List of Corrections

Ph.D. Thesis
Doctor of Philosophy

DTU Compute
Department of Applied Mathematics and Computer Science

An adventure in data structures

Søren Vind

Kongens Lyngby 2015

DTU Compute
Department of Applied Mathematics and Computer Science
Technical University of Denmark

Matematiktorvet
Building 303B
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Abstract

Write English Abstract

Resume

Write Danish Resume

Preface

This dissertation is the result of my work within (mostly) theoretical computer science, more specifically, in the area of algorithms and data structures. It was prepared during my enrollment as a PhD student in the ITMAN graduate school at the Department of Applied Mathematics and Computer Science in partial fulfillment of the requirements for acquiring a PhD degree at the Technical University of Denmark.

The thesis contains a general introduction that contains a basic introduction to the field of algorithms and data structures and a description of the results I have obtained during my PhD, followed by seven included original papers. The papers are the full body of research I have produced during my enrollment, and five of them were peer-reviewed and published at international conferences, with the remaining two in submission.

Managed Video as a Service My PhD scholarship was funded by a grant from the Danish National Advanced Technology Foundation (Højteknologifonden) for the project *Managed Video as a Service*. The project was conceived by Aalborg University, the Technical University of Denmark, Milestone Systems and Securitas in collaboration, scheduled to last for four years and to involve four PhD students (and advisors), as well as funding for improvements to a software system built by Milestone Systems.

My part of the project was called *Algorithms for Metadata*. Its goal was to research and develop compressed data structures for indexing massive amounts of meta data that supports efficient queries, in the software system built by Milestone Systems to support massive deployments of surveillance cameras. One of my papers is a direct result of my involvement in this project, being an efficient index for finding movement in surveillance footage. This paper also contains the only prototype I have created, and my only experimental results. All other works are motivated by theoretical curiosity.

We vs I

Acknowledgements

- advisors
- external visit
- friends and family
- office
- section

- institute
- university
- coauthors: Philip Bille, Patrick Hagge Cording, Roberto Grossi, Inge Li Gørtz, Markus Jalsenius, Giulia Menconi, Nadia Pisanti, Benjamin Sach, Frederik Rye Skjoldjensen, Roberto Trani, Hjalte Wedel Vildhøj

CONTENTS

1 Introduction

Data Structures are the basic building blocks in software engineering; they are the organizational method that allow us to store and access information in our computers efficiently. An *Algorithm* specifies the steps we perform to complete some task on an input in order to produce the correct output, relying on underlying data structures. Naturally, deep knowledge and understanding of algorithms and data structures are core competences for software engineers, absolutely vital to developing efficient and predictable software. In this thesis, we consider the *design and analysis of algorithms and data structures*:

> Our objective is to use resources efficiently. We *design* data structures and algorithms that solve a *problem*, and *analyse* proposed designs in a *machine model* that allows us to predict and compare the efficiency of different solutions on real computers.

The following Section 1.1 is a brief general introduction to the field of algorithmic research, and may be skipped by familiar readers. The remainder of this chapter is an overview and introduction to our contributions. The later chapters each include one of the following papers that are published in or submitted to peer-reviewed conference proceedings.

Fingerprints in Compressed Strings. By Philip Bille, Patrick Hagge Cording, Inge Li Gørtz, Benjamin Sach, Hjalte Wedel Vildhøj and Søren Vind. At Algorithms and Data Structures Symposium (WADS) 2013 [Bil+13].

Colored Range Searching In Linear Space. By Roberto Grossi and Søren Vind. At Scandinavian Symposium and Workshops on Algorithm Theory (SWAT) 2014 [Gro+14b].

Indexing Motion Detection Data for Surveillance Video. By Søren Vind, Philip Bille and Inge Li Gørtz. At IEEE International Symposium on Multimedia (ISM) 2014 [Vin+14].

Output-Sensitive Pattern Extraction in Sequences. By Roberto Grossi, Giulia Menconi, Nadia Pisanti, Roberto Trani and Søren Vind. At Foundations of Software Technology and Theoretical Computer Science (FSTTCS) 2014 [Gro+14a].

Compressed Data Structures for Range Searching. By Philip Bille, Inge Li Gørtz and Søren Vind. At International Conference on Language and Automata Theory and Applications (LATA) 2015 [Bil+15].

Compressed Pattern Matching in the Annotated Streaming Model. By Markus Jalsenius, Benjamin Sach and Søren Vind. In submission.

Dynamic Relative Compression and Dynamic Partial Sums. By Philip Bille, Patrick Hagge Cording, Inge Li Gørtz, Frederik Rye Skjoldjensen, Hjalte Wedel Vildhøj and Søren Vind. In submission.

The papers appear in separate chapters in their original form (except for formatting), meaning that notation, language and terminology has not been changed and may not be consistent across chapters. The chapter titles are equal to the original paper titles. Authors are listed as per the tradition in the field (alphabetically except for *Indexing Motion Detection Data for Surveillance Video* which was published at a multimedia conference).

The journal version of the following paper appeared during my PhD, but as the results were obtained and the conference version published prior to starting the PhD programme, the paper is omitted from this dissertation.

String Indexing for Patterns with Wildcards. By Philip Bille, Inge Li Gørtz, Hjalte Wedel Vildhøj and Søren Vind. At Scandinavian Symposium and Workshops on Algorithm Theory (SWAT) 2012 [Bil+12b]. In Theory of Computing Systems 2014. [Bil+14]

1.1 Bits of Background and Context

In his pioneering work *"On Computable Numbers, with an Application to the Entscheidungsproblem"* [Tur36] from 1936, Alan Turing in a single stroke became the father of the field of *theoretical computer science*. He gave the first general model of the theoretical capabilities of computers with the *Turing Machine*, and (among others) gave a formalisation of *algorithms*. In the following 80 years theoretical computer science naturally expanded, with the computer revolution and the first computer science institutions being established around fifty years ago[1]. Today, the field is founded on the modern day equivalents of the concepts introduced by Turing:

Machine Model We use an abstract model of a computer that ignores most details and allows us to understand its behaviour and reason about performance. For example, computation in the very common Word RAM model resembles the capabilities of modern day CPUs: memory is modeled as a sequence of words with w bits. We can read or write a word in unit time and perform arithmetic and word-operations on a constant number of words in unit time.

Algorithms In its most general formulation, an algorithm is a method for solving a problem on an input that produces the correct output. The *problem* is a central concept which states the desired properties of the input and output. One classic example of an algorithmic problem is *sorting*: given as input a list of numbers, return the same numbers in non-decreasing order.

[1]In 1965, the Department of Computer Science was founded at Carnegie Mellon University, as possibly the first such department in the world.

Data Structures A data structure is a method for maintaining a representation of some data while supporting a set of *operations* on the data: allow *updates*, and answer *queries* on the updated data. An example of a data structure problem is *sorted list representation*: store a list of numbers subject to insertions and deletions, and support queries for the k'th smallest number.

We characterise proposed solutions using several parameters. First, the output of an algorithm or query must always be (one of the possible) *correct* answers. The *performance* of a solution is the amount of resources required to execute it in our machine model, calculated relative to the size n of a finite input: typically the *space usage* is the number of memory words required, and the *time* is the number of machine operations necessary. We generally consider *worst-case analysis*, meaning the performance obtained when given the input that cause the solution to perform as poorly as possible.

Data Structure Example: Two Possible Solutions We now give a simple example with two possible solutions to a basic data structure problem called *membership*, where we must store a set of distinct numbers X and support member queries: given a number y, does X contain y? We use a simplified machine model that models time by how many comparisons of numbers are made, and ignores space.

The first solution is to store the unordered set X in a list. To answer the query, we start at one end of the list and compare y with a list element x: if $x = y$ we answer yes; otherwise we move to the next element. If we did not see y after checking all elements in the list $y \notin X$ and we answer no. Observe that a query answer always requires $O(|X|)$ comparisons if $y \notin X$.

An alternate solution is to store the numbers sorted in ascending order. This allows us to repeatedly use a single comparison to reduce the size of the list where y can possibly exist by half. The technique is called *binary search*, and it works as follows. We start at the middle number in the ordered list and compare its value x to y. No matter the result, we know that y can only possibly be in one half of the list and we can thus ignore the other half (if $x = y$ we answer yes; if $x < y$ we ignore the lower half, and conversely if $x > y$ we ignore the upper). The search continues by comparing y to the remaining non-ignored part of the list and expanding the ignored part until the possible list has size one, meaning $y \notin X$. In the worst case, we use $O(\lg |X|)$ comparisons as each allows us to ignore half of the remaining possible list, and $\lg |X|$ is the number of halvings of $|X|$ until $|X| \leq 1$.

In the above descriptions we ignored the time taken to construct the data structures. This can be done in $O(|X|)$ time for the unsorted list and $O(|X| \lg |X|)$ time for the sorted solution. As the list is only created once and then queried repeatedly, the sorted solution spend less time in total if we make more than $O(\lg |X|)$ queries.

Traditional *deterministic* solutions have inherently bad worst-case performance when solving some problems. To this end, *randomized* solutions relaxes the strict requirements on correctness and worst-case analysis. Instead, *monte carlo* solutions have worst case performance guarantees but some probability of giving an incorrect output. On the contrary, *las vegas* solutions always give a correct output, but with a risk of bad performance.

Generally, research in a particular problem takes two different angles. One is to prove that it is impossible to solve the problem using less resources than some *lower bound* for *any* solution in a given machine model. The opposite direction is to show the existence of a solution where the amount of used resources can be limited by some *upper bound*.

Lower Bound Example: Sorting It is easy to show a lower bound of $\Omega(n \lg n)$ for *sorting n* numbers using comparisons and swaps of pairs of numbers. If all n numbers are distinct they have $n!$ possible permutations, only one of which is the correct ordering. Then the number of number-pair comparisons to identify a single permutation uniquely is $\Omega(\lg(n!)) = \Omega(n \lg n)$. There are several matching upper bounds in the form of algorithms solving the problem using $O(n \lg n)$ comparisons[2].

Observe that the sorting lower bound immediately implies a lower bound on the time per operation for any *sorted list representation* data structure $\mathscr{D}$. The argument is as follows: First insert the entire list of n numbers into $\mathscr{D}$, and then repeatedly get, store, and remove the smallest number in $\mathscr{D}$. Then the result is an ordered version of the original list, created in $O(n)$ operations, meaning that at least one of the operations must take time $\Omega(\lg n)$.

Generally, lower bounds in weaker models may be circumvented by a stronger machine model. For example, there is a sorting algorithm taking $O(n \lg \lg n)$ time in the Word RAM model.

1.2 My Contributions

As initially stated, I consider data structures to one of the most fundamental building blocks for efficiently solving problems. Consequently, my main interest has been in designing new solutions for core data structure problems. A particular interest of mine is to find solutions that require relatively little space. Each of the proposed data structures exhibit this particular property, though they span several different subfields of algorithmic research, namely combinatorial pattern matching, geometry, and compression.

The papers included in the remaining chapters of this thesis each contribute at least one non-trivial newly-designed data structure for solving a particular problem. In some instances, the data structure is a cornerstone in giving a solution to an algorithmic problem, while in other cases it is the singular contribution. All but one of the papers are purely theoretical, meaning that although the proposed solutions may be good in practice, their efficiency have only been established in theory. In a single case, the proposed data structure was implemented and tested experimentally in practical scenarios.

In the following, I will give a brief overview of some of the fundamental data structure problems that are relevant for the later chapters, and provide a background for how our contributions fit in the larger world of data structures. The chapter is meant to give a brief overview of the field from our perspective, not to provide an exhaustive history.

For each paper or subject considered in the later chapters, make a subsection. Then in that subsection clarify:

- Our Contributions
- Future Directions

1.2.1 Model

The computation model used in all papers is the *Word RAM model*, where the memory is modeled as an array of w-bit words. To be able to index into the array in constant time, we always require $w \geq \lg n$ where n is the problem size. In this model, words can be accessed and modified in constant time. Comparisons, arithmetic operations and common word-operations (AND, OR, NOT) each take constant time when operating on a constant number of words.

1.2.2 Strings

A *string* (or text) is a sequence of characters from some *alphabet*. Classically, we store the sequence as is, with each character taking up a single word in memory. Strings is a core data abstraction, and much data may be stored and transmitted in the form of text, such as log files, time series databases, text documents or DNA sequences such as ACGATTAGATAAGC.

We may also consider storing strings in compressed form. *Phrase-compression* is a standard compression model, where the text is compressed into a sequence of phrases and each phrase is an extension of some previous phrase in the sequence. A *straight line program* is such a phrase-based compression scheme, where a new phrase is always either 1) a terminal character or 2) the concatenation of two previous phrases. This theoretical scheme captures many compression schemes with low overhead, such as the LZ77 and LZ78 schemes invented by Lempel and Ziv [Ziv+77; Ziv+78]. When dealing with compressed text, we denote by n the size of the compressed text (the number of phrases), and let N be the size of the decompressed text.

Four papers related to strings are included in this dissertation. First, we consider *compressed fingerprints*, showing how augment a straight line program with linear additional storage to support efficient fingerprint queries. We also consider the *compressed pattern matching* problem, which is to find the occurrences of a given *pattern* in a given *compressed text* (i.e. the starting positions where a substring of the text matches the pattern). A single paper give the first output-sensitive solution to *pattern extraction*, which may be considered the opposite of pattern matching. It is to extract unknown and *important patterns* from a given text (where importance is measured as the number of occurrences of the pattern). Finally, we consider *dynamic compression*, where we must maintain a representation of a compressed string that can be modified.

Compressed Fingerprints

In 1987, Karp and Rabin [Kar+87] proposed a classic pattern matching algorithm for uncompressed text of length N. Their approach is *randomized*, relying on *fingerprints* to efficiently compare substrings. The randomization is one-sided: all occurrences will be reported, but there is a small risk of reporting occurrences that do not exist. Their *Karp-Rabin fingerprints* have subsequently been used for a multitude of purposes, as a cornerstone in solving many string problems [Ami+92; And+06; Col+03; Cor+05; Cor+07; Far+98; Gas+96; Kal02; Por+09]. They have a number of key properties that make them extremely useful:

1. they support composition in constant time, allowing us to find the fingerprint of a string from the fingerprints of two halves,

2. they can be stored in constant space per fingerprint,

3. the fingerprints of two strings are different with high probability if the strings are different (if the strings are equal the fingerprints are as well).

These properties mean, for example, that we can create and store fingerprints of all prefixes of a string s in $O(N)$ time and space to allow (probabilistic) substring comparisons in s in constant time. While the same bound can be achieved deterministically by storing a suffix tree, we will use the same properties on compressed strings.

Our Contribution If storing a compressed string we cannot afford linear space in the text length to store fingerprints. In *Fingerprints for Compressed Strings* we provide a data structure for storing fingerprints in compressed space that supports efficient retrieval. Our data structure is for straight line programs, uses $O(n)$ space and allows us to retrieve the Karp-Rabin fingerprint of any (decompressed) substring in $O(\lg N)$ time. A previous solution with the same query time was given by Gagie for the much simpler case of balanced straight line programs, but no previous results were known in the general case [Gag+12a]. Our data structure builds on techniques introduced in a seminal paper by Bille, Landau, Raman, Sadakane, Satti, and Weimann [Bil+11], where they show how to store a straight line program in linear space to support random access to a character in the decompressed string in $O(\lg N)$ time.

The consequence of our result is that (probabilistically) comparing two substrings can now be done in the same time as accessing a single character. This is the same situation as for uncompressed strings. As an application of our fingerprint queries, we also show how to answer *longest common extension* queries, asking for the length ℓ of the longest matching substring starting from two positions in the uncompressed text, in time $O(\lg \ell \lg N)$. This is a fundamental primitive used in many string algorithms [3]. This data structure immediately implies a compressed space implementation of all previous algorithms relying

[3] zillionpapers

on Karp-Rabin fingerprints or longest common extension queries. It also admits a compressed space probabilistic implementation with $O(\lg N)$ overhead of all deterministic algorithms relying on constant time substring comparisons on the decompressed string.

Future Work Very recently, XXX et al.[4] have shown that random access in straight line programs can be solved in optimal time $O(\lg N / \lg\lg N)$ if increasing the space use to be superlinear. This matches a lower bound by somepeople[5]. A natural question is if their technique can be extended to improve the query time for fingerprint queries?

We answer longest common extension queries by an exponential search followed by a binary search, both using fingerprint queries to compare substrings. The fingerprint queries are extremely structured, and it feels like this may be exploited to improve the query time. However, having looked at this for a long time, I believe a different approach or new techniques are required. This is an obvious interesting research direction.

[4] patrick++
[5] RASLPlowerbound

Compressed Pattern Matching

The canonical problem on strings is *pattern matching*, where we must find occurrences of a *pattern* in a text. In the classic problem, the pattern consist of characters from the same alphabet as the text, and an occurrence of the pattern in the text is the location of a substring in the text that is equal to the pattern. In *compressed pattern matching*, the text is compressed into n phrases (the pattern is not) and we must solve pattern matching on the decompressed text: report the occurrences of the pattern of length m in the decompressed text of length N.

For both the classic and compressed problem good solutions exist. In the classic variant there are multiple optimal solutions, some of which even work in the *streaming model* [Mun+80; Fla+85; Alo+99]. In this model the text is received in a stream one character at a time, occurrences must be reported when they occur, and only sublinear space $o(N)$ is allowed. Known solutions for the compressed case does not work in the streaming model with the sequence of phrases received one at a time. In fact there is a space lower bound showing that any solution to compressed matching in the streaming model must use space $\Omega(n)$.

In order to model the current state of affairs in computing with easy and cheap access to massive computational power over the internet, the *annotated streaming model* introduced by Chakrabarti, Cormode and McGregor [Cha+09; Cha+14] expands the classic streaming model by introducing an untrustworthy *annotator*. This annotator is a theoretical version of "the cloud", assumed to have unbounded computational resources and storage, and it assists a *client* in solving some problem by providing an *annotated data stream* that is transmitted along with the normal input stream (i.e. the client-annotator communication is one-way). Software and hardware faults, as well as intentional attempts at deceit by the annotator (as could reasonably be the case), are modeled by assuming that the annotator cannot be trusted. In this model, the relevant performance parameters is the space required on the client, the time to process a phrase and the amount of annotation sent per item in the input stream.

Our Contribution In *Compressed Pattern Matching in the Annotated Streaming Model*, we give a trade-off solution to compressed pattern matching in the annotated streaming model that requires $O(\lg n)$ space on the client, and uses $O(\lg n)$ words of annotation and time per phrase received. The result is possibly the first to show the power of cloud computing in solving traditional problems on strings (the existing early work focuses on graph algorithms).

Future Work There are multiple natural directions to go in extending our work in the annotated streaming model. Generally, research into decreasing the amount of annotation per stream element even further would be of great interest. It seems likely that our obtained result can be extended to dictionary matching, where matches with any pattern from a dictionary of patterns must be reported. Of course, we would hope to get bounds better than just repeating the same algorithm for an entire dictionary of patterns.

A natural and much more powerful extension of our patterns is that of regular expressions. Intuitively, access to an untrusted annotator with unbounded computational power seems like it should help, as answering regular expression matching typically require many resources. However, extending our techniques to this case was non-trivial, but is a promising area of further research.

Pattern Extraction

In *pattern extraction*, the task is to extract the “most important” and frequently occurring patterns from a text S. We define the occurrence of a pattern in S as in *pattern matching* (pattern p occurs at position i in S iff p appears as a substring of S starting at position i). The importance of a pattern depends on its statistical relevance, namely, if the number of occurrences is above a certain threshold.

Pattern extraction should *not* be confused with pattern matching. In fact, we may consider the problems inverse: the former gets a text S from the user, and extracts patterns P and their occurrences from S, where both are unknown to the user (and P meets some criteria); the latter gets S and a given pattern p from the user, and searches for occurrences of p in S, and thus only the pattern occurrences are unknown to the user.

As is the case in pattern matching, one can imagine generalizing the standard patterns that consist of characters from the text alphabet in many ways (making the problem more real-world applicable). Indeed, many notions of patterns exist in the pattern extraction world [**CominV13**; **CunialA12**; **ApostolicoPU11**; **rime**; **Eskin04**; **IliopoulosMPPRS05**; **IsaacAU05**; Gro+11; Sag98; Ukk09; Ari+07]. The natural variation we study allows the special don’t care character $\star$ in a pattern to mean that the position inside the pattern occurrences in S can be ignored (so $\star$ matches any single character in S). For example, $\texttt{TA}\star\texttt{C}\star\texttt{ACA}\star\texttt{GTG}$ is such a pattern for DNA sequences.

Our Contribution In *Output-Sensitive Pattern Extraction in Sequences*, we show how to extract patterns with at most k don’t cares and a minimum number of occurrences. We extract only *maximal* patterns, meaning that we do not report patterns that can be made more precise (by extending them or replacing a don’t care by an alphabet symbol) without losing occurrences.

We give the first truly output-sensitive bounds for any variation of the pattern extraction problem, with all previous bounds that claimed to be output-sensitive still requiring polynomial time in n to report each pattern. Instead, our solution requires $O(k^3)$ time per pattern occurrence, which is far superior as k is typically a small constant. Our algorithm is simple, but the analysis is complex.

The outline here.

Future Work

- Our paper was analyzing how the construction algorithm worked for the motif trie. We amortized the construction cost for the entire trie on the number of actual maximal motifs it stores.

- Does the same kind of analysis extend to the *frequent itemset problem*? Here algorithms also perform bad in the worst case, but often not so bad in practice (as was the case for motif extraction before). We feel that the same situation may occur here.

Dynamic Compression

Dynamic Relative Compression

- Maintaining an asymptotically optimal compression of dynamic strings

Future Work

1.2.3 Range Searching

Much real world data has a natural spatial representation, as is the case for locations of interesting landmarks, visual data, or visitors to websites. Furthermore, much data can be represented spatially. For example, each row in a relational database can be thought of as a multidimensional point (with one dimension for each column). Consequently, geometric data structures is an important and well-studied research topic.

One of the core problems is *orthogonal range searching*, where we must store a set of d-dimensional points P to support *reporting* and *counting* queries. The input to a query is a d-dimensional rectangle, and the answer is the list of points in P that are contained in the rectangle (or the number of points). If we allow updates, they are typically in the form of point insertions or deletions.

If the points are in a single dimension, we can typically solve problems efficiently using integer data structures. We will focus on points of dimension at least two, where there are multiple efficient data structures for range searching. However, only the simplest variant is understood, with upper and lower bounds for higher dimensions being far apart.

Three of our papers consider variations of range searching. First, we consider *colored range searching* where the points each have a single color and the colors inside a query rectangle must be reported or counted. We then give a result on *compressed range searching*, showing a compression scheme for almost any classic range searching data structure that does not increase query times. Finally, we give experimental results on supporting *threshold range searching*, where each point has some weight and only the points exceeding some weight threshold must be reported.

Points with Colors

We consider *colored range searching*, which is a natural variation of range searching where points each have a single color. Colors are picked from some color alphabet, and two points can have the same color. The problem is also sometimes called generalized or categorical range searching, as we can think of the colors as categories that the points are put in. Queries ask for information about the colors of the points contained in a query range (instead of the actual points). That is, an answer to a reporting query is the distinct colors in a range, and a counting query must return the number of distinct colors.

Perhaps surprisingly, the colored variations of range searching are known to be much harder than their non-colored counterparts of the same dimensionality [6]. This is due to the non-uniqueness of colors in the query range, which means that reporting distinct colors incur additional computation that cannot be counted towards the time spent outputting points normally we count the time spent outputting points separately, as this is a required part of the output.

Counting hardness

Counting "harder" than reporting

Observe that for any constant dimensionality, we can always obtain linear space and linear time by simply storing the points in a list and scanning throught it to find (the distinct colors of) the points in the range. Furthermore, the well-known range tree [7] gives logarithmic query time for the non-colored problem using space $O(n \lg n)$, and in linear space we can obtain $O(\sqrt{n})$ time with the K-D tree. However, to obtain sublinear query times for the colored variant, previous solutions required excessive amounts of space of roughly $O(n^d)$.

Our Contribution In the paper *Colored Range Searching in Linear Space*, we give the first data structure for two-dimensional points in linear space and with sub-linear query time. We also give high dimensional structures with almost-linear space of $O(n \lg \lg^d n)$. In addition, we propose the first dynamic solutions in for any number of dimensions.

No previous papers considered how to obtain a non-trivial solution to colored range searching in (almost) linear space, and consequently, the proposed solution uses much less space than previous solutions.

Our result is obtained by partitioning points into groups depending on their color. Each group stores all the points for at most $\lg n$ specific colors. We then further partition the points in each group into buckets, each containing a polylogarithmic number of points. The points in each bucket are stored in a modified range tree data structure,

[6] lower bounds

[7] range tree

storing for each node in the range tree an auxiliary data structure identifying the colors contained in its range. To answer a query, we solve the problem independently in each bucket and get the final answer by merging the bucket results.

Future Work

- Non-trivial linear space data structure for at least 3 dimensions
- Improved bounds for two-dimensional colored range searching (we know of a tabulation improvement)
- What if the points have multiple colors?
- What if the points are moving?

Compressed Point Sets

The orthogonal range searching problem has been studied extensively over the last 40 years, creating a variety of classic proposed solutions [Ben75; Ben79; Ore82; Ben+80; Lue78; Lee+80; Gut; Cla83; Kan+99; Kre+91; Gae+98; Bay+72; Arg+08; Rob; Pro+03; Com79; Epp+08] (Samet presents an overview in [Sam90]). The classic solutions typically store the points in a some tree, using some strategy for dividing the d-dimensional universe into smaller areas and possibly introducing multiple copies of the same point to answer queries efficiently.

There are applications where the point set stored in an orthogonal range searching data structure commonly contains geometric repetitions, that is, subsets of points that are identical to each other but where the subset is offset by some translation vector. Range searching on points sets with geometric repetitions arise naturally in several scenarios such as data and image analysis [Tet+01; Paj+00; Dic+09], GIS applications [Sch+08; Zhu+02; Hae+10; Dic+09], and in compactly representing sparse matrices and web graphs [Gal+98; Bri+09; Gar+14; Ber+13].

Our Contribution In *Compressed Data Structures for Range Searching* we show that geometric repetitions can be exploited to compress all classic data structures that are based on trees can be efficiently compressed into a directed acyclic graph (DAG) representation. Any query on the original data structure can be answered by simulation on the compressed representation with only constant overhead.

We give a canonical range searching data structure, which is a tree that models almost all classic range searching data structures with constant overhead. We then show how to represent this using relative coordinates, allowing us to compress the relative tree using a minimal DAG representation that can be constructed in linear time due to Downey, Sethi and Tarjan [Dow+80].

As the classic data structures are not designed to be compressed, they may be uncompressible though the underlying points they store are. To this end, we also give a new hierarchical clustering algorithm that ensures that compressible point sets are indeed compressed (under certain constraints). The result is a new compressed data structure for which we can guarantee that the points are compressed.

Future Work

- Improve clustering algorithm run time
- Clustering for larger range of point distributions
- Give bounds for compression of specific point distributions

One could imagine other types of repetitions, such as subsets of points that are identical under rotation or scaling. We do not consider those in the paper, though the canonical range searching data structure supports storing such repetitions (as long as they can be described by a constant number of words per edge). An open question is how to efficiently find such repetitions and construct a corresponding relative tree.

Threshold Range Searching in Practice

Yet another variation of range searching arise if we assume that the points each have an integer weight. We can then naturally consider queries asking us to report or count the number of points in some query area where the weight of a reported point exceed some threshold. Alternatively, we can think of the weight as a coordinate in a new dimension and the queries as being unbounded in one direction in that new dimension.

An instance of the 4-dimensional threshold range counting problem occurred naturally in a software package for video surveillance built by Milestone Systems. Their system stores surveillance videos on disk in compressed form for later analysis. The goal was to be able to answer *motion detection* queries on video from a (statically mounted) surveillance camera to find "interesting clips". More precisely, given a time interval T, a video area A and two motion parameters p and q, report all the timestamps in T where area A of the video had more than p% of the pixels changed by more than q values. That is, each pixel is a point and the pixel value change is its weight. For this particular practical problem, the previously implemented solution performed extremely poorly. It simply decompressed the entire relevant portion of the compressed video file and computed the query. This made the functionality useless in practice.

Our Contribution In *Indexing Motion Detection Data for Surveillance Video*, we suggest an index for answering *motion detection* queries. The solution 1) creates a single data structure for each frame, 2) reduces the resolution of the queries allowed by partitioning frames into a number of regions, 3) creates histogram for the number of different weights in each region, and 4) stores the histograms after having compressed them using a standard compressor. To answer queries, the relevant histograms are decompressed, and the answer can be directly looked up.

We implemented a prototype and performed a number of experiments with realistic data sets. The index is space-efficient in practice (using around 10% of space required by the video file), quick to construct and answers queries quickly (speedup of at least $30x$).

Real World Implications A modified version of the suggested solution has already been implemented by Milestone in their software package. Their implementation is a restricted and optimized version of the prototype. First, by also implementing automatic noise detection they filter out the image noise. This allows them to reduce the histogram resolution, only storing a handful of difference values for each histogram. The end result is a performance improvement of 100x compared to their previous solution, for a space requirement of 1KB per second of video. This has enabled them to use the search function as an integral part of their product as it can now deliver results in real time.

Future Work The reason we are solving threshold range searching in the first place is that data structures for storing moving points are quite complicated and seem to perform relatively poorly. What we *actually* want is a simple kinetic data structure that supports efficient threshold queries. It should be able to compete with our current approach of just creating a new data structure per time stamp.

1.2.4 Bonus: Integer Data Structures

One of the most fundamental and important fields of data structures is that of *integer data structures*, where we operate on a sequence of integers, here denoted X. In our work, we have suggested new solutions to variants of two core problems called *predecessor* and *prefix sums*.

Predecessor

We must store X in order to support the *predecessor* query $\textsc{pred}(q)$, asking for the largest element in X smaller than q (a *successor* query for the smallest element larger than q must also be supported). Early data structures for the problem only supported that query, but later developments also included support for insertions and deletions in X as well as other closely related queries. The latest solutions are extremely advanced data structures; they are the culmination of decades of work and essentially optimal for all possible parameters[8], yielding a query time of $O(\lg\lg|X|)$ include fusion tree bounds.

Our Contribution Predecessor data structures are used in several of the included papers as a black box. But we also introduced a new data structure in *Fingerprints in Compressed Strings* for answering *finger predecessor* queries: The query is as before except that it also gives a reference to an existing element $\ell \in X$. For this variation, we give a dynamic solution with query time $O(\lg\lg|\ell - q|)$ number of elements between q, ℓ?, which is much better than the general solution when the reference is close to the query point. This is used in the paper to reduce the query time for multiple repeated queries to the same data structure.

Future Work

- Predecessor (and Finger Predecessor) essentially solved
- Succinct
- Deterministic?
- Finger Trees

[8] vEB, PatrascuThorup, Fusion Trees

Partial Sums

We must store the sequence of integers X to support three operations: *update* the value of an integer at some position by adding some value to it, find the *sum* of the first i elements, and *search* for the smallest sum larger than some query integer q. It is natural to consider the search operation as a successor query among the prefix sums in X.

The partial sums problem is extremely well-studied, with a long line of papers showing improved upper and lower bounds[9]. It was shown early that $O(\lg n / \lg\lg n)$ time per operation is sufficient[10]. Due to Patrascu and Demaine [**PatrascuDemaine**; **Others?**], we now have matching upper and lower bounds of $O(\lg n / \lg(w/\Delta))$ check this time per query in the Word RAM model, where Δ is the maximal number of bits allowed in the update argument and w is the maximal number of bits per integer in X.

Our Contribution In *Dynamic Relative Compression and Dynamic Partial Sums* we give a new improvement to the existing (optimal-time) data structure that also allow insertions and deletions in the sequence X. All previous data structures that allow modifications in X only worked for very small integers (where $w = O(\lg\lg n)$) check this.

Future Work

- Succinct
- Dynamic Partial Sums essentially solved
- Match the lower bound also for new operations
- No hope of improving parameter possibilities for insert/delete
- Further new operations may be considered

[9] ManyPapers
[10] somePaper

Bibliography

[Abo+04] Mohamed Ibrahim Abouelhoda, Stefan Kurtz, and Enno Ohlebusch. "Replacing suffix trees with enhanced suffix arrays". In: *JDA* 2.1 (2004), pp. 53–86.

[Abo+10] Mohamed Ibrahim Abouelhoda and Moustafa Ghanem. "String Mining in Bioinformatics". In: *Scientific Data Mining and Knowledge Discovery*. 2010, pp. 207–247.

[Aga+02] Pankaj K Agarwal, Sathish Govindarajan, and S Muthukrishnan. "Range searching in categorical data: Colored range searching on grid". In: *Proc. 10th ESA*. 2002, pp. 17–28.

[Alo+99] Noga Alon, Yossi Matias, and Mario Szegedy. "The Space Complexity of Approximating the Frequency Moments". In: *JCSS* 58.1 (1999), pp. 137–147 (cit. on p. 8).

[Als+00] Stephen Alstrup and Jacob Holm. "Improved algorithms for finding level ancestors in dynamic trees". In: *Proc. 27th ICALP*. 2000, pp. 73–84.

[Ami+92] Amihood Amir, Martin Farach, and Yossi Matias. "Efficient randomized dictionary matching algorithms". In: *Proc. 3rd CPM*. 1992, pp. 262–275 (cit. on p. 6).

[And+06] Alexandr Andoni and Piotr Indyk. "Efficient algorithms for substring near neighbor problem". In: *Proc. 17th SODA*. 2006, pp. 1203–1212 (cit. on p. 6).

[And+07] Arne Andersson and Mikkel Thorup. "Dynamic ordered sets with exponential search trees". In: *Journal of the ACM (JACM)* 54.3 (2007), p. 13.

[And99] Arne Andersson. "General balanced trees". In: *J. Algorithms* 30.1 (1999), pp. 1–18.

[Apo+06] Alberto Apostolico, Matteo Comin, and Laxmi Parida. "Bridging lossy and lossless compression by motif pattern discovery". In: *General Theory of Information Transfer and Combinatorics*. 2006, pp. 793–813.

[Arg+08] Lars Arge, Mark De Berg, Herman Haverkort, and Ke Yi. "The Priority R-tree: A practically efficient and worst-case optimal R-tree". In: *ACM TALG* 4.1 (2008), p. 9 (cit. on p. 15).

[Ari+07] Hiroki Arimura and Takeaki Uno. "An efficient polynomial space and polynomial delay algorithm for enumeration of maximal motifs in a sequence". In: *JCO* (2007) (cit. on p. 10).

[Bab85] László Babai. "Trading group theory for randomness". In: *Proc. 17th STOC*. 1985, pp. 421–429.

[Bak95] Brenda S Baker. “On finding duplication and near-duplication in large software systems”. In: *Proc. 2nd WCRE*. 1995, pp. 86–95.

[Bar+10] Jérémy Barbay, Francisco Claude, and Gonzalo Navarro. “Compact rich-functional binary relation representations”. In: *Proc. 9th LATIN*. 2010, pp. 170–183.

[Bay+72] R Bayer and EM McCreight. “Organization and maintenance of large ordered indexes”. In: *Acta Informatica* 1.3 (1972), pp. 173–189 (cit. on p. 15).

[Bel+12] Djamal Belazzougui, Paolo Boldi, and Sebastiano Vigna. “Predecessor search with distance-sensitive query time”. In: *arXiv:1209.5441* (2012).

[Ben+04] M.A. Bender and M. Farach-Colton. “The level ancestor problem simplified”. In: *Theoret. Comput. Sci.* 321 (2004), pp. 5–12.

[Ben+80] Jon Louis Bentley and James B Saxe. “Decomposable searching problems I. Static-to-dynamic transformation”. In: *J. Algorithms* 1.4 (1980), pp. 301–358 (cit. on p. 15).

[Ben75] Jon Louis Bentley. “Multidimensional binary search trees used for associative searching”. In: *Comm. ACM* 18.9 (1975), pp. 509–517 (cit. on p. 15).

[Ben79] Jon Louis Bentley. “Multidimensional binary search trees in database applications”. In: *IEEE Trans. Softw. Eng.* 4 (1979), pp. 333–340 (cit. on p. 15).

[Ber+08] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. 3rd. 2008.

[Ber+13] Guillermo de Bernardo, Sandra Álvarez-Garca, Nieves R Brisaboa, Gonzalo Navarro, and Oscar Pedreira. “Compact Querieable Representations of Raster Data”. In: *Proc. 20th SPIRE*. 2013, pp. 96–108 (cit. on p. 15).

[Ber+94] O. Berkman and U. Vishkin. “Finding level-ancestors in trees”. In: *J. Comput. System Sci.* 48.2 (1994), pp. 214–230.

[Bil+11] P. Bille, G.M. Landau, R. Raman, K. Sadakane, S.R. Satti, and O. Weimann. “Random access to grammar-compressed strings”. In: *Proc. 22nd SODA*. 2011, pp. 373–389 (cit. on p. 6).

[Bil+12a] Philip Bille, Inge Li Gørtz, Benjamin Sach, and Hjalte Wedel Vildhøj. “Time-Space Trade-Offs for Longest Common Extensions”. In: *Proc. 23rd CPM*. 2012, pp. 293–305.

[Bil+12b] Philip Bille, Inge Gørtz, Hjalte Vildhøj, and Søren Vind. “String Indexing for Patterns with Wildcards”. In: *Proc. 13th SWAT* (2012), pp. 283–294 (cit. on p. 2).

[Bil+13] Philip Bille, Patrick Hagge Cording, Inge Li Gørtz, Benjamin Sach, Hjalte Wedel Vildhøj, and Søren Vind. “Fingerprints in Compressed Strings”. In: *Proc. 13th WADS*. 2013, pp. 146–157 (cit. on p. 1).

[Bil+14] Philip Bille, Inge Li Gørtz, Hjalte Wedel Vildhøj, and Søren Vind. “String indexing for patterns with wildcards”. In: *Theory of Computing Systems* 55.1 (2014), pp. 41–60 (cit. on p. 2).

[Bil+15] Philip Bille, Inge Li Gørtz, and Søren Vind. “Compressed Data Structures for Range Searching”. In: *Proc. 9th LATA* (2015) (cit. on p. 1).

[Bos+09] Prosenjit Bose, Meng He, Anil Maheshwari, and Pat Morin. “Succinct orthogonal range search structures on a grid with applications to text indexing”. In: *Proc. 11th WADS*. 2009, pp. 98–109.

[Boz+95] Panayiotis Bozanis, Nectarios Kitsios, Christos Makris, and Athanasios K Tsakalidis. “New Upper Bounds for Generalized Intersection Searching Problems”. In: *Proc. 22nd ICALP*. 1995, pp. 464–474.

[Bri+09] Nieves R Brisaboa, Susana Ladra, and Gonzalo Navarro. “K2-trees for compact web graph representation”. In: *Proc. 16th SPIRE*. 2009, pp. 18–30 (cit. on p. 15).

[Bri+95] Sergey Brin, James Davis, and Hector Garcia-Molina. “Copy detection mechanisms for digital documents”. In: *Proc. ACM SIGMOD*. Vol. 24. 2. 1995, pp. 398–409.

[Cha+03] Chia-Hui Chang, Chun-Nan Hsu, and Shao-Cheng Lui. “Automatic information extraction from semi-structured web pages by pattern discovery”. In: *Decis Support Syst* 34.1 (2003), pp. 129–147.

[Cha+05] Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, Amit Sahai, and Abbi Shelat. “The smallest grammar problem”. In: *IEEE Trans. Inf. Theory* 51.7 (2005), pp. 2554–2576.

[Cha+09] Amit Chakrabarti, Graham Cormode, and Andrew McGregor. “Annotations in data streams”. In: *Proc. 36th ICALP*. 2009, pp. 222–234 (cit. on p. 8).

[Cha+14] Amit Chakrabarti, Graham Cormode, Andrew McGregor, and Justin Thaler. “Annotations in data streams”. In: *ACM Transactions on Algorithms (TALG)* 11.1 (2014), p. 7 (cit. on p. 8).

[Cha+15] Amit Chakrabarti, Graham Cormode, Andrew McGregor, Justin Thaler, and Suresh Venkatasubramanian. “Verifiable Stream Computation and Arthur–Merlin Communication”. In: *CCC*. 2015.

[Che+04] Xin Chen, Brent Francia, Ming Li, Brian Mckinnon, and Amit Seker. “Shared information and program plagiarism detection”. In: *IEEE Trans Inf Theory* 50.7 (2004), pp. 1545–1551.

[Cla+11] Francisco Claude and Gonzalo Navarro. “Self-indexed grammar-based compression”. In: *Fundamenta Informaticae* 111.3 (2011), pp. 313–337.

[Cla83] Kenneth L Clarkson. “Fast algorithms for the all nearest neighbors problem”. In: *Proc. 24th FOCS*. Vol. 83. 1983, pp. 226–232 (cit. on p. 15).

[Cli+12] Raphaël Clifford, Markus Jalsenius, Ely Porat, and Benjamin Sach. “Pattern matching in multiple streams”. In: *Proc. 23rd CPM*. 2012, pp. 97–109.

[Col+03] Richard Cole and Ramesh Hariharan. “Faster suffix tree construction with missing suffix links”. In: *SIAM J. Comput.* 33.1 (2003), pp. 26–42 (cit. on p. 6).

[Com79] Douglas Comer. “Ubiquitous B-tree”. In: *ACM CSUR* 11.2 (1979), pp. 121–137 (cit. on p. 15).

[Cor+05] Graham Cormode and S Muthukrishnan. “Substring compression problems”. In: *Proc. 16th SODA*. 2005, pp. 321–330 (cit. on p. 6).

[Cor+07] Graham Cormode and S Muthukrishnan. “The string edit distance matching problem with moves”. In: *ACM Trans. Algorithms* 3.1 (2007), p. 2 (cit. on p. 6).

[Cor+12] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. “Practical verified computation with streaming interactive proofs”. In: *Proc. 3rd ITCS*. 2012, pp. 90–112.

[Cor+13] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. “Streaming graph computations with a helpful advisor”. In: *Algorithmica* 65.2 (2013), pp. 409–442.

[Cut+00] Ross Cutler and Larry S. Davis. “Robust real-time periodic motion detection, analysis, and applications”. In: *IEEE Trans. PAMI* 22.8 (2000), pp. 781–796.

[Deb+99] Hervé Debar, Marc Dacier, and Andreas Wespi. “Towards a taxonomy of intrusion-detection systems”. In: *Computer Networks* 31.8 (1999), pp. 805–822.

[Dic+09] Christian Dick, Jens Schneider, and Ruediger Westermann. “Efficient Geometry Compression for GPU-based Decoding in Realtime Terrain Rendering”. In: *CGF* 28.1 (2009), pp. 67–83 (cit. on p. 15).

[Die91] Paul F. Dietz. “Finding Level-Ancestors in Dynamic Trees”. In: *Proc. 2nd WADS*. 1991, pp. 32–40.

[Dow+80] Peter J Downey, Ravi Sethi, and Robert Endre Tarjan. “Variations on the common subexpression problem”. In: *J. ACM* 27.4 (1980), pp. 758–771 (cit. on p. 15).

[Du+14] Shan Du, Choudhury A Rahman, Saika Sharmeen, and Wael Badawy. “Event Detection by Spatio-Temporal Indexing of Video Clips.” In: *IJCTE* 6.1 (2014).

[Emd+76] Peter van Emde Boas, Rob Kaas, and Erik Zijlstra. “Design and implementation of an efficient priority queue”. In: *Theory Comput. Syst.* 10.1 (1976), pp. 99–127.

[Epp+08] David Eppstein, Michael T Goodrich, and Jonathan Z Sun. “Skip quadtrees: Dynamic data structures for multidimensional point sets”. In: *IJCGA* 18.01n02 (2008), pp. 131–160 (cit. on p. 15).

[Far+14] Arash Farzan, Travis Gagie, and Gonzalo Navarro. “Entropy-bounded representation of point grids”. In: *CGTA* 47.1 (2014), pp. 1–14.

[Far+98] Martin Farach and Mikkel Thorup. “String Matching in Lempel–Ziv Compressed Strings”. In: *Algorithmica* 20.4 (1998), pp. 388–404 (cit. on p. 6).

[Fed+09] Maria Federico and Nadia Pisanti. “Suffix tree characterization of maximal motifs in biological sequences”. In: *Theor. Comput. Sci.* 410.43 (2009), pp. 4391–4401.

[Fla+85] Philippe Flajolet and G Nigel Martin. “Probabilistic counting algorithms for data base applications”. In: *JCSS* 31.2 (1985), pp. 182–209 (cit. on p. 8).

[Fre+84] Michael L Fredman, János Komlós, and Endre Szemerédi. “Storing a sparse table with 0(1) worst case access time”. In: *J. ACM* 31.3 (1984), pp. 538–544.

[Fre+93] Michael L. Fredman and Dan E. Willard. “Surpassing the information theoretic bound with fusion trees”. In: *J. Comput. System Sci.* 47.3 (1993), pp. 424–436.

[Gae+98] Volker Gaede and Oliver Günther. “Multidimensional access methods”. In: *ACM CSUR* 30.2 (1998), pp. 170–231 (cit. on p. 15).

[Gag+12a] Travis Gagie, Pawe Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. “A faster grammar-based self-index”. In: *arXiv:1209.5441* (2012) (cit. on p. 6).

[Gag+12b] Travis Gagie, Juha Kärkkäinen, Gonzalo Navarro, and Simon J Puglisi. “Colored range queries and document retrieval”. In: *TCS* (2012).

[Gal+98] Nicola Galli, Bernhard Seybold, and Klaus Simon. “Compression of Sparse Matrices: Achieving Almost Minimal Table Size”. In: *Proc. ALEX*. 1998, pp. 27–33 (cit. on p. 15).

[Gar+14] Sandra Alvarez Garcia, Nieves R Brisaboa, Guillermo de Bernardo, and Gonzalo Navarro. “Interleaved K2-Tree: Indexing and Navigating Ternary Relations”. In: *Proc. DCC*. 2014, pp. 342–351 (cit. on p. 15).

[Gas+05] Leszek Gasieniec, Roman Kolpakov, Igor Potapov, and Paul Sant. “Real-time traversal in grammar-based compressed files”. In: *Proc. 15th DCC*. 2005, p. 458.

[Gas+96] Leszek Gasieniec, Marek Karpinski, Wojciech Plandowski, and Wojciech Rytter. “Randomized efficient algorithms for compressed strings: The fingerprint approach”. In: *Proc. 7th CPM*. 1996, pp. 39–49 (cit. on p. 6).

[Gaw13] Pawe Gawrychowski. “Optimal pattern matching in LZW compressed strings”. In: *ACM TALG* 9.3 (2013), p. 25.

[Gol+85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. “The knowledge complexity of interactive proof-systems”. In: *Proc. 17th STOC*. 1985, pp. 291–304.

[Gol+86] Shafi Goldwasser and Michael Sipser. “Private coins versus public coins in interactive proof systems”. In: *Proc. 18th STOC*. 1986, pp. 59–68.

[Gol+91] Oded Goldreich, Silvio Micali, and Avi Wigderson. “Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems”. In: *J. ACM* 38.3 (1991), pp. 690–728.

[Göö+15] Mika Göös, Toniann Pitassi, and Thomas Watson. “Zero-Information Protocols and Unambiguity in Arthur-Merlin Communication”. In: *Proc. 6th ITCS*. 2015, pp. 113–122.

[Gro+11] Roberto Grossi, Andrea Pietracaprina, Nadia Pisanti, Geppino Pucci, Eli Upfal, and Fabio Vandin. “MADMX: A strategy for maximal dense motif extraction”. In: *J. Comp. Biol.* 18.4 (2011), pp. 535–545 (cit. on p. 10).

[Gro+14a] Roberto Grossi, Giulia Menconi, Nadia Pisanti, Roberto Trani, and Søren Vind. “Output-Sensitive Pattern Extraction in Sequences”. In: *Proc. 34th FSTTCS* 29 (2014), pp. 303–314 (cit. on p. 1).

[Gro+14b] Roberto Grossi and Søren Vind. “Colored Range Searching in Linear Space”. In: *Proc. 14th SWAT*. 2014, pp. 229–240 (cit. on p. 1).

[Gup+95] Prosenjit Gupta, Ravi Janardan, and Michiel Smid. “Further Results on Generalized Intersection Searching Problems: Counting, Reporting, and Dynamization”. In: *J. Algorithms* 19.2 (1995), pp. 282–317.

[Gup+97] Prosenjit Gupta, Ravi Janardan, and Michiel Smid. “A technique for adding range restrictions to generalized searching problems”. In: *Inform. Process. Lett.* 64.5 (1997), pp. 263–269.

[Gut] Antonin Guttman. “R-trees: A dynamic index structure for spatial searching”. In: *Proc. 1984 ACM SIGMOD*. Vol. 14. 2, pp. 47–57 (cit. on p. 15).

[Hae+10] Simon Haegler, Peter Wonka, Stefan Mueller Arisona, Luc Van Gool, and Pascal Mueller. “Grammar-based Encoding of Facades”. In: *CGF* 29.4 (2010), pp. 1479–1487 (cit. on p. 15).

[Hag98] Torben Hagerup. “Sorting and Searching on the Word RAM”. In: *Proc. 15th STACS*. 1998, pp. 366–398.

[Har+84] Dov Harel and Robert E. Tarjan. “Fast algorithms for finding nearest common ancestors”. In: *SIAM J. Comput.* 13.2 (1984), pp. 338–355.

[Hu+04] Weiming Hu, Tieniu Tan, Liang Wang, and Steve Maybank. “A survey on visual surveillance of object motion and behaviors”. In: *IEEE Trans. SMC* 34.3 (2004), pp. 334–352.

[Hua11] Shih-Chia Huang. “An advanced motion detection algorithm with video quality analysis for video surveillance systems”. In: *IEEE Trans. CSVT* 21.1 (2011), pp. 1–14.

[JáJ+05] Joseph JáJá, Christian W Mortensen, and Qingmin Shi. “Space-efficient and fast algorithms for multidimensional dominance reporting and counting”. In: *Proc. 15th ISAAC*. 2005, pp. 558–568.

[Jan+93] Ravi Janardan and Mario Lopez. “Generalized intersection searching problems”. In: *IJCGA* 3.01 (1993), pp. 39–69.

[Kal02] Adam Kalai. “Efficient pattern-matching with don’t cares”. In: *Proc. 13th SODA*. 2002, pp. 655–656 (cit. on p. 6).

[Kan+99] Kothuri Venkata Ravi Kanth and Ambuj Singh. “Optimal Dynamic Range Searching in Non-replicating Index Structures”. In: *Proc. 7th ICDT*. 1999, pp. 257–276 (cit. on p. 15).

[Kao+] Wei Chieh Kao, Shih Hsuan Chiu, and Che Y. Wen. “An effective surveillance video retrieval method based upon motion detection”. In: *IEEE ISI 2008* (), pp. 261–262.

[Kap+07] Haim Kaplan, Natan Rubin, Micha Sharir, and Elad Verbin. “Counting colors in boxes”. In: *Proc. 18th SODA*. 2007, pp. 785–794.

[Kap+13] Alexis Kaporis, Christos Makris, Spyros Sioutas, Athanasios Tsakalidis, Kostas Tsichlas, and Christos Zaroliagis. “Improved Bounds for Finger Search on a RAM”. In: *Algorithmica* (2013), pp. 1–38.

[Kar+87] Richard M Karp and Michael O Rabin. “Efficient randomized pattern-matching algorithms”. In: *IBM J. Res. Dev.* 31.2 (1987), pp. 249–260 (cit. on p. 6).

[Kla+13] Hartmut Klauck and Ved Prakash. “Streaming computations with a loquacious prover”. In: *Proc. 4th ITCS*. 2013, pp. 305–320.

[Kla+14] Hartmut Klauck and Ved Prakash. “An improved interactive streaming algorithm for the distinct elements problem”. In: *Proc. 41st ICALP*. 2014, pp. 919–930.

[Kre+91] Marc J van Kreveld and Mark H Overmars. “Divided k-d trees”. In: *Algorithmica* 6.1-6 (1991), pp. 840–858 (cit. on p. 15).

[Kre92] Marc van Kreveld. *New results on data structures in computational geometry*. PhD thesis, Department of Computer Science, University of Utrecht, Netherlands, 1992.

[Lar+12] Kasper Green Larsen and Rasmus Pagh. “I/O-efficient data structures for colored range and prefix reporting”. In: *Proc. 23rd SODA*. 2012, pp. 583–592.

[Lar+13] Kasper Green Larsen and Freek van Walderveen. “Near-Optimal Range Reporting Structures for Categorical Data.” In: *Proc. 24th SODA*. 2013, pp. 265–276.

[Lee+80] DT Lee and CK Wong. “Quintary trees: a file structure for multidimensional datbase sytems”. In: *ACM TODS* 5.3 (1980), pp. 339–353 (cit. on p. 15).

[Lue78] George S Lueker. “A data structure for orthogonal range queries”. In: *Proc. 19th FOCS*. 1978, pp. 28–34 (cit. on p. 15).

[Mab+10] Nizar R Mabroukeh and Christie I Ezeife. “A taxonomy of sequential pattern mining algorithms”. In: *ACM CSUR* 43.1 (2010), p. 3.

[Mäk+07] Veli Mäkinen and Gonzalo Navarro. “Rank and select revisited and extended”. In: *TCS* 387.3 (2007), pp. 332–347.

[McC76] Edward M. McCreight. “A Space-Economical Suffix Tree Construction Algorithm”. In: *Journal of the ACM* 23.2 (April 1976), pp. 262–272.

[Meh+90] Kurt Mehlhorn and Stefan Näher. “Bounded ordered dictionaries in $O(\lg\lg N)$ time and $O(n)$ space”. In: *Inform. Process. Lett.* 35.4 (1990), pp. 183–189.

[Mor03] Christian W Mortensen. *Generalized static orthogonal range searching in less space*. Tech. rep. TR-2003-22, The IT University of Copenhagen, 2003.

[Mun+80] Ian Munro and Mike S Paterson. “Selection and sorting with limited storage”. In: *TCS* 12.3 (1980), pp. 315–323 (cit. on p. 8).

[Nek+13] Yakov Nekrich and Jeffrey Scott Vitter. “Optimal color range reporting in one dimension”. In: *Proc. 21st ESA*. 2013, pp. 743–754.

[Nek09] Yakov Nekrich. “Orthogonal Range Searching in Linear and Almost-linear Space”. In: *Comput. Geom. Theory Appl.* 42.4 (2009), pp. 342–351.

[Nek12] Yakov Nekrich. “Space-efficient range reporting for categorical data”. In: *Proc. 31st PODS*. 2012, pp. 113–120.

[Nek14] Yakov Nekrich. “Efficient range searching for categorical and plain data”. In: *ACM TODS* 39.1 (2014), p. 9.

[Ore82] Jack A Orenstein. “Multidimensional tries used for associative searching”. In: *Inform. Process. Lett.* 14.4 (1982), pp. 150–157 (cit. on p. 15).

[Ove87] Mark H Overmars. *Design of Dynamic Data Structures*. 1987. isbn: 038712330X.

[Paj+00] Renato Pajarola and Peter Widmayer. “An image compression method for spatial search”. In: *IEEE Trans. Image Processing* 9.3 (2000), pp. 357–365 (cit. on p. 15).

[Par+00] Laxmi Parida, Isidore Rigoutsos, Aris Floratos, Daniel E. Platt, and Yuan Gao. “Pattern discovery on character sets and real-valued data: linear bound on irredundant motifs and an efficient polynomial time algorithm”. In: *Proc. 11th SODA*. 2000, pp. 297–308.

[Par+01] Laxmi Parida, Isidore Rigoutsos, and Dan Platt. “An Output-Sensitive Flexible Pattern Discovery Algorithm”. In: *Proc. 12th CPM*. 2001, pp. 131–142.

[Pat07] Mihai Patrascu. “Lower bounds for 2-dimensional range counting”. In: *Proc. 39th STOC*. 2007, pp. 40–46.

[Pic+06] Luká Pichl, Takuya Yamano, and Taisei Kaizoji. “On the symbolic analysis of market indicators with the dynamic programming approach”. In: *Advances in Neural Networks-ISNN*. 2006, pp. 432–441.

[Por+09] Benny Porat and Ely Porat. “Exact and approximate pattern matching in the streaming model”. In: *Proc. 50th FOCS*. 2009, pp. 315–323 (cit. on p. 6).

[Pro+03] Octavian Procopiuc, Pankaj K Agarwal, Lars Arge, and Jeffrey Scott Vitter. “Bkd-tree: A dynamic scalable kd-tree”. In: *Proc. 8th SSTD*. 2003, pp. 46–65 (cit. on p. 15).

[Rig+04] Isidore Rigoutsos and Tien Huynh. “Chung-Kwei: a Pattern-discovery-based System for the Automatic Identification of Unsolicited E-mail Messages”. In: *CEAS*. 2004.

[Rob] John T Robinson. “The KDB-tree: a search structure for large multidimensional dynamic indexes”. In: *Proc. 1981 ACM SIGMOD*, pp. 10–18 (cit. on p. 15).

[Ryt03] Wojciech Rytter. “Application of Lempel–Ziv factorization to the approximation of grammar-based compression”. In: *Theoret. Comput. Sci.* 302.1 (2003), pp. 211–222.

[Sac+94] Todd S Sachs, Craig H Meyer, Bob S Hu, Jim Kohli, Dwight G Nishimura, and Albert Macovski. “Real-time motion detection in spiral MRI using navigators”. In: *Magnetic resonance in medicine* 32.5 (1994), pp. 639–645.

[Sag98] Marie-France Sagot. “Spelling approximate repeated or common motifs using a suffix tree”. In: *Proc. 3rd LATIN*. 1998, pp. 374–390 (cit. on p. 10).

[Sam90] Hanan Samet. *Applications of spatial data structures*. Addison-Wesley, 1990 (cit. on p. 15).

[Sch+08] Grant Schindler, Panchapagesan Krishnamurthy, Roberto Lublinerman, Yanxi Liu, and Frank Dellaert. “Detecting and matching repeated patterns for automatic geo-tagging in urban environments”. In: *CVPR*. 2008, pp. 1–7 (cit. on p. 15).

[She+06] Reza Sherkat and Davood Rafiei. “Efficiently evaluating order preserving similarity queries over historical market-basket data”. In: *Proc. 22nd ICDE*. 2006, pp. 19–19.

[Shi+05] Qingmin Shi and Joseph JáJá. “Optimal and near-optimal algorithms for generalized intersection reporting on pointer machines”. In: *Inform. Process. Lett.* 95.3 (2005), pp. 382–388.

[Sim93] Imre Simon. “String matching algorithms and automata”. In: *Proc. 1st SPIRE*. 1993, pp. 151–157.

[Tet+01] Igor V Tetko and Alessandro EP Villa. “A pattern grouping algorithm for analysis of spatiotemporal patterns in neuronal spike trains.” In: *J. Neurosci. Meth.* 105.1 (2001), pp. 1–14 (cit. on p. 15).

[Tha14] Justin Thaler. “Semi-Streaming Algorithms for Annotated Graph Streams”. In: *arXiv:1407.3462* (2014).

[Tia+] Ying-Li Tian and Arun Hampapur. “Robust salient motion detection with complex background for real-time video surveillance”. In: *WACV 2005*.

[Tur36] Alan Mathison Turing. “On computable numbers, with an application to the Entscheidungsproblem”. In: *J. of Math* 58 (1936), pp. 345–363 (cit. on p. 2).

[Ukk09] Esko Ukkonen. “Maximal and minimal representations of gapped and non-gapped motifs of a string”. In: *Theor. Comput. Sci.* 410.43 (2009), pp. 4341–4349 (cit. on p. 10).

[Vin+14] Søren Vind, Philip Bille, and Inge Li Gørtz. “Indexing Motion Detection Data for Surveillance Video”. In: *Proc. IEEE ISM* (2014) (cit. on p. 1).

[Vin14] Søren Vind. *Source code for motion detection experiments*. `https://github.com/sorenvind/phd-motiondetectionindex`. 2014.

[Wei73] Peter Weiner. "Linear pattern matching algorithms". In: *Proc. 14th SWAT*. 1973, pp. 1–11.

[Wil12] Virginia Vassilevska Williams. "Multiplying matrices faster than Coppersmith-Winograd". In: *Proc. 44th STOC*. 2012, pp. 887–898.

[Wil83] Dan E Willard. "Log-logarithmic worst-case range queries are possible in space $\Theta(N)$". In: *Inform. Process. Lett.* 17.2 (1983), pp. 81–84.

[Zhu+02] Qing Zhu, Xuefeng Yao, Duo Huang, and Yetin Zhang. "An Efficient Data Management Approach for Large Cyber-City GIS". In: *ISPRS Archives* (2002), pp. 319–323 (cit. on p. 15).

[Ziv+77] Jacob Ziv and Abraham Lempel. "A universal algorithm for sequential data compression". In: *Information Theory, IEEE Trans. Inf. Theory* 23.3 (1977), pp. 337–343 (cit. on p. 5).

[Ziv+78] Jacob Ziv and Abraham Lempel. "Compression of individual sequences via variable-rate coding". In: *Information Theory, IEEE Trans. Inf. Theory* 24.5 (1978), pp. 530–536 (cit. on p. 5).