all such boundary functions that pass below q. Thus, our framework can be applied to dominance reporting as well, if

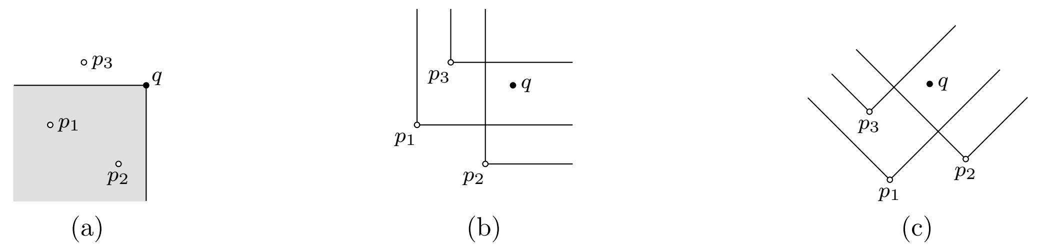


Figure 6.5: Reduction of dominance queries to aboveness queries in 2-d. (The construction in 3-d is identical, except that the last step requires two 45°-rotations.) (a) A point set (white) and a query range defined by the black corner point q. Points p_1 and p_2 are contained in the query range; p_3 is not. (b) Piecewise linear functions corresponding to the white points. q is a above the functions defined by query points p_1 and p_2, but not above the one defined by p_3. (c) Rotating the figure 45° to the left makes all functions totally defined and does not change aboveness.

we can verify that any collection of such boundary functions satisfies conditions (i)–(iii). It is not difficult to show, however, that the lower envelope of this set of functions has linear complexity (see, e.g., [2]). Thus, conditions (i′) and (i) are satisfied once again. Furthermore, as with halfspace queries, conditions (ii) and (iii) reduce to point location in a planar straight-line subdivision [2] and, hence, can be satisfied using the point location structure by Bender et al.

$\square$

6.3 Range Reporting

We can use the approximate range counting structure provided by Theorem 6.1 as a building block to quite easily obtain a cache-oblivious data structure that answers range reporting queries for any problem that fits in our framework. This data structure uses $O(N \log N)$ space and achieves the optimal query bound.

Given a set $\mathcal{F}$ of N functions satisfying conditions (i)–(iii), such a data structure can be obtained as follows. For $0 \leq i \leq \log N$, let $\mathcal{C}_i$ be a shallow cutting for the ($\leq 2^i$)-level of $\mathcal{F}$. For each cell $C \in \mathcal{C}_i$, we store the conflict list Δ_C contiguously. Since $\mathcal{C}_i$ contains $O(N/2^i)$ cells and each cell has a conflict list of size $O(2^i)$, the representation of each shallow cutting $\mathcal{C}_i$ uses linear space. As there are $\log N$ shallow cuttings, the total space consumption is $O(N \log N)$. Finally, we add a 2-approximate counting structure for $\mathcal{F}$ as in Theorem 6.1, as well as a point location structure $\mathcal{L}(\mathcal{C}_i)$ for each shallow cutting $\mathcal{C}_i$. This adds only $O(N)$

to the total space bound.

To answer a range reporting query with a query point q, we query the counting structure to obtain a 2-approximation K' of K. This incurs $\mathrm{O}(\log_B N)$ block transfers. Next we use another $\mathrm{O}(\log_B N)$ block transfers to query $\mathcal{L}(\mathcal{C}_i)$, for $i = \lceil \log K' \rceil$, and determine the cell $C \in \mathcal{C}_i$ that contains the point q. Finally, we scan the conflict list Δ_C and output all functions in Δ_C that pass below q. This incurs another $\mathrm{O}(1 + |\Delta_C|/B) = \mathrm{O}(1 + K/B)$ block transfers. The total query cost is thus $\mathrm{O}(\log_B N + K/B)$, and we obtain the following theorem.

Theorem 6.7. *For a given set $\mathcal{F}$ of N functions satisfying conditions (i)–(iii), there exists a cache-oblivious data structure that uses $\mathrm{O}(N \log N)$ space and supports range reporting queries using $\mathrm{O}(\log_B N + K/B)$ block transfers, where K is the output size of the query.*

Following the discussion in Section 6.2.5, this immediately implies the following corollary.

Corollary 6.8. *There exist cache-oblivious data structures that use $\mathrm{O}(N \log N)$ space and support 3-d dominance reporting and 3-d halfspace range reporting queries using $\mathrm{O}(\log_B N + K/B)$ block transfers.*

Using the reductions of Section 2.4.1, Corollary 6.8 immediately implies further results on cache-oblivious 3-sided range reporting and circular range reporting. Moreover, the construction of Theorem 6.7 can also be used to obtain a cache-oblivious data structure for K-nearest neighbour searching in the plane: using the reduction from circular range reporting to 3-d halfspace range reporting of Lemma 2.3, a K-nearest neighbour query in the plane can be converted into the problem of reporting the K lowest planes in 3-d stabbed by a vertical line ℓ; we can identify these planes using $\mathrm{O}(\log_B N + K/B)$ block transfers by identifying the shallow cutting $\mathcal{C}_i$ with $i = \lceil \log K \rceil$, using $\mathcal{L}(\mathcal{C}_i)$ (which is a planar point location structure on the xy-projection of $\mathcal{C}_i$) to find a cell $C \in \mathcal{C}_i$ stabbed by ℓ, and finally applying a linear-time selection algorithm (e.g., see [58, Chapter 9]) to Δ_C to find the K lowest planes in Δ_C stabbed by ℓ. Except for 3-sided range reporting, similar results were not known in the cache-oblivious model before.

Corollary 6.9. *There exist cache-oblivious data structures that use $\mathrm{O}(N \log N)$ space and achieve the optimal query bound of $\mathrm{O}(\log_B N + K/B)$ block transfers for 3-sided and circular range reporting in the plane, and for 2-d K-nearest neighbour searching.*

The final consequence of Corollary 6.8 is the first cache-oblivious data structure for 3-d orthogonal range reporting using the optimal query bound. This structure is obtained using a standard reduction of this problem to 3-d dominance reporting [56].

Corollary 6.10. *There exists a cache-oblivious 3-d range reporting structure that uses* $\mathrm{O}\big(N \log^4 N\big)$ *space and supports queries using* $\mathrm{O}(\log_B N + K/B)$ *block transfers.*

6.4 Approximate Conflict Lists

For the construction of the δ-approximate conflict lists in Lemma 6.2, we use the notion of an ε-approximation from VC-dimension theory. A set system $(S, \mathbb{R})$ consists of a base set S and a collection $\mathbb{R} \subseteq 2^S$ of subsets of S. An *ε-approximation* of $(S, \mathbb{R})$ is a subset $\tilde{S} \subseteq S$ such that

$$\left| \frac{\left|\tilde{S} \cap R\right|}{\left|\tilde{S}\right|} - \frac{|R|}{|S|} \right| \leq \varepsilon,$$

for all $R \in \mathbb{R}$.

The conflict list Δ_C of a cell C defines a set system $(\Delta_C, \mathbb{R}_C)$, where $\mathbb{R}_C = \{R_p \mid p \in \mathbb{R}^3\}$ and R_p is the set of functions in Δ_C that pass below the point p. We choose the δ-approximate conflict list $\tilde{\Delta}_C$ of the cell C to be a $(\delta/2)$-approximation of the set system $(\Delta_C, \mathbb{R}_C)$. For a query point $q \in C$, its level is $K = |R_q|$, and its level in $\tilde{\Delta}_C$ is $\tilde{K} := \left|R_q \cap \tilde{\Delta}_C\right|$. Since $\tilde{\Delta}_C$ is a $(\delta/2)$-approximation of Δ_C, we have $K - (\delta/2)\,|\Delta_C| \leq \tilde{K}\,|\Delta_C| \,/\, \left|\tilde{\Delta}_C\right| \leq K + (\delta/2)\,|\Delta_C|$, and the value $K' := \tilde{K}\,|\Delta_C| \,/\, \left|\tilde{\Delta}_C\right| + (\delta/2)\,|\Delta_C|$ satisfies $K \leq K' \leq K + \delta\,|\Delta_C|$, which is the approximation of K we require in Lemma 6.2.

By results from [97, 118] we can find a $(\delta/2)$-approximation of $(\Delta_C, \mathbb{R}_C)$ of size $\mathrm{O}(\delta^{-2} \log \delta^{-1})$ for every conflict list Δ_C. Vapnik and Chervonenkis showed that every set system of constant VC-dimension has an ε-approximation of size $\mathrm{O}(\varepsilon^{-2} \log \varepsilon^{-1})$ [118]. The set system defined by a set of algebraic functions of constant degree has constant VC-dimension [97].

Chapter 7

Conclusions

In this thesis we have outlined the natural importance of range searching problems, and the emerging importance of data locality in data structures for solving them. We have discussed practical heuristic data structures, as well as theoretically optimal data structures in the cache-oblivious model. We have also explored the problem in depth in the cache-oblivious model providing a strong space separation result between the I/O model and the cache-oblivious model.

On the heuristic side of things we have explored the use of space-filling curves, notably the Hilbert curve. We reverse engineered Butz's algorithm presenting it in a much more intuitive and rigorous geometric setting. We identified a common short-coming in regular usage which either imposes an $\log N$-factor time overhead, or an $\mathrm{O}\left(\frac{mn}{X}\right)$-space overhead when sorting data sets by Hilbert curve position. Couching Butz's algorithm in a geometric setting permits a key insight into the nature of this redundancy. We introduced the novel idea of a compact Hilbert index, as well as efficient algorithms for computing them. These indices allow for sorting by Hilbert curve position in optimal space and time complexity. Experimental results have validated our theorized improvements, and have shown that our implementation is able to compete with the best existing implementations. Since its initial publication the software developed as a part of this research has been adopted by several projects.

For the problem of space-filling curves, there remain many interesting questions. Our approach to compact Hilbert indices breaks one aspect of the fundamental behaviour of Hilbert curves. We were interested in preserving exactly the ordering provided by the full Hilbert curve, thus the generated compact curves no longer take unit step sizes between every two successive points. It would be interesting to explore alternative techniques that preserve this property rather than the original ordering. Similarly, it would be interesting to generalize the ideas of compact Hilbert indices to input hyperboxes whose sidelengths are *not* powers of two. In a recent paper Haverkort [72] explores why some space-filling curves behave better than others at preserving data locality though the exploration of Arwwid numbers; it would be interesting to determine and compare the Arwwid numbers of various competing compact space-filling curves.

We also explored range searching in the cache-oblivious model, providing a general framework for constructing cache-oblivious data structures for approximate range counting and exact range reporting for range searching problems that have appropriate shallow cuttings. This includes 3-sided range searching, for which matching results were obtained before using different techniques, as well as 3-d dominance searching and 3-d halfspace range searching, for which no such cache-oblivious structures were previously known.

The obtained counting structures use linear space, while the reporting structures use $\mathrm{O}(N \log N)$ space, which is a $\log N$ factor away from the space needed to obtain equivalent query complexities in internal memory or in the I/O model. However, we also showed that it is in fact impossible to achieve the optimal query bound of $\mathrm{O}(\log_B N + K/B)$ for these range reporting problems using linear space.

Our lower bound result shows an $\Omega((\log \log N)^{\varepsilon})$ gap between the space bounds of range reporting data structures with optimal query bounds in the I/O and CO models. While previous separation results between the two models had been obtained (with considerable technical difficulty and using sophisticated techniques), our result is the first one that proves a gap that grows with the input size. Our proof of the lower bound continues to hold even if the data structure is aware of the block sizes we use. That is, the lower bound holds even for cache-aware multi-level memory hierarchy models. In that case, however, our proof makes the somewhat unrealistic assumption that the memory hierarchy has $\sqrt{\log N}$ levels.

While we have made progress on the problems we have considered, there are still questions left open. Most obviously, the gap between the range searching data structures of Chapter 6 and the lower bound of Chapter 5 remains open. Recent results by Afshani and Zeh [11] have extended the work presented here, tightening the lower bound from $\Omega(N(\log \log N)^{\varepsilon})$ to $\Omega(N(\log N)^{\varepsilon})$[1]. The same paper uses the approximate counting structure of Section 6.2 as the basis of an $\mathrm{O}\big(N\sqrt{\log N} \log \log N\big)$-space structure for optimal 3-d dominance and 3-sided range reporting, only an $\mathrm{O}(\log \log N)$-factor away from matching the tightened lower bound. These results have narrowed the gap even further, but it still remains open. It seems plausible that the lower bound of $\mathrm{O}(N \log^{\varepsilon} N)$ space should be achievable for these problems,

[1]The proof is quite similar but starts with 'harder' point and query sets that have a natural 'depth' of $\Omega(\log N)$ rather than $\Omega(\log \log N)$.

which would close the gap completely.

Our reporting structures follow the standard framework of cache-oblivious geometric search structures: obtain an approximation of the output size of the query and then query the appropriate level in a multi-level reporting structure whose levels are tailored to support different output sizes. Since our counting structures use linear space and provide good enough approximations of the output size, the main challenge in obtaining more space-efficient cache-oblivious range reporting structures is to reduce the space required by such structures that know the output size. The recent results by Afshani and Zeh [11] succeed in this task for 3-d dominance and 3-sided range reporting, but the approach is limited to these particular problems and does not appear easily generalizable. It also does not address the fundamental issue of secondary data structures in the CO model, which remains an interesting open problem.

This thesis presents a comprehensive review of current heuristic, applied and theoretical results for range searching data structures with cache locality, and introduces new results for each. These results have already formed the basis for further improvements [11], and it is hoped that the techniques and results developed in this thesis will continue to see use.

Bibliography

[1] David J. Abel and David M. Mark. A comparative analysis of some two-dimensional orderings. *International Journal of Geographic Information Systems*, 4(1):21–31, January 1990.

[2] Peyman Afshani. On dominance reporting in 3D. In *Proceedings of the 16th European Symposium on Algorithms*, volume 5193 of *Lecture Notes in Computer Science*, pages 41–51. Springer-Verlag, 2008.

[3] Peyman Afshani. *On Geometric Range Searching, Approximate Counting and Depth Problems*. PhD thesis, University of Waterloo, September 2008.

[4] Peyman Afshani, Lars Arge, and Kasper Dalgaard Larsen. Orthogonal range reporting in three and higher dimensions. In *FOCS'09: Proceedings of the 50th IEEE Symposium on Foundations of Computer Science*, pages 149–158, 2009.

[5] Peyman Afshani, Lars Arge, and Kasper Dalgaard Larsen. Orthogonal range reporting: query lower bounds optimal structures in 3-d and higher dimensional improvements. In *SOCG'10: Proceedings of the 26th ACM Symposium on Computational Geometry*, pages 240–246, 2010.

[6] Peyman Afshani and Timothy M. Chan. On approximate range counting and depth. *Discrete and Computational Geometry*, 42:3–21, 2009.

[7] Peyman Afshani and Timothy M. Chan. Optimal halfspace range reporting in three dimensions. In *Proceedings of the 20th ACM-SIAM Symposium on Discrete Algorithms*, pages 180–186, 2009.

[8] Peyman Afshani, Chris H. Hamilton, and Norbert Zeh. Cache-oblivious range reporting with optimal queries requires superlinear space. In *Proceedings of the 25th ACM Symposium on Computational Geometry*, pages 277–286, June 2009.

[9] Peyman Afshani, Chris H. Hamilton, and Norbert Zeh. A general approach for cache-oblivious range reporting and approximate range counting. In *Proceedings of the 25th ACM Symposium on Computational Geometry*, pages 287–295, June 2009.

[10] Peyman Afshani, Chris H. Hamilton, and Norbert Zeh. A general approach for cache-oblivious range reporting and approximate range counting. *Computational Geometry: Theory and Applications*, 43(8):700–712, October 2010.

[11] Peyman Afshani and Norbert Zeh. Improved space bounds for cache-oblivious range reporting. In *SODA'11: Proceedings of the 22nd ACM-SIAM Symposium on Discrete Algorithms*, pages 1745–1758, 2011.

[12] Pankaj K. Agarwal. Range searching. In Jacob E. Goodman and Joseph O'Rourke, editors, *CRC Handbook of Discrete and Computational Geometry*. CRC Press, Inc., 2004.

[13] Pankaj K. Agarwal, Lars Arge, Andrew Danner, and Bryan Holland-Minkley. Cache-oblivious data structures for orthogonal range searching. In *Proceedings of the 19th ACM Symposium on Computational Geometry*, pages 237–245, 2003.

[14] Pankaj K. Agarwal, Lars Arge, Jeff Erickson, Paulo G. Franciosa, and Jeffrey Scott Vitter. Efficient searching with linear constraints. *Journal of Computer and System Sciences*, 61(2):194–216, 2000.

[15] Pankaj K. Agarwal, Boris Aronov, Timothy M. Chan, and Micha Sharir. On levels in arrangements of lines, segments, planes, and triangles. *Discrete and Computational Geometry*, Volume 19:315–331, 1998.

[16] Pankaj K. Agarwal, Boris Aronov, and Micha Sharir. On levels in arrangements of lines, segments, planes, and triangles. In *SCG'97: Proceedings of the 13th ACM Symposium on Computational Geometry*, pages 30–38. ACM, 1997.

[17] Pankaj K. Agarwal, Alon Efrat, and Micha Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. *SIAM Journal on Computing*, 29(3):912–953, 2000.

[18] Pankaj K. Agarwal, Eran Nevo, János Pach, Rom Pinchasi, Micha Sharir, and Shakhar Smorodinsky. Lenses in arrangements of pseudo-circles and their applications. *Journal of the ACM*, 51(2):139–186, 2004.

[19] Pankaj K. Agarwal and Micha Sharir. Pseudo-line arrangements: duality, algorithms, and applications. *SIAM Journal on Computing*, 34(3):526–552, 2005.

[20] Alok Aggarwal, Bowen Alpern, Ashok K. Chandra, and Marc Snir. A model for hierarchical memory. In *STOC'87: Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 305–314, May 1987.

[21] Alok Aggarwal and Jeffrey Scott Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, September 1988.

[22] Bowen Alpern, Larry Carter, and Ephraim Feig. Uniform memory hierarchies. In *FOCS'90: Proceedings of the 31st IEEE Symposium on Foundations of Computer Science*, pages 600–608, October 1990.

[23] Charles J. Alpert and Andrew B. Kahng. Multi-way partitioning via spacefilling curves and dynamic programming. In *Proceedings of the 31st Annual Conference on Design Automation*, pages 652–657, San Diego, California, June 6-10 1994.

[24] Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. New data structures for orthogonal range searching. In *FOCS'00: Proceedings of the 41st IEEE Symposium on Foundations of Computer Science*, page 198. IEEE Computer Society, 2000.

[25] Lars Arge, Gerth Stølting Brodal, Rolf Fagerberg, and Morten Laustsen. Cache-oblivious planar orthogonal range searching and counting. In *Proceedings of the 21st ACM Symposium on Computational Geometry*, pages 160–169, 2005.

[26] Lars Arge, Mark de Berg, and Herman J. Haverkort. Cache-oblivious R-trees. In *Proceedings of the 21st ACM Symposium on Computational Geometry*, pages 170–179, 2005.

[27] Lars Arge, Mark de Berg, Herman J. Haverkort, and Ke Yi. The priority R-tree: A practically efficient and worst-case optimal R-tree. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 347–358, 2004.

[28] Lars Arge, Vasilis Samoladas, and Jeffrey Scott Vitter. On two-dimensional indexability and optimal range search indexing. In *PODS'99: Proceedings of the 18th Symposium on Principles of Database Systems*, pages 346–357. ACM, 1999.

[29] Lars Arge, Vasilis Samoladas, and Ke Yi. Optimal external memory planar point enclosure. In *ESA'04: Proceedings of the 14th European Symposium on Algorithms*, pages 40–52, 2004.

[30] Lars Arge and Norbert Zeh. Simple and semi-dynamic structures for cache-oblivious orthogonal range searching. In *Proceedings of the 22nd ACM Symposium on Computational Geometry*, pages 158–166, 2006.

[31] Boris Aronov and Sariel Har-Peled. On approximating the depth and related problems. In *SODA'05: Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms*, pages 886–894. Society for Industrial and Applied Mathematics, 2005. see `http://valis.cs.uiuc.edu/~sariel/research/papers/04/depth/`

[32] Boris Aronov, Sariel Har-Peled, and Micha Sharir. On approximate halfspace range counting and relative epsilon-approximations. In *SCG'07: Proceedings of the 23rd ACM Symposium on Computational Geometry*, pages 327–336. ACM, 2007.

[33] John J. Bartholdi III and Paul Goldsman. Vertex-labeling algorithms for the Hilbert spacefilling curve. *Software–Practice and Experience*, 31(5):395–408, May 2001.

[34] Rudolf Bayer and Edward M. McCreight. Organization and maintenance of large ordered indices. *Acta Informatica*, 1:173–189, 1972.

[35] Bruno Becker, Stephan Gschwind, Thomas Ohler, Bernhard Seeger, and Peter Widmayer. An asymptotically optimal multiversion B-tree. *The VLDB Journal*, 5(4):264–275, 1996.

[36] Michael A. Bender, Gerth Stølting Brodal, Rolf Fagerberg, Dongdong Ge, Simai He, Haodong Hu, John Iacono, and Alejandro López-Ortiz. The cost of cache-oblivious searching. In *Proceedings of the 44th IEEE Symposium on Foundations of Computer Science*, pages 271–282, 2003.

[37] Michael A. Bender, Richard Cole, and Rajeev Raman. Exponential structures for efficient cache-oblivious algorithms. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, volume 2380 of *Lecture Notes in Computer Science*, pages 195–207. Springer-Verlag, 2002.

[38] Christian Böhm, Stefan Berchtold, and Daniel A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322–373, September 2001.

[39] Greg Breinholt and Christoph Schierz. Algorithm 781: Generating Hilbert's space-filling curve by recursion. *ACM Transactions on Mathematical Software*, 24(2):184–189, June 1998.

[40] Gerth Stølting Brodal and Rolf Fagerberg. On the limits of cache-obliviousness. In *Proceedings of the 35th ACM Symposium on Theory of Computing*, pages 307–315, 2003.

[41] Gerth Stølting Brodal, Rolf Fagerberg, and Kristoffer Vinther. Engineering a cache-oblivious sorting algorithm. *ACM Journal of Experimental Algorithmics*, 12, June 2008. Article 2.2.

[42] Arthur R. Butz. Convergence with Hilbert's space-filling curve. *Journal of Computer and System Sciences*, 3(2):128–146, May 1969.

[43] Arthur R. Butz. Alternative algorithm for Hilbert's space-filling curve. *IEEE Transactions on Computers*, pages 424–426, April 1971.

[44] Timothy M. Chan. Random sampling, halfspace range reporting, and construction of $(\leq k)$-levels in three dimensions. In *FOCS'98: Proceedings of the 39th IEEE Symposium on Foundations of Computer Science*, pages 586–595, 1998.

[45] Timothy M. Chan. Low-dimensional linear programming with violations. *SIAM Journal on Computing*, 34:879–893, 2000.

[46] Timothy M. Chan. Random sampling, halfspace range reporting, and construction of $(\leq k)$-levels in three dimensions. *SIAM Journal on Computing*, 30(2):561–575, 2000.

[47] Timothy M. Chan. On levels in arrangements of curves. *Discrete and Computational Geometry*, 29:375–393, 2003.

[48] Timothy M. Chan. On levels in arrangements of curves, II: A simple inequality and its consequences. *Discrete and Computational Geometry*, 34:11–24, 2004.

[49] Timothy M. Chan. On levels in arrangements of surfaces in three dimensions. In *SODA'05: Proceedings of the 16th annual ACM-SIAM symposium on Discrete algorithms*, pages 232–240. Society for Industrial and Applied Mathematics, 2005.

[50] Timothy M. Chan. On levels in arrangements of curves, III: further improvements. In *SCG'08: Proceedings of the 24th ACM Symposium on Computational Geometry*, pages 85–93, 2008.

[51] Siddhartha Chatterjee, Alvin R. Lebeck, Praveen K. Patnala, and Mithuna Thottethodi. Recursive array layouts and fast parallel matrix multiplication. In *Proceedings of the Eleventh Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA 1999*, pages 222–231, Saint-Malo, France, June 27-30 1999.

[52] Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal on Computing*, 17(3):427–462, 1988.

[53] Bernard Chazelle. Lower bounds on the complexity of polytope range searching. *Journal of the American Mathematical Society*, 2:637–666, 1989.

[54] Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete and Computational Geometry*, 9(2):145–158, 1993.

[55] Bernard Chazelle. Cuttings. In *Handbook of Data Structures and Applications*. Chapman and Hall/CRC, 2005.

[56] Bernard Chazelle and Leonidas J. Guibas. Fractional cascading: II. applications. *Algorithmica*, 1:163–191, 1986.

[57] Kenneth L. Clarkson and Peter W. Shor. Applications of random sampling in computational geometry, II. *Discrete and Computational Geometry*, 4:387–421, 1989.

[58] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.

[59] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 1997.

[60] Frank Dehne, Todd Eavis, and Andrew Rau-Chaplin. Parallel multi-dimensional ROLAP indexing. In *CCGrid'03: Proceedings of the 2003 IEEE International Symposium on Cluster Computing and the Grid*, pages 86–93, 2003.

[61] Christian A. Duncan, Michael T. Goodrich, and Stephen Kobourov. Balanced aspect ratio trees: combining the advantages of k-d trees and octrees. *Journal of Algorithms*, 38(1):303–333, 2001.

[62] Paul Erdős, László Lovász, Gustavus J. Simmons, and Ernst G. Straus. Dissection graphs of planar point sets. In Jagdish N. Srivastava, editor, *A Survey of Combinatorial Theory*, pages 139–154. North-Holland, 1973.

[63] James A. Fill and Svante Janson. The number of bit comparisons used by quicksort: an average-case analysis. In *SODA'04: Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms*, pages 300–307, 2004.

[64] Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, pages 285–297, 1999.

[65] Frank Gray. Pulse code communication. US Patent Number 2,632,058, March 17 1953.

[66] Roberto Grossi and Giuseppe F. Italiano. Efficient cross-tree for external memory. In James M. Abello and Jeffrey Scott Vitter, editors, *External Memory Algorithms and Visualization*, pages 87–106. American Mathematical Society, 1999.

[67] Roberto Grossi and Giuseppe F. Italiano. Efficient splitting and merging algorithms for order decomposable problems. *Information and Computation*, 154(1):1–33, 1999.

[68] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pages 47–57, 1984.

[69] Chris H. Hamilton and Andrew Rau-Chaplin. Compact Hilbert Indices for multi-dimensional data. In *CISIS'07: Proceedings of the 1st International Conference on Complex, Intelligent and Software Intensive Systems*, pages 139–146, 2007.

[70] Chris H. Hamilton and Andrew Rau-Chaplin. Compact Hilbert Indices: Space-filling curves for domains with unequal side lengths. *Information Processing Letters*, 105:155–163, February 2008.

[71] Sariel Har-Peled and Micha Sharir. Relative ε-approximations in geometry, 2006. see `http://valis.cs.uiuc.edu/~sariel/research/papers/06/relative/`

[72] Herman J. Haverkort. Recursive tilings and space-filling curves with little fragmentation. In *EuroCG'10: Proceedings of the 26th European Workshop on Computational Geometry*, pages 185–189, 2010. Full manuscript acommpanying original abstract at `http://arxiv.org/abs/1002.1843`.

[73] Joseph M. Hellerstein, Elias Koutsoupias, and Christos H. Papadimitriou. On the analysis of indexing schemes. In *Proceedings of the 16th ACM Symposium on Principles of Database Systems*, pages 249–256, 1997.

[74] David Hilbert. Über die stetige Abbildung einer Linie auf ein Flächenstück. *Mathematische Annalen*, 38:459–460, 1891.

[75] Guohua Jin and John M. Mellor-Crummey. SFCGen: A framework for efficient generation of multi-dimensional space-filling curves by recursion. *ACM Transactions on Mathematical Software*, 31(1):120–148, March 2005.

[76] Guohua Jin, John M. Mellor-Crummey, and Robert J. Fowler. Increasing temporal locality with skewing and recursive blocking. In *Proceedings of the 2001 ACM/IEEE Conference on Supercomputing*, page 43, Denver, Colorado, November 10-16 2001.

[77] Maher Kaddoura, Chao-Wei Ou, and Sanjay Ranka. Partitioning unstructured computational graphs for nonuniform and adaptive environments. *IEEE Parallel and Distributed Technology: Systems and Technology*, 3(3):63–69, September 1995.

[78] Ibrahim Kamel and Christos Faloutsos. Hilbert R-tree: An improved R-tree using fractals. In *Proceedings of the Twentieth International Conference on Very Large Databases*, pages 500–509, Santiago, Chile, September 1994.

[79] K. V. Ravi Kanth and Ambuj K. Singh. Optimal dynamic range searching in non-replicated index structures. In *Proceedings of the 7th International Conference on Database Theory*, volume 1540 of *Lecture Notes in Computer Science*, pages 257–276. Springer-Verlag, 1999.

[80] Haim Kaplan, Edgar A. Ramos, and Micha Sharir. The overlay of minimization diagrams in a randomized incremental construction. Manuscript, 2007.

[81] Haim Kaplan, Edgar A. Ramos, and Micha Sharir. Range minima queries with respect to a random permutation, and approximate range counting. *Discrete and Computational Geometry*, 2011. to appear.

[82] Haim Kaplan and Micha Sharir. Randomized incremental constructions of three-dimensional convex hulls and planar Voronoi diagrams, and approximate range counting. In *Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms*, pages 484–493. ACM Press, 2006.

[83] Naoki Katoh and Takeshi Tokuyama. -levels of concave surfaces$k$. *Discrete and Computational Geometry*, 27:567–584, 2002.

[84] David Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12:28–35, 1983.

[85] Elias Koutsoupias and David Scot Taylor. Tight bounds for 2-dimensional indexing schemes. In *Proceedings of the 17th ACM Symposium on Principles of Database Systems*, pages 52–58, 1998.

[86] Claude-Henri Lamarque and Frédéric Robert. Image analysis using space-filling curves and 1d wavelet bases. *Pattern Recognition*, 29(8):1309–1322, August 1996.

[87] Jonathan K. Lawder. Calculations of mappings between one and n-dimensional values using the Hilbert space-filling curve. Technical Report JL1/00, Birkbeck College, University of London, August 2000.

[88] Jonathan K. Lawder and Peter J. H. King. Querying multi-dimensional data indexed using the Hilbert space-filling curve. *SIGMOD Record*, 30(1):19–24, March 2001.

[89] László Lovász. On the number of halving lines. *Annal. Univ. Scie. Budapest. de Rolando Eötvös Nominatae, Sectio Math.*, 14:107–108, 1971.

[90] Christos Makris and Athanasios Tsakalidis. Algorithms for three-dimensional dominance searching in linear space. *Information Processing Letters*, 66(6):277–283, 1998.

[91] Adam Marcus and Gábor Tardos. Intersection reverse sequences and geometric applications. *Journal of Combinatorial Theory, Series A*, 113(4):675–691, 2006.

[92] Yossi Matias and Adi Shamir. A video scrambling technique based on space filling curves. In *Proceedings of Advances in Cryptology - CRYPTO'87*, pages 398–417, Santa Barbara, California, August 16-20 1987.

[93] Jiři Matoušek. Efficient partition trees. *Discrete and Computational Geometry*, 8(3):315–334, 1992.

[94] Jiři Matoušek. Reporting points in halfspaces. *Computational Geometry: Theory and Applications*, 2(3):169–186, 1992.

[95] Jiři Matoušek. Range searching with efficient hierarchical cuttings. *Discrete and Computational Geometry*, 10(2):157–182, 1993.

[96] Jiři Matoušek. Geometric range searching. *ACM Computing Surveys*, 26:421–461, 1994.

[97] Jiři Matoušek. Geometric set systems. *European Congress of Mathematics*, 2:1–27, 1998.

[98] Edward M. McCreight. Priority search trees. *SIAM Journal on Computing*, 14(2):257–276, 1985.

[99] Baback Moghaddam, Kenneth J. Hintz, and Clayton V. Stewart. Space-filling curves for image compression. In *Proceedings of the First Annual SPIE Conference on Automatic Object Recognition*, volume 1471, pages 414–421, Orlando, Florida, April 1-5 1991.

[100] Bongki Moon, Hosagrahar V. Jagadish, Christos Faloutsos, and Joel H. Saltz. Analysis of the clustering properties of the Hilbert space-filling curve. *Knowledge and Data Engineering*, 13(1):124–141, January 2001.

[101] Doug Moore. Fast Hilbert curve generation, sorting, and range queries. see `http://web.archive.org/web/20050212162158/http://www.caam.rice.edu/~dougm/twiddle/Hilbert/`, 1999.

[102] Edward A. Patrick, Douglas R. Anderson, and Friend K. Bechtel. Mapping multidimensional space to one dimension for computer output display. *IEEE Transactions on Computers*, 17(10):949–953, October 1968.

[103] Giuseppe Peano. Sur une courbe, qui remplit toute une aire plane. *Mathematische Annalen*, 36:157–160, 1890.

[104] Rom Pinchasi and Radoa Radoicic. Topological graphs with no self-intersecting cycle of length 4. In *SCG'03: Proceedings of the 19th ACM Symposium on Computational Geometry*, pages 98–103. ACM, 2003.

[105] Loren K. Platzman and John J. Bartholdi III. Spacefilling curves and the planar travelling salesman problem. *Journal of the ACM*, 36(4):719–737, October 1989.

[106] Octavian Procopiuc, Pankaj K. Agarwal, Lars Arge, and Jeffrey Scott Vitter. Bkd-tree: A dynamic scalable kd-tree. In *Proceedings of the 8th International Symposium on Advances in Spatial and Temporal Databases*, volume 2750 of *Lecture Notes in Computer Science*, pages 46–65. Springer-Verlag, 2003.

[107] Subramanian Ramaswamy. The p-range tree: A new data structure for range searching in secondary memory. In *Proceedings of the 6th ACM-SIAM Symposium on Discrete Algorithms*, pages 378–387, 1995.

[108] Edgar A. Ramos. On range reporting, ray shooting and k-level construction. In *Proceedings of the 15th ACM Symposium on Computational Geometry*, pages 390–399, 1999.

[109] John T. Robinson. The K-D-B tree: A search structure for large dimensional dynamic indexes. In *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data*, pages 10–18, 1981.

[110] Vasilis Samoladas and Daniel P. Miranker. A lower bound theorem for indexing schemes and its application to multidimensional range queries. In *Proceedings of the 17th ACM Symposium on Principles of Database Systems*, pages 44–51, 1998.

[111] Cristina Schmidt and Manish Parashar. Enabling flexible queries with guarantees in P2P systems. *IEEE Internet Computing*, 8(3):19–26, 2004.

[112] Cristina Schmidt, Manish Parashar, Wenjin Chen, and David J. Foran. Engineering a peer-to-peer collaboratory for tissue microarray research. In *CLADE'04: Proceedings of the 2nd International Workshop on Challenges of Large Applications in Distributed Environments*, page 64, 2004.

[113] Micha Sharir, Shakhar Smorodinsky, and Gábor Tardos. An improved bound for k-sets in three dimensions. *Discrete and Computational Geometry*, 26:195–204, 2001.

[114] Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[115] Hisao Tamaki and Takeshi Tokuyama. How to cut pseudoparabolas into segments. *Discrete and Computational Geometry*, 19:265–290, 1998.

[116] Hisao Tamaki and Takeshi Tokuyama. A characterization of planar graphs by pseudo-line arrangements. *Algorithmica*, 35:269–285, 2003.

[117] Spencer W. Thomas. Utah raster toolkit. Internet: `http://web.mit.edu/afs/athena/contrib/urt/src/urt3.1/urt-3.1b.tar.gz`, 1991.

[118] Vladimir Vapnik and Alexey Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16:264–280, 1971.

[119] Darren Erik Vengroff and Jeffrey Scott Vitter. Efficient 3-D range searching in external memory. In *Proceedings of the 28th ACM Symposium on Theory of Computing*, pages 192–201. ACM, 1996.

[120] Jeffrey Scott Vitter. External memory algorithms and data structures: dealing with massive data. *ACM Computing Surveys*, 33(2):209–271, 2001. see `http://www.cs.purdue.edu/~jsv/Papers/Vit.IO\\_survey.pdf`

[121] John Wilkes and Chris Ruemmler. An introduction to disk drive modelling. *IEEE Computer*, 27(3):17–28, 1994.

[122] Yuefeng Zhang and Robert E. Webber. Space diffusion: An improved parallel halftoning technique using space-filling curves. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 93*, pages 305–312, Anaheim, California, August 2-6 1993.