ISSN 2186-7437

NII Shonan Meeting Report

No. 2017-15

Computation over Compressed Structured Data

Kunihiko Sadakane (The University of Tokyo, Japan)
Gonzalo Navarro (University of Chile, Chile)

October 09–12, 2017

National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-Ku, Tokyo, Japan

Computation over Compressed Structured Data

Organizers:
Kunihiko Sadakane (The University of Tokyo, Japan)
Gonzalo Navarro (University of Chile, Chile)

October 09–12, 2017

The aim of this meeting was to bring together researchers from various areas related to compression of structured data. Thirty-five participants attended the 4-day meeting; we list them at the next page. From the 35 participants, 13 came from Asia (37%), 13 from Europe (37%), 8 from the Americas (23%) and 1 from Australia (3%). In another classification, 33 were from Academia (94%) and 2 from Industry (6%). Finally, 10 of them (29%) were students or had obtained their PhD in the last 5 years.

One of our goals as organizers was to ensure that the meeting was more than a series of talks that filled up all the available time. We then organized talks only in the mornings, leaving all the afternoons for discussion and work in groups without imposing any structure.

We solicited talks aiming at current topics of research that could stimulate collaboration during the meeting. As a result, 14 long and short talks were selected; these are listed later in the report.

In the last day, we collected the collaborations that had arised during the meeting and asked the involved researchers to write a short description of their findings. An impressive number of 18 collaborations were identified; their descriptions are also added to this report.

We believe that the meeting was a success and that it will have a significant impact in various collaborations in the near future. The survey on the opinion of the participants is similarly very positive: in a scale 1–5, 13 participants valued the seminar with a 5, 4 with a 4, and 1 with a 3. All said they learned something for their work or research. All but one would come again to Shonan. We are particularly happy from off-the-record comments of some researchers from Asia that said that this was their only opportunity to meet in person so many prestigious researchers from all over the world.

The answers of the survey also include a couple of suggestions for future improvement that are interesting. The first is that leaving the whole afternoon without structure may let people get distracted in long-term activities of other kinds, whereas distributing the talks into more blocks (not only in the morning) with shorter breaks for work could keep the participants more focused. A second one is that the use of space could be organized better to favor people gathering. For example, people would welcome more spaces where work and socialization can happen at the same time, like the small room of the coffee breaks.

We thank the NII Shonan Meeting organization for their help in making our work so easy, to the participants for their involvement and enthusiasm, and to Sankar Deep Chakraborty for helping us collect the material for this report.

List of Participants and Institutions

- Kunihiko Sadakane, The University of Tokyo
- Gonzalo Navarro, University of Chile
- Wing Kai Hon, National Tsing Hua University
- Rahul Shah, Luisiana State University
- J. Ian Munro, University of Waterloo
- Sharma Thankachan, University of Central Florida
- Johannes Fischer, TU Dortmund University
- Travis Gagie, Diego Portales University
- Simon Gog, Karlsruhe Institute of Theoretical Informatics (KIT and eBay)
- Hiroshi Sakamoto, Kyushu Institute of Technology
- Yakov Nekrich, University of Waterloo
- Philip Bille, Technical University of Denmark
- Simon Puglisi, University of Helsinki
- Markus Lohrey, University of Siegen
- Roberto Grossi, University of Pisa
- Srinivasa Rao Satti, Seoul National University
- Tetsuo Shibuya, The University of Tokyo
- Shuhei Denzumi, The University of Tokyo
- Hiroki Arimura, Hokkaido University
- Inge Li Gørtz, Technical University of Denmark
- Nicola Prezza, Technical University of Denmark
- Sebastian Maneth, University of Bremen
- Hideo Bannai, Kyushu University
- Tomohiro I, Kyushu Institute of Technology
- Sankar Deep Chakraborty, National Institute of Informatics
- Giovanni Manzini, Universita Piemonte Orientale
- Diego Arroyuelo, Universidad Técnica Federico Santa María, Chile
- Keisuke Goto, Fujitsu Laboratories
- José Fuentes-Sepúlveda, University of Chile

- Daniel Valenzuela, University of Helsinki
- Juha Kärkkäinen, University of Helsinki
- Fabian Peternek, University of Edinburgh
- Matthias Petri, University of Melbourne
- Ayumi Shinohara, Tohoku University
- Roberto Konow, eBay Inc.
- Takuya Takagi, Hokkaido University

Overview of the Talks

Wheeler Graphs: A Framework for BWT-Based Data Structures

Giovanni Manzini, University of Piemonte Orientale, Italy

Abstract: The famous Burrows-Wheeler Transform was originally defined for single strings but variations have been developed for sets of strings, labelled trees, de Bruijn graphs, alignments, etc. In this talk we propose a unifying view that includes many of these variations and that we hope will simplify the search for more. Somewhat surprisingly we get our unifying view by considering the Nondeterministic Finite Automata related to different pattern-matching problems. We show that the state graphs associated with these automata have common properties that we summarize with the concept of a Wheeler graph. Using the notion of a Wheeler graph, we show that it is possible to process strings efficiently even if the automaton is nondeterministic. In addition, we show that Wheeler graphs can be compactly represented and traversed using up to three arrays with additional data structures supporting efficient rank and select operations. It turns out that these arrays coincide with, or are substantially equivalent to, the output of many Burrows-Wheeler Transform variants described in the literature.

This is joint work with Travis Gagie and Jouni Siren.

Compressed graph processing

Kunihiko Sadakane, The University of Tokyo, Japan

Abstract: We consider network and related problems based on graph decomposition. Our algorithms first construct indices (data structures) from a given graph, then use them for solving the problems. A basic problem is the all pairs maximum flow problem. To solve the problems efficiently, we decompose the input graph into small subgraphs such as triconnected components. We want to develop efficient algorithms based on such decompositions.

Isomorphism of Unordered Compressed Trees

Sebastian Maneth, University of Bremen, UK

Abstract: The talk is based on an ICALP 2015 paper by Lohrey, Maneth, and Peternek. Isomorphism of unordered unrooted trees can be solved in linear time, as described in the 1974 book on algorithms by Aho, Hopcroft, and Ullman. An unordered tree can be compressed by applying known compression methods (for ordered trees) to an ordered version of the tree. Our choice of compression method are linear straight-line context-free tree grammars (TSLPs), which generalize the sharing of common subtrees (of DAGs) to the sharing of connected subgraphs of a tree. We show that isomorphism of two unordered trees given by TSLPs can be solved in polynomial time. The idea is to construct TSLPs for the canonical ordered trees. Canonical ordered trees are obtained by sorting subtrees of a node according to the length-lexicographical ordering of their traversal strings. This sorting can be directly carried out on the given TSLPs by exploiting a normal form and reducing sorting to binary search plus equality checks of (string) SLPs for traversal strings.

Querying regular languages over sliding-windows

Markus Lohrey, University of Siegen, Germany

Abstract: Sliding-window streaming algorithms get as input a stream of input data and have to answer queries about the last n symbols for a certain window size n. In the talk we consider queries that are given by regular languages. More precisely, we consider the so-called sliding window word problem for a regular language L: Given a data stream of symbols $a_1a_2a_3\cdots$, answer at every time instant t, whether $a_{t-n+1}\cdots a_t$ belongs to L. We are mainly interested in the space complexity of this problem measured in the window length n. For regular languages, we prove that this space complexity is either constant, logarithmic, or linear. Moreover, for the constant and logarithmic space classes we provide very natural characterizations: For every regular language L the sliding window word problem can be solved in

- constant space if and only if L is a boolean combination of regular length languages and suffix-testable languages;
- logarithmic space if and only if L is a boolean combination of regular length languages and regular left ideals.

For context-free languages the above space trichotomy does not hold: For every natural number c there is a context-free language for which the optimal space bound for the sliding window word problem is $n^{1/c}$.

This is joint work with Moses Ganardi, Danny Hucke and Konstantinos Mamouras.

Fast and compact planar embeddings

José Fuentes-Sepúlveda, University of Chile, Chile

Abstract: There are many representations of planar graphs, but few are as

elegant as Turan's (1984): it is simple and practical, uses only 4 bits per edge, can handle self-loops and multi-edges, and can store any specified embedding. Its main disadvantage has been that "it does not allow efficient searching" (Jacobson, 1989). In this talk we show how to add a sublinear number of bits to Turan's representation such that it supports fast navigation while retaining simplicity. As a consequence of the inherited simplicity, we offer the first efficient parallel construction of a compact encoding of a planar graph embedding. Our experimental results show that the resulting representation uses about 6 bits per edge in practice, supports basic navigation operations within a few microseconds, and can be built sequentially at a rate below 1 microsecond per edge, featuring a linear speedup with a parallel efficiency around 50 percent for large datasets.

Towards Constant-Delay Traversal of Grammar-Compressed Graphs

Fabian Peternek, University of Edinburgh, UK

Abstract: We present a data structure based on pointers to traverse certain hyperedge replacement graph grammars such that a single traversal step requires constant time. The general idea revolves about precomputing the possible paths in the derivation tree that can be induced by a single step. This precomputation is done in such a way that the generated structures can be efficiently combined to derive representations of successor nodes. We further give some intuition why previous methods used to traverse strings and trees are unlikely to generalize to graph grammars. The result assumes that the rank of the grammar (i.e., the amount of nodes any nonterminal hyperedge is attached to) is bounded by a constant.

Locally-Adaptive Compressed Dictionaries and Bitmaps

Diego Arroyuelo, Diego Arroyuelo, Universidad Técnica Federico Santa María, Chile

Abstract: Gap and run-length are usual ways to compress dictionaries and bitmaps. However, choosing an encoding that minimizes the space usage can be difficult in scenarios where the data has different local regularities. This talk proposes locally-adaptive data-aware measures for compressing static dictionaries and bitmaps. Unlike gap and run-length encoding, the idea is to take advantage of the local regularities that arise in the data. A locally-adaptive data-aware measure will be proposed, and shown to be smaller or equal than both gap and run-length encoding measures. The talk will also discuss how to support rank, select, and membership queries in dictionaries, while using space close to the proposed locally-adaptive measure.

Fast Locating with the RLFM-Index

Travis Gagie, Diego Portales University, Chile, and Gonzalo Navarro, University of Chile

Abstract: The run-length encoded FM-index is an adaptation of the FM-

index for repetitive datasets. It achieves excellent compression on such datasets and supports fast counting queries but locating queries have been a major weakness: we cannot sample many entries of the suffix array without ruining the compression, so current implementations often use thousands of rank queries to locate each occurrence. In this talk we will see how we can sample suffix array entries only at the endpoints of runs and still locate each occurrence in doubly-logarithmic time.

This is a joint work with Nicola Prezza.

Tree Compression and Top Trees

Philip Bille, Technical University of Denmark, Denmark

Abstract: Top tree are a simple tree compression scheme that offers strong theoretical compression guarantees, support efficient navigation, and is practical. We discuss the basic concepts, compare the scheme with other related compression schemes, and present a few new results.

Range LCP with k-Mismatches

Sharma Thankachan, University of Central Florida, USA

Abstract: A range LCP query (a, b) over a text $T[1..n]$ asks to report "largest element in $\{|LCP(T[x, n], Y[y, n])|a =< x < y =< b\}$", where LCP is the longest common prefix function. Amir et al. [ISAAC 2011] introduced this problem and presented an $O(n \log n)$ space solution with $O(poly \log(n))$ query time. We present an $O(n \log^{k+1} n)$ space and $O(poly \log(n))$ query time solution for the k-mismatch case.

Compressed Data Structure for Approximate Color Counting

Yakov Nekrich, University of Waterloo, Canada

Abstract: In this talk we describe a data structure that supports approximate color counting queries in $O(1)$ time on array $A[1..n]$. For any query range $[i..j]$, $1 \leq i \leq j \leq n$, we can estimate the number of distinct elements in subarray $A[i..j]$. Our data structure uses $O(n)$ bits of space and we do not need to store the array A.

This is a joint work with Ian Munro and Hicham El-Zein.

String attractors

Nicola Prezza, Technical University of Denmark, Denmark

Abstract: A well-known fact in the field of lossless text compression is that high-order entropy is a weak model when the input contains long repetitions. Motivated by this fact, decades of research have generated myriads of so-called

dictionary compressors: algorithms able to reduce the textâĂŹs size by exploiting its repetitiveness. Lempel-Ziv 77 is probably one of the most successful and known tools of this kind, followed by straight-line programs, run-length Burrows-Wheeler transform, macro schemes, collage systems, and the compact directed acyclic word graph. In this work, we show that these techniques are only different solutions to the same, elegant, combinatorial problem: to find a small set of positions capturing all distinct textâĂŹs substrings. We call string attractor such a set. We first show reductions between dictionary compressors and string attractors. This gives us the approximation ratios of dictionary compressors with respect to the smallest string attractor and allows us to solve several open problems related to the asymptotic relations between the output sizes of different dictionary compressors. We then show that k-attractor problem âĂŤ that is, deciding whether a text has a size-t set of positions capturing all substrings of length at most k– is NP-complete for $k \geq 3$. This, in particular, implies the NP-completeness of the full string attractor problem. We provide several approximation techniques for the smallest k-attractor, show that the problem belongs to the APX class for constant k, and give strong inapproximability results. To conclude, we use string attractors to design a universal data structure for random access on dictionary-compressed text supporting queries in near-optimal time.

This is a joint work with Dominik Kempa.

Succinct Data Structures ... the Expected Case

J. Ian Munro, University of Waterloo, Canada

Abstract: Work on succinct data structures generally focuses on finding a representation requiring a number of bits within a lower order term of lg of the number of objects of the given size. We turn our attention to minimizing the expected space requirement when given a probability distribution of the objects of the given size. In particular we look at binary trees from the distribution one gets by inserting n elements, by the naive algorithm, in random order. This is equivalent to the distribution of Cartesian trees on n random points in a rectangle. We show that about 1.736... n bits are necessary and sufficient for this to support basic navigation operations in constant time.

This is joint work with Patrick Nicholson.

Compact Order Preserving Pattern Matching

Rahul Shah, Luisiana State University, USA

Abstract: Developments in Compressed Text Indexing in last two decades have made it possible to compress the all text data structures to the information theoretically minimum space required for the original (or compressed) text data. Based on Burrows-Wheeler Transform (BWT) and LF-mapping (or alternatively Phi-function), this lead to the development of what is known as Compressed Suffix Tree (CST) which admits theoretically optimal space.

Many text indexing and sequence matching problems, use Suffix Trees with some augmenting information. One line of work in the area has been to replace Suffix Tree with a CST as a black box and then separately encode the augmenting data to squeeze it into optimal compressed space. However, less chartered territory here is to develop compressed index structures for the problems which use variants of Suffix Trees. These variants have different structural properties than regular Suffix Trees, making use of the usual notion of BWT or CST ineffective in achieving the space optimality.

Building over our previous work on Parameterized Pattern Matching, we design a compact index for order-preserving pattern matching where alphabet set is totally ordered and two strings R, S match if they represent the same permutation. That is for all i, j we have $R[i] < R[j]$ implies $S[i] < S[j]$ and $R[i] = R[j]$ implies $S[i] = S[j]$. To compute LF-mapping, we develop a new method called LF-successor.

List of Collaborative Projects

LZ77-like Parsing in Small Space using RLZ

Gonzalo Navarro, Department of Computer Science, University of Chile
Simon Puglisi, Department of Computer Science, University of Helsinki
Daniel Valenzuela, Department of Computer Science, University of Helsinki

Lempel-Ziv 77 (LZ77) is one of the most successful techniques to compress repetitive collections. Even though it can be computed in linear time, when the collections are too large to fit in main memory the only choice is to resort to external memory, resulting in prohibitive computation times. Relative Lempel-Ziv (RLZ) [46] is a variant that uses a smaller reference to factorize the data that needs to be compressed. In the original setup the data is a collection of similar genomes and one of them is used as reference to compress the whole collection. When there is no clear choice for a "good reference" a reference is built by sampling the input [47]. While RLZ works very well in practice, there is still a gap between its compression levels and what LZ77 can achieve. Some of the efforts to improve RLZ consist in look-ahead heuristics trying to avoid harmful phrase breaks [24]. In the less-structured setup of web-collections, this challenge materializes as how to build the reference in the most favourable way.

We want to use RLZ as a device to find an LZ77-like parsing using sublinear memory, even if we do not obtain the optimal LZ77 parse. The key idea is to use the RLZ parse as a small intermediate representation on top of which we can find a reasonably good LZ77-like parse. Consider for instance a collection of genomes where there are two clusters of highly similar genomes. As we choose one of the genomes as a reference, all the genomes belonging to the same cluster will get satisfactory compression, however, all the genomes from the other cluster will be compressed in a less satisfactory way. Our idea is to exploit that after computing the RLZ parsing, those genomes that did not compress well with respect to the reference will produce similar phrases. We would like to capture this by computing the LZ77 parsing of the output of RLZ. For instance, if two genomes that did not compress well are identical, their RLZ parsing will be the same, and easily captured by LZ77. Unfortunately, if those genomes have a

slight difference at the beginning, their RLZ parsing may be entirely different as the greedy strategy may always choose different sources to get the longest phrases. In such case, LZ77 would not capture their similarity at all. To avoid that those small differences produce widely different RLZ parsings, we plan to restrict the positions where phrases can end, to a multiple of some parameter.

In summary, our proposed algorithm is as follows: First we compute the (restricted) RLZ parsing. Then we compute the LZ77 parsing over the (pos, len) sequence output by the first step. Finally, we transform this last parsing back to pairs (pos, len) in the original text, so that our output is again a LZ77-like parsing.

Online LZ77 Factorization in Compressed Space

Hideo Bannai, Department of Informatics, Kyushu University, Japan
Travis Gagie, EIT, Diego Portales University, Chile
Tomohiro I, Frontier Research Academy, Kyushu Institute of Technology, Japan

Policriti and Prezza [64, 65, 63] recently showed how a run-length compressed FM-index for a text T, with suffix array (SA) samples at the boundaries of the runs in the Burrows-Wheeler Transform (BWT), can be used to build the LZ77 parse of T in $O(n \log r)$ time using $O(r)$ words of workspace, where n is the length of T and r is the number of runs in the BWT. Their key result was a lemma showing that this SA sample can be used during a backward search to find the position of at least one occurrence of the pattern. Gagie, Navarro and Prezza [32] have now shown how we can use Policriti and Prezza's lemma and an auxiliary $O(r)$-space data structure to find all the occurrences' positions efficiently.

Also recently, Ohno, Takabatake, I and Sakamoto [62] described a new algorithm for updating a run-length compressed BWT as characters are appended to the underlying string, which uses $O(n \log r)$ time and $O(r)$ workspace to build online the run-length compressed BWT of T. Notably, they have implemented their algorithm and shown that it is practical. Extending his work with Navarro and Prezza, Gagie developed an unpublished algorithm for updating Policriti and Prezza's SA sample as characters are appended to the underlying string, with the same time- and space-bounds as Ohno et al.'s algorithm for updating the BWT.

During the Shonan seminar and subsequent discussions, we realized that combining Ohno et al.'s and Gagie's algorithms lets us maintain online a run-length compressed FM-index. Combining that with Policriti and Prezza's technique, we obtain the first online algorithm for building the LZ77 parse of T in $O(n \log r)$ time and $O(r)$ workspace. We also found an alternative version of Policriti and Prezza's lemma, which is conceptually slightly simpler and could be more practical. I has now implemented this combined algorithm and found it to be practical. We plan to incorportate Gagie's algorithm and our new results in the journal version of Ohno et al.'s paper.

Top-trees for Suffix Trees of Repetitive Collections

Phil Bille, DTU Compute, Technical University of Denmark
Inge Li Gørtz, DTU Compute, Technical University of Denmark

Gonzalo Navarro, Department of Computer Science, University of Chile

Compressed Suffix Trees (CSTs) typically have three main components [68]: (1) a Compressed Suffix Array (CSA), (2) a Longest Common Prefix (LCP) array, and (3) a compressed topology representation. On repetitive text collections, the size of the CSA component can be drastically reduced by exploiting runs [51], especially with the latest developments that do not require a regular text sampling in order to locate pattern occurrences [28]. The other two components can also be reduced significantly on repetitive texts [1, 59] but, so far, not as much as the CSA. Besides, while it is possible to avoid representing the tree topology directly, this has resulted in an orders-of-magnitude-slower representation [1, 59].

One way to compress the tree topology is to detect isomorphic subtrees and factor them out. This is essentially what the CDAWG-based suffix trees do [9, 10, 8], and it is also close to what is obtained with a grammar-based compression of the parentheses representation of the topology [16, 59]. As said, this compresses the suffix trees of repetitive sequences significantly, but not as much as the way runs compress the CSA. Since we are in both cases representing the same combinatorial object, it seems the compression of the topology is missing some key aspect that is being captured in the compression of the CSA.

We plan to experiment with a technique that captures more than identical subtrees. Top-tree compression [15, 14] is able to capture repeated internal parts of the trees as well. While not as powerful as tree grammars [49], top-tree compression retains better operation times, usually logarithmic. This is comparable with the time obtained with CDAWGs and grammar-compressed sequences, so compression with top-trees seems promising in principle.

It must be noted that, unlike for whole subtrees, it is not obvious which regularities should be revealed as repeated internal subtrees, and thus the nature of this research is completely exploratory. If we turn out to capture repeated substructures that were not exploited before, we will study which new regularities they owe to.

Top-tree compression compresses labeled trees, which would be very convenient for storing the letter of the incoming edge of each node. This speeds up the operation of descending to a child, which is usually very slow in CSTs. Instead, we could discard this information in order to further improve compression.

Faster Sequence Alignment using Relative Lempel-Ziv

Gonzalo Navarro, Department of Computer Science, University of Chile
Daniel Valenzuela, Department of Computer Science, University of Helsinki

Consider a large set of genomes from a single or closely related species. Relative Lempel-Ziv (RLZ) [46] is a compressed representation technique that chooses a *representative* sequence (e.g., one of the genomes, in this case) and then describes the others as a sequence of *blocks* copied from the representative, plus a few explicit symbols. It works particularly well in this case and allows fast extraction of any substring from any sequence.

Our idea is to use RLZ to perform rapid sequence alignment in the genomes. This includes computation of longest common subsequences, edit distances, local alignments, etc. in pairs of sequences or even within the same sequence. In all

those cases, one fills a dynamic programming matrix $M[1..n][1..n']$ to align two sequences $S[1..n]$ and $S'[1..n']$, typically at cost $O(n \cdot n')$.

In our case, since both sequences are aligned to the representative, we can speed up the computation of large areas of M by first aligning the reference $R[1..r]$ to itself, in a matrix $M_R[1..r][1..r]$. Consider the case of local sequence alignment. If, at some point of the computation, we have to fill $M[a..b][a'..b']$ and it turns out that $S[a..b] = R[x..y]$ is a block and $S'[a'..b'] = R[x'..y']$ is another block, then most of the area $M[a..b][a'..b']$ will be equal to $M_R[x..y][x'..y']$, except for possibly a short band below the row $M_R[x][x'..y']$ and to the right of the column $M_R[x..y][x']$. The same is likely to happen for global similarity measures if we represent M_R in some differential form. We can then proceed block-wise in both S and S' and work only around the perimeters of the areas.

This idea has been considered before [23] to obtain subquadratic sequence alignment algorithms, by using LZ78 compression [73] on S and on S' separately. If they obtained z and z' phrases on S and S', respectively, then their total time was $O(nz' + n'z)$. While with RLZ we might not obtain worst-case bounds, we also expect the alignment cost to be close to that, where z and z' are now the number of RLZ blocks into which S and S' are decomposed, respectively.

A Linear-Space Poly-Logarithmic Query-Time Data Structure for Range LCP Queries

Wing-Kai Hon, National Tsing Hua University, Taiwan
Yakov Nekrich, University of Waterloo, Canada
Kunihiko Sadakane, The University of Tokyo, Japan
Rahul Shah, Louisiana State University, USA
Sharma V. Thankachan, University of Central Florida, USA

Let T be a text of length n and T_i be its ith longest suffix. A range LCP query (α, β) on T asks to report $\max\{|\mathsf{LCP}(T_i, T_j)| \mid \alpha \leq i < j \leq \beta\}$, where $\mathsf{LCP}(T_i, T_j)$ is the longest common prefix T_i and T_j. Amir et al. [ISAAC 2011] proposed an $O(n \log^{1+\epsilon} n)$ space structure with $O(\log \log n)$ query time for this problem. Additionally, they presented a linear space structure with query time $O(d \log \log n)$, where $d = \beta - \alpha + 1$. Later Patil et al. [SPIRE 2013] improved its query time to $O(\sqrt{d} \log^{\epsilon} d)$. We revisit this problem and present a linear space data structure with query time $O(\log^{1+\epsilon} n)$.

Parameterized Text Indexing with One Wildcard

Wing-Kai Hon, National Tsing Hua University, Taiwan
Rahul Shah, Louisiana State University, USA
Sharma V. Thankachan, University of Central Florida, USA

Let X and Y are two equal-length strings over an alphabet set Σ of size σ. We say X and Y is a parameterized match iif X can be transformed to Y by renaming the character $X[i]$ to the character $Y[i]$ for $1 \leq i \leq |X|$ by using a one-to-one function from the set of symbols in X to the set of symbols in Y. The parameterized text indexing problems is defined as follows: Index a text T of n characters over an alphabet set Σ of size σ, such that whenever a pattern $P[1, p]$ comes as a query, we can report all *occ* parameterized occurrences of

P in T. A position $i \in [1, n]$ is a parameterized occurrence of P in T, iff P and $T[i..(i + p - 1)]$ is a parameterized match. Such queries can be answered in optimal $O(p + occ)$ time using a linear space data structure, knows as the parameterized suffix tree [Baker, STOC 1993]. Recently, Ganguly et al. [SODA 17] proposed a space efficient data structure of $n \log \sigma + O(n)$ bits and $O((p + occ \cdot \log n) \log \sigma)$ query time.

We study an interesting generalization of this problem, where the pattern contains a single wild-card character $\phi \notin \Sigma$. The wild-card character can match with any other character in Σ. We show that such queries can be answered in optimal $O(p + occ)$ time using an $O(n \log n)$ space index. We then show how to compress our index into $O(n \log \sigma)$ words, but with a higher query cost.

Recognizing Wheeler Graphs

Jarno Niklas Alanko, University of Helsinki, Finland
Travis Gagie, Diego Portales University, Chile
Giovanni Manzini, University of Piemonte Orientale, Italy
Tuukka Norri, University of Helsinki, Finland

Gagie, Manzini and Sirén [30] recently introduced Wheeler graphs as a framework for designing data structures based on the Burrows-Wheeler Transform. A Wheeler graph is a directed edge-labelled graph with the property that there is an ordering of the nodes such that the edges' order by destination is the same as their order by label with ties broken by origin. Gagie et al. showed how to index the path labels in a Wheeler graph using little memory such that we can answer pattern-matching queries on them quickly, and showed that the graphs arising in several applications are Wheeler graphs. However, they left as an open problem designing a polynomial-time algorithm that, given an arbitrary graph, decides whether or not it is a Wheeler graph (and ideally, if it is, returns an ordering of the nodes as a witness).

We have been discussing generalizing the Karp-Miller-Rozenberg "doubling algorithm" for building a suffix array (which is the node-ordering for a cyclic Wheeler graph), to compute an order for a Wheeler graph. Our idea is that we can order nodes based on the co-lexicographically least and greatest path labels leading to them: if α_u, β_u, α_v and β_v are the co-lexicographically least and greatest path labels leading to nodes u and v, then we think

- if $\alpha_u = \alpha_v$ and $\beta_u = \beta_v$ then u and v can be assigned any order with respect to each other
- otherwise, if $\alpha_u \preceq \beta_u \preceq \alpha_v \preceq \beta_v$ but $\alpha_u \neq \alpha_v$, then u must precede v in the ordering
- otherwise, the graph is not a Wheeler graph.

Computing the co-lexicographically least and greatest path labels leading to each node is complicated by the fact that in general graphs, unlike cycles, nodes can have many predecessors and we must keep track of *all* the nodes before a node v at the distance we are currently considering, from which we can reach v along the co-lexicographically least and greatest paths. This seems to require us to compute distance matrices by repeated squaring, slowing the algorithm down from $O(n \log n)$ to $O(n^3 \log n)$ (using naïve Boolean-matrix multiplication).

We have also been considering expressing this problem as a constraint-satisfaction problem using Horn clauses — enforcing a total order on the nodes satisfying the requirements stated above — which can then be solved in time linear in the total formula size, $O(|V|^3 + |E|^2)$, where V and E are the vertex and edge sets of the input graph. Among other things, this suggests that we can reduce suffix sorting to a tractable logic problem; although it may not be practical, this seems interesting.

Approximate Document Counting on Compressed Repetitive Collections

Travis Gagie, Diego Portales University, Chile
Gonzalo Navarro, Department of Computer Science, University of Chile
Yakov Nekrich, University of Waterloo, Canada
Nicola Prezza, Technical University of Denmark, Denmark

Munro, Nekrich and El-Zein [57] recently showed how, given an array $A[1..n]$ and a positive constant ϵ, we can store $n\,\mathrm{polylog}(n)$ words such that later, given i and j, in constant time we can return a number between $|\{x : x \in A[i..j]\}|$ and $(1+\epsilon)\,|\{x : x \in A[i..j]\}|$. It follows that, given a collection of documents of total length n over an alphabet of size $\sigma \in n^{\mathrm{polylog}(n)}$ and a positive constant ϵ, we can store $n\,\mathrm{polylog}(n)$ words such that later, given a pattern $P[1..m]$, in $O(m \log\log \sigma)$ time we can return a number between the number of documents containing P and $1+\epsilon$ times that number. To do this, we store an FM-index for counting (i.e., without a suffix-array sample) for the collection and an instance of their data structure for the document array; given P, we use the former to find P's interval in the suffix array and the latter to estimate the number of distinct elements in the corresponding interval of the document array.

Combining their data structure with Kempa and Prezza's [44] generalization of block trees to string attractors, we obtain the following result: given A, a string attractor of size s for A and a positive constant ϵ, we can store $s\,\mathrm{polylog}(n)$ words such that later, given i and j, in $\mathrm{polylog}(n)$ time we can return a number between $|\{x : x \in A[i..j]\}|$ and $(1+\epsilon)\,|\{x : x \in A[i..j]\}|$. That is, we can reduce the space bound by a factor of n/s at the cost of increasing the time bound on queries to $\mathrm{polylog}(n)$. To do this, we use the generalized block tree to find an occurrence of $A[i..j]$ that includes an element in the string attractor, then estimate the number of distinct elements in that occurrence. It follows that we can tighten the our space bound for approximate document counting to $(r+s)\,\mathrm{polylog}(n)$, where r is the number of runs in the Burrows-Wheeler Transform (BWT) of the collection and s is the size of a given string attractor for the document array. To do this, we store a run-length compressed FM-index for counting for the collection, a generalized block tree for the string attractor of the document, and an instance of Munro, Nekrich and El-Zein's data structure for the string attractor.

If we take the union of set of positions of characters at the ends of runs in the BWT and the set of positions of the first characters in each document, then we obtain a string attractor of size at most $d + r$, where d is the number of documents in the collection. To see why, consider that if an interval $D[i..j]$ of the document array does not contain an element of the resulting set, then none of the characters in $\mathrm{BWT}[i..j]$ are the first character in a run in BWT or the

first character in a document, so $D[\mathrm{LF}[i]..\mathrm{LF}[j]]$ is equal to $D[i..j]$. By repeated application of the LF function, eventually we reach an occurrence that includes an element of the set, so the set is a string attractor. With this string attractor, we obtain a $(d + r)\,\mathrm{polylog}(n)$-space data structure for approximate document counting with $\mathrm{polylog}(n)$ query time.

Locally-Adaptive Compressed Dictionaries

Diego Arroyuelo, Department of Informatics, Universidad Técnica Federico Santa María, Chile
Juha Kärkkäinen, Department of Computer Science, University of Helsinki
Srinivasa Rao Satti, Department of Computer Science and Engineering, Seoul National University

Dictionaries are one of the most fundamental data-structure problems: Let $\mathcal{U} = \{1, \ldots, u\} \subset \mathbb{N}$ be a universe and $\mathcal{S} = \{x_1, \ldots, x_n\} \subseteq \mathcal{U}$ a set of n elements, for $1 \leq x_1 < \cdots < x_n \leq u$, we want to support membership queries on $\mathcal{S}$: given $x \in \mathcal{U}$, does $x \in \mathcal{S}$? A fully-indexable dictionary (FID) also supports the fundamental operations:

- $\mathsf{rank}(\mathcal{S}, x)$: given an element $x \in \mathcal{U}$, yields $|\{x_j \in \mathcal{S},\ x_j \leq x\}|$, and
- $\mathsf{select}(\mathcal{S}, j)$: given $j \in \mathbb{N}$, yields x_j.

Given the amount of data managed by many applications nowadays, it is also important to support these operations using as less space as possible. Hence, the succinct and compressed representation of FIDs has been an important research topic in the last decades [58].

Typically, compression proceeds by first choosing a compression model, and then encoding the data using that model. Different models try to take advantage of different regularities that arise in the data. Typical examples are entropy-compressed [67], gap-compressed [69, 50, 39], and run-length-compressed [58] FIDs. However, in practice, data has local regularities, meaning that different compression models would be more adequate for different parts of the data. Hence, there has been a proliferation of hybrid approaches in the literature, where basically the data is divided into blocks, and then each block is compressed using the most suitable model for it. The main research communities where this has been done are:

- Information Retrieval: indexing of versioned document collections [19], and reordered document collections [3],
- Databases: hybrid compression schemes for bitmap indexes, like WAH [72] and relatives, and
- Succinct Data Structures: hybrid bit vectors, mostly used to encode the Burrows-Wheeler transform of repetitive text collections [43, 41].

This collaboration aims at exploring alternative hybrid encodings for locally-adaptive dictionaries. In his talk, D. Arroyuelo proposed a model where gap and run-length encodings are combined to take the best from both worlds. However, it is not clear if there are more efficient combinations. We propose to pursue

this line. In particular, a scheme that seems promising is the compression based on variable-to-fixed encoding [42]. Another interesting line of research is that of streaming compression of dictionaries: we must compress a set without knowing the universe size nor the set size, using a limited amount of main memory and with a single pass on the data. Since the dictionary is not known in advance, a compression model that is able to automatically adapt to the data regularities is desirable. There has been some progress on streaming data compression, as for instance the computation of the LZ78 parsing of a text [2]. However, more work needs to be done in this line.

Run-length FM indexes made practical

Travis Gagie, EIT, Diego Portales University, Chile
Simon Gog, Karslruhe Insitute of Technology
Nicola Prezza, DTU Compute, Technical University of Denmark

The FM-index [27] is, to date, the gold standard for full-text searches on entropy-compressible datasets in many domains. Lately, the rise of repetitive datasets has however generated a lot of interest in compressed self-indexes able to beat the entropy lower-bound by exploiting the dataset's repetitiveness. A typical example is represented by the advances in next-generation DNA sequencing: more and more genomes are being sequenced, and the ability to store and index them in compressed format is of crucial importance in bioinformatics. Run-length FM indexes have been one of the first solutions able to efficiently solve this problem: repetitions in the dataset translate to long equal-letter runs in its Burrows-Wheeler transform, therefore run-length compression of this text transformation can reduce the size of the index by orders of magnitude. The state-of-the art index belonging to this family — the RLCSA [55] — reports competitive speeds w.r.t. entropy-compressed FM-indexes while at the same time taking orders of magnitude less space. However, one particular weakness of the RLCSA prevents it to be fully practical on extremely repetitive datasets: in order to being able to locate each pattern occurrence in time proportional to s, the RLCSA needs a suffix array sampling of size n/s, n being the dataset's size. On extremely repetitive datasets, s needs to be large in order to keep space usage under control, with the result that locate queries are very slow.

Very recently, Gagie, Navarro, and Prezza [33] solved this long-standing problem by showing how to sample $2r$ suffix array entries (r being the number of runs in the BWT of the dataset) while supporting locate queries in log-logarithmic time each. Preliminary results showed that their index is up to three orders of magnitude faster than the RLCSA, while using comparable space. This solution is even faster — both asymptotically and in practice — than classical FM-indexes, where the locate time is usually poly-logarithmic (the price to pay being space in the case the text is not compressible at all). Given the practical relevance of this new index, we expect that an optimized implementation will be of great importance in any area currently making use of (classic) FM-indexes. The sdsl C++ library [36] is, to date, the gold-standard for implementing space-efficient data structures for string processing. The library already collects the most efficient implementations of (entropy/run-length) FM-indexes; in order to make an efficient implementation of the run-length FM index [33] available to the community, a natural step to take now is therefore to integrate it in the sdsl

library, which is precisely the goal of this collaboration started at the Shonan meeting.

Implementing linear-time BWT construction in $O(n \lg \sigma)$ bits

José Fuentes-Sepúlveda, Department of Computer Science, University of Chile
Gonzalo Navarro, Department of Computer Science, University of Chile
Yakov Nekrich, Cheriton School of Computer Science, University of Waterloo

The *Burrows-Wheeler transform (BWT)* is a central tool for several compression algorithms and compact data structures. The transform exploits the high-order entropy of sequences, generating a suitable input for compression techniques. In the last years, sequential and parallel algorithms have been proposed to construct the BWT. Sequentially, the transform can be constructed in $O(n)$ time and $O(n \lg \sigma)$ bits of space [25, 56], where n is the length of the sequence and σ is the alphabet size. In parallel, the transform can be constructed in $O(n + \sigma^{2 \lg^* n})$ sequential time, $O(\lg^2 n)$ parallel time and $O(n^2 + \sigma^{2 \lg^* n})$ bits of space [26], or $O(n \lg^2 n)$ sequential time, $O(\lg \sigma \lg^5 n)$ parallel time and $O(n \lg \lg n + p \lg n)$ bits of space [40], where p is the number of available cores. Recently, in the context of bioinformatics, Liu et al. [48] presented a new algorithm to compute the Burrows-Wheeler Transform, called *deBWT*. The introduced algorithm represents and organizes the suffixes of the input text using Bruijn graphs, facilitating the comparison between suffixes with a long common prefix. Alternatively, parallel algorithms to construct suffix arrays can be used to obtain the BWT using $O(n \lg n)$ bits of space.

We plan to provide the first practical implementation and experimental study of the linear-time and linear-space algorithm of Munro et al. [56]. The algorithm computes the BWT by dividing the input sequence into segments of size $\Delta = \lg_\sigma n$ and then incrementally constructing the BWT in Δ steps. Additionally, we will discuss the parallelization of the algorithm of Munro et al. retaining the linear space consumption. Finally, we will provide an implementation of our parallel algorithm for multicore architectures.

The main challenge is to find practical alternatives to some theoretical solutions that are used in the original paper to find the desired bound. We will focus on the most interesting case, $\sigma = O(\text{polylog } n)$, and will proceed by physically inserting the n/Δ new symbols in the current BWT array in each step. By using bit-parallel operations and multiary wavelet trees, we expect to retain the $O(n)$ construction time within $O(n \log \sigma)$ bits of space. We also plan to exploit multi-core concurrency to speed up the insertions of all the symbols across the BWT array.

Succinct Run-length Encoded Rank/Select Data Structure

José Fuentes-Sepúlveda, Department of Computer Science, University of Chile
Juha Kärkkäinen, Department of Computer Science, University of Helsinki
Dmitry Kosolobov, Department of Computer Science, University of Helsinki
Simon J. Puglisi, Department of Computer Science, University of Helsinki

Succinct representations of sequences with rank and select support have been

a fundamental component of practical and theoretical results on compressed full-text indexes. Several rank/select representations of general sequences have been proposed, representing a sequence in optimal space and supporting efficient queries [38, 37, 6, 22, 54]. However, in some applications, such as the FM-index, the sequences that we need to represent may contain large subsequences of one symbol. Such subsequences are called *runs*. In a more general application, the *Burrows-Wheeler Transform (BWT)* of a repetitive sequence is more likely to contain large runs. In such cases, it is possible of design rank/select structures with better complexities than for general sequences. For example, see [32, 53, 12, 52].

In this collaborative work we propose a new succinct representation of sequences with runs and rank/select support. Given a sequence of length n, r runs and an alphabet of size σ, our representation uses $r \log \frac{n\sigma}{r} + o(r \log \frac{n\sigma}{r})$ bits, where $2 \leq \sigma \leq r \leq n/\log^{\omega(1)} n$, and supports rank in $O(\log \frac{\log(n\sigma/r)}{\log \log n})$ time and select in $O(\log \frac{\log(n/r)}{\log \log n})$ time. Additionally, our structure supports access query with the same bounds than select. The rank, select and access queries are time optimal whenever $r \geq 2^{\log^{\delta} n}$, for an arbitrary positive constant δ.

We also provide an implementation of the proposed structure. In preliminaries experiments we compare our structure with the state of the art, showing that the closest competitors consume on 31%-46% more space at the cost of increasing the query time.

Universal Compression with Access and Indexing

Gonzalo Navarro, Department of Computer Science, University of Chile
Nicola Prezza, Technical University of Denmark, Denmark

The rise of repetitive datasets has lately generated a lot of interest in compressed self-indexes based on dictionary compression. For each such compression scheme, several different indexing solutions have been proposed. To date, the fastest indexes for repetitive texts are based on the run-length compressed Burrows-Wheeler transform [31] and on the Compact Directed Acyclic Word Graph [11]. The most space-efficient indexes, on the other hand, are based on the Lempel-Ziv parsing [29, 7] and on grammar compression [20, 21]. Recently, a fast and practical index based on Block Trees [13] has been proposed by Navarro [60]. The space of this index can be bounded in terms of the size of the Lempel-Ziv parsing of the text. Indexes for more universal schemes such as collage systems and macro schemes have not yet been proposed.

Very recently, Prezza showed in [66] that all dictionary compressors fall under a very general and universal scheme: they can all be seen as approximations of the smallest string attractor, i.e. a set of text positions capturing all distinct substrings. Despite the simplicity of this definition, it can be shown that the property underlying string attractors is sufficient to design a universal data structure for random access supporting queries in near-optimal time [66] (that is, close to a known lower bound). This goal is achieved by generalizing the idea underlying Block Trees [13] so that it works on any string attractor. Importantly, this data structure is universal: we can build it on top of any dictionary-compressed text representation.

The goal of this work is to explore if the generality of string attractors can

be exploited also to design a universal compressed index, i.e. an index that can be built on top of any dictionary- compressed text representation. Given that Block Trees can be enhanced to support indexing queries [60], we expect this to be possible. Moreover, we will explore if the simple property underlying string attractors is sufficient also to compress components such as the suffix array, inverse suffix array, and LCP array. To this end, we will develop ideas first explored in [31], namely: text repetitions generate corresponding repetitions in the differential SA, ISA, LCP arrays. These repetitions can be captured around attractor positions with the extended Block Tree presented in [66] for random access on text. As an ultimate goal, this will yield a universally-compressed fully-functional suffix tree.

Indexing the Bijective BWT

Hideo Bannai, Kyushu University, Fukuoka, Japan
Juha Kärkkäinen, University of Helsinki, Finland
Dominik Köppl, Department of Computer Science, TU Dortmund, Germany
Marcin Piątkowski, Nicolaus Copernicus University, Toruń, Poland

The Burrows-Wheeler transform (BWT) [17] is a reversible transformation permuting all symbols of a given string s. The output is formed by the characters preceding each suffix in the lexicographical order of all suffixes of s. Such an operation tends to group identical characters together, which has many applications in data compression and text indexing.

Notice that all conjugates (cyclic rotations) of a given string share the same BWT. Moreover, some strings cannot be considered as valid BWTs (e.g. `bccaab` cannot be reversed). However, one can consider a bijective version of BWT [35, 45] based on the Lyndon factorization [18] of the input string. In this case the output consists of the last symbols of the lexicographically sorted cyclic rotations of all Lyndon factors of the input. Since each string has a unique factorization into lexicographically nonincreasing Lyndon words such a transformation induces a bijection between strings of a given length n and multisets of Lyndon words of total length n.

It is well known that the traditional BWT can be used as a text index by implementing LF-mapping computations [61]; Given a pattern p and the traditional BWT of T, the occurrences of p in a text T can be computed with $O(|p|)$ LF-mapping computations. In this light, one may ask whether it is possible to build similar index data structures by exchanging the traditional BWT with the bijective variant.

We previously answered the question affirmatively [5], showing that the search on the bijective BWT can be conducted with $O(|p|m)$ LF-mapping computations, where m is the size of the Lyndon factorization of p, which can be $O(|p|)$ in the worst case, i.e., the algorithm requires $O(|p|^2)$ LF-mapping computations in the worst case. Based on discussions during and after the Shonan seminar, we now believe that the search on the bijective BWT can be conducted with $O(|p|m')$ LF-mapping computations, where m' is the number of distinct factors in the Lyndon factorization of the longest pre-Lyndon suffix of p and is actually in $O(\log |p|)$, thus reducing the number of LF-mapping computations

to $O(|p| \log |p|)$. The algorithm and analysis are based on combinatoric properties of Lyndon words and the bijective BWT, and we plan to write a paper describing these results.

Compressing the LCP array of parameterized suffix trees

Kunihiko Sadakane, The University of Tokyo, Japan
Sharma V. Thankachan, University of Central Florida, USA

The suffix trees [71] are data structures for indexing strings. For a string of length n, its suffix tree uses $\mathrm{O}(n \log n)$ bits of memory. The compressed suffix trees [70] are compressed version of the suffix trees and use $\mathrm{O}(n \log \sigma)$ bits where σ is the alphabet size of the string. The parameterized suffix trees [4] are variants of the suffix trees which are used for parameterized pattern matching. The size of the parameterized suffix tree is $\mathrm{O}(n \log n)$ bits. Though there exist a linear size ($\mathrm{O}(n \log \sigma)$ bits) data structure for parameterized pattern matching [34], it does not represent the parameterized suffix tree. To represent it in linear space, we need to compress the LCP (longest common prefix) array into $\mathrm{O}(n)$ bits, and this is an open problem.

Algorithms for Tree Pattern Matching

Phil Bille, DTU Compute, Technical University of Denmark
Inge Li Gørtz, DTU Compute, Technical University of Denmark
Sebastian Maneth, Universität Bremen

Trees are among the most common and well-studied combinatorial structures in computer science. The problem of comparing trees occurs in areas as diverse as structured text data bases (XML), computational biology, compiler optimization, natural language processing, and image analysis. A labeled tree T is a rooted, ordered tree, where each node has a label from an alphabet. In the tree pattern matching problem the problem is to find all occurrences of a given tree pattern P in a tree T. We have been working on improving the best known tree pattern matching algorithms using ideas from data structures and string matching.

Grammar-Compressed Query Reporting

Phil Bille, DTU Compute, Technical University of Denmark
Johannes Fischer, Universität Dortmund
Inge Li Gørtz, DTU Compute, Technical University of Denmark
Markus Lohrey, Universität Siegen
Sebastian Maneth, Universität Bremen

Grammar-based compression offers a mathematically clean abstraction of important compression schemes. There are several known methods of selecting positions of a string (or nodes of a tree), that can be performed efficiently directly over the grammar, e.g., finding matches of a given pattern, or finding the positions where a given finite automaton reaches a certain state. The matching positions can be compactly represented using a grammar again. In this project

we are interested in indexing such a "grammar of match positions" to speed-up certain operations. For instance, we would like to be able to enumerate match position with only constant-delay between each reported position.

Top Tree Compression

Phil Bille, DTU Compute, Technical University of Denmark
Inge Li Gørtz, DTU Compute, Technical University of Denmark
Roberto Grossi, Department of Computer Science, University of Pisa

A labeled tree T is a rooted, ordered tree, where each node has a label. Labelled trees are one of the most frequently used nonlinear data structures in computer science, appearing in the form of e.g. suffix trees, XML files, tries, and dictionaries. These trees are frequently very large, prompting a need for compression for on-disk storage. Top-tree compression [15, 14] is a compression method for trees that is able to capture repeated subtrees and tree structure repeats and that can compress exponentially better than DAG compression. We have been discussing new potential uses of top tree compression by applying them to parse trees and sequences. The discussion is still preliminary as we are planning some experiments to see if using top trees improves the compression in this scenario.

References

[1] A. Abeliuk, R. Cánovas, and G. Navarro. Practical compressed suffix trees. *Algorithms*, 6(2):319–351, 2013.

[2] D. Arroyuelo, R. Canovas, G. Navarro, and R. Raman. LZ78 compression in low main memory space. In *Prof. of 24th International Symposium on String Processing and Information Retrieval (SPIRE)*, pages 38–50, 2017.

[3] D. Arroyuelo, S. González, M. Oyarzún, and V. Sepulveda. Document identifier reassignment and run-length-compressed inverted indexes for improved search performance. In *Proc. of 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 173–182, 2013.

[4] Brenda S. Baker. Parameterized Pattern Matching: Algorithms and Applications. *Journal of Computer and System Sciences*, 52(1):28–42, feb 1996.

[5] Hideo Bannai, Juha Kärkkäinen, Dominik Köppl, and Marcin Piątkowski. Indexing the bijective BWT. WCTA 2017 (12th Workshop on Compression, Text and Algorithms), 2017.

[6] Jérémy Barbay, Francisco Claude, Travis Gagie, Gonzalo Navarro, and Yakov Nekrich. Efficient fully-compressed sequence representations. *Algorithmica*, 69(1):232–268, May 2014.

[7] Jérémy Barbay and Gonzalo Navarro. On compressing permutations and adaptive sorting. *Theor. Comput. Sci.*, 513:109–123, 2013.

[8] D. Belazzougui and F. Cunial. Representing the suffix tree with the CDAWG. In *Proc. 28th CPM*, LIPIcs 78, page article 7, 2017.

[9] D. Belazzougui, F. Cunial, T. Gagie, N. Prezza, and M. Raffinot. Composite repetition-aware data structures. In *Proc. 26th CPM*, LNCS 9133, pages 26–39, 2015.

[10] D. Belazzougui, F. Cunial, T. Gagie, N. Prezza, and M. Raffinot. Practical combinations of repetition-aware data structures. *CoRR*, abs/1604.06002, 2016.

[11] Djamal Belazzougui and Fabio Cunial. Fast label extraction in the CDAWG. In *String Processing and Information Retrieval - 24th International Symposium, SPIRE 2017, Palermo, Italy, September 26-29, 2017, Proceedings*, pages 161–175, 2017.

[12] Djamal Belazzougui, Fabio Cunial, Travis Gagie, Nicola Prezza, and Mathieu Raffinot. *Composite Repetition-Aware Data Structures*, pages 26–39. Springer International Publishing, Cham, 2015.

[13] Djamal Belazzougui, Travis Gagie, Pawel Gawrychowski, Juha Kärkkäinen, Alberto Ordóñez Pereira, Simon J. Puglisi, and Yasuo Tabei. Queries on lz-bounded encodings. In *2015 Data Compression Conference, DCC 2015, Snowbird, UT, USA, April 7-9, 2015*, pages 83–92, 2015.

[14] P. Bille, F. Fernstrøm, and I. Li Gørtz. Tight bounds for top tree compression. In *Proc. 24th SPIRE*, pages 97–102, 2017.

[15] P. Bille, I. Li Gørtz, G. M. Landau, and O. Weimann. Tree compression with top trees. *Inf. Comput.*, 243:166–177, 2015.

[16] P. Bille, G. M. Landau, R. Raman, K. Sadakane, S. S. Rao, and O. Weimann. Random access to grammar-compressed strings and trees. *SIAM J. Comp.*, 44(3):513–539, 2015.

[17] M. Burrows and D. J. Wheeler. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, Palo Alto, California, 1994.

[18] Kuo Tsai Chen, Ralph H. Fox, and Roger C. Lyndon. Free differential calculus, IV. The quotient groups of the lower central series. *Annals of Mathematics*, pages 81–95, 1958.

[19] F. Claude, A. Fariña, M. A. Martinez-Prieto, and G. Navarro. Universal indexes for highly repetitive document collections. *Information Systems*, 61:1–23, 2016.

[20] Francisco Claude and Gonzalo Navarro. Self-indexed grammar-based compression. *Fundam. Inform.*, 111(3):313–337, 2011.

[21] Francisco Claude and Gonzalo Navarro. Improved grammar-based compressed indexes. In *String Processing and Information Retrieval - 19th International Symposium, SPIRE 2012, Cartagena de Indias, Colombia, October 21-25, 2012. Proceedings*, pages 180–192, 2012.

[22] Francisco Claude and Gonzalo Navarro. *The Wavelet Matrix*, pages 167–179. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[23] M. Crochemore, G. M. Landau, and M. Ziv-Ukelson. A subquadratic sequence alignment algorithm for unrestricted scoring matrices. *SIAM J. Comp.*, 32(6):1654–1673, 2003.

[24] Sebastian Deorowicz and Szymon Grabowski. Robust relative compression of genomes with random access. *Bioinformatics*, 27(21):2979–2986, 2011.

[25] Djamal Belazzougui, Fabio Cunial, Juha Kärkkäinen, and Veli Mäkinen. Linear-time string indexing and analysis in small space. *CoRR*, 2016.

[26] James A. Edwards and Uzi Vishkin. Parallel algorithms for burrowsâĂŞwheeler compression and decompression. *Theoretical Computer Science*, 525:10 – 22, 2014. Advances in Stringology.

[27] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 390–398. IEEE, 2000.

[28] T. Gagie, G. Navarro, and N. Prezza. Optimal-time text indexing in BWT-runs bounded space. *CoRR*, abs/1705.10382, 2017.

[29] Travis Gagie, Pawel Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. Lz77-based self-indexing with faster pattern matching. In *LATIN 2014: Theoretical Informatics - 11th Latin American Symposium, Montevideo, Uruguay, March 31 - April 4, 2014. Proceedings*, pages 731–742, 2014.

[30] Travis Gagie, Giovanni Manzini, and Jouni SirÃČÂľn. Wheeler graphs: A framework for bwt-based data structures. *Theoretical Computer Science*, 698(Supplement C):67 – 78, 2017. Algorithms, Strings and Theoretical Approaches in the Big Data Era (In Honor of the 60th Birthday of Professor Raffaele Giancarlo).

[31] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fast locating with the RLBWT. *CoRR*, abs/1705.10382, 2017.

[32] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Optimal-time text indexing in BWT-runs bounded space. In *Proceedings of the 19th Symposium on Discrete Algorithms (SODA)*, 2018. To appear.

[33] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Optimal-Time Text Indexing in BWT-runs Bounded Space. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, NA, USA, January 7-10, 2018*. SIAM, 2018. Accepted for publication. Preprint available at: https://arxiv.org/abs/1705.10382.

[34] Arnab Ganguly, Rahul Shah, and Sharma V. Thankachan. pBWT: achieving succinct data structures for parameterized pattern matching and related problems. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 397–407. Society for Industrial and Applied Mathematics, 2017.

[35] Joseph Yossi Gil and David Allen Scott. A bijective string sorting transform. *arXiv*, abs/1201.3077, 2012.

[36] Simon Gog, Timo Beller, Alistair Moffat, and Matthias Petri. From theory to practice: Plug and play with succinct data structures. In *13th International Symposium on Experimental Algorithms, (SEA 2014)*, pages 326–337, 2014.

[37] Alexander Golynski, J. Ian Munro, and S. Srinivasa Rao. Rank/select operations on large alphabets: A tool for text indexing. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 368–373, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics.

[38] Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '03, pages 841–850, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[39] A. Gupta, W.-K. Hon, R. Shah, and J. S. Vitter. Compressed data structures: Dictionaries and data-aware measures. *Theoretical Computer Science*, 387(3):313–331, 2007.

[40] S. Hayashi and K. Taura. Parallel and memory-efficient burrows-wheeler transform. In *Big Data, 2013 IEEE International Conference on*, pages 43–50, Oct 2013.

[41] H. Huo, L. Chen, H. Zhao, J. S. Vitter, Y. Nekrich, and Q. Yu. A data-aware FM-index. In *Prof. of 17th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 10–23, 2015.

[42] S. Jo, S. Joannou, D. Okanohara, R. Raman, and S. Rao Satti. Compressed bit vectors based on variable-to-fixed encodings. *Computer Journal*, 60(5):761–775, 2017.

[43] J. Kärkkäinen, D. Kempa, and S. Puglisi. Hybrid compression of bitvectors for the FM-index. In *Data Compression Conference (DCC)*, pages 302–311, 2014.

[44] Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: String attractors. In *Technical Report 1710.10964, arXiv.org, 2017*, 2017.

[45] Manfred Kufleitner. On bijective variants of the Burrows-Wheeler transform. In *Proc. PSC*, pages 65–79, 2009.

[46] Shanika Kuruppu, Simon J Puglisi, and Justin Zobel. Relative Lempel-Ziv compression of genomes for large-scale storage and retrieval. In *Proc. 17th International Symposium on String Processing and Information Retrieval (SPIRE)*, pages 201–206. Springer, 2010.

[47] Kewen Liao, Matthias Petri, Alistair Moffat, and Anthony Wirth. Effective construction of relative Lempel-Ziv dictionaries. In *Proc. of the 25th International Conference on World Wide Web (WWW)*, pages 807–816. International World Wide Web Conferences Steering Committee, 2016.

[48] Bo Liu, Dixian Zhu, and Yadong Wang. debwt: parallel construction of burrowsâĂŞwheeler transform for large collection of genomes with de bruijn-branch encoding. *Bioinformatics*, 32(12):i174–i182, 2016.

[49] M. Lohrey, S. Maneth, and R. Mennicke. XML tree structure compression using RePair. *Inf. Syst.*, 38(8):1150–1167, 2013.

[50] V. Mäkinen and G. Navarro. Rank and select revisited and extended. *Theoretical Computer Science*, 387(3):332–347, 2007.

[51] V. Mäkinen, G. Navarro, J. Sirén, and N. Välimäki. Storage and retrieval of highly repetitive sequence collections. *J. Comp. Biol.*, 17(3):281–308, 2010.

[52] V. Mäkinen, G. Navarro, J. Sirén, and N. Välimäki. Storage and retrieval of highly repetitive sequence collections. *Journal of Computational Biology*, 17(3):281–308, 2010.

[53] Veli Mäkinen and Gonzalo Navarro. Succinct suffix arrays based on run-length encoding. *Nordic J. of Computing*, 12(1):40–66, March 2005.

[54] Veli Mäkinen and Gonzalo Navarro. *Position-Restricted Substring Searching*, pages 703–714. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[55] Veli Mäkinen, Gonzalo Navarro, Jouni Sirén, and Niko Välimäki. Storage and retrieval of highly repetitive sequence collections. *Journal of Computational Biology*, 17(3):281–308, 2010.

[56] J. Ian Munro, Gonzalo Navarro, and Yakov Nekrich. Space-efficient construction of compressed indexes in deterministic linear time. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 408–424, Philadelphia, PA, USA, 2017. Society for Industrial and Applied Mathematics.

[57] J. Ian Munro, Yakov Nekrich, and Hicham El-Zein. Succinct color searching in one dimension. In *Proceedings of the 28th International Symposium on Algorithms and Computation (ISAAC)*.

[58] G. Navarro. *Compact Data Structures – A Practical Approach*. Cambridge University Press, 2016.

[59] G. Navarro and A. Ordóñez. Faster compressed suffix trees for repetitive text collections. *J. Exper. Alg.*, 21(1):article 1.8, 2016.

[60] Gonzalo Navarro. A self-index on block trees. In *String Processing and Information Retrieval - 24th International Symposium, SPIRE 2017, Palermo, Italy, September 26-29, 2017, Proceedings*, pages 278–289, 2017.

[61] Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Comput. Surv.*, 39(1):article 2, 2007.

[62] Tatsuya Ohno, Yoshimasa Takabatake, Tomohiro I, and Hiroshi Sakamoto. A faster implementation of online run-length Burrows-Wheeler transform. In *Proceedings of the 28th International Workshop on Combinatorial Algorithms (IWOCA)*, 2017. To appear.

[63] Alberto Policriti and Nicola Prezza. LZ77 computation based on the run-length encoded BWT. *Algorithmica*. To appear.

[64] Alberto Policriti and Nicola Prezza. Computing LZ77 in run-compressed space. In *Proceedings of the Data Compression Conference (DCC)*, pages 23–32, 2016.

[65] Alberto Policriti and Nicola Prezza. From LZ77 to the run-length encoded Burrows-Wheeler transform, and back. In *Proceedings of the 28th Symposium on Combinatorial Pattern Matching (CPM)*, pages 17:1–17:10, 2017.

[66] Nicola Prezza. String attractors. *CoRR*, abs/1709.05314, 2017.

[67] R. Raman, V. Raman, and S. Rao Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Transactions on Algorithms*, 3(4):43, 2007.

[68] K. Sadakane. Compressed suffix trees with full functionality. *Theo. Comp. Sys.*, 41(4):589–607, 2007.

[69] K. Sadakane and R. Grossi. Squeezing succinct data structures into entropy bounds. In *Proc. of 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1230–1239, 2006.

[70] Kunihiko Sadakane. Compressed Suffix Trees with Full Functionality. *Theory of Computing Systems*, 41(4):589–607, dec 2007.

[71] P. Weiner. Linear Pattern Matching Algorithms. In *Proceedings of the 14th IEEE Symposium on Switching and Automata Theory*, pages 1–11, 1973.

[72] K. Wu, E. J. Otoo, and A. Shoshani. Optimizing bitmap indices with efficient compression. *ACM Transactions on Database Systems*, 31(1):1–38, 2006.

[73] J. Ziv and A. Lempel. Compression of individual sequences via variable length coding. *IEEE Trans. Inf. Theory*, 24(5):530–536, 1978.