Forty Years of Suffix Trees

Alberto Apostolico, Maxime Crochemore, Martín Farach-Colton, Zvi Galil, S. Muthukrishnan

Forty Years of Suffix Trees

Alberto Apostolico
College of Computing
Georgia Institute of Technology
801 Atlantic Drive
Atlanta, GA 30332-0280, USA
axa@cc.gatech.edu

Maxime Crochemore
King's College London
London WC2R 2LS, UK
and Université Paris-Est, France
maxime.crochemore@kcl.ac.uk

Martin Farach-Colton
Department of Computer Science
Rutgers University
Piscataway, NJ 08854, USA
farach@cs.rutgers.edu

Zvi Galil
College of Computing
Georgia Institute of Technology
801 Atlantic Drive
Atlanta, GA 30332-0280, USA
galil@cc.gatech.edu

S. Muthukrishnan
Department of Computer Science
Rutgers University
Piscataway, NJ 08854, USA
muthu@cs.rutgers.edu

ABSTRACT

This paper reviews the first 40 years in the life of suffix trees, their many incarnations, and their applications. The paper is non-technical but assumes some familiarity with the structures and constructions discussed. It is not meant to be exhaustive. It is meant to be a tribute to a ubiquitous tool of string matching — the suffix tree and its variants — and one of the most persistent subjects of study in the theory of algorithms.

Keywords: pattern matching, string searching, bi-tree, suffix tree, dawg, suffix automaton, factor automaton, suffix array, FM-index, wavelet tree.

1. PROLOG

When William Legrand finally decrypted the string:

53‡‡†305))6*,48264‡.)4‡);806”,48†8P60))85;1‡
(;:‡*8†83(88)5*†,46(;88*96*?;8)*‡(;485);5*†2:*‡
(;4956*2(5*Ñ4)8P8*;4069285););6‡8)4‡‡;1(‡9;48081;8:
8‡1;4885;4)485†528806*81(ddag9;48;(88;4(‡?34;
48)4‡;161;:188;‡?;

it did not seem to make much more sense than it did before. The decoded message read: "A good glass in the bishop's hostel in the devil's seat forty-one degrees and thirteen minutes northeast and by north main branch seventh limb east side shoot from the left eye of the death's-head a bee line from the tree through the shot fifty feet out." But at least it did sound more like natural language, and eventually guided the main character of Edgar Allan Poe's "The Gold Bug" [36] to discover the treasure he had been after. Legrand solved a substitution cipher using symbol frequencies. He first looked for the most frequent symbol and changed it into the most frequent letter of English, then similarly inferred the most frequent word, then punctuation marks, and so on.

Both before and after 1843, the natural impulse when faced with some mysterious message has been to count frequencies of individual tokens or subassemblies in search of a clue. Perhaps one of the most intense and fascinating subjects for this kind of scrutiny have been bio-sequences. As soon as some such sequences became available, statistical analysts tried to link characters or blocks of characters to relevant biological functions. With the early examples of whole genomes emerging in the mid 1990's, it seemed natural to count the occurrences of all blocks of size 1, 2, etc., up to any desired length, looking for statistical characterizations of coding regions, promoter regions, etc.

This review is not about cryptography. It is about a data structure and its variants, and the many surprising and useful features it carries. Among these is the fact that, to set up a statistical table of occurrences for *all* substrings (also called factors), of *any* length, of a text string of n characters, it only takes time and space linear in the length of the text string. While nobody would be so foolish as to solve the problem by first generating all exponentially many possible strings and then counting their occurrences one by one, a text string may still contain $\Theta(n^2)$ distinct substrings, so that tabulating all of them in linear space, never mind linear time, already seems puzzling.

Over the years, such structures have held center stage in text searching, indexing, statistics, and compression as well as in the assembly, alignment and comparison of biosequences. Their range of scope extends to areas as diverse as detecting plagiarism, finding surprising substrings in a text, testing the unique decipherability of a code, and more. Their impact on Computer Science and IT at large cannot be overstated. Text searching and Bioinformatics would not be the same without them. In 2013, the Combinatorial Pattern Matching symposium celebrated the 40th anniversary of the appearance of Weiner's paper [41] with a special session entirely dedicated to that event.

2. HISTORY BITS AND PIECES

At the dawn of "stringology," Don Knuth conjectured that the problem of finding the longest substring common to two long text sequences of total length n required $\Omega(n \log n)$ time. An $O(n \log n)$-time had been provided by Karp, Miller and Rosenberg [26]. That construction was destined to play a role in parallel pattern matching, but Knuth's conjecture was short lived: in 1973, Peter Weiner showed that the problem admitted an elegant linear-time solution [41], as long as the alphabet of the string was fixed. Such a solution was actually a byproduct of a construction he had originally set up for a different purpose, i.e., identifying any substring of a text file without specifying all of them. In doing so, Weiner introduced the notion of a textual inverted index that would elicit refinements, analyses and applications for forty years and counting, a feature hardly shared by any other data structure.

Weiner's original construction processed the text file from right to left. As each new character was read in, the structure, which he called a "bi-tree", would be updated to accommodate longer and longer suffixes of the text file. Thus this was an inherently off-line construction, since the text had to be known in its entirety before the construction could begin. Alternatively, one could say that the algorithm would build the structure for the reverse of the text on-line. About three years later, Ed McCreight provided a left-to-right algorithm and changed the name of the structure to "suffix tree," a name that would stick [32].

Let x be a string of $n-1$ symbols over some alphabet Σ and \$ an extra character not in Σ. The *expanded suffix tree* T_x associated with x is a digital search tree collecting all suffixes of $x\$$. Specifically, T_x is defined as follows.

1. T_x has n leaves, labeled from 1 to n.

2. Each arc is labeled with a symbol of $\Sigma \cup \{\$\}$. For any i, $1 \leq i \leq n$, the concatenation of the labels on the path from the root of T_x to leaf i is precisely the suffix $suf_i = x_i x_{i+1} ... x_{n-1}\$$.

3. For any two suffixes suf_i and suf_j of $x\$$, if w_{ij} is the longest common prefix that suf_i and suf_j have in common, then the path in T_x relative to w_{ij} is the same for suf_i and suf_j.

An example of expanded suffix tree is given in Figure 1.

The tree can be interpreted as the state transition diagram of a deterministic finite automaton where all nodes and leaves are final states, the root is the initial state, and the labeled arcs, which are assumed to point downwards, represent part of the state-transition function. The state transitions not specified in the diagram lead to a unique non-final *sink* state. Our automaton recognizes the (finite) language consisting of all substrings of string x. This observation also clarifies how the tree can be used in an on-line search: letting y be the pattern, we follow the downward path in the tree in response to consecutive symbols of y, one symbol at a time. Clearly, y occurs in x if and only if this process leads to a final state. In terms of T_x, we say that the *locus* of a string y is the node α, if it exists, such that the path from the root of T_x to α is labeled y.

An algorithm for the direct construction of the expanded T_x (often called *suffix trie*) is readily derived (see Figure 2). We start with an empty tree and add to it the suffixes of $x\$$ one at a time. This procedure takes time $\Theta(n^2)$ and $O(n^2)$ space, however, it is easy to reduce space to $O(n)$ thereby producing a suffix tree in compact form (Figure 3). Once

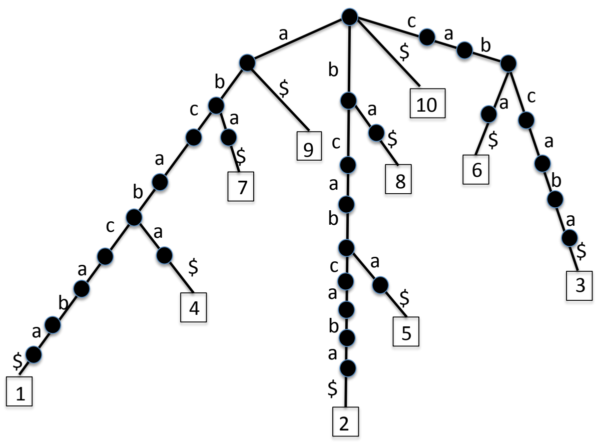


Figure 1: The expanded suffix tree of the string $x = abcabcaba$

this is done, it becomes possible to aim for an expectedly non-trivial $O(n)$ time construction.

At the CPM Conference of 2013, McCreight revealed that his $O(n)$ time construction was not born as an alternative to Weiner's: he had developed it in an effort to understand Weiner's paper, but when he showed it to Weiner asking him to confirm that he had understood that paper the answer was "No, but you have come up with an entirely different and elegant construction!" In unpublished lecture notes of 1975, Vaughan Pratt displayed the duality of this structure and Weiner's "repetition finder" [37]. McCreight's algorithm was still inherently off-line, and it immediately triggered a search for an on-line version. Some partial attempts at an on-line algorithm were made, but such a variant had to wait almost two decades for Esko Ukkonen's paper in 1995 [39]. In all these linear-time constructions, linearity was based on the assumption of a finite alphabet and took $\Theta(n \log n)$ time without that assumption. In 1997, Martin Farach introduced an algorithm that abandoned the one-suffix-at-time approach prevalent until then; this algorithm gives a linear-time reduction from suffix-tree construction to character sorting, and thus is optimal for all alphabets [17]. In particular, it runs in linear time for a larger class of alphabets, for example, when the alphabet size is polynomial in input length.

Around 1984, Anselm Blumer et al. [9] and Maxime Crochemore [14] exposed the surprising result that the smallest finite automaton recognizing all and only the suffixes of a string of n characters has only $O(n)$ states and edges. Initially coined a directed acyclic word graph (DAWG), it can even be further reduced if all states are terminal states [14]. It then accepts all substrings of the string and is called the factor/substring automaton. There is a nice relation between the index data structures when the string has no end-marker and its suffixes are marked with terminal states in the trie.

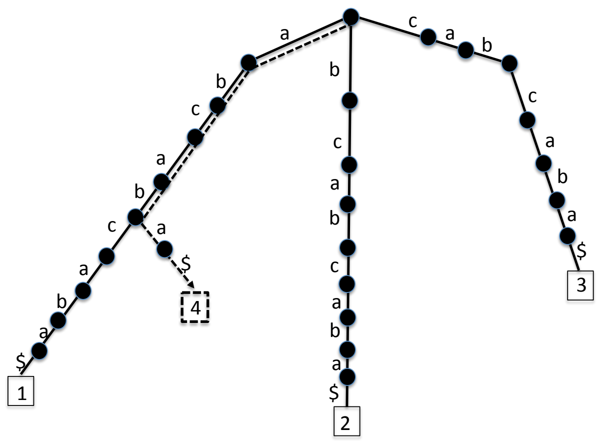


Figure 2: Building an expanded suffix tree by insertion of consecutive suffixes (showing here the insertion of $abcaba\$$). The insertion of suffix suf_i ($i = 1, 2, ..., n$) consists of two phases. In the first phase, we search for suf_i in T_{i-1}. Note that the presence of $ guarantees that every suffix will end in a distinct leaf. Therefore, this search will end with failure sooner or later. At that point, we will have identified the longest prefix of suf_i that has a locus (i.e., a terminal node) in T_{i-1}. Let $head_i$ ($abcab$ in the example) be this prefix and α the locus of $head_i$. We can write $suf_i = head_i \cdot tail_i$ with $tail_i$ ($a\$$ in the example) nonempty. In the second phase, we need to add to T_{i-1} a path leaving node α and labeled $tail_i$. This achieves the transformation of T_{i-1} into T_i.

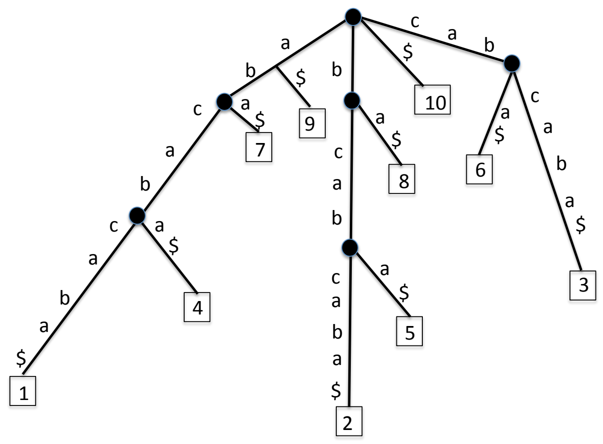


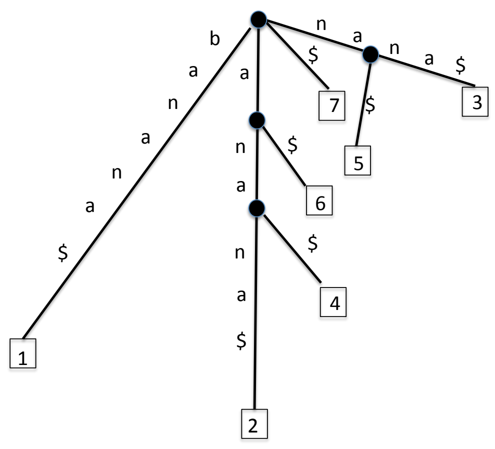
Figure 3: A suffix tree in compact form. This is obtained by first collapsing every chain formed by nodes with only one child into a single arc. The resulting *compact* version of T_x has at most n internal nodes, since there are $n + 1$ leaves in total and every internal node is branching. The labels of the generic arc are now a substring, rather than a symbol of $x\$$. However, arc labels can be expressed by suitable pairs of pointers to a common copy of $x\$$ thus achieving $O(n)$ space bound overall.

Then, the suffix tree is the edge-compacted version of the trie and its number of nodes can be minimized like with any automaton thereby providing the compact DAWG of the string. Permuting the two operations, compaction and minimization, leads to the same structure. Apparently Anatoli Slissenko (see Appendix) ended up with a similar structure for his work on the detection of repetitions in strings. These automata provide another more efficient counterexample to Knuth's conjecture when they are used, against the grain, as pattern matching machines (see Figure 4).

The appearance of suffix trees dovetailed with some interesting and independent developments in information theory. In his famous approach to the notion of information, Kolmogorov equated the information or structure in a string to the length of the shortest program that would be needed to produce that string by a Universal Turing Machine. The unfortunate thing is that this measure is not computable and even if it were, most long strings are incompressible (i.e., lack a short program producing them), since there are increasingly many long strings and comparatively much fewer short programs (themselves strings).

The regularities exploited by Kolmogorov's universal and omniscient machine could be of any conceivable kind, but what if one limited them to the syntactic redundancies affecting a text in the form of repeated substrings? If a string is repeated many times one could profitably encode all occurrences by a pointer to a common copy. This copy could be internal or external to the text. In the former case one could have pointers going in both directions or only in one direction, allow or forbid nesting of pointers, etc. In his doctoral thesis, Jim Storer showed that virtually all such "macro schemes" are intractable, *except one.* Not long before that, in a landmark paper entitled "On the Complexity of Finite Sequences" [30], Abraham Lempel and Jacob Ziv had proposed a variable-to-block encoding, based on a simple parsing of the text with the feature that the compression achieved would match, in the limit, that produced by a compressor tailored to the source probabilities. Thus, by a remarkable alignment of stars, the compression method brought about by Lempel and Ziv was not only optimal in the information theoretic sense, but it found an optimal, linear-time implementation by the suffix tree, as was detailed immediately by Michael Rodeh, Vaugham Pratt and Shimon Even [38].

In his original paper, Weiner listed a few applications of his "bi-tree" including most notably off-line string searching: preprocessing a text file to support queries that return the occurrences of a given pattern in time linear in the length of the pattern. And of course, the "bi-tree" addressed



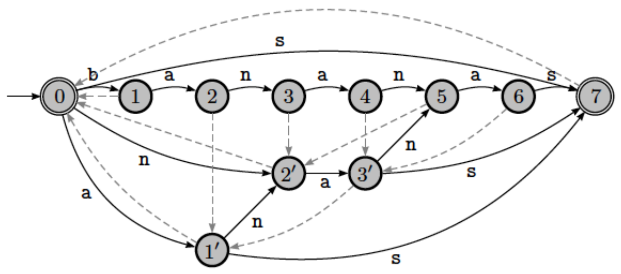


Figure 4: The compact suffix tree (top) and the suffix automaton (bottom) of the string "bananas". Failure links are represented by the dashed arrows. Despite the fact that it is an index on the string, the same automaton can be used as a pattern-matching machine to locate substrings of "bananas" in another text or to compute their longest common substring. The process runs on-line on the second string. Assume for example "bana" has just been scanned from the second string and the current state of the automaton is state 4. If the next letter is "n", the common substring is "banan" of length 5 and the new state is 5. If the next letter is "s", the failure link is used and from state 3' corresponding to a common substring "ana" of length 3 we get the common substring "anas" with the new state 7. If the next letter is "b", iterating the failure link leads to state 0 and we get the common substring "b" with the new state 1. Finally, any other next letter will produce the empty common substring and state 0.

Knuth's conjecture, by showing how to find the longest substring common to two files in linear time for a finite alphabet. There followed unpublished notes by Pratt entitled "Improvements and Applications for the Weiner Repetition Finder" [37]. A decade later, Alberto Apostolico would list more applications in a paper entitled "The Myriad Virtues of Suffix Trees" [2], and two decades later suffix trees and companion structures with their applications gave rise to several chapters in reference books by Crochemore and Rytter, Dan Gusfield, and Crochemore, Hancart and Lecroq (see Appendix).

The space required by suffix trees has been a nuisance in applications where they were needed the most. With genomes on the order of gigabytes, for instance, the space difference between 20 times larger than the source versus, say, only 11 times larger, can be substantial. For a few lustra, Stefan Kurtz and his co-workers devoted their effort to cleverly allocating the tree and some of its companion structures [28]. In 2001 David R. Clark and J. Ian Munro proposed one of the best space-saving methods on secondary storage [13]. Clark and Munro's "succinct suffix tree" sought to preserve as much of the structure of the suffix tree as possible. Udi Manber and Eugene W. Myers took a different approach, however. In 1990, they introduced the "suffix array" [31], which eliminated most of the structure of the suffix tree, but was still able to implement many of the same operations, requiring space equal to 2 integers per text character and searching in time $O(|P| + \log n)$ (reducible to 1 by accepting search time $O(|P| \log n)$). The suffix array stores the suffixes of the input in lexicographic order and can be seen as the sequence of leaves' labels as found in the suffix tree by a preorder traversal that would expand each node according to the lexicographic order. Although the suffix array seemed at first to be a different data structure than the suffix tree, the distinction has receded. For example, Manber and Myers's original construction of the suffix array took $O(n \log n)$ time for any alphabet, but the suffix array could be constructed in linear time from the suffix tree for any alphabet. In 2001, Toru Kasai et al. [27] showed that the suffix tree could be constructed in linear time from the suffix array. Therefore the suffix array was shown to be a succinct representation of the suffix tree. In 2003, three groups (see Appendix) presented three different modifications of Farach's algorithm for suffix tree construction to give the first linear-time algorithms for directly constructing the suffix array; that is, the first linear-time algorithms for computing suffix arrays that did not first compute the full suffix tree. Since then, there have been many algorithms for fast construction of suffix arrays, notably by Nong, Zhang and Chan [35], which is linear time and fast in practice. With fast construction algorithms and small space required, the suffix array is the suffix-tree variant that has gained the most widespread adoption in software systems. A more recent succinct suffix tree and array, which take $O(n)$ bits to represent for a binary alphabet ($O(n \log \sigma)$ bits otherwise), was presented by Grossi and Vitter [21].

Actually, the histories of suffix trees and compression are tightly intertwined. This should not come as a surprise, since the redundancies that pattern discovery tries to unearth are ideal candidates to be removed for purposes of compression. In 1994, M. Burrows and D. J. Wheeler proposed a breakthrough compression method based on suffix sorting [11]. Circa 1995, Amihood Amir, Gary Benson and Martin Farach posed the problem of searching in compressed texts [1]. In 2000, Paolo Ferragina and Giovanni Manzini introduced the FM-index, a compressed suffix array based on the Burrows-Wheeler transform [19]. This structure, which may be smaller than the source file, supports searching without decompression. This was extended to compressed tree indexing problems in [18] using a modification of the Burrows-Wheeler transform.

3. FALLOUT, EXTENSIONS AND CHALLENGES

As highlighted in our prolog, there has been hardly any application of text processing that did not need these indexes at one point or another. A prominent case has been searching with errors, a problem first efficiently tackled in 1985 by Gad Landau in his PhD thesis [29]. In this kind of search, one looks for substrings of the text that differ from the pattern in a limited number of errors such as a single character deletion, insertion or substitution. To efficiently solve this problem, Landau combined Suffix Trees with a clever solution to the so-called lowest common ancestor (LCA) problem. The LCA problem assumes that a rooted tree is given and then it seeks, for any pair of nodes, the lowest node in the tree that is an ancestor of both [23]. It is seen that following a linear-time preprocessing of the tree any LCA query can be answered in constant time. Landau used LCA queries on Suffix Trees to perform constant-time jumps over segments of the text that would be guaranteed to match the pattern. When k errors are allowed, the search for an occurrence at any given position can be abandoned after k such jumps. This leads to an algorithm that searches for a pattern with k errors in a text of n characters in $O(nk)$ steps.

Among the basic primitives supported by suffix trees and arrays, one finds of course the already mentioned search for a pattern in a text in time proportional to the length of the pattern rather than the text. In fact, it is even possible to enumerate occurrences in time proportional to their number and, with trivial preprocessing of the tree, tell the total number of occurrences for any query pattern in time proportional to the pattern size. The problem of finding the longest substring appearing twice in a text or shared between two files has been already mentioned: this is probably where it all started. A germane problem is that of detecting squares, repetitions and maximal periodicities in a text, a problem rooted in work by Axel Thue dated more than a century ago with multiple contemporary applications in compression and DNA analysis. A square is a pattern consisting of two consecutive occurrences of the same string. Suffix trees have been used to detect in optimal $O(n \log n)$ time all squares (or repetitions) in a text, each with its set of starting positions [5], and later to find and store all distinct square substrings in a text in linear time. Squares play a role in an augmentation of the suffix tree suitable to report, for any query pattern, the number of its non-overlapping occurrences [6, 10].

There are multiple uses of suffix trees in setting up some kind of signature for text strings, as well as measures of similarity or difference. Among the latter, there is the problem of computing the forbidden or absent words of a text, which are minimal strings that do not appear in the text (while all their proper substrings do) [8, 15]. Such words lead to, among other things, an original approach to text compression [16]. Once regarded as the succinct representation of the "bag-of-words" of a text, suffix trees can be used to assess the similarity of two text files, thereby supporting clustering, document classification and even phylogeny [4, 12, 40]. Intuitively, this is done by assessing how much the trees for the two input sequences have in common. Suitably enriched with the probability of the substring ending at each node, a tree can be used to detect surprisingly over-represented substrings of any length [3], e.g., in the quest of promoter regions in biosequences.

The suffix tree of the concatenation of say, $k \geq 2$ text files, supports efficient solutions to problems arising in domains ranging from plagiarism detection to motif discovery in biosequences. The need for k distinct end-markers poses some subtleties in maintaining linear time, for which the reader is referred to [22]. In its original form, the problem of indexing multiple texts was called the "color problem" and seeks to report, for any given query string and in time linear in the query, how many documents out of the total of k contain at least one occurrence of the query. A simple and elegant solution was given in 1992 by Lucas C. K. Hui [25]. Recently, the combined suffix trees of many strings (also known as the *generalized* suffix tree) was used to solve a variety of *document listing* problems. Here, a set of text documents are preprocessed as a combined suffix tree. The problem is to return the list of all documents that contain a query pattern in time proportional to the number of such documents, not to the total number of occurrences (occ) which can be significantly larger. This problem was solved in [33] by reducing it to Range Minimum Queries. This basic document listing problem has since been extended to many other problems including listing the top-k in various string and information distances. For example, in [24], the structure of generalized suffix tree is crucially used to design a linear machine-word data structure to return the top-k most frequent documents containing a pattern p in time nearly linear in pattern size.

One surprising variant of the suffix tree was introduced by Brenda Baker for purposes of detection of plagiarism in student reports as well as optimization in software development [7]. This variant of pattern matching, called "parameterized matching", enables one to find program segments that are identical up to a systematic change of parameters, or substrings that are identical up to a systematic relabeling or permutation of the characters in the alphabet.

One obvious extension of the notion of a suffix tree is to more than one dimension, albeit the mechanics of the extension itself are far from obvious [34]. Among more distant relatives, one finds "wavelet trees". Originally proposed as a representation of compressed suffix arrays [20], wavelet trees enable one to perform on general alphabets the ranking and selection primitives previously limited to bit vectors, and more.

The list could go on and on, but the scope of this paper was not meant to be exhaustive. Actually, after forty years of unrelenting developments, it is fair to assume that the list will continue to grow. Open problems also abound. For instance, many of the observed sequences are expressed in numbers rather than characters, and in both cases are affected by various types of errors. While the outcome of a two character comparison is just one bit, two numbers can be more or less close, depending on their difference or some other metric. Likewise, two text strings can be more or less similar, depending on the number of elementary steps necessary to change one in the other. The most disruptive aspect of this framework is the loss of the transitivity property that leads to the most efficient exact string matching solutions.

And yet indexes capable of supporting fast and elegant approximate pattern queries of the kind just highlighted would be immensely useful. Hopefully, they will come up soon and, in time, have their own 40th anniversary celebration.

Acknowledgements: We are grateful to Ed McCreight, Ronnie Martin, Vaughan Pratt, Peter Weiner, and Jacob Ziv for discussions and help. We are indebted to the Referees for their careful scrutiny of an earlier version of the paper, which led to many improvements.

4. REFERENCES

[1] A. Amir, G. Benson, and M. Farach. Let sleeping files lie: Pattern matching in Z-compressed files. In *Proceedings of the 5th ACM-SIAM Annual Symposium on Discrete Algorithms*, pages 705–714, Arlington, VA, 1994.

[2] A. Apostolico. The myriad virtues of suffix trees. In A. Apostolico and Z. Galil, editors, *Combinatorial Algorithms on Words*, volume 12 of *NATO Advanced Science Institutes, Series F*, pages 85–96. Springer-Verlag, Berlin, 1985.

[3] A. Apostolico, M. E. Bock, and S. Lonardi. Monotony of surprise and large-scale quest for unusual words. *Journal of Computational Biology*, 10(3/4):283–311, 2003.

[4] A. Apostolico, O. Denas, and A. Dress. Efficient tools for comparative substring analysis. *Journal of Biotechnology*, 149(3):120–126, 2010.

[5] A. Apostolico and F. P. Preparata. Optimal off-line detection of repetitions in a string. *Theor. Comput. Sci.*, 22(3):297–315, 1983.

[6] A. Apostolico and F. P. Preparata. Data structures and algorithms for the strings statistics problem. *Algorithmica*, 15(5):481–494, May 1996.

[7] B. S. Baker. Parameterized duplication in strings: Algorithms and an application to software maintenance. *SIAM J. Comput.*, 26(5):1343–1362, 1997.

[8] M.-P. Béal, F. Mignosi, and A. Restivo. Minimal forbidden words and symbolic dynamics. In *STACS 96, 13th Annual Symposium on Theoretical Aspects of Computer Science, Grenoble, France, February 22-24, 1996, Proceedings*, volume 1046 of *Lecture Notes in Computer Science*, pages 555–566. Springer, 1996.

[9] A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler, M. T. Chen, and J. Seiferas. The smallest automaton recognizing the subwords of a text. *Theor. Comput. Sci.*, 40(1):31–55, 1985.

[10] G. S. Brodal, R. B. Lyngsø, A. Östlin, and C. N. S. Pedersen. Solving the string statistics problem in time $O(n \log n)$. In *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, volume 2380 of *Lecture Notes in Computer Science*, pages 728–739. Springer, 2002.

[11] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipments Corporation, May 1994.

[12] S. Chairungsee and M. Crochemore. Using minimal absent words to build phylogeny. *Theoretical Computer Science*, 450(1):109–116, 2012.

[13] D. R. Clark and J. I. Munro. Efficient suffix trees on secondary storage. In *Proceedings of the 7th ACM-SIAM Annual Symposium on Discrete Algorithms*, pages 383–391, Atlanta, Georgia, 1996.

[14] M. Crochemore. Transducers and repetitions. *Theor. Comput. Sci.*, 45(1):63–86, 1986.

[15] M. Crochemore, F. Mignosi, and A. Restivo. Automata and forbidden words. *Information Processing Letters*, 67(3):111–117, 1998.

[16] M. Crochemore, F. Mignosi, A. Restivo, and S. Salemi. Data compression using antidictonaries. *Proceedings of the IEEE.*, 88(11):1756–1768, 2000. Special issue *Lossless data compression* edited by J. Storer.

[17] M. Farach. Optimal suffix tree construction with large alphabets. In *Proceedings of the 38th IEEE Annual Symposium on Foundations of Computer Science*, pages 137–143, Miami Beach, FL, 1997.

[18] P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan. Compressing and indexing labeled trees, with applications. *J. ACM*, 57(1), 2009.

[19] P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *FOCS*, pages 390–398, 2000.

[20] R. Grossi, A. Gupta, and J. S. Vitter. High-order entropy-compressed text indexes. In *SODA*, pages 841–850, 2003.

[21] R. Grossi and J. S. Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. In *Proceedings ACM Symposium on the Theory of Computing*, pages 397–406, Portland, Oregon, 2000. ACM Press.

[22] D. Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology.* Cambridge University Press, Cambridge, 1997.

[23] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.

[24] W.-K. Hon, R. Shah, and J. S. Vitter. Space-efficient framework for top-k string retrieval problems. In *FOCS*, pages 713–722. IEEE Computer Society, 2009.

[25] L. C. K. Hui. Color set size problem with applications to string matching. In A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, editors, *Proceedings of the 3rd Annual Symposium on Combinatorial Pattern Matching*, number 644 in Lecture Notes in Computer Science, pages 230–243, Tucson, AZ, 1992. Springer-Verlag, Berlin.

[26] R. M. Karp, R. E. Miller, and A. L. Rosenberg. Rapid identification of repeated patterns in strings, trees and arrays. In *Proceedings of the 4th ACM Symposium on the Theory of Computing*, pages 125–136, Denver, CO, 1972. ACM Press.

[27] T. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *CPM*, pages 181–192. Springer-Verlag, 2001.

[28] S. Kurtz. Reducing the space requirements of suffix trees. *Softw. Pract. Exp.*, 29(13):1149–1171, 1999.

[29] G. M. Landau. *String matching in erroneus input.* Ph. D. Thesis, Department of Computer Science, Tel-Aviv University, 1986.

[30] A. Lempel and J. Ziv. On the complexity of finite sequences. *IEEE Trans. Inf. Theory*, 22:75–81, 1976.

[31] U. Manber and G. Myers. Suffix arrays: a new method for on-line string searches. In *Proceedings of the 1st ACM-SIAM Annual Symposium on Discrete Algorithms*, pages 319–327, San Francisco, CA, 1990.

[32] E. M. McCreight. A space-economical suffix tree construction algorithm. *J. Algorithms*, 23(2):262–272, 1976.

[33] S. Muthukrishnan. Efficient algorithms for document listing problems. In *Proceedings of the 13th ACM-SIAM Annual Symposium on Discrete Algorithms*, pages 657–666, 2002.

[34] J. C. Na, P. Ferragina, R. Giancarlo, and K. Park. Two-dimensional pattern indexing. In *Encyclopedia of Algorithms*. 2008.

[35] G. Nong, S. Zhang, and W. H. Chan. Two efficient algorithms for linear time suffix array construction. *IEEE Trans. Comput.*, 60(10):1471–1484, 2011.

[36] E. A. Poe. *The Gold-Bug and Other Tales*. Dover Thrift Editions Series. Dover, 1991.

[37] V. Pratt. Improvements and applications for the Weiner repetition finder. Manuscript, 1975.

[38] M. Rodeh, V. Pratt, and S. Even. Linear algorithm for data compression via string matching. *J. Assoc. Comput. Mach.*, 28(1):16–24, 1981.

[39] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[40] I. Ulitsky, D. Burstein, T. Tuller, and B. Chor. The average common substring approach to phylogenomic reconstruction. *Journal of Computational Biology*, 13(2):336–350, 2006.

[41] P. Weiner. Linear pattern matching algorithms. In *Proceedings of the 14th Annual IEEE Symposium on Switching and Automata Theory*, pages 1–11, Washington, DC, 1973.

5. APPENDIX: TO KNOW MORE

A preliminary version of this article associated with the special session celebrating the 40th anniversary of the appearance of Weiner's paper appeared as: A. Apostolico, M. Crochemore, M. Farach-Colton, Z. Galil, and S. Muthukrishnan: Forty years of text indexing. In J. Fischer and P. Sanders, editors, Combinatorial Pattern Matching, number 7922 in LNCS, pages 1–10. Springer, 2013. The story of William Legrand is from: E. A. Poe: The Gold-Bug and Other Tales. Dover Thrift Editions Series. Dover, 1991. Weiner's cited publication was also motivated by file compression, see P. Weiner and R. W. Tuttle: The file transmission problem, Research Report TR016, Computer Science, Yale University, 1973. Books on string algorithms include: M. Crochemore and W. Rytter: Text algorithms. Oxford University Press, 1994; D. Gusfield: Algorithms on strings, trees and sequences: computer science and computational biology. Cambridge University Press, Cambridge, 1997; M. Crochemore, C. Hancart, and T. Lecroq.: Algorithms on Strings. Cambridge University Press, Cambridge, 2007. Statistical characterizations of coding and promoter regions abound in the post-genome, starting with J. van Helden, B. André, and J. Collado-Vides: Extracting regulatory sites from the upstream region of the yeast genes by computational analysis of oligonucleotides. J. Mol. Biol., 281:827-842, 1998. For parallel constructions based on Karp-Miller-Rozenberg paradigm see: Z. Galil: Optimal parallel algorithms for string matching. In Proceedings of the 16th ACM Symposium on the Theory of Computing, pages 240-248, Washington, D.C., 1984. ACM Press. Z. Galil: Optimal parallel algorithms for string matching. Inf. Control, 67(1-3):144-157, 1985. A. Apostolico, C. Iliopoulos, G. M. Landau, B. Schieber, and U. Vishkin. Parallel construction of a suffix tree with applications. Algorithmica, 3:347-365, 1988. M. Crochemore and W. Rytter: Usefulness of the Karp-Miller-Rosenberg algorithm in parallel computations on strings and arrays. Theor. Comput. Sci., 88(1):59-82, 1991. Partial attempts at online ST construction were contributed by M. Kempf, R. Bayer, and U. Güntzer: Time optimal left to right construction of position trees. Acta Inform., 24(4):461-474, 1987 and M. E. Majster and A. Ryser: Efficient on-line construction and correction of position trees. SIAM J. Comput., 9(4):785-807, 1980. Further relationships among suffix trees and DAWGs are described by M. Crochemore in Reducing space for index implementation. Theor. Comput. Sci., 292(1):185-197, 2003. And by S. Inenaga, H. Hoshino, A. Shinohara, M. Takeda, S. Arikawa, G. Mauri, and G. Pavesi. On-line construction of compact directed acyclic word graphs, Discrete Applied Mathematics, 146(2):156-179, 2005. For Slissenko's work on string matching see A. O. Slisenko: Determination in real time of all the periodicities in a word. Sov. Math. Dokl., 21:392-295, 1980. and A. O. Slisenko: Detection of periodicities and string matching in real time. J. Sov. Math., 22:1316-1386, 1983. Kolmogorov's seminal paper appeared in English as A. N. Kolmogorov: Three approaches to the quantitative definition of information. Problems of Information Transmission, 1(1):1-7, 1965. Additional references on macro schemes can be found in J. A. Storer and T. G. Szymanski: The macro model for data compression. In Proceedings of the 10th ACM Symposium on the Theory of Computing, pages 30-39, San Diego, CA, 1978. ACM Press. J. A. Storer and T. G. Szymanski: Data compression via textual substitution. J. Assoc. Comput. Mach., 29(4):928-951, 1982. An important step towards construction in secondary memory was achieved by P. Ferragina, R. Grossi: The String B-tree: a new data structure for string search in external memory and its applications, Journal of the ACM, 46(2): 236-280, 1999. The journal version of Suffix arrays appeared as U. Manber and G. Myers: Suffix arrays: a new method for on-line string searches. SIAM J. Comput., 22(5):935-948, 1993. The modifications to Farach's algorithm were proposed in J. Karkkainen and P. Sanders: Simple linear-work suffix array construction. In J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger, editors, Automata, Languages and Programming, 30th International Colloquium, ICALP 2003, Eindhoven, The Netherlands, June 30 - July 4, 2003. Proceedings, volume 2719 of Lecture Notes in Computer Science, pages 943-955, 2003. Suffix arrays in linear time are in D. K. Kim, J. S. Sim, H. Park, and K. Park: Constructing suffix arrays in linear time. J. Discrete Algorithms, 3(2-4):126-142, 2005. See also P. Ko and S. Aluru: Space-efficient linear-time construction of suffix arrays. J. Discrete Algorithms, 3(2-4):143-156, 2005. Searching with errors was published in G. M. Landau and U.

Vishkin, Fast parallel and serial approximate string matching, Journal of Algorithms, 10, 2, 157–169 (1989). Early work on approximate indexed searches includes E. Ukkonen: Approximate String-Matching over Suffix Trees. CPM 1993: 228-242, A. Cobbs: Fast approximate matching using suffix trees. In Proc. 6th Ann. Symp. on Combinatorial Pattern Matching (CPM'95), LNCS 807, pages 41-54, 1995. The constant-time LCA solution was due to D. Harel and R. E. Tarjan: Fast algorithms for finding nearest common ancestors. SIAM J. Comput., 13(2):338-355, 1984. A simpler implementation was proposed in M. A. Bender and M. Farach-Colton: The LCA problem revisited. In LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings, volume 1776 of Lecture Notes in Computer Science, pages 88-94. Springer, 2000. Axel Thue's landmark paper on square-free morphism is A. Thue: Uber die gegenseitige lage gleicher teile gewisser zeichenreichen. Nor. Vidensk. Selsk. Skr. Mat. Nat. Kl., 1:1-67, 1912. See also D. Gusfield and J. Stoye: Linear-time algorithms for finding and representing all the tandem repeats in a string. J. Comput. Syst. Sci., 69(4):525-546, 2004. For orthogonal range queries see M. Lewenstein: Orthogonal range searching for text indexing. In A. Brodnik, A. Lopez-Ortiz, V. Raman, and A. Viola, editors, Space-Efficient Data Structures, Streams, and Algorithms, volume 8066 of LNCS, pages 267-302. Springer, 2013. For 2D suffix trees see J. C. Na, R. Giancarlo, and K. Park: On-line construction of two-dimensional suffix trees in $O(n^2 \log n)$ time. Algorithmica, 48(2):173-186, 2007. For applications of wavelet trees see P. Ferragina, R. Giancarlo, and G. Manzini: The myriad virtues of wavelet trees. Inf. Comput., 207(8):849-866, 2009. Suffix trees and their derivatives find countless applications in bioinformatics, more recently for the search of short strings produced by sequencing into massive genomes. For a couple of examples, see H. Li and R. Durbin: Fast and accurate long read alignment with Burrows-Wheeler Transform. Bioinformatics, 26:589-595, 2010; B. Langmead, C. Trapnell, M. Pop and S. L Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. Nat Biotechnol. 27(5): 455-457, 2009. One more notable problem of computational biology is string barcoding, that finds use in the classification of organisms and virus identification. For this, see, e.g. S. Rash and D. Gusfield. Uncovering Optimal Virus Signatures. Proceedings of the Annual International Conference on on Computational Molecular Biology (RECOMB) ACM press, 254-261, 2002.