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

c

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 xxx thesis was prepared at the department of Applied Mathematics and Computer Science at the Technical University of Denmark in fulfillment of the requirements for acquiring a yyy degree in zzz.

The thesis contains a general introduction as well as .....

This dissertation is the result of my research in data structures during my enrollment as a PhD student at DTU Compute.

About my project, collaboration, funding, etc

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. Presented at Algorithms and Data Structures Symposium (WADS), 2013.

Colored Range Searching In Linear Space. By Roberto Grossi and Søren Vind. Presented at Scandinavian Symposium and Workshops on Algorithm Theory (SWAT), 2014.

Indexing Motion Detection Data for Surveillance Video. By Søren Vind, Philip Bille and Inge Li Gørtz. Presented at IEEE International Symposium on Multimedia (ISM), 2014.

Output-Sensitive Pattern Extraction in Sequences. By Roberto Grossi, Giulia Menconi, Nadia Pisanti, Roberto Trani and Søren Vind. Presented at Foundations of Software Technology and Theoretical Computer Science (FSTTCS), 2014.

Compressed Data Structures for Range Searching. By Philip Bille, Inge Li Gørtz and Søren Vind. Presented at International Conference on Language and Automata Theory and Applications (LATA), 2015.

Annotated Data Streams with Multiple Queries. By Markus Jalsenius, Benjamin Sach and Søren Vind. In submission.

Dynamic Relative Compression 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. Presented at Scandinavian Symposium and Workshops on Algorithm Theory (SWAT), 2012. In Theory of Computing Systems, 2014.

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.

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

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

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. However, we may consider storing the string in compressed form. One canonical model of compression is the *phrase-compression*, where the text is stored as a sequence of phrases, with each phrase being 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 [3]. 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 are related to strings. 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

Karp and Rabin [4] proposed a classic pattern matching algorithm for uncompressed text of length N in XXX. Their approach is *randomized*, relying on fingerprints to efficiently

[3] lz77,lz78
[4] karprabin

compare substrings (with a risk of giving a wrong answer). Their *Karp-Rabin fingerprints* have subsequently been used for a multitude of purposes, as a cornerstone in solving many string problems [5]. A key property is that they support composition in constant time, allowing us to find the fingerprint of a string from the fingerprints of two halves. This allow us to store fingerprints in $O(N)$ space to allow substring comparisons in constant time.

However, if compressing a string the fingerprints are not readily available in compressed space. This is what we provide in Fingerprints for Compressed Strings. We give a data structure for straight line programs that in $O(n)$ space allow us to retrieve the Karp-Rabin fingerprint of any (decompressed) substring in $O(\lg N)$ time. The techniques build on those used in a paper by Bille et al. [6] that in the same time and space bound allows random access to a character in the decompressed string, and afterwards supports linear time decompression of substrings. That is, we can now support comparing two substrings in the same time as accessing a character. As an application, we 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 data structure immediately implies a compressed space implementation of allow all previous algorithms relying on Karp-Rabin fingerprints or longest common extension queries.

Compressed Pattern Matching

In Annotated Data Streams with Multiple Queries, we show how to perform pattern matching when the compressed text arrives one phrase at a time. The pattern matching is performed on a client using the assistance of a powerful but untrusted *annotator*. 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).

- Fingerprints and Longest Common Extension
- In compressed text arriving in a stream

Pattern Extraction

The third and final included paper on strings is Output-Sensitive Pattern Extraction in Sequences. In it, we show how to extract patterns with a bounded number of wildcards that each match a single character, and which have a minimum number of occurrences.

- Extract patterns from a text

[5] manymanypapers
[6] bille2011

Dynamic Compression

Dynamic Relative Compression

- Maintaining an asymptotically optimal compression of dynamic strings

1.2.3 Range Searching

Intro to geometric data structures

In ORTHOGONAL RANGE SEARCHING, 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.

Observe that if the points are in a single dimension we can solve the problem using a predecessor data structure as follows: the query identifies two ends of an interval for which we can find the end points using predecessor queries, and we must report the points between the ends.

In two or more dimensions, there are multiple data structures for range searching.

Points with Colors

We consider COLORED RANGE SEARCHING, which is a natural variation of range searching where points each have a single color from some color alphabet. This is also sometimes called generalized or categorical range searching. Query results are on the colors of the points contained in a query range; answering reporting we must report the distinct colors and we must report the number of distinct colors in a counting query.

Perhaps surprisingly, the colored variations of range searching are known to be much harder than their non-colored counterparts of the same dimensionality [7]. Where we know simple bounds for non-colored and colored here in two dimensions. Consequently, in order to obtain logarithmic query times to the colored problems previous solutions required excessive amounts of space of around $O(n^d)$. In the paper Colored Range Searching, we give the first data structure with linear space use in two dimensions and sub-linear query time. We also give higher dimensional structures with almost-linear space of $O(n \lg \lg^d n)$. This is obtained by collecting results from many small data structures, each storing an auxiliary data structure with each node in a range tree of a limited size. The resulting query time is $O(n/\lg n)$ check this, which is only slightly sub-linear.

[7] lower bounds

Compressed Point Sets

Threshold Range Searching in Practice

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. We will focus on two core problems, called *predecessor* and *prefix sums*.

Predecessor

We must store X in order to support the *predecessor* query $\text{PRED}(q)$, 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.

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 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 that variant, 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.

Prefix 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 *prefix sum* of the first i elements, and *search* for the smallest prefix 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 prefix 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[11], 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. In DynamicRelative we give a new improvement to the existing (optimal-time) data structure that also allow insertions and

[8] vEB, PatrascuThorup, Fusion Trees
[9] ManyPapers
[10] somePaper
[11] PatrascuDemaine, Others?

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.

BIBLIOGRAPHY

[] `https://github.com/sorenvind/phd-motiondetectionindex`.

[ABHY08] 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), page 9.

[ACP06] 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, pages 793–813.

[AFM92] Amihood Amir, Martin Farach, and Yossi Matias. "Efficient randomized dictionary matching algorithms". In: *Proc. 3rd CPM*. 1992, pages 262–275.

[AG10] Mohamed Ibrahim Abouelhoda and Moustafa Ghanem. "String Mining in Bioinformatics". In: *Scientific Data Mining and Knowledge Discovery*. 2010, pages 207–247.

[AGM02] Pankaj K Agarwal, Sathish Govindarajan, and S Muthukrishnan. "Range searching in categorical data: Colored range searching on grid". In: *Proc. 10th ESA*. 2002, pages 17–28.

[AH00] Stephen Alstrup and Jacob Holm. "Improved algorithms for finding level ancestors in dynamic trees". In: *Proc. 27th ICALP*. 2000, pages 73–84.

[AI06] Alexandr Andoni and Piotr Indyk. "Efficient algorithms for substring near neighbor problem". In: *Proc. 17th SODA*. 2006, pages 1203–1212.

[AKO04] Mohamed Ibrahim Abouelhoda, Stefan Kurtz, and Enno Ohlebusch. "Replacing suffix trees with enhanced suffix arrays". In: *JDA* 2.1 (2004), pages 53–86.

[AMS99] N Alon, Y Matias, and M Szegedy. "The Space Complexity of Approximating the Frequency Moments". In: *JCSS* 58.1 (1999), pages 137–147.

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

[AT07] Arne Andersson and Mikkel Thorup. "Dynamic ordered sets with exponential search trees". In: *Journal of the ACM (JACM)* 54.3 (2007), page 13.

[AU07] Hiroki Arimura and Takeaki Uno. "An efficient polynomial space and polynomial delay algorithm for enumeration of maximal motifs in a sequence". In: *JCO* (2007).

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

[BÁBNP13] 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, pages 96–108.

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

[BBV12] Djamal Belazzougui, Paolo Boldi, and Sebastiano Vigna. “Predecessor search with distance-sensitive query time”. In: *arXiv:1209.5441* (2012).

[BCKO08] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. 3rd. 2008.

[BCN10] Jérémy Barbay, Francisco Claude, and Gonzalo Navarro. “Compact rich-functional binary relation representations”. In: *Proc. 9th LATIN*. 2010, pages 170–183.

[BDG95] Sergey Brin, James Davis, and Hector Garcia-Molina. “Copy detection mechanisms for digital documents”. In: *Proc. ACM SIGMOD*. Volume 24. 2. 1995, pages 398–409.

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

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

[BF04] M.A. Bender and M. Farach-Colton. “The level ancestor problem simplified”. In: *Theoret. Comput. Sci.* 321 (2004), pages 5–12.

[BGSV12] Philip Bille, Inge Li Gørtz, Benjamin Sach, and Hjalte Wedel Vildhøj. “Time-Space Trade-Offs for Longest Common Extensions”. In: *Proc. 23rd CPM*. 2012, pages 293–305.

[BHMM09] 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, pages 98–109.

[BKMT95] Panayiotis Bozanis, Nectarios Kitsios, Christos Makris, and Athanasios K Tsakalidis. “New Upper Bounds for Generalized Intersection Searching Problems”. In: *Proc. 22nd ICALP*. 1995, pages 464–474.

[BLN09] Nieves R Brisaboa, Susana Ladra, and Gonzalo Navarro. “K2-trees for compact web graph representation”. In: *Proc. 16th SPIRE*. 2009, pages 18–30.

[BLRSSW11] 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, pages 373–389.

[BM72] R Bayer and EM McCreight. “Organization and maintenance of large ordered indexes”. In: *Acta Informatica* 1.3 (1972), pages 173–189.

[BS80] Jon Louis Bentley and James B Saxe. “Decomposable searching problems I. Static-to-dynamic transformation”. In: *J. Algorithms* 1.4 (1980), pages 301–358.

[BV94] O. Berkman and U. Vishkin. “Finding level-ancestors in trees”. In: *J. Comput. System Sci.* 48.2 (1994), pages 214–230.

[CCM09] Amit Chakrabarti, Graham Cormode, and Andrew McGregor. “Annotations in data streams”. In: *Proc. 36th ICALP*. 2009, pages 222–234.

[CCMT14] Amit Chakrabarti, Graham Cormode, Andrew McGregor, and Justin Thaler. “Annotations in data streams”. In: *ACM Transactions on Algorithms (TALG)* 11.1 (2014), page 7.

[CCMTV15] Amit Chakrabarti, Graham Cormode, Andrew McGregor, Justin Thaler, and Suresh Venkatasubramanian. “Verifiable Stream Computation and Arthur–Merlin Communication”. In: *CCC*. 2015.

[CD00] Ross Cutler and Larry S. Davis. “Robust real-time periodic motion detection, analysis, and applications”. In: *IEEE Trans. PAMI* 22.8 (2000), pages 781–796.

[CFLMS04] 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), pages 1545–1551.

[CH03] Richard Cole and Ramesh Hariharan. “Faster suffix tree construction with missing suffix links”. In: *SIAM J. Comput.* 33.1 (2003), pages 26–42.

[CHL03] 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), pages 129–147.

[CJPS12] Raphaël Clifford, Markus Jalsenius, Ely Porat, and Benjamin Sach. “Pattern matching in multiple streams”. In: *Proc. 23rd CPM*. 2012, pages 97–109.

[Cla83] Kenneth L Clarkson. “Fast algorithms for the all nearest neighbors problem”. In: *Proc. 24th FOCS*. Volume 83. 1983, pages 226–232.

[CLLPPSS05] 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), pages 2554–2576.

[CM05] Graham Cormode and S Muthukrishnan. “Substring compression problems”. In: *Proc. 16th SODA*. 2005, pages 321–330.

[CM07] Graham Cormode and S Muthukrishnan. “The string edit distance matching problem with moves”. In: *ACM Trans. Algorithms* 3.1 (2007), page 2.

[CMT12] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. “Practical verified computation with streaming interactive proofs”. In: *Proc. 3rd ITCS*. 2012, pages 90–112.

[CMT13] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. “Streaming graph computations with a helpful advisor”. In: *Algorithmica* 65.2 (2013), pages 409–442.

[CN11] F. Claude and G. Navarro. “Self-indexed grammar-based compression”. In: *Fundamenta Informaticae* 111.3 (2011), pages 313–337.

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

[DDW99] Hervé Debar, Marc Dacier, and Andreas Wespi. “Towards a taxonomy of intrusion-detection systems”. In: *Computer Networks* 31.8 (1999), pages 805–822.

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

[DRSB14] Shan Du, Choudhury A Rahman, Saika Sharmeen, and Wael Badawy. “Event Detection by Spatio-Temporal Indexing of Video Clips.” In: *IJCTE* 6.1 (2014).

[DST80] Peter J Downey, Ravi Sethi, and Robert Endre Tarjan. “Variations on the common subexpression problem”. In: *J. ACM* 27.4 (1980), pages 758–771.

[DSW09] Christian Dick, Jens Schneider, and Ruediger Westermann. “Efficient Geometry Compression for GPU-based Decoding in Realtime Terrain Rendering”. In: *CGF* 28.1 (2009), pages 67–83.

[EGS08] David Eppstein, Michael T Goodrich, and Jonathan Z Sun. “Skip quadtrees: Dynamic data structures for multidimensional point sets”. In: *IJCGA* 18.01n02 (2008), pages 131–160.

[EKZ76] P. van Emde Boas, R. Kaas, and E. Zijlstra. “Design and implementation of an efficient priority queue”. In: *Theory Comput. Syst.* 10.1 (1976), pages 99–127.

[FGN14] Arash Farzan, Travis Gagie, and Gonzalo Navarro. “Entropy-bounded representation of point grids”. In: *CGTA* 47.1 (2014), pages 1–14.

[FKS84] 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), pages 538–544.

[FN85] Philippe Flajolet and G Nigel Martin. “Probabilistic counting algorithms for data base applications”. In: *JCSS* 31.2 (1985), pages 182–209.

[FP09] Maria Federico and Nadia Pisanti. “Suffix tree characterization of maximal motifs in biological sequences”. In: *Theor. Comput. Sci.* 410.43 (2009), pages 4391–4401.

[FT98] Martin Farach and Mikkel Thorup. “String Matching in Lempel–Ziv Compressed Strings”. In: *Algorithmica* 20.4 (1998), pages 388–404.

[FW93] Michael L. Fredman and Dan E. Willard. “Surpassing the information theoretic bound with fusion trees”. In: *J. Comput. System Sci.* 47.3 (1993), pages 424–436.

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

[GBBN14] Sandra Alvarez Garcia, Nieves R Brisaboa, Guillermo de Bernardo, and Gonzalo Navarro. “Interleaved K2-Tree: Indexing and Navigating Ternary Relations”. In: *Proc. DCC*. 2014, pages 342–351.

[GG98] Volker Gaede and Oliver Günther. “Multidimensional access methods”. In: *ACM CSUR* 30.2 (1998), pages 170–231.

[GGKNP12] Travis Gagie, Pawe Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. “A faster grammar-based self-index”. In: *arXiv:1209.5441* (2012).

[GJS95] P Gupta, R Janardan, and M Smid. “Further Results on Generalized Intersection Searching Problems: Counting, Reporting, and Dynamization”. In: *J. Algorithms* 19.2 (1995), pages 282–317.

[GJS97] Prosenjit Gupta, Ravi Janardan, and Michiel Smid. “A technique for adding range restrictions to generalized searching problems”. In: *Inform. Process. Lett.* 64.5 (1997), pages 263–269.

[GKNP12] Travis Gagie, Juha Kärkkäinen, Gonzalo Navarro, and Simon J Puglisi. “Colored range queries and document retrieval”. In: *TCS* (2012).

[GKPR96] Leszek Gasieniec, Marek Karpinski, Wojciech Plandowski, and Wojciech Rytter. “Randomized efficient algorithms for compressed strings: The finger-print approach”. In: *Proc. 7th CPM*. 1996, pages 39–49.

[GKPS05] Leszek Gasieniec, Roman Kolpakov, Igor Potapov, and Paul Sant. “Real-time traversal in grammar-based compressed files”. In: *Proc. 15th DCC*. 2005, page 458.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. “The knowledge complexity of interactive proof-systems”. In: *Proc. 17th STOC*. 1985, pages 291–304.

[GMW91] 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), pages 690–728.

[GPPPUV11] 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), pages 535–545.

[GPW15] Mika Göös, Toniann Pitassi, and Thomas Watson. “Zero-Information Protocols and Unambiguity in Arthur-Merlin Communication”. In: *Proc. 6th ITCS*. 2015, pages 113–122.

[GS86] Shafi Goldwasser and Michael Sipser. “Private coins versus public coins in interactive proof systems”. In: *Proc. 18th STOC*. 1986, pages 59–68.

[GSS98] Nicola Galli, Bernhard Seybold, and Klaus Simon. “Compression of Sparse Matrices: Achieving Almost Minimal Table Size”. In: *Proc. ALEX*. 1998, pages 27–33.

[Gut] Antonin Guttman. “R-trees: A dynamic index structure for spatial searching”. In: *Proc. 1984 ACM SIGMOD*. Volume 14. 2, pages 47–57.

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

[HT84a] D. Harel and R. E. Tarjan. “Fast algorithms for finding nearest common ancestors”. In: *SIAM J. Comput.* 13.2 (1984), pages 338–355.

[HT84b] D. Harel and R.E. Tarjan. “Fast algorithms for finding nearest common ancestors”. In: *SIAM J. Comput.* 13.2 (1984), pages 338–355.

[HTWM04] 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), pages 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), pages 1–14.

[HWAVM10] Simon Haegler, Peter Wonka, Stefan Mueller Arisona, Luc Van Gool, and Pascal Mueller. “Grammar-based Encoding of Facades”. In: *CGF* 29.4 (2010), pages 1479–1487.

[JL93] Ravi Janardan and Mario Lopez. “Generalized intersection searching problems”. In: *IJCGA* 3.01 (1993), pages 39–69.

[JMS05] Joseph JáJá, Christian W Mortensen, and Qingmin Shi. “Space-efficient and fast algorithms for multidimensional dominance reporting and counting”. In: *Proc. 15th ISAAC*. 2005, pages 558–568.

[Kal02] Adam Kalai. “Efficient pattern-matching with don’t cares”. In: *Proc. 13th SODA*. 2002, pages 655–656.

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

[KMSTTZ13] Alexis Kaporis, Christos Makris, Spyros Sioutas, Athanasios Tsakalidis, Kostas Tsichlas, and Christos Zaroliagis. “Improved Bounds for Finger Search on a RAM”. In: *Algorithmica* (2013), pages 1–38.

[KO91] Marc J van Kreveld and Mark H Overmars. “Divided k-d trees”. In: *Algorithmica* 6.1-6 (1991), pages 840–858.

[KP13] Hartmut Klauck and Ved Prakash. “Streaming computations with a loquacious prover”. In: *Proc. 4th ITCS*. 2013, pages 305–320.

[KP14] Hartmut Klauck and Ved Prakash. “An improved interactive streaming algorithm for the distinct elements problem”. In: *Proc. 41st ICALP*. 2014, pages 919–930.

[KR87] Richard M Karp and Michael O Rabin. “Efficient randomized pattern-matching algorithms”. In: *IBM J. Res. Dev.* 31.2 (1987), pages 249–260.

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

[KRSV07] Haim Kaplan, Natan Rubin, Micha Sharir, and Elad Verbin. “Counting colors in boxes”. In: *Proc. 18th SODA*. 2007, pages 785–794.

[KS99] KV Ravi Kanth and Ambuj Singh. “Optimal Dynamic Range Searching in Non-replicating Index Structures”. In: *Proc. 7th ICDT*. 1999, pages 257–276.

[LP01] I. Rigoutsos L. Parida and D. E. Platt. “An Output-Sensitive Flexible Pattern Discovery Algorithm”. In: *Proc. 12th CPM*. 2001, pages 131–142.

[LP12] Kasper Green Larsen and Rasmus Pagh. “I/O-efficient data structures for colored range and prefix reporting”. In: *Proc. 23rd SODA*. 2012, pages 583–592.

[Lue78] George S Lueker. “A data structure for orthogonal range queries”. In: *Proc. 19th FOCS*. 1978, pages 28–34.

[LW13] Kasper Green Larsen and Freek van Walderveen. “Near-Optimal Range Reporting Structures for Categorical Data.” In: *Proc. 24th SODA*. 2013, pages 265–276.

[LW80] DT Lee and CK Wong. “Quintary trees: a file structure for multidimensional datbase sytems”. In: *ACM TODS* 5.3 (1980), pages 339–353.

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

[ME10] Nizar R Mabroukeh and Christie I Ezeife. “A taxonomy of sequential pattern mining algorithms”. In: *ACM CSUR* 43.1 (2010), page 3.

[MN07] Veli Mäkinen and Gonzalo Navarro. “Rank and select revisited and extended”. In: *TCS* 387.3 (2007), pages 332–347.

[MN90] K. Mehlhorn and S. Näher. “Bounded ordered dictionaries in $O(\lg\lg N)$ time and $O(n)$ space”. In: *Inform. Process. Lett.* 35.4 (1990), pages 183–189.

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

[MP80] J Ian Munro and Mike S Paterson. "Selection and sorting with limited storage". In: *TCS* 12.3 (1980), pages 315–323.

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

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

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

[NV13] Yakov Nekrich and Jeffrey Scott Vitter. "Optimal color range reporting in one dimension". In: *Proc. 21st ESA*. 2013, pages 743–754.

[Ore82] Jack A Orenstein. "Multidimensional tries used for associative searching". In: *Inform. Process. Lett.* 14.4 (1982), pages 150–157.

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

[PAAV03] Octavian Procopiuc, Pankaj K Agarwal, Lars Arge, and Jeffrey Scott Vitter. "Bkd-tree: A dynamic scalable kd-tree". In: *Proc. 8th SSTD*. 2003, pages 46–65.

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

[PP09] Benny Porat and Ely Porat. "Exact and approximate pattern matching in the streaming model". In: *Proc. 50th FOCS*. 2009, pages 315–323.

[PRFPG00] 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, pages 297–308.

[PW00] Renato Pajarola and Peter Widmayer. "An image compression method for spatial search". In: *IEEE Trans. Image Processing* 9.3 (2000), pages 357–365.

[PYK06] 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, pages 432–441.

[RH04] 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*, pages 10–18.

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

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

[Sam90] Hanan Samet. *Applications of spatial data structures*. Addison-Wesley, 1990.

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

[SJ05] Qingmin Shi and Joseph JáJá. “Optimal and near-optimal algorithms for generalized intersection reporting on pointer machines”. In: *Inform. Process. Lett.* 95.3 (2005), pages 382–388.

[SKLLD08] 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, pages 1–7.

[SMHKNM94] 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), pages 639–645.

[SR06] Reza Sherkat and Davood Rafiei. “Efficiently evaluating order preserving similarity queries over historical market-basket data”. In: *Proc. 22nd ICDE*. 2006, pages 19–19.

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

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

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

[TV01] 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), pages 1–14.

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

[Wei73] P. Weiner. “Linear pattern matching algorithms”. In: *Proc. 14th SWAT*. 1973, pages 1–11.

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

[Wil83] D.E. Willard. “Log-logarithmic worst-case range queries are possible in space Θ(N)”. In: *Inform. Process. Lett.* 17.2 (1983), pages 81–84.

[ZL77] Jacob Ziv and Abraham Lempel. “A universal algorithm for sequential data compression”. In: *Information Theory, IEEE Trans. Inf. Theory* 23.3 (1977), pages 337–343.

[ZL78] Jacob Ziv and Abraham Lempel. “Compression of individual sequences via variable-rate coding”. In: *Information Theory, IEEE Trans. Inf. Theory* 24.5 (1978), pages 530–536.

[ZYHZ02] Qing Zhu, Xuefeng Yao, Duo Huang, and Yetin Zhang. “An Efficient Data Management Approach for Large Cyber-City GIS”. In: *ISPRS Archives* (2002), pages 319–323.