Succinct Data Structures, Adaptive (analysis of) Algorithms: Overview, Combination, and Perspective

Jérémy Barbay

Technical Report CS-2007-33, Cheriton School of Computer Science, University of Waterloo, Canada.

Abstract. Succinct data structures replace static instances of pointer based data structures, improving performance in both time and space in the word RAM model (a restriction of the RAM model where the size of each machine-word is bounded). The adaptive analysis of algorithms considers the complexity in a finer way than merely grouping the instances by size, yielding more precise lower and upper bounds on the complexity of a problem. We give a quick overview of those two techniques, some brief examples of how they can be combined on various search problems to obtain near optimal solutions, and some general perspective on the development and application of those techniques to other problems and in under different models. The slides corresponding to this abstract are available at the following address: http://www.cs.uwaterloo.ca/~jbarbay/Recherche/Publishing/Lectures/succinctAdaptive_handout.pdf.

1 Introduction

A succinct data structure for a given abstract data type is a representation that uses an amount of space "close" to the information theoretic lower bound of the underlying combinatorial object together, along with algorithms that supports the operations of the abstract data type "efficiently". A natural example is the representation of a binary tree [22]: an arbitrary binary tree on n nodes can be represented in $2n + o(n)$ bits while supporting a variety of operations on any node, which include finding its parent, its left or right child, and returning the size of its subtree, each in constant time. As there are $\binom{2n}{n}/(n+1)$ binary trees on n nodes and the logarithm of this term[1] is $2n - o(n)$, the space used by this representation is optimal to within a lower order term. Preprocessing such data-structures so as to be able to perform searches is a complex process requiring a variety of subordinate structures, which we review here.

Adaptive algorithms are algorithms that take advantage of "easy" instances of the problem at hand, i.e. their complexity depends on some measure of difficulty, for example a function of the size of the instance and of other parameters. For example, Kirkpatrick and Seidel [24] proposed an algorithm for computing the convex hull that has complexity $\mathcal{O}(n \lg h)$, where n is the number of input vertices, and h is the number of output vertices in the resulting convex hull. As previously known algorithms guarantee only a running time of $\mathcal{O}(n \lg n)$ in the worst case, clearly, the adaptive algorithm performs better when the size of the convex hull size is small (e.g. a triangle).

We describe the fundamental principles of those two techniques and we illustrate them by a selection of results, respectively in Section 2 and 3. We describe in Section 4 how they can be combined on various search problems to obtain solutions near from non-deterministic optimality, and some perspective of research concerning those techniques.

2 Succinct Data Structures

2.1 Bit Vectors

Jacobson introduced the concept of succinct data structures encoding *bit vectors* [22] and supporting efficiently some basic operators on it, as a constructing block for other data-structures, such as tree structures

[1] All logarithms are taken to the base 2. By convention, $\lg \lg x$ is noted $\operatorname{llg} x$ and $\lg \lg \lg x$ is noted $\operatorname{lllg} x$.

and planar graphs. Given a bit vector $B[0,\ldots,n{-}1]$, a bit $\alpha \in \{0,1\}$, an object $x \in [n] = \{0,\ldots,n{-}1\}$ and an integer $r \in \{1,\ldots,n\}$, the operator $\texttt{bin\_rank}_B(\alpha,x)$ returns the number of occurrences of α in $B[0,...,x{-}1]$, and the operator $\texttt{bin\_select}_B(\alpha,r)$ returns the position of the r-th label α in B (we omit the subscript B when it is clear from the context). To illustrate these operators, consider the bit vector "0 0 0 **1** 0 0 **0** 0 **1** 0 0" on the binary alphabet. Counting the number of 1-bits among the six first bits corresponds to the operation $\texttt{bin\_rank}(1,6) = 1$, while searching for the second 1-bit corresponds to the operation $\texttt{bin\_select}(1,2) = 7$.

Those operators can be supported on a bit vector of length n in *constant time* in the $\Theta(\lg n)$-word RAM model, using an index of $\frac{n \lg\lg n}{\lg n} + \mathcal{O}(\frac{n}{\lg n})$ additional bits [17], which is asymptotically negligible (i.e. it is $o(n)$) compared to the n bits required to encode the bit vector itself, in the worst case over all bit vectors of n bits. As the index is separated from the encoding of the binary string, the results holds even if the binary string has exactly v bits set to one and is compressed to $\lg \binom{n}{v}$ bits, as long as the encoding supports in constant time the access to a machine word of the string [30]. The space used by the resulting data-structures is optimal up to asymptotically negligible terms among the encodings keeping the index separated from the encoding of the binary string [17]. Obtaining the same lower bound or a better encoding in the general case is still open.

2.2 Ordinal Trees and Planar Graphs

An *ordinal tree* is a rooted tree in which the children of a node are ordered and specified by their rank. The basic operators on ordinal trees are $\texttt{child}(x,i)$, the i-th child of node x for $i \geq 1$; $\texttt{childrank}(x)$, the number of siblings preceding x in preorder; $\texttt{leveled\_ancestor}(x,i)$, the i-th ancestor of node x (x is its own 0-th ancestor); $\texttt{depth}(x)$, the number of ancestors of x; $\texttt{nbdesc}(x)$, the number of descendants of x; and $\texttt{degree}(x)$, the number of children of x; $\texttt{tree\_rank}_{pre/post/dfuds}(x)$, the position of node x in the given tree-traversal; $\texttt{tree\_select}_{pre/post/dfuds}(r)$, the r-th node in the given tree-traversal.

Several techniques permit to encode ordinal trees while supporting various sets of basic operators in constant time in the $\Theta(\lg n)$-word RAM model, using the fact that ordinal trees are in bijection with strings of well balanced parenthesis [27]; using a sequence of node degrees [7]; or using a recursive decomposition of the tree [15].

The most general encoding [15] supports all those operators in constant time (in the $\Theta(\lg n)$-word RAM model) while encoding an ordinal tree of n nodes and its index using a total of $2n + o(n)$ bits, which is asymptotically tight with the lower bound suggested by information theory, and better than traditional solutions (such as using $2n \lg n$ bits and supporting a subset of the navigation operators in constant time through pointers). Once again, the space used by the resulting data-structures is optimal up to asymptotically negligible terms among the data-structures keeping the index separated from the encoding of the binary string. It is possible to obtain a better encoding in the general case [19], but obtaining a lower bound for the general case is an open problem.

The design of succinct encodings for planar graphs supporting navigation operators is similar, using the fact that planar graphs can be decomposed in a finite number of ordinal trees, through a book embedding [8] or through realizers [11, 12]; or decomposed recursively in a similar ways to trees [10].

2.3 Permutations and Functions

A basic building block for the structures described below is the representation of a *permutation* of the integers $\{0,\ldots,n{-}1\}$, again denoted by $[n]$. The basic operators on a permutation are the image of a number i through the permutation, through its inverse or through the k-th power of it (i.e. π iteratively applied k times starting at i, where k can be any integer so that π^{-1} is the inverse of π).

A straightforward array of n words encode a permutation π and supports the application of π^1 in constant time. An additional index composed of n/t shortcuts [16] cutting the largest cycles of π in loops of less than t elements supports the inverse permutation π^{-1} in at most t word-accesses. Using such an encoding for the permutation mapping a permutation π to one of its cyclic representation, one can also support the application of $\pi^k(i)$ in at most t word-accesses, with the same space constraints [26]. Those results extend to functions on finite sets [28] by a simple combination with the tree encodings described above, and the space used by the resulting data-structures is optimal up to asymptotically negligible terms [18].

2.4 Strings

Another basic abstract data type is the *string*, composed of n characters taken from an alphabet of arbitrary size σ (as opposed to binary for the bit vector). The basic operations on a string are to access it, and to search and count the occurrences of a pattern, such as a simple character from $[\sigma]$ in the simplest case [21]. Formally, for any character α, position x in the string and positive integer r, it corresponds to the operators `string_access`(x), the x-th character of the string; `string_rank`(α, x), the number of α-occurrences before position x; and `string_select`(α, r), the position of the r-th α-occurrence.

Golynski *et al.* [20] showed how to encode a string of lenght n over the alphabet $[\sigma]$ via n/σ permutations over $[\sigma]$, and how to support the string operators using the operators on those permutations. Choosing a value of $t = \lg\sigma$ in the encoding of the permutation yields an encoding using $n\big(\lg\sigma + o(\lg\sigma)\big)$ bits in order to support the operators in at most $\mathcal{O}(\operatorname{llg}\sigma)$ word accesses. Observing that the encoding of permutations already separates the data from the index, Barbay *et al.* [4] properly separated the data and the index of strings, yielding a succinct index using the same space and supporting the operators in $\mathcal{O}(\operatorname{llg}\sigma \operatorname{lllg}\sigma(f(n,\sigma) + \operatorname{llg}\sigma))$ word access, where each word of the data can be accessed in $\mathcal{O}(f(n,\sigma))$ word accesses. Although this supporting time is slightly larger than for the succinct encoding ($\mathcal{O}(\operatorname{llg}\sigma \operatorname{lllg}\sigma)$ instead of $\mathcal{O}(\operatorname{llg}\sigma)$ with an encoding of the data supporting constant time access), this succinct index has the advantage of removing any restrictions on the encoding of the data of the string (hence allowing compression, among other advantages). The space used by the resulting data-structures is optimal up to asymptotically negligible terms, among all possible succinct indexes [18]. A lower bound on succinct encodings is still an open problem.

2.5 Binary Relations

Given two ordered sets of sizes σ and n, represented by $[\sigma]$ and $[n]$, a *binary relation* R between these sets is a subset of their Cartesian product, i.e. $R \subset [\sigma]\times[n]$. In some applications, it represents the relation between a set of labels $[\sigma]$ (e.g. keywords entered by users in conjunctive queries) and a set of objects $[n]$ (e.g. web pages indexed by a search engine).

Although a string can be seen as a particular case of a binary relation, where the objects are positions and exactly one label is associated to each position, the search operations on binary relations are more diverse, including operators on both labels and objects. For any label α, object x, and integer r, the basic operators on binary relations are $\texttt{label\_rank}_R(\alpha, x)$: the number of objects labeled α preceding or equal to x; $\texttt{label\_select}_R(\alpha, r)$: the position of the r-th object labeled α if any, or ∞ otherwise; $\texttt{label\_nb}_R(\alpha)$, the number of objects with label α; $\texttt{object\_rank}_R(x, \alpha)$: the number of labels associated with object x preceding or equal to label α; $\texttt{object\_select}_R(x, r)$: the r-th label associated with object x, if any, or ∞ otherwise; $\texttt{object\_nb}_R(x)$: the number of labels associated with object x; and $\texttt{table\_access}_R(x, \alpha)$: checks whether object x is associated with label α.

Such a binary relation, consisting of t pairs from $[n]\times[\sigma]$, can be encoded as a text string S listing the t label occurrences, and a bit vector B indicating how many labels are associated with each object [3], so that search operations on the objects associated with a fixed label are reduced to a combination of operators on text and binary strings: such a representation uses $t(\lg\sigma + o(\lg\sigma))$ bits. Using a more direct reduction to the encoding of permutations, the index of the binary relation can be separated from its encoding, and even more operators can be supported, taking literals (covering labels and negation of labels) as parameters [4]. The space used by the resulting data-structures is optimal up to asymptotically negligible terms [18] among succinct indexes, but open in the general case.

2.6 Labelled Trees and Planar Graphs

A *labeled tree* T with any number of labels per node can be represented by an ordinal tree coding its structure [23] and a binary relation R associating to each node its labels [3]. If the nodes are considered in preorder (resp. in DFUDS order) the search operators enumerate all the descendants (resp. children) of a node matching some literal α. Using succinct indexes, a single encoding of the labels and the support of a permutation between orders is sufficient to implement both enumerations and other search operators on the labels [4]. Since a binary relation can be seen as a very flat labeled tree, the lower bounds on binary relations obviously also hold for labeled trees.

Similarly to the unlabeled version, the succinct encodings for labeled planar graphs take advantage of the results on labeled trees [2], whether the labels are associated to the nodes or to the edges.

3 Adaptive Analysis

3.1 Convex Hull

The *convex hull* of a finite set of n points S is the smallest convex polygon containing the set. By convention, the size of this polygon is noted h. In two dimensions, sorting the vertices by their relative slope to a pivot yields an algorithm computing the convex hull in $\mathcal{O}(n \lg n)$ operations. This complexity is optimal in the worst case over instances of fixed size n, but unacceptable in practice, where n is often very large.

Rather, using a divide-and-conquer technique, one can compute the convex-hull in $\mathcal{O}(n \lg h)$ operations [24]. This algorithm is *output-sensitive* (i.e. adaptive to the size of the output) in the sense that it performs better on instances of both small input and output size, and better on instances of small output size among all instances of fixed input size. This bound is tight among all instances of fixed input and output size.

3.2 Sorting

Sorting an array A of numbers is a basic problem, where the size of the output of an instance is always equal to its input size. Still, some instances are easier than others to sort (e.g. a sorted array, which can be checked/sorted in linear time). Instead of the output size, one can consider the disorder in an array as a measure of the difficulty of sorting this array [9, 25].

There are many ways to measure this disorder: one can consider the number of exchanges required to sort an array; the number of adjacent exchanges required; the number of pairs (i, j) such that $A[i] > A[j]$, but there are many others [29]. For each disorder measure, the logarithm of the number of instances with a fixed size and disorder forms a natural lower bound to the worst case complexity of any sorting algorithm in the comparison model, as a correct algorithm must at least be able to distinguish all instances. As a consequence, there could be as many optimal algorithms as there are difficulty measures. Instead, one can reduce difficulty measures between themselves, which yields a hierarchy of disorder measures [14].

3.3 Union of Sorted Sets

A problem where the output size can vary but is not a good measure of difficulty, is the *description* of the *sorted union of sorted sets*: given k sorted sets, describe their union. On the one hand, the sorted union of $A = \{0, 1, 2, 3, 4\}$ and $B = \{5, 6, 7, 8, 9\}$ is easier to describe (all values from A in their original order, followed by all values from B, in their original order) than the union of $C = \{0, 2, 4, 6, 8\}$ and $D = \{1, 3, 5, 7, 9\}$. On the other hand, a deterministic algorithm must *find* this description, which can take much more time than to output it when computing the union of many sets at once.

Possible measures of difficulty are then the minimal encoding size $\mathcal{C}$ of a *certificate* [13], a set of comparisons required to check the correction of the output of the algorithm (yielding complexity $\Theta(\mathcal{C})$); or the non-deterministic complexity [6], the number of steps δ performed by a non deterministic algorithm to solve the instance, or equivalently the minimal number of comparisons of a certificate (yielding complexity $\Theta(\delta k \lg(n/\delta k)))$. For both measures, there is an algorithm proved to be optimal which is not optimal for the other measure. Finding a more general measure of difficulty is an open problem.

3.4 Intersection and Threshold Set of Sorted Arrays

A related problem, with applications to conjunctive queries in indexed search engines, is the *intersection of sorted arrays*: as an indexed search engine maintains for each keyword a sorted list of the related objects, answering a conjunctive queries composed of k keywords correspond to intersect the k sets associated to those keywords. Reusing the arrays from the previous example, while both intersections are empty, it is easier to prove that the intersection of $A = \{0,1,2,3,4\}$ and $B = \{5,6,7,8,9\}$ is empty (the largest element from A is smaller than the smallest element from B) than it is for the intersection of $C = \{0,2,4,6,8\}$ and $D = \{1,3,5,7,9\}$ [13].

The difficulty measures considered are the minimal encoding size $\mathcal{G}$ of a part of the certificate [13] (yielding complexity $\Theta(k\mathcal{G})$) and the minimal number δ of comparisons of a certificate [5] (yielding complexity $\Theta(\delta k \lg(n/\delta k)))$ as for the union, and a measure ρ related to the number of possible short certificates of the answer, to take into account features making the instance easier for randomized algorithms [1] (yielding complexity $\Theta(\rho k \lg(n/\rho k)))$.

As an empty intersection corresponds to the null answer to a conjunctive query, it is natural to consider a relaxation of the intersection of sorted arrays, the *threshold set* composed of all the elements which are contained in *at least* t arrays. For $t = k$ this is obviously the intersection, for $t = 1$ it is obviously the union, and for $t = 2$ the description of the threshold set corresponds exactly to the description of the union discussed above. The same techniques yields very similar results, even when weights are associated to the terms of the query (simulating the repetition of an array in the intersection), or with the pairs of the binary relation (to distinguish different level of association between labels and objects) [31].

3.5 Pattern Matching in Labelled Trees

Given a multi-labeled tree and k labels, the *path subset pattern matching* consists in finding each node x such that the path from the root to x matches the labels. Given a multi-labeled tree and k labels, each node of the corresponding *lowest common ancestor* set is such that its descendants match *all the keywords*, but none of them is a lowest common ancestor itself [32]. Supposing an encoding of the multi-labeled tree allowing efficient search operators (such as the one described in Section 2.6 or an equivalent one based on sorted arrays), the technique described to compute the intersection of sorted arrays generalizes easily to solve those queries [3], and to solve their generalization to the threshold set.

4 Combining Approaches and Perspective

Data structures and algorithms are complementary. Any choice of data-structure can be combined with any algorithm, and the best performance is obtained only when they interact well.

For instance, a naive approach would be to replace sorted arrays by binary vectors, and to replace the binary or doubling searches in those arrays by a combination of `bin_rank` and `bin_select` operators, reducing the time from $\mathcal{O}(\lg n)$ to constant. Among many other results, this yields an intersection algorithm solving conjunctive queries in $\mathcal{O}(\delta k)$ operations in the word RAM model instead of $\Theta(\delta k \lg(n/\delta k))$ in the comparison model (Section 3.4), but at an excessive cost in space. A wiser approach is to take a larger perspective and to reconsider the abstract data types from the beginning. For instance, in the example above, the complexity of the intersection algorithm can be improved to $\mathcal{O}(\delta k \operatorname{llg} \sigma)$ by encoding all the sorted arrays in a single binary relation [3], replacing the binary or doubling searches by a combination of `label_rank` and `label_select` operators, while potentially reducing the space taken by the index from $t(\lg n) + \sigma$ to $t(\lg \sigma + o(\lg \sigma))$. The same approach yields similar results for pattern matching queries in labeled trees [3].

In those examples the complexity $\mathcal{O}(\delta k \operatorname{llg} \sigma)$ achieved by adaptive algorithms using succinct data structures is getting very close to the theoretical non-deterministic lower bound of $\Omega(\delta)$ *for the instance*: in practice k is often quite small, and the function $\operatorname{llg} \sigma$ grows so slowly with σ that it would require some unrealistic data set for it to be larger than 6 (e.g. $2 \uparrow 6 > 10^{19}$): the worst case analysis cannot be refined further in the asymptotic perspective. The next challenges are of course to apply those techniques to more problems, but also to extend the techniques to other computational models, for instance to multicore systems and extended memory hierarchy, and to study the values taken experimentally by the difficulty measures for each particular class of application, in the hope to further adapt the encoding and algorithmic solutions.

Bibliography

[1] J. Barbay. Optimality of randomized algorithms for the intersection problem. In *Proceedings of the Symposium on Stochastic Algorithms, Foundations and Applications (SAGA)*, volume 2827 of *Lecture Notes in Computer Science (LNCS)*, pages 26–38. Springer, 2003.

[2] J. Barbay, L. C. Aleardi, M. He, and J. I. Munro. Succinct representation of labeled graphs. In *Proceedings of the 18th International Symposium Algorithms and Computation (ISAAC)*, volume 2906 of *Lecture Notes in Computer Science (LNCS)*, pages 575–584. Springer, 2007.

[3] J. Barbay, A. Golynski, J. I. Munro, and S. S. Rao. Adaptive searching in succinctly encoded binary relations and tree-structured documents. In *Proceedings of the 17th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 4009 of *Lecture Notes in Computer Science (LNCS)*, pages 24–35. Springer-Verlag, 2006.

[4] J. Barbay, M. He, J. I. Munro, and S. S. Rao. Succinct indexes for strings, binary relations and multi-labeled trees. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 680–689. ACM, 2007.

[5] J. Barbay and C. Kenyon. Adaptive intersection and t-threshold problems. In *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 390–399. Society for Industrial and Applied Mathematics (SIAM), 2002.

[6] J. Barbay and C. Kenyon. Deterministic algorithm for the t-threshold set problem. In *Proceedings of the 14th International Symposium Algorithms and Computation (ISAAC)*, volume 2906 of *Lecture Notes in Computer Science (LNCS)*, pages 575–584. Springer, 2003.

[7] D. Benoit, E. D. Demaine, J. I. Munro, R. Raman, V. Raman, and S. S. Rao. Representing trees of higher degree. *Algorithmica*, pages 275–292, 2005.

[8] F. Bernhart and P. C. Kainen. The book thickness of a graph. *Journal of Combinatorial Theory, Series B*, 27(3):320–331, 1979.

[9] W. H. Burge. Sorting, trees, and measures of order. *Information and Control*, 1(3):181–197, 1958.

[10] L. Castelli-Aleardi, O. Devillers, and G. Schaeffer. Succinct representation of triangulations with a boundary. In *Proc. 9th Workshop on Algorithms and Data Structures*, volume 3608 of *LNCS*, pages 134–145. Springer, 2005.

[11] Y.-T. Chiang, C.-C. Lin, and H.-I. Lu. Orderly spanning trees with applications to graph encoding and graph drawing. *SODA*, pages 506–515, 2001.

[12] R.-N. Chuang, A. Garg, X. He, M.-Y. Kao, and H.-I. Lu. Compact encodings of planar graphs via canonical orderings and multiple parentheses. *Automata, Laguages and Programming*, pages 118–129, 1998.

[13] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Adaptive set intersections, unions, and differences. In *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 743–752, 2000.

[14] V. Estivill-Castro and D. Wood. A survey of adaptive sorting algorithms. *ACM Computing Surveys*, 24(4):441–476, 1992.

[15] R. F. Geary, R. Raman, and V. Raman. Succinct ordinal trees with level-ancestor queries. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1–10, 2004.

[16] R. Gennaro and L. Trevisan. Lower bounds on the efficiency of generic cryptographic constructions. In *IEEE Symposium on Foundations of Computer Science*, pages 305–313, 2000.

[17] A. Golynski. Optimal lower bounds for rank and select indexes. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, 2006.

[18] A. Golynski. *Upper and Lower Bounds for Text Indexing Data Structures.* PhD thesis, University of Waterloo, 2007.

[19] A. Golynski, R. Grossi, A. Gupta, R. Raman, and S. S. Rao. On the size of succinct indices. In L. Arge, M. Hoffmann, and E. Welzl, editors, *ESA*, volume 4698 of *Lecture Notes in Computer Science*, pages 371–382. Springer, 2007.

[20] A. Golynski, J. I. Munro, and S. S. Rao. Rank/select operations on large alphabets: a tool for text indexing. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 368–373. ACM, 2006.

[21] R. Grossi, A. Gupta, and J. S. Vitter. High-order entropy-compressed text indexes. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete algorithms (SODA)*, pages 841–850. ACM, 2003.

[22] G. Jacobson. Space-efficient static trees and graphs. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 549–554, 1989.

[23] J. Jansson, K. Sadakane, and W.-K. Sung. Ultra-succinct representation of ordered trees. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 575–584. ACM, 2007.

[24] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM J. Comput.*, 1986. 15(1):287–299.

[25] H. Mannila. Measures of presortedness and optimal sorting algorithms. *IEEE Trans. Computers*, 34(4):318–325, 1985.

[26] J. I. Munro, R. Raman, V. Raman, and S. S. Rao. Succinct representations of permutations. In *Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 2719 of *Lecture Notes in Computer Science (LNCS)*, pages 345–356. Springer-Verlag, 2003.

[27] J. I. Munro and V. Raman. Succinct representation of balanced parentheses, static trees and planar graphs. In *IEEE Symposium on Foundations of Computer Science*, pages 118–126, 1997.

[28] J. I. Munro and S. S. Rao. Succinct representations of functions. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3142 of *Lecture Notes in Computer Science (LNCS)*, pages 1006–1015. Springer-Verlag, 2004.

[29] O. Petersson and A. Moffat. A framework for adaptive sorting. *Discrete Applied Mathematics*, 59(2):153–179, 1995.

[30] R. Raman, V. Raman, and S. S. Rao. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete algorithms*, pages 233–242, 2002.

[31] A. Veraskouski. Adaptive algorithms for weighted queries on weighted binary relations and labeled trees. Master's thesis, Cheriton School of Computer Science, University of Waterloo, 2007.

[32] Y. Xu and Y. Papakonstantinou. Efficient keyword search for smallest LCAs in XML databases. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 527–538, New York, NY, USA, 2005. ACM Press.