A separation between RLSLPs and LZ77

In their ground-breaking paper on grammar-based compression, Charikar et al. (2005) gave a separation between straight-line programs (SLPs) and Lempel Ziv ’77 (LZ77): they described an infinite family of strings such that the size of the smallest SLP generating a string of length n in that family, is an $\Omega(\log n / \log \log n)$-factor larger than the size of the LZ77 parse of that string. However, the strings in that family have run-length SLPs (RLSLPs) — i.e., SLPs in which we can indicate many consecutive copies of a symbol by only one copy with an exponent — as small as their LZ77 parses. In this paper we modify Charikar et al.’s proof to obtain the same $\Omega(\log n / \log \log n)$-factor separation between RLSLPs and LZ77.
Compressed and Practical Data Structures for Strings

In this dissertation, I will cover a number of different topics related to strings in compressed and practical settings. I will first present some fundamental techniques from the area, and then cover 6 different topics within the area. A short introduction to each of these topics is given in the following.

Finger Search in Grammar-Compressed Strings. Grammar-based compression, where one replaces a long string by a small context-free grammar that generates the string, is a simple and powerful paradigm that captures many popular compression schemes. Given a grammar, the random access problem is to compactly represent the grammar while supporting random access, that is, given a position in the original uncompressed string report the character at that position. We study the random access problem with the finger search property, that is, the time for a random access query should depend on the distance between a specified index f, called the finger, and the query index i. We consider both a static variant, where we first place a finger and subsequently access indices near the finger efficiently, and a dynamic variant where also moving the finger such that the time depends on the distance moved is supported. Let n be the size of the grammar, and let N be the size of the string. For the static variant we give a linear space representation that supports placing the finger in $O(\log N)$ time and subsequently accessing in $O(\log D)$ time, where $D$ is the distance between the finger and the accessed index. For the dynamic variant we give a linear space representation that supports placing the finger in $O(\log N)$ time and accessing and moving the finger in $O(\log D + \log\log N)$ time. Compared to the best linear space solution to random access, we improve a $O(\log N)$ query bound to $O(\log D)$ for the static variant and to $O(\log D + \log\log N)$ for the dynamic variant, while maintaining linear space. As an application of our results we obtain an improved solution to the longest common extension problem in grammar compressed strings. To obtain our results, we introduce several new techniques of independent interest, including a novel van Emde Boas style decomposition of grammars.

Compressed Indexing with Signature Grammars. The compressed indexing problem is to preprocess a string S of length n into a compressed representation that supports pattern matching queries. That is, given a string P of length m report all occurrences of P in S. We present a data structure that supports pattern matching queries in $O(m + \text{occ}(\log m + \log z))$ time using $O(z \log(n/z))$ space where $z$ is the size of the LZ77 parse of S and $> 0$, when the alphabet size or the compression ratio is at least polynomial. We also present two data structures for the general case; one where the space is increased by $O(z \log z)$, and one where the query time changes from worst-case to expected. In all cases, the results improve the previously best known solutions. Notably, this is the first data structure that decides if P occurs in S in $O(m)$ time using $O(z \log(n/z))$ space. Our results are mainly obtained by a novel combination of a randomized grammar construction algorithm with well known techniques relating pattern matching to 2D-range reporting. Dynamic Relative Compression, Dynamic Partial Sums, and Substring Concatenation. Given a static reference string R and a source string S, a relative compression of S with respect to R is an encoding of S as a sequence of references to substrings of R. Relative compression schemes are a classic model of compression and have recently proved very successful for compressing highly-repetitive massive data sets such as genomes and web-data. We initiate the study of
relative compression in a dynamic setting where the compressed source string $S$ is subject to edit operations. The goal is
to maintain the compressed representation compactly, while supporting edits and allowing efficient random access to the
(uncompressed) source string. We present new data structures that achieve optimal time for updates and queries while
using space linear in the size of the optimal relative compression, for nearly all combinations of parameters. We also
present solutions for restricted and extended sets of updates. To achieve these results, we revisit the dynamic partial
sums problem and the substring concatenation problem. We present new optimal or near optimal bounds for these
problems. Plugging in our new results we also immediately obtain new bounds for the string indexing for patterns with
We consider the well-studied partial sums problem in succinct space where one is to maintain an array of $n$-bit integers
subject to updates such that partial sums queries can be efficiently answered.
We present two succinct versions of the Fenwick Tree—known for its simplicity and practicality. Our results hold in the
encoding model where one is allowed to reuse the space from the input data. Our main result is the first that only requires
$nk + o(n)$ bits of space while still supporting sum/update in $O(\log_b n)$ time where $2 \leq b \leq \log O(1) n$. The
second result shows how optimal time for sum/update can be achieved while only slightly increasing the space usage to
$nk + o(nk)$ bits. Beyond Fenwick Trees, the results are primarily based on bit-packing and sampling—making them very
practical—and they also allow for simple optimal parallelization. Fast Dynamic Arrays. We present a highly optimized
implementation of tiered vectors, a data structure for maintaining a sequence of $n$ elements supporting access in time $O(1)$
and insertion and deletion in time $O(n)$ for $n > 0$ while using $o(n)$ extra space. We consider several different implementation
optimizations in C++ and compare their performance to that of vector and multiset from the standard library on sequences
with up to 108 elements. Our fastest implementation uses much less space than multiset while providing speedups of 40×
for access operations compared to multiset and speedups of 10,000× compared to vector for insertion and deletion
operations while being competitive with both data structures for all other operations. Parallel Lookups in String Indexes.
Here we consider the indexing problem on in the parallel random access machine model. Recently, the first PRAM
algorithms were presented for looking up a pattern in a suffix tree. We improve the bounds, achieving optimal results for all
parameters but the preprocessing. Given a text $T$ of length $n$ we create a data structure of size $O(n)$ that answers pattern
matching queries for a pattern $P$ of length $m$ in $O(\log m)$ time and $O(m)$ work.

Compressed Communication Complexity of Longest Common Prefixes
We consider the communication complexity of fundamental longest common prefix $LCP(A, B)$ problems. In the simplest version, two parties, Alice and Bob, each hold a string, $A$ and $B$, and we want to determine the
length of their longest common prefix $\ell = LCP(A, B)$ using as few rounds and bits of
communication as possible. We show that if the longest common prefix of $A$ and $B$ is compressible, then we can
significantly reduce the number of rounds compared to the optimal uncompressed protocol, while achieving the same (or
fewer) bits of communication. Namely, if the longest common prefix has an LZ77 parse of $z$ phrases, only $SO(\lg z)$$\$ rounds and $SO(\lg \ell)$$\$ total communication is necessary. We extend the result to the natural case when Bob holds a
set of strings $SB_1, ldots, SB_k$$\$, and the goal is to find the length of the maximal longest prefix shared by $A$ and any of
$SB_1, ldots, SB_k$$$. Here, we give a protocol with $SO(\log z)$$\$ rounds and $SO(\lg z + \lg k + \lg \ell)$$\$ total
communication. We present our result in the public-coin model of computation but by a standard technique our results
generalize to the private-coin model. Furthermore, if we view the input strings as integers the problems are the greater-
than problem and the predecessor problem.

General information
State: Published
Organisations: Department of Applied Mathematics and Computer Science, Algorithms and Logic
Contributors: Christiansen, A. R., Bille, P., Gørtz, I. L.
Number of pages: 149
Publication date: 2018

Publication information
Publisher: DTU Compute
Original language: English
Electronic versions:
phd464_Christiansen_AR.pdf
Research output: Research › Ph.D. thesis – Annual report year: 2018

Compressed Communication Complexity of Longest Common Prefixes
We consider the communication complexity of fundamental longest common prefix $LCP(A, B)$ problems. In the simplest version, two parties, Alice and Bob, each hold a string, $A$ and $B$, and we want to determine the
length of their longest common prefix $\ell = LCP(A, B)$ using as few rounds and bits of
communication as possible. We show that if the longest common prefix of $A$ and $B$ is compressible, then we can
significantly reduce the number of rounds compared to the optimal uncompressed protocol, while achieving the same (or
fewer) bits of communication. Namely, if the longest common prefix has an LZ77 parse of $z$ phrases, only $SO(\lg z)$$\$ rounds and $SO(\lg \ell)$$\$ total communication is necessary. We extend the result to the natural case when Bob holds a
set of strings $SB_1, ldots, SB_k$$\$, and the goal is to find the length of the maximal longest prefix shared by $A$ and any of
$SB_1, ldots, SB_k$$$. Here, we give a protocol with $SO(\log z)$$\$ rounds and $SO(\lg z + \lg k + \lg \ell)$$\$ total
communication. We present our result in the public-coin model of computation but by a standard technique our results
generalize to the private-coin model. Furthermore, if we view the input strings as integers the problems are the greater-
than problem and the predecessor problem.

General information
State: Published
Organisations: Department of Applied Mathematics and Computer Science, Algorithms and Logic, University of Pisa
Contributors: Bille, P., Berggreen Ettienne, M., Grossi, R., Görtz, I. L., Rotenberg, E.
Pages: 74-87
Publication date: 2018

Host publication information
Title of host publication: String Processing and Information Retrieval
Publisher: Springer
Dynamic Relative Compression, Dynamic Partial Sums, and Substring Concatenation

Given a static reference string R and a source string S, a relative compression of S with respect to R is an encoding of S as a sequence of references to substrings of R. Relative compression schemes are a classic model of compression and have recently proved very successful for compressing highly-repetitive massive data sets such as genomes and web-data. We initiate the study of relative compression in a dynamic setting where the compressed source string S is subject to edit operations. The goal is to maintain the compressed representation compactly, while supporting edits and allowing efficient random access to the (uncompressed) source string. We present new data structures that achieve optimal time for updates and queries while using space linear in the size of the optimal relative compression, for nearly all combinations of parameters. We also present solutions for restricted and extended sets of updates. To achieve these results, we revisit the dynamic partial sums problem and the substring concatenation problem. We present new optimal or near optimal bounds for these problems. Plugging in our new results we also immediately obtain new bounds for the string indexing for patterns with wildcards problem and the dynamic text and static pattern matching problem.

General information
State: Published
Organisations: Department of Applied Mathematics and Computer Science, Algorithms and Logic
Number of pages: 18
Pages: 3207-3224
Publication date: 2018
Peer-reviewed: Yes

Publication information
Journal: Algorithmica
Volume: 80
Issue number: 11
ISSN (Print): 0178-4617
Ratings:
BFI (2019): BFI-level 2
Web of Science (2019): Indexed yes
BFI (2018): BFI-level 2
Web of Science (2018): Indexed yes
BFI (2017): BFI-level 2
Scopus rating (2017): CiteScore 1.27 SJR 0.56 SNIP 1.354
Web of Science (2017): Impact factor 0.667
Web of Science (2017): Indexed yes
BFI (2016): BFI-level 2
Scopus rating (2016): CiteScore 1.11 SJR 0.648 SNIP 1.184
Web of Science (2016): Impact factor 0.735
Web of Science (2016): Indexed yes
BFI (2015): BFI-level 2
Scopus rating (2015): CiteScore 1.15 SJR 0.739 SNIP 1.16
Web of Science (2015): Impact factor 0.795
Web of Science (2015): Indexed yes
BFI (2014): BFI-level 2
Scopus rating (2014): CiteScore 1.2 SJR 0.854 SNIP 1.197
Web of Science (2014): Impact factor 0.791
Web of Science (2014): Indexed yes
BFI (2013): BFI-level 2
Scopus rating (2013): CiteScore 1.26 SJR 0.952 SNIP 1.395
Finger Search in Grammar-Compressed Strings

Grammar-based compression, where one replaces a long string by a small context-free grammar that generates the string, is a simple and powerful paradigm that captures many popular compression schemes. Given a grammar, the random access problem is to compactly represent the grammar while supporting random access, that is, given a position in the original uncompressed string report the character at that position. In this paper we study the random access problem with the finger search property, that is, the time for a random access query should depend on the distance between a specified index $f$, called the finger, and the query index $i$. We consider both a static variant, where we first place a finger and subsequently access indices near the finger efficiently, and a dynamic variant where also moving the finger such that the time depends on the distance moved is supported. Let $n$ be the size the grammar, and let $N$ be the size of the string. For the static variant we give a linear space representation that supports placing the finger in $O(\log N)$ time and subsequently accessing in $O(\log D)$ time, where $D$ is the distance between the finger and the accessed index. For the dynamic variant we give a linear space representation that supports placing the finger in $O(\log N)$ time and accessing and moving the finger in $O(\log D + \log \log N)$ time. Compared to the best linear space solution to random access, we improve a $O(\log N)$ query bound to $O(\log D)$ for the static variant and to $O(\log D + \log \log N)$ for the dynamic variant, while maintaining linear space. As an application of our results we obtain an improved solution to the longest common extension problem in grammar compressed strings. To obtain our results, we introduce several new techniques of independent interest, including a novel van Emde Boas style decomposition of grammars.
Fingerprints in compressed strings

In this paper we show how to construct a data structure for a string S of size N compressed into a context-free grammar of size n that supports efficient Karp–Rabin fingerprint queries to any substring of S. That is, given indices i and j, the answer to a query is the fingerprint of the substring S[i..j]. We present the first O(n) space data structures that answer fingerprint queries without decompressing any character. For Straight Line Programs (SLP) we get O(logN) query time, and for Linear SLPs (an SLP derivative that captures LZ78 compression and its variations) we get O(loglogN) query time. We extend the result to solve the longest common extension problem in query time O(logNlogℓ) and O(logℓloglogℓ+loglogN) for SLPs and Linear SLPs, respectively. Here, ℓ denotes the length of the LCE.
Subsequence automata with default transitions

Let $S$ be a string of length $n$ with characters from an alphabet of size $\sigma$. The subsequence automaton of $S$ (often called the directed acyclic subsequence graph) is the minimal deterministic finite automaton accepting all subsequences of $S$. A straightforward construction shows that the size (number of states and transitions) of the subsequence automaton is $O(n\sigma)$ and that this bound is asymptotically optimal. In this paper, we consider subsequence automata with default transitions, that is, special transitions to be taken only if none of the regular transitions match the current character, and which do not consume the current character. We show that with default transitions, much smaller subsequence automata are possible, and provide a full trade-off between the size of the automaton and the delay, i.e., the maximum number of consecutive default transitions followed before consuming a character. Specifically, given any integer parameter $k$, $1 < k \leq \sigma$, we present a subsequence automaton with default transitions of size $O(n \log k \sigma)$ and delay $O(\log k \sigma)$. Hence, with $k=2$ we obtain an automaton of size $O(n \log \sigma)$ and delay $O(\log \sigma)$. At the other extreme, with $k=\sigma$, we obtain an automaton of size $O(n \sigma)$ and delay $O(1)$, thus matching the bound for the standard subsequence automaton construction. Finally, we generalize the result to multiple strings. The key component of our result is a novel hierarchical automata construction of independent interest.

General information

State: Published
Organisations: Department of Applied Mathematics and Computer Science, Algorithms and Logic, Technical University of Denmark
Contributors: Bille, P., Gørtz, I. L., Skjoldjensen, F. R.
Number of pages: 8
Pages: 48-55
Publication date: 1 May 2017
Peer-reviewed: Yes

Publication information

Journal: Journal of Discrete Algorithms
Volume: 44
ISSN (Print): 1570-8667
Ratings:
BFI (2019): BFI-level 1
We present a new algorithm for subsequence matching in grammar compressed strings. Given a grammar of size $n$ compressing a string of size $N$ and a pattern string of size $m$ over an alphabet of size $\sigma$, our algorithm uses $O(n + \frac{n\sigma}{w})$ space and $O(n + \frac{n\sigma}{w} + m \log N \log w \cdot \text{occ})$ or $O(n + \frac{n\sigma}{w} \log w + m \log N \log \text{occ})$ time. Here $w$ is the word size and $\text{occ}$ is the number of minimal occurrences of the pattern. Our algorithm uses less space than previous algorithms and is also faster for $\text{occ}=o(\frac{n}{\log N})$ occurrences. The algorithm uses a new data structure that allows us to efficiently find the next occurrence of a given character after a given position in a compressed string. This data structure in turn is based on a new data structure for the tree color problem, where the node colors are packed in bit strings.

Compressed Subsequence Matching and Packed Tree Coloring

We present a new algorithm for subsequence matching in grammar compressed strings. Given a grammar of size $n$ compressing a string of size $N$ and a pattern string of size $m$ over an alphabet of size $\sigma$, our algorithm uses $O(n + \frac{n\sigma}{w})$ space and $O(n + \frac{n\sigma}{w} + m \log N \log w \cdot \text{occ})$ or $O(n + \frac{n\sigma}{w} \log w + m \log N \log \text{occ})$ time. Here $w$ is the word size and $\text{occ}$ is the number of minimal occurrences of the pattern. Our algorithm uses less space than previous algorithms and is also faster for $\text{occ}=o(\frac{n}{\log N})$ occurrences. The algorithm uses a new data structure that allows us to efficiently find the next occurrence of a given character after a given position in a compressed string. This data structure in turn is based on a new data structure for the tree color problem, where the node colors are packed in bit strings.
Deterministic indexing for packed strings

Given a string $S$ of length $n$, the classic string indexing problem is to preprocess $S$ into a compact data structure that supports efficient subsequent pattern queries. In the deterministic variant the goal is to solve the string indexing problem without any randomization (at preprocessing time or query time). In the packed variant the strings are stored with several characters in a single word, giving us the opportunity to read multiple characters simultaneously. Our main result is a new string index in the deterministic and packed setting. Given a packed string $S$ of length $n$ over an alphabet $\sigma$, we show how to preprocess $S$ in $O(n)$ (deterministic) time and space $O(n)$ such that given a packed pattern string of length $m$ we can support queries in (deterministic) time $O \left( \frac{m}{\alpha} + \log m + \log \log \sigma \right)$, where $\alpha = \frac{w}{\log \sigma}$ is the number of characters packed in a word of size $w = \Theta(\log n)$. Our query time is always at least as good as the previous best known bounds and whenever several characters are packed in a word, i.e., $\log \sigma$.
Immersive Algorithms: Better Visualization with Less Information

Visualizing algorithms, such as drawings, slideshow presentations, animations, videos, and software tools, is a key concept to enhance and support student learning. A typical visualization of an algorithm shows the data and then performs computation on the data. For instance, a standard visualization of a standard binary search on an array shows an array of sorted numbers and then illustrates the action of the algorithm in a step-by-step fashion. However, this approach does not fully capture the computational environment from the perspective of the algorithm. Specifically, the algorithm does not "see" the full sorted array, but only the single position that it accesses during each step of the computation. To fix this discrepancy we introduce the immersive principle that states that at any point in time, the displayed information should closely match the information accessed by the algorithm.

We give several examples of immersive visualizations of basic algorithms and data structures, discuss methods for implementing it, and briefly evaluate it.

Lempel-Ziv Compression in a Sliding Window

We present new algorithms for the sliding window Lempel-Ziv (LZ77) problem and the approximate rightmost LZ77 parsing problem. Our main result is a new and surprisingly simple algorithm that computes the sliding window LZ77 parse in $O(w)$ space and either $O(n)$ expected time or $O(n \log \log w + z \log \log \sigma)$ deterministic time. Here, $w$ is the window size, $n$ is the size of the input string, $z$ is the number of phrases in the parse, and $\sigma$ is the size of the alphabet. This matches the space and time bounds of previous results while removing constant size restrictions on the alphabet size. To achieve our result, we combine a simple modification and augmentation of the suffix tree with periodicity properties of sliding windows. We also apply this new technique to obtain an algorithm for the approximate rightmost LZ77 problem that uses $O(n(\log z + \log \log n))$ time and $O(n)$ space and produces a $(1 + \epsilon)$-approximation of the rightmost parsing (any constant $\epsilon > 0$). While this does not improve the best known time-space trade-offs for exact rightmost parsing, our algorithm is significantly simpler and exposes a direct connection between sliding window parsing and the approximate rightmost matching problem.
Space-Efficient Re-Pair Compression

Re-Pair [5] is an effective grammar-based compression scheme achieving strong compression rates in practice. Let n, σ, and d be the text length, alphabet size, and dictionary size of the final grammar, respectively. In their original paper, the authors show how to compute the Re-Pair grammar in expected linear time and $5n + 4\sigma^2 + 4d + \sqrt{n}$ words of working space on top of the text. In this work, we propose two algorithms improving on the space of their original solution. Our model assumes a memory word of $\log_2 n$ bits and a re-writable input text composed by n such words. Our first algorithm runs in expected $O(n/\varepsilon)$ time and uses $(1+\varepsilon)n + \sqrt{n}$ words of space on top of the text for any parameter $0 < \varepsilon \leq 1$ chosen in advance. Our second algorithm runs in expected $O(n \log n)$ time and improves the space to $n + \sqrt{n}$ words.

Tight bounds for top tree compression

We consider compressing labeled, ordered and rooted trees using DAG compression and top tree compression. We show that there exists a family of trees such that the size of the DAG compression is always a logarithmic factor smaller than the size of the top tree compression (even for an alphabet of size 1). The result settles an open problem from Bille et al. (Inform. and Comput., 2015).
Time-space trade-offs for Lempel-Ziv compressed indexing

Given a string $S$, the compressed indexing problem is to preprocess $S$ into a compressed representation that supports fast substring queries. The goal is to use little space relative to the compressed size of $S$ while supporting fast queries. We present a compressed index based on the Lempel-Ziv 1977 compression scheme. Let $n$, and $z$ denote the size of the input string, and the compressed LZ77 string, respectively. We obtain the following time-space trade-offs. Given a pattern string $P$ of length $m$, we can solve the problem in (i) $O(m + \text{occ} \log \log n)$ time using $O(z \log (n/z) \log z)$ space, or (ii) $(m(1 + \log z/\log(n/z)) + \text{occ}(\log \log n + \log z))$ time using $O(z \log (n/z))$ space, for any $0 < \eta < 1$. In particular, (i) improves the leading term in the query time of the previous best solution from $O(m \log m)$ to $O(m)$ at the cost of increasing the space by a factor $\log \log z$. Alternatively, (ii) matches the previous best space bound, but has a leading term in the query time of $O(m(1 + \log z/\log(n/z)))$. However, for any polynomial compression ratio, i.e., $z = O(n^{1-\delta})$, for constant $\delta > 0$, this becomes $O(m)$. Our index also supports extraction of any substring of length $\ell$ in $O(\ell + \log(n/z))$ time. Technically, our results are obtained by novel extensions and combinations of existing data structures of independent interest, including a new batched variant of weak prefix search.

Boxed Permutation Pattern Matching.

Given permutations $T$ and $P$ of length $n$ and $m$, respectively, the Permutation Pattern Matching problem asks to find all $m$-length subsequences of $T$ that are order-isomorphic to $P$. This problem has a wide range of applications but is known to be NP-hard. In this paper, we study the special case, where the goal is to only find the boxed subsequences of $T$ that are order-isomorphic to $P$. This problem was introduced by Bruner and Lackner who showed that it can be solved in $O(n^3)$ time. Cho et al. [CPM 2015] gave an $O(n^m)$ time algorithm and improved it to $O(n^2 \log m)$. In this paper we present a solution that uses only $O(n^2)$ time. In general, there are instances where the output size is $\Omega(n^2)$ and hence our bound is optimal. To achieve our results, we introduce several new ideas including a novel reduction to 2D offline dominance counting. Our algorithm is surprisingly simple and straightforward to implement.
Capacitated Vehicle Routing with Non-Uniform Speeds

The capacitated vehicle routing problem (CVRP) involves distributing identical items from a depot to a set of demand locations using a single capacitated vehicle. We introduce the heterogeneous capacitated vehicle routing problem, a generalization of CVRP to the setting of multiple vehicles having nonuniform speeds, and present for it a constant-factor approximation algorithm.

Our main contribution is an approximation algorithm for the heterogeneous traveling salesman problem, which is the special case of heterogeneous CVRP with uncapacitated vehicles. Given a metric denoting distances between vertices, a depot \( r \) containing \( k \) vehicles having respective speeds \( \lambda_k \), the objective in heterogeneous TSP is to find a tour for each vehicle (starting and ending at \( r \)) so that every vertex is covered in some tour and the maximum completion time is minimized; the completion time of a vehicle is the distance traveled divided by its speed.

Our algorithm relies on a new approximate minimum spanning tree construction called \textit{Level-Prim}, which is related to but different from \textit{Light Approximate Shortest-path Trees}. We also extend the widely used tour-splitting technique to nonuniform speeds, using ideas from the 2-approximation algorithm for scheduling in unrelated machines.
Distance labeling schemes for trees

We consider distance labeling schemes for trees: given a tree with n nodes, label the nodes with binary strings such that, given the labels of any two nodes, one can determine, by looking only at the labels, the distance in the tree between the two nodes. A lower bound by Gavoille et al. [Gavoille et al., J. Alg., 2004] and an upper bound by Peleg [Peleg, J. Graph Theory, 2000] establish that labels must use Θ(log^2(n)) bits. Gavoille et al. [Gavoille et al., ESA, 2001] show that for very small approximate stretch, labels use Θ(log(n) log(log(n))) bits. Several other papers investigate various variants such as, for example, small distances in trees [Alstrup et al., SODA, 2003]. We improve the known upper and lower bounds of exact distance labeling by showing that 1/4 log2(n) bits are needed and that 1/2 log2(n) bits are sufficient. We also give (1 + ε)-stretch labeling schemes using Θ(log(n)) bits for constant ε > 0. (1 + ε)-stretch labeling schemes with polylogarithmic label size have previously been established for doubling dimension graphs by Talwar [Talwar, STOC, 2004]. In addition, we present matching upper and lower bounds for distance labeling for caterpillars, showing that labels must have size 2log n - Θ(log log n). For simple paths with k nodes and edge weights in [1,n], we show that labels must have size (k - 1)/k log n + Θ(log k).

General information

State: Published
Organisations: Department of Applied Mathematics and Computer Science, Algorithms and Logic, University of Copenhagen, Bar-Ilan University
Contributors: Alstrup, S., Gørtz, I. L., Bistrup Halvorsen, E., Porat, E.
Pages: 1-16
Publication date: 2016

Host publication information
Title of host publication: Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)
Article number: 132
Keywords: Distributed computing, Distance labeling, Graph theory, Routing, Trees
Electronic versions: LIPIcs.ICALP.2016.132.pdf
DOIs: 10.4230/LIPIcs.ICALP.2016.132
Dynamic Relative Compression, Dynamic Partial Sums, and Substring Concatenation

Given a static reference string $R$ and a source string $S$, a relative compression of $S$ with respect to $R$ is an encoding of $S$ as a sequence of references to substrings of $R$. Relative compression schemes are a classic model of compression and have recently proved very successful for compressing highly-repetitive massive data sets such as genomes and web-data. We initiate the study of relative compression in a dynamic setting where the compressed source string $S$ is subject to edit operations. The goal is to maintain the compressed representation compactly, while supporting edits and allowing efficient random access to the (uncompressed) source string. We present new data structures that achieve optimal time for updates and queries while using space linear in the size of the optimal relative compression, for nearly all combinations of parameters. We also present solutions for restricted and extended sets of updates. To achieve these results, we revisit the dynamic partial sums problem and the substring concatenation problem. We present new optimal or near optimal bounds for these problems. Plugging in our new results we also immediately obtain new bounds for the string indexing for patterns with wildcards problem and the dynamic text and static pattern matching problem.

Finger Search in Grammar-Compressed Strings

Grammar-based compression, where one replaces a long string by a small context-free grammar that generates the string, is a simple and powerful paradigm that captures many popular compression schemes. Given a grammar, the random access problem is to compactly represent the grammar while supporting random access, that is, given a position in the original uncompressed string report the character at that position. In this paper we study the random access problem with the finger search property, that is, the time for a random access query should depend on the distance between a specified index $f$, called the finger, and the query index $i$. We consider both a static variant, where we first place a finger and subsequently access indices near the finger efficiently, and a dynamic variant where also moving the finger such that the time depends on the distance moved is supported.

Let $n$ be the size the grammar, and let $N$ be the size of the string. For the static variant we give a linear space representation that supports placing the finger in $O(\log(N))$ time and subsequently accessing in $O(\log(D))$ time, where $D$ is the distance between the finger and the accessed index. For the dynamic variant we give a linear space representation that supports placing the finger in $O(\log(N))$ time and accessing and moving the finger in $O(\log(D) + \log(\log(N)))$ time. Compared to the best linear space solution to random access, we improve a $O(\log(N))$ query bound to $O(\log(D))$ for the static variant and to $O(\log(D) + \log(\log(N)))$ for the dynamic variant, while maintaining linear space. As an application of our results we obtain an improved solution to the longest common extension problem in grammar compressed strings. To obtain our results, we introduce several new techniques of independent interest, including a novel van Emde Boas style decomposition of grammars.
Locating Depots for Capacitated Vehicle Routing

We study a location-routing problem in the context of capacitated vehicle routing. The input to the k-location capacitated vehicle routing problem (k-LocVRP) consists of a set of demand locations in a metric space and a fleet of k identical vehicles, each of capacity Q. The objective is to locate k depots, one for each vehicle, and compute routes for the vehicles so that all demands are satisfied and the total cost is minimized. Our main result is a constant-factor approximation algorithm for k-LocVRP. In obtaining this result, we introduce a common generalization of the k-median and minimum spanning tree problems (called k median forest), which might be of independent interest. We give a local-search based (3+ε)-approximation algorithm for k median forest, which leads to a (12+ε)-approximation algorithm for k-LocVRP, for any constant ε>0.

General information

State: Published
Organisations: Department of Applied Mathematics and Computer Science , Algorithms and Logic , University of Michigan, Ann Arbor
Contributors: Gørtz, I. L., Nagarajan, V.
Pages: 94-103
Publication date: 2016
Peer-reviewed: Yes

Publication information

Journal: Networks
Volume: 68
Issue number: 2
ISSN (Print): 0028-3045
Ratings:
BFI (2019): BFI-level 2
Web of Science (2019): Indexed yes
BFI (2018): BFI-level 2
Web of Science (2018): Indexed yes
BFI (2017): BFI-level 1
Scopus rating (2017): CiteScore 1.44 SJR 0.94 SNIP 1.38
Web of Science (2017): Impact factor 1.121
Web of Science (2017): Indexed yes
BFI (2016): BFI-level 1
Scopus rating (2016): CiteScore 1.22 SJR 1.119 SNIP 1.084
Web of Science (2016): Impact factor 1.213
Web of Science (2016): Indexed yes
BFI (2015): BFI-level 2
Scopus rating (2015): CiteScore 1.21 SJR 0.896 SNIP 0.974
Web of Science (2015): Impact factor 0.943
BFI (2014): BFI-level 2
Scopus rating (2014): CiteScore 1.03 SJR 0.941 SNIP 1.349
The longest common extension (LCE) of two indices in a string is the length of the longest identical substrings starting at these two indices. The LCE problem asks to preprocess a string into a compact data structure that supports fast LCE queries.

In this paper we generalize the LCE problem to trees and suggest a few applications of LCE in trees to tries and XML databases. Given a labeled and rooted tree $T$ of size $n$, the goal is to preprocess $T$ into a compact data structure that support the following LCE queries between subpaths and subtrees in $T$. Let $v_1$, $v_2$, $w_1$, and $w_2$ be nodes of $T$ such that $w_1$ and $w_2$ are descendants of $v_1$ and $v_2$ respectively.

- **LCEPP**($v_1$, $w_1$, $v_2$, $w_2$): (path-path LCE) return the longest common prefix of the paths $v_1 \rightarrow w_1$ and $v_2 \rightarrow w_2$.

- **LCEPT**($v_1$, $w_1$, $v_2$): (path-tree LCE) return maximal path-path LCE of the path $v_1 \rightarrow w_1$ and any path from $v_2$ to a descendant leaf.

- **LCETT**($v_1$, $v_2$): (tree-tree LCE) return a maximal path-path LCE of any pair of paths from $v_1$ and $v_2$ to descendant leaves.
We present the first non-trivial bounds for supporting these queries. For LCEPP queries, we present a linear-space solution with $O(\log^* n)$ query time. For LCEPT queries, we present a linear-space solution with $O((\log \log n)^2)$ query time, and complement this with a lower bound showing that any path-tree LCE structure of size $O(n \text{polylog}(n))$ must necessarily use $O(\log \log n)$ time to answer queries. For LCETT queries, we present a time-space trade-off, that given any parameter $\tau$, $1 \leq \tau \leq n$, leads to an $O(n\tau)$ space and $O(n/\tau)$ query-time solution. This is complemented with a reduction to the set intersection problem implying that a fast linear space solution is not likely to exist.

**General information**

State: Published
Organisations: Department of Applied Mathematics and Computer Science, Algorithms and Logic, University of Warsaw, University of Haifa
Pages: 98-107
Publication date: 2016
Peer-reviewed: Yes

**Publication information**

Journal: Theoretical Computer Science
Volume: 638
ISSN (Print): 0304-3975
Ratings:
- BFI (2019): BFI-level 2
- Web of Science (2019): Indexed yes
- BFI (2018): BFI-level 2
- Web of Science (2018): Indexed yes
- BFI (2017): BFI-level 2
- Scopus rating (2017): CiteScore 1.08 SJR 0.488 SNIP 0.996
- Web of Science (2017): Impact factor 0.772
- Web of Science (2017): Indexed yes
- BFI (2016): BFI-level 2
- Scopus rating (2016): CiteScore 0.97 SJR 0.547 SNIP 0.996
- Web of Science (2016): Impact factor 0.698
- Web of Science (2016): Indexed yes
- BFI (2015): BFI-level 2
- Scopus rating (2015): CiteScore 1 SJR 0.592 SNIP 1.123
- Web of Science (2015): Impact factor 0.643
- Web of Science (2015): Indexed yes
- BFI (2014): BFI-level 2
- Scopus rating (2014): CiteScore 1.08 SJR 0.669 SNIP 1.148
- Web of Science (2014): Impact factor 0.657
- Web of Science (2014): Indexed yes
- BFI (2013): BFI-level 2
- Scopus rating (2013): CiteScore 1.17 SJR 0.717 SNIP 1.297
- Web of Science (2013): Impact factor 0.516
- ISI indexed (2013): ISI indexed yes
- BFI (2012): BFI-level 2
- Scopus rating (2012): CiteScore 1.16 SJR 0.78 SNIP 1.21
- Web of Science (2012): Impact factor 0.489
- ISI indexed (2012): ISI indexed yes
- Web of Science (2012): Indexed yes
- BFI (2011): BFI-level 2
- Scopus rating (2011): CiteScore 1.17 SJR 0.747 SNIP 1.246
- Web of Science (2011): Impact factor 0.665
- ISI indexed (2011): ISI indexed yes
- Web of Science (2011): Indexed yes
- BFI (2010): BFI-level 2
- Scopus rating (2010): SJR 0.839 SNIP 1.278
In this work, we present efficient algorithms for constructing sparse suffix trees, sparse suffix arrays, and sparse position heaps for $b$ arbitrary positions of a text $T$ of length $n$ while using only $O(b)$ words of space during the construction. Attempts at breaking the naïve bound of $\Omega(nb)$ time for constructing sparse suffix trees in $O(b)$ space can be traced back to the origins of string indexing in 1968. First results were not obtained until 1996, but only for the case in which the $b$ suffixes were evenly spaced in $T$. In this article, there is no constraint on the locations of the suffixes. Our main contribution is to show that the sparse suffix tree (and array) can be constructed in $O(n \log_2 b)$ time. To achieve this, we develop a technique that allows one to efficiently answer $b$ longest common prefix queries on suffixes of $T$, using only $O(b)$ space. We expect that this technique will prove useful in many other applications in which space usage is a concern. Our first solution is Monte Carlo, and outputs the correct tree with high probability. We then give a Las Vegas algorithm, which also uses $O(b)$ space and runs in the same time bounds with high probability when $b = O(\sqrt{n})$. Additional trade-offs between space usage and construction time for the Monte Carlo algorithm are given. Finally, we show that, at the expense of slower pattern queries, it is possible to construct sparse position heaps in $O(n + \log_b b)$ time and $O(b)$ space.

**General information**

**State:** Published

**Organisations:** Department of Applied Mathematics and Computer Science, Algorithms and Logic, Technical University of Dortmund, Weizmann Institute of Science, University of Bristol

**Contributors:** Bille, P., Fischer, J., Gertz, I. L., Kopelowitz, T., Sach, B., Vildhøj, H. W.

**Pages:** 1-19

**Publication date:** 2016

**Peer-reviewed:** Yes

**Publication Information**

**Journal:** A C M Transactions on Algorithms

**Volume:** 12

**Issue number:** 3
Subsequence Automata with Default Transitions

Let $S$ be a string of length $n$ with characters from an alphabet of size $\sigma$. The subsequence automaton of $S$ (often called the directed acyclic subsequence graph) is the minimal deterministic finite automaton accepting all subsequences of $S$. A straightforward construction shows that the size (number of states and transitions) of the subsequence automaton is $O(n\sigma)$ and that this bound is asymptotically optimal. In this paper, we consider subsequence automata with default transitions, that is, special transitions to be taken only if none of the regular transitions match the current character, and which do not consume the current character. We show that with default transitions, much smaller subsequence automata are possible, and provide a full trade-off between the size of the automaton and the delay, i.e., the maximum number of consecutive default transitions followed before consuming a character. Specifically, given any integer parameter $k$, $1 < k \leq \sigma$, we present a subsequence automaton with default transitions of size $O(nk\log_k \sigma)$ and delay $O(\log_k \sigma)$. Hence, with $k = 2$ we obtain an automaton of size $O(n \log \sigma)$ and delay $O(\log \sigma)$. On the other extreme, with $k = \sigma$, we obtain an automaton of size $O(n \sigma)$ and delay $O(1)$, thus matching the bound for the standard subsequence automaton construction. The key component of our result is a novel hierarchical automata construction of independent interest.

General information
State: Published
Organisations: Department of Applied Mathematics and Computer Science, Algorithms and Logic
Contributors: Bille, P., Gørtz, I. L., Skjoldjensen, F. R.
Number of pages: 9
Pages: 208-216
Publication date: 2016

Host publication information
Volume: 9587
Publisher: Springer
ISBN (Print): 978-3-662-49191-1
ISBN (Electronic): 978-3-662-49192-8
(Lecture Notes in Computer Science).
Electronic versions: 1510.08748.pdf
DOIs: 10.1007/978-3-662-49192-8_17
Source: FindIt
Source-ID: 2304141877
Research output: Research - peer-review › Article in proceedings – Annual report year: 2016

Algorithms and data structures for grammar-compressed strings

Textual databases for e.g. biological or web-data are growing rapidly, and it is often only feasible to store the data in compressed form. However, compressing the data comes at a price. Traditional algorithms for e.g. pattern matching requires all data to be decompressed - a computationally demanding task. In this thesis we design data structures for accessing and searching compressed data efficiently.

Our results can be divided into two categories. In the first category we study problems related to pattern matching. In particular, we present new algorithms for counting and comparing substrings, and a new algorithm for finding all occurrences of a pattern in which we may insert gaps. In the other category we deal with accessing and decompressing parts of the compressed string. We show how to quickly access a single character of the compressed string, and present a data structure that supports fast decompression of substrings from prespecified positions.

General information
State: Published
Organisations: Department of Applied Mathematics and Computer Science, Algorithms and Logic
Contributors: Cording, P. H., Bille, P., Gørtz, I. L.
Number of pages: 66
Publication date: 2015

Publication information
Place of publication: Kgs. Lyngby
Publisher: Technical University of Denmark (DTU)
Compressed Fingerprints. The Karp-Rabin fingerprint of a string is a useful type of hash value that has multiple applications due to its strong properties. Given a string $S$ of length $N$ compressed into a straight line program (SLP) of size $n$, we show a $O(n)$ space data structure that supports fingerprint queries, retrieving the fingerprint of any substring of $S$. Queries are answered in $O(lg N)$ time. If the compression is a Linear SLP (capturing LZ78 compression and variations), we get $O(lg lg N)$ query time.

Our structure matches the best known query time bound for random access in SLPs, and is the first for general (unbalanced) SLPs that answers fingerprint queries without decompressing any text. We also support longest common extension queries, returning the length $r$ that the substrings from two given positions in $S$ are equal. Answers are correct w.h.p. and take time $O(lg N lg r)$ and $O(lg lg N + lg r lg lg r)$ for SLPs and Linear SLPs, respectively.

Dynamic Compression. In the dynamic relative compression scheme, we compress a string $S$ of length $N$ into $n$ substrings of a given reference string of length $r$. We give data structures that maintain an asymptotically optimal compression in the scheme and support access, replace, insert and delete operations on $S$. Our solutions support each operation in $O(lg n/lg lg n + lg lg r)$ time and $O(n lg r)$ space; or $O(lg n/lg lg n)$ time and $O(n lg r)$ space. They can be naturally generalized to compress multiple strings.

Our solutions obtain almost-optimal bounds, and are the first to dynamically maintain a string under a compression scheme that can achieve better than entropy compression. We also give improved results for the substring concatenation problem, and an extension of our structure can be used as a black box to get an improved solution to the previously studied dynamic text static pattern problem.

Compressed Pattern Matching. In the streaming model, input data flows past a client one item at a time, but is far too large for the client to store. The annotated streaming model extends the model by introducing a powerful but untrusted annotator (representing “the cloud”) that can annotate input elements with additional information, sent as one-way communication to the client. We generalize the annotated streaming model to be able to solve problems on strings and present a data structure that allows us to trade off client space and annotation size. This lets us exploit the power of the annotator.

In compressed pattern matching we must report occurrences of a pattern of length $m$ in a text compressed into $n$ phrases (capturing LZ78 compression and variations). In the streaming model, any solution to the problem requires $\Omega(n)$ space. We show that the problem can be solved in $O(lg n)$ client space in the annotated streaming model, using $O(lg n)$ time and $O(lg n)$ words of annotation per phrase. Our solution shows that the annotator let us solve previously impossible problems, and it is the first solution to a classic problem from combinatorial pattern matching in the annotated streaming model.

Pattern Extraction. The problem of extracting important patterns from text has many diverse applications such as data mining, intrusion detection and genomic analysis. Consequently, there are many variations of the pattern extraction problem with different notions of patterns and importance measures. We study a natural variation where patterns must 1) contain at most $k$ don’t cares that each match a single character, and 2) have at least $q$ occurrences. Both $k$ and $q$ are input parameters.

We show how to extract such patterns and their occurrences from a text of length $n$ in $O(nk+3occ)$ time and space, where $occ$ is the total number of pattern occurrences. Our bound is the first output-sensitive solution for any approximate variation of the pattern extraction problem, with all previous solutions requiring $\Omega(n^2)$ time per reported pattern. Our algorithm is relatively simple, but requires a novel analysis technique that amortizes the cost of creating the index over the number of pattern occurrences.

Compressed Point Sets. Orthogonal range searching on a set of points is a classic geometric data structure problem. Given a query range, solutions must either count or report the points inside the range. Variants of this problem has numerous classic solutions, typically storing the points in a tree.

We show that almost any such classic data structure can be compressed without asymptotically increasing the time spent answering queries. This allows us to reduce the required space use if the point set contains geometric repetitions (copies of equal point set that are translated relative to each other). Our result captures most classic data structures, such as Range Trees, KD-trees, R-trees and Quad Trees. We also show a hierarchical clustering algorithm for ensuring that geometric repetitions are compressed.

Points with Colors. Colored orthogonal range searching is a natural generalization of orthogonal range searching which allows us to perform statistic analysis of a point set. We must store $n$ points that each have a color (sometimes called a category) and support queries that either count or report the distinct colors of the points inside a query range.
We show data structures that support both types of queries in sublinear time, storing two-dimensional points in linear space and high-dimensional points in almost-linear space. These are the first (almost) linear space solutions with sublinear query time. We also give the first dynamic solution with sublinear query time for any dimensionality. Previous solutions answer queries faster, but require much more space.

Points with Weights in Practice. If points are each assigned a weight, it is natural to consider the threshold range counting problem. A data structure must store the points and be able to count the number of points within a query range with a weight exceeding some threshold. This query appears naturally in a software system built by Milestone Systems, and allows detecting motion in video from surveillance cameras.

We implement a prototype of an index for 3-dimensional points that use little space and answers threshold queries efficiently. In experiments on realistic data sets, our prototype shows a speedup of at least a factor 30 at the expense of 10% additional space use compared to the previous approach. An optimized version of our proposed index is implemented in the latest version of the Milestone Systems software system.

Finger Predecessor. The predecessor problem is to store a set of n integers from a universe of size N to support predecessor queries, returning the largest integer in the set smaller than a given integer q. We study a variation where the query additionally receives a finger to an integer i in the set from which to start the search. We show a linear space data structure that answers such finger predecessor queries in $O(lg lg |i - q|)$ time. This generalizes and improves the $O(lg lg N)$ time solutions for the standard predecessor problem. Our data structure is the first with a query time that only depends on the numerical distance between the finger and the query integer.

Dynamic Partial Sums. The well-studied partial sums problem is to store a sequence of n integers with support for sum and search queries. The sequence is static in the sense that its length cannot change, but the update operation can be used to change the value of an integer in the sequence by a given value. There are matching lower and upper bounds showing that the problem can be solved on the w-bit Word RAM in linear space and $O((lg n = lg(w=.)))$ time per operation, where $._.$ is the maximum number of bits allowed in updates.

As a natural generalization we consider dynamic partial sums, allowing insertions and deletions in the sequence. Our solution requires linear space and supports all operations in optimal worst-case time $O(lg n/ lg(w/δ))$, matching lower bounds for all supported operations. Our data structure is the first dynamic partial sums solution that matches the lower bounds, and the first to support storing integers of more than $lg w$ bits.
Longest Common Extensions in Sublinear Space

The longest common extension problem (LCE problem) is to construct a data structure for an input string $T$ of length $n$ that supports $LCE(i,j)$ queries. Such a query returns the length of the longest common prefix of the suffixes starting at positions $i$ and $j$ in $T$. This classic problem has a well-known solution that uses $O(n)$ space and $O(1)$ query time. In this paper we show that for any trade-off parameter $1 \leq \tau \leq n$, the problem can be solved in $O(n/\tau)$ space and $O(\tau)$ query time. This significantly improves the previously best known time-space trade-offs, and almost matches the best known time-space product lower bound.

General information

State: Published
Organisations: Department of Applied Mathematics and Computer Science, Algorithms and Logic, University of Copenhagen, Bar-Ilan University
Pages: 65-76
Publication date: 2015

Host publication information

Title of host publication: Combinatorial Pattern Matching : 26th Annual Symposium, CPM 2015, Ischia Island, Italy, June 29 -- July 1, 2015, Proceedings
Publisher: Springer
ISBN (Print): 978-3-319-19928-3
ISBN (Electronic): 978-3-319-19929-0
(Lecture Notes in Computer Science, Vol. 9133).
Electronic versions:
paper_58.pdf
DOIs:
10.1007/978-3-319-19929-0_6
Research output: Research - peer-review › Article in proceedings – Annual report year: 2015

Longest Common Extensions in Trees

The longest common extension (LCE) of two indices in a string is the length of the longest identical substrings starting at these two indices. The LCE problem asks to preprocess a string into a compact data structure that supports fast LCE queries.

In this paper we generalize the LCE problem to trees and suggest a few applications of LCE in trees to tries and XML databases. Given a labeled and rooted tree $T$ of size $n$, the goal is to preprocess $T$ into a compact data structure that support the following LCE queries between subpaths and subtrees in $T$. Let $v_1$, $v_2$, $w_1$, and $w_2$ be nodes of $T$ such that $w_1$ and $w_2$ are descendants of $v_1$ and $v_2$ respectively.

- $\text{LCEPP}(v_1, w_1, v_2, w_2)$: (path-path LCE) return the longest common prefix of the paths $v_1 \rightarrow w_1$ and $v_2 \rightarrow w_2$.
- $\text{LCEPT}(v_1, v_2)$: (path-tree LCE) return maximal path-path LCE of the path $v_1 \rightarrow w_1$ and any path from $v_2$ to a descendant leaf.
- $\text{LCETT}(v_1, v_2)$: (tree-tree LCE) return a maximal path-path LCE of any pair of paths from $v_1$ and $v_2$ to descendant leaves.

We present the first non-trivial bounds for supporting these queries. For $\text{LCEPP}$ queries, we present a linear-space solution with $O(\log^* n)$ query time. For $\text{LCEPT}$ queries, we present a linear-space solution with $O(\log \log n)$ query time, and complement this with a lower bound showing that any path-tree LCE structure of size $O(n \polylog(n))$ must necessarily use $\Omega(\log \log n)$ time to answer queries. For $\text{LCETT}$ queries, we present a time-space trade-off, that given any parameter
τ, 1 ≤ τ ≤ n, leads to an O(nτ) space and O(n/τ) query-time solution. This is complemented with a reduction to the set intersection problem implying that a fast linear space solution is not likely to exist.

**General information**

State: Published
Organisations: Department of Applied Mathematics and Computer Science, Algorithms and Logic, University of Warsaw, University of Haifa
Pages: 52-64
Publication date: 2015

**Host publication information**

Title of host publication: Proceedings of the 26th Annual Symposium on Combinatorial Pattern Matching, CPM 2015
Publisher: Springer
Editors: Cicalese, F., Porat, E., Vaccaro, U.
ISBN (Print): 978-3-319-19928-3
ISBN (Electronic): 978-3-319-19929-0
(Lecture Notes in Computer Science, Vol. 9133).
Electronic versions:
paper_45.pdf
DOIs:
10.1007/978-3-319-19929-0_5
Source: FindIt
Source-ID: 275282304
Research output: Research - peer-review > Article in proceedings – Annual report year: 2015

**Minimum Makespan Multi-Vehicle Dial-a-Ride**

Dial-a-Ride problems consist of a set V of n vertices in a metric space (denoting travel time between vertices) and a set of m objects represented as source-destination pairs {(s(i), t(i))}i=1(m), where each object requires to be moved from its source to destination vertex. In the multi-vehicle Dial-a-Ride problem, there are q vehicles, each having capacity k and where each vehicle j has its own depot-vertex r(j) epsilon V. A feasible schedule consists of a capacitated route for each vehicle (where vehicle j originates and ends at its depot r(j)) that together move all objects from their sources to destinations. The objective is to find a feasible schedule that minimizes the maximum completion time (i.e., makespan) of vehicles, where the completion time of vehicle j is the time when it returns to its depot r(j) at the end of its route. We study the preemptive version of multi-vehicle Dial-a-Ride, in which an object may be left at intermediate vertices and transported by more than one vehicle, while being moved from source to destination. Our main results are an O(log(3) n)-approximation algorithm for preemptive multi-vehicle Dial-a-Ride, and an improved O(log t)-approximation for its special case when there is no capacity constraint (here t
This dissertation studies problems in the general theme of combinatorial pattern matching. More specifically, we study the following topics:

Longest Common Extensions. We revisit the longest common extension (LCE) problem, that is, preprocess a string $T$ into a compact data structure that supports fast LCE queries. An LCE query takes a pair $(i, j)$ of indices in $T$ and returns the length of the longest common prefix of the suffixes of $T$ starting at positions $i$ and $j$. Such queries are also commonly known as longest common prefix (LCP) queries. We study the time-space trade-offs for the problem, that is, the space used for the data structure vs. the worst-case time for answering an LCE query. Let $n$ be the length of $T$. Given a parameter $\tau$, $1 \leq \tau \leq n$, we show how to achieve either $O(n/\sqrt{\tau})$ space and $O(\tau)$ query time, or $O(n/\tau)$ space and $O(\tau \log |LCE(i, j)/\tau|)$ query time, where $|LCE(i, j)|$ denotes the length of the LCE returned by the query. These bounds provide the first smooth trade-offs for the LCE problem and almost match the previously known bounds at the extremes when $\tau = 1$ or $\tau = n$. We apply the result to obtain improved bounds for several applications where the LCE problem is the computational bottleneck, including approximate string matching and computing palindromes. We also present an efficient technique to reduce LCE queries on two strings to one string. Finally, we give a lower bound on the time-space product for LCE data structures in the non-uniform cell probe model showing that our second trade-off is nearly optimal.
Fingerprints in Compressed Strings. The Karp-Rabin fingerprint of a string is a type of hash value that due to its strong properties has been used in many string algorithms. We show how to construct a data structure for a string $S$ of size $N$ compressed by a context-free grammar of size $n$ that supports fingerprint queries. That is, given indices $i$ and $j$, the answer to a query is the fingerprint of the substring $S[i..j]$. We present the first $O(n)$ space data structures that answer fingerprint queries without decompressing any characters. For Straight Line Programs (SLP) we get $O(\log N)$ query time, and for Linear SLPs (an SLP derivative that captures LZ78 compression and its variations) we get $O(\log \log N)$ query time. Hence, our data structures have the same time and space complexity as for random access in SLPs. We utilize the fingerprint data structures to solve the longest common extension problem in query time $O(\log N \log e)$ and $O(\log e \log \log e + \log \log N)$ for SLPs and Linear SLPs, respectively. Here, $e = |\text{LCE}(i, j)|$ denotes the length of the LCE.

Sparse Text Indexing. We present efficient algorithms for constructing sparse suffix trees, sparse suffix arrays and sparse positions heaps for $b$ arbitrary positions of a text $T$ of length $n$ while using only $O(b)$ words of space during the construction. Our main contribution is to show that the sparse suffix tree (and array) can be constructed in $O(n \log^2 b)$ time. To achieve this we develop a technique, that allows to efficiently answer $b$ longest common prefix queries on suffixes of $T$, using only $O(b)$ space. Our first solution is Monte-Carlo and outputs the correct tree with high probability. We then give a Las-Vegas algorithm which also uses $O(b)$ space and runs in the same time bounds with high probability when $b = O(\sqrt{n})$. Furthermore, additional tradeoffs between the space usage and the construction time for the Monte-Carlo algorithm are given. Finally, we show that at the expense of slower pattern queries, it is possible to construct sparse position heaps in $O(n + b \log b)$ time and $O(b)$ space.

The Longest Common Substring Problem. Given $m$ documents of total length $n$, we consider the problem of finding a longest string common to at least $d \geq 2$ of the documents. This problem is known as the longest common substring (LCS) problem and has a classic $O(n)$ space and $O(n)$ time solution (Weiner [FOCS'73], Hui [CPM'92]). However, the use of linear space is impractical in many applications. We show several time-space trade-offs for this problem. Our main result is that for any trade-off parameter $1 \leq \tau \leq n$, the LCS problem can be solved in $O(\tau)$ space and $O(n^2/\tau)$ time, thus providing the first smooth deterministic time-space trade-off from constant to linear space. The result uses a new and very simple algorithm, which computes a $\tau$-additive approximation to the LCS in $O(n^2/\tau)$ time and $O(\tau)$ space. We also show a time-space trade-off lower bound for deterministic branching programs, which implies that any deterministic RAM algorithm solving the LCS problem on documents from a sufficiently large alphabet in $O(\tau)$ space must use $\Omega(n^2/\tau \log(n/\tau \log n))$ time.

Structural Properties of Suffix Trees. We study structural and combinatorial properties of suffix trees. Given an unlabeled tree $T$ on $n$ nodes and suffix links of its internal nodes, we ask the question “Is $T$ a suffix tree?”, i.e., is there a string $S$ whose suffix tree has the same topological structure as $T$? We place no restrictions on $S$, in particular we do not require that $S$ ends with a unique symbol. This corresponds to considering the more general definition of implicit or extended suffix trees. Such general suffix trees have many applications and are for example needed to allow efficient updates when suffix trees are built online. We prove that $T$ is a suffix tree if and only if it is realized by a string $S$ of length $n - 1$, and we give a linear-time algorithm for inferring $S$ when the first letter on each edge is known.

General information
State: Published
Organisations: Department of Applied Mathematics and Computer Science, Algorithms and Logic
Contributors: Vildhøj, H. W., Gørtz, I. L., Bille, P.
Number of pages: 147
Publication date: 2015

Publication information
Place of publication: Kgs. Lyngby
Publisher: Technical University of Denmark (DTU)
Original language: English
(DTU Compute PHD-2014; No. 348).
Electronic versions:
phd348_Vildhoj_HW.pdf
Research output: Research › Ph.D. thesis – Annual report year: 2015

Tree compression with top trees
We introduce a new compression scheme for labeled trees based on top trees. Our compression scheme is the first to simultaneously take advantage of internal repeats in the tree (as opposed to the classical DAG compression that only exploits rooted subtree repeats) while also supporting fast navigational queries directly on the compressed representation. We show that the new compression scheme achieves close to optimal worst-case compression, can compress exponentially better than DAG compression, is never much worse than DAG compression, and supports navigational queries in logarithmic time.

General information
State: Published
Compact q-gram profiling of compressed strings

We consider the problem of computing the q-gram profile of a string T of size N compressed by a context-free grammar with n production rules. We present an algorithm that runs in O(N−α) expected time and uses O(n+q+kT,q) space, where N−α≤qnN−α≤qn is the exact number of characters decompressed by the algorithm and kT,q≤N−α is the number of distinct q-grams in T. This simultaneously matches the current best known time bound and improves the best known space bound. Our space bound is asymptotically optimal in the sense that any algorithm storing the grammar and the q-gram profile must use Ω(n+q+kT,q) space. To achieve this we introduce the q-gram graph that space-efficiently captures the structure of a string with respect to its q-grams, and show how to construct it from a grammar.
BFI (2019): BFI-level 2
Web of Science (2019): Indexed yes
BFI (2018): BFI-level 2
Web of Science (2018): Indexed yes
BFI (2017): BFI-level 2
Scopus rating (2017): CiteScore 1.08 SJR 0.488 SNIP 0.996
Web of Science (2017): Impact factor 0.772
Web of Science (2017): Indexed yes
BFI (2016): BFI-level 2
Scopus rating (2016): CiteScore 0.97 SJR 0.547 SNIP 0.996
Web of Science (2016): Impact factor 0.698
Web of Science (2016): Indexed yes
BFI (2015): BFI-level 2
Scopus rating (2015): CiteScore 1 SJR 0.592 SNIP 1.123
Web of Science (2015): Impact factor 0.643
Web of Science (2015): Indexed yes
BFI (2014): BFI-level 2
Scopus rating (2014): CiteScore 1.08 SJR 0.669 SNIP 1.148
Web of Science (2014): Impact factor 0.657
Web of Science (2014): Indexed yes
BFI (2013): BFI-level 2
Scopus rating (2013): CiteScore 1.17 SJR 0.717 SNIP 1.297
Web of Science (2013): Impact factor 0.516
ISI indexed (2013): ISI indexed yes
BFI (2012): BFI-level 2
Scopus rating (2012): CiteScore 1.16 SJR 0.78 SNIP 1.21
Web of Science (2012): Impact factor 0.489
ISI indexed (2012): ISI indexed yes
Web of Science (2012): Indexed yes
BFI (2011): BFI-level 2
Scopus rating (2011): CiteScore 1.17 SJR 0.747 SNIP 1.246
Web of Science (2011): Impact factor 0.665
ISI indexed (2011): ISI indexed yes
Web of Science (2011): Indexed yes
BFI (2010): BFI-level 2
Scopus rating (2010): SJR 0.839 SNIP 1.278
Web of Science (2010): Impact factor 0.838
Web of Science (2010): Indexed yes
BFI (2009): BFI-level 2
Scopus rating (2009): SJR 0.876 SNIP 1.434
BFI (2008): BFI-level 2
Scopus rating (2008): SJR 1.098 SNIP 1.587
Web of Science (2008): Indexed yes
Scopus rating (2007): SJR 0.931 SNIP 1.576
Scopus rating (2006): SJR 0.834 SNIP 1.457
Web of Science (2006): Indexed yes
Scopus rating (2005): SJR 0.747 SNIP 1.438
Web of Science (2005): Indexed yes
Scopus rating (2004): SJR 0.717 SNIP 1.324
Web of Science (2004): Indexed yes
Scopus rating (2003): SJR 0.848 SNIP 1.494
Scopus rating (2002): SJR 0.729 SNIP 1.32
Web of Science (2002): Indexed yes
Compressed Subsequence Matching and Packed Tree Coloring

We show how to compactly index video data to support fast motion detection queries. A query specifies a time interval $T$, an area $A$ in the video, and two thresholds $v$ and $p$. The answer to a query is a list of timestamps in $T$ where at least $p\%$ of $A$ has changed by at least $v$ values. Our results show that by building a small index, we can support queries with a speedup of two to three orders of magnitude compared to motion detection without an index. For high resolution video, the index size is about 20% of the compressed video size.
String Indexing for Patterns with Wildcards

We consider the problem of indexing a string $t$ of length $n$ to report the occurrences of a query pattern $p$ containing $m$ characters and $j$ wildcards. Let $\text{occ}$ be the number of occurrences of $p$ in $t$, and $\sigma$ the size of the alphabet. We obtain the following results. A linear space index with query time $O(m+\sigma j \log \log n + \text{occ})$. This significantly improves the previously best known linear space index by Lam et al. (in Proc. 18th ISAAC, pp. 846-857, [2007]), which requires query time $\Theta(jn)$ in the worst case. An index with query time $O(m+j+\text{occ})$ using space $\text{(Mathematical expression)}$, where $k$ is the maximum number of wildcards allowed in the pattern. This is the first non-trivial bound with this query time. A time-space trade-off, generalizing the index by Cole et al. (in Proc. 36th STOC, pp. 91-100, [2004]). We also show that these indexes can be generalized to allow variable length gaps in the pattern. Our results are obtained using a novel combination of well-known and new techniques, which could be of independent interest.

General information
State: Published
Organisations: Department of Applied Mathematics and Computer Science, Algorithms and Logic
Contributors: Bille, P., Gørtz, I. L., Vildhøj, H. W., Vind, S. J.
Number of pages: 20
Pages: 41-60
Publication date: 2014
Peer-reviewed: Yes

Publication information
Journal: Theory of Computing Systems
Volume: 55
Issue number: 1
ISSN (Print): 1432-4350
Ratings:
BFI (2019): BFI-level 2
Web of Science (2019): Indexed yes
BFI (2018): BFI-level 2
Web of Science (2018): Indexed yes
BFI (2017): BFI-level 2
Scopus rating (2017): CiteScore 0.82 SJR 0.372 SNIP 0.979
Web of Science (2017): Impact factor 0.458
Web of Science (2017): Indexed yes
BFI (2016): BFI-level 2
Scopus rating (2016): CiteScore 0.78 SJR 0.512 SNIP 0.957
Web of Science (2016): Impact factor 0.645
BFI (2015): BFI-level 2
Scopus rating (2015): CiteScore 0.79 SJR 0.601 SNIP 1.117
Web of Science (2015): Impact factor 0.719
BFI (2014): BFI-level 2
Scopus rating (2014): CiteScore 0.96 SJR 0.664 SNIP 1.077
Web of Science (2014): Impact factor 0.533
Web of Science (2014): Indexed yes
BFI (2013): BFI-level 2
Scopus rating (2013): CiteScore 0.93 SJR 0.686 SNIP 1.065
Web of Science (2013): Impact factor 0.452
ISI indexed (2013): ISI indexed yes
BFI (2012): BFI-level 2
Scopus rating (2012): CiteScore 0.83 SJR 0.756 SNIP 0.896
Web of Science (2012): Impact factor 0.477
ISI indexed (2012): ISI indexed yes
Web of Science (2012): Indexed yes
BFI (2011): BFI-level 2
Scopus rating (2011): CiteScore 0.65 SJR 0.525 SNIP 0.939
Web of Science (2011): Impact factor 0.442
ISI indexed (2011): ISI indexed yes
Substring Range Reporting

We revisit various string indexing problems with range reporting features, namely, position-restricted substring searching, indexing substrings with gaps, and indexing substrings with intervals. We obtain the following main results.

We give efficient reductions for each of the above problems to a new problem, which we call substring range reporting. Hence, we unify the previous work by showing that we may restrict our attention to a single problem rather than studying each of the above problems individually.

We show how to solve substring range reporting with optimal query time and little space. Combined with our reductions this leads to significantly improved time-space trade-offs for the above problems. In particular, for each problem we obtain the first solutions with optimal time query and $O(n \log \log n)$ space, where $n$ is the length of the indexed string.

We show that our techniques for substring range reporting generalize to substring range counting and substring range emptiness variants. We also obtain non-trivial time-space trade-offs for these problems.

Our bounds for substring range reporting are based on a novel combination of suffix trees and range reporting data structures. The reductions are simple and general and may apply to other combinations of string indexing with range reporting.

General information

State: Published
Organisations: Department of Applied Mathematics and Computer Science, Algorithms and Logic
Contributors: Bille, P., Gørtz, I. L.
Pages: 384-396
Publication date: 2014
Peer-reviewed: Yes

Publication information

Journal: Algorithmica
Volume: 69
Issue number: 2
ISSN (Print): 0178-4617
Ratings:
BFI (2019): BFI-level 2
Web of Science (2019): Indexed yes
BFI (2018): BFI-level 2
Time–space trade-offs for longest common extensions

We revisit the longest common extension (LCE) problem, that is, preprocess a string $T$ into a compact data structure that supports fast LCE queries. An LCE query takes a pair $(i,j)$ of indices in $T$ and returns the length of the longest common prefix of the suffixes of $T$ starting at positions $i$ and $j$. We study the time–space trade-offs for the problem, that is, the space used for the data structure vs. the worst-case time for answering an LCE query. Let $n$ be the length of $T$. Given a parameter $\tau$, $1 \leq \tau \leq n$, we show how to achieve either $O(n/\tau)$ space and $O(\tau)$ query time, or $O(n/\tau)$ space and $O(\log(|LCE(i,j)|/\tau))$ query time, where $|LCE(i,j)|$ denotes the length of the LCE returned by the query. These bounds provide the first smooth trade-offs for the LCE problem and almost match the previously known bounds at the extremes when $\tau=1$ or $\tau=n$. We apply the result to obtain improved bounds for several applications where the LCE problem is the computational bottleneck, including approximate string matching and computing palindromes. We also present an efficient technique to reduce LCE queries on two strings to one string. Finally, we give a lower bound on the time–space product for LCE data structures in the non-uniform cell probe model showing that our second trade-off is nearly optimal.
Union-Find with Constant Time Deletions

A union-find data structure maintains a collection of disjoint sets under the operations makeset, union, and find. Kaplan, Shafrir, and Tarjan [SODA 2002] designed data structures for an extension of the union-find problem in which items of the sets maintained may be deleted. The cost of a delete operation in their implementations is essentially the same as the cost of a find operation; namely, \(O(\log n)\) worst-case and \(O(\alpha_{\frac{M}{N}(n)})\) amortized, where \(n\) is the number of items in the set returned by the find operation, \(N\) is the total number of makeset operations performed, \(M\) is the total number of find operations performed, and \(\alpha_{\frac{M}{N}}(n)\) is a functional inverse of Ackermann’s function. They left open the question whether delete operations can be implemented more efficiently than find operations, for example, in \(o(\log n)\) worst-case time. We resolve this open problem by presenting a relatively simple modification of the classical union-find data structure that supports delete, as well as makeset and union operations, in constant worst-case time, while still supporting find operations in \(O(\log n)\) worst-case time and \(O(\alpha_{\frac{M}{N}}(n))\) amortized time. Our analysis supplies, in particular, a very concise potential-based amortized analysis of the standard union-find data structure that yields an \(O(\alpha_{\frac{M}{N}}(n))\) amortized bound on the cost of find operations. All previous potential-based analyses yielded the weaker amortized bound of \(O(\alpha_{\frac{M}{N}}(N))\). Furthermore, our tighter analysis extends to one-path variants of the path compression technique such as path splitting.

General information

State: Published
Organisations: Department of Applied Mathematics and Computer Science, Algorithms and Logic, University of Copenhagen, Octoshape ApS, Tel Aviv University
Contributors: Alstrup, S., Thorup, M., Gørtz, I. L., Rauhe, T., Zwick, U.
Number of pages: 28
Publication date: 2014
Peer-reviewed: Yes

Publication information

Journal: ACM Transactions on Algorithms
Volume: 11
Issue number: 1
ISSN (Print): 1549-6325
Ratings:
  BFI (2019): BFI-level 2
  Web of Science (2019): Indexed yes
  BFI (2018): BFI-level 2
  Web of Science (2018): Indexed yes
  BFI (2017): BFI-level 2
  Scopus rating (2017): CiteScore 2.11 SJR 0.804 SNIP 1.701
  Web of Science (2017): Impact factor 0.652
  Web of Science (2017): Indexed yes
  BFI (2016): BFI-level 2
  Scopus rating (2016): CiteScore 2.1 SJR 1.32 SNIP 1.951
  Web of Science (2016): Impact factor 1.458
  Web of Science (2016): Indexed yes
  BFI (2015): BFI-level 2
  Scopus rating (2015): CiteScore 1.94 SJR 1.443 SNIP 1.77
  Web of Science (2015): Impact factor 0.776
  Web of Science (2015): Indexed yes
  BFI (2014): BFI-level 2
  Scopus rating (2014): CiteScore 1.61 SJR 1.325 SNIP 1.471
  Web of Science (2014): Impact factor 0.895
  Web of Science (2014): Indexed yes
  BFI (2013): BFI-level 2
Compact q-gram Profiling of Compressed Strings

We consider the problem of computing the q-gram profile of a string T of size N compressed by a context-free grammar with n production rules. We present an algorithm that runs in $O(N-\alpha)$ expected time and uses $O(n+k_T,q)$ space, where $N-\alpha \leq qn$ is the exact number of characters decompressed by the algorithm and $k_T,q \leq N-\alpha$ is the number of distinct q-grams in T. This simultaneously matches the current best known time bound and improves the best known space bound. Our space bound is asymptotically optimal in the sense that any algorithm storing the grammar and the q-gram profile must use $\Omega(n+k_T,q)$ space. To achieve this we introduce the q-gram graph that space-efficiently captures the structure of a string with respect to its q-grams, and show how to construct it from a grammar.

General information
State: Published
Organisations: Department of Applied Mathematics and Computer Science, Algorithms and Logic
Contributors: Bille, P., Cording, P. H., Gørtz, I. L.
Pages: 62-73
Publication date: 2013
Fingerprints in Compressed Strings

The Karp-Rabin fingerprint of a string is a type of hash value that due to its strong properties has been used in many string algorithms. In this paper we show how to construct a data structure for a string S of size N compressed by a context-free grammar of size n that answers fingerprint queries. That is, given indices i and j, the answer to a query is the fingerprint of the substring S[i..j]. We present the first O(n) space data structures that answer fingerprint queries without decompressing any characters. For Straight Line Programs (SLP) we get O(logN) query time, and for Linear SLPs (an SLP derivative that captures LZ78 compression and its variations) we get O(loglogN) query time. Hence, our data structures has the same time and space complexity as for random access in SLPs. We utilize the fingerprint data structures to solve the longest common extension problem in query time O(logNlogℓ) and O(logloglogℓ + loglogN) for SLPs and Linear SLPs, respectively. Here, ℓ denotes the length of the LCE.

General information
State: Published
Organisations: Department of Applied Mathematics and Computer Science, Algorithms and Logic, University of Warwick
Contributors: Bille, P., Cording, P. H., Gørtz, I. L., Sach, B., Vildhøj, H. W., Vind, S. J.
Pages: 146-157
Publication date: 2013

Sparse suffix tree construction in small space

We consider the problem of constructing a sparse suffix tree (or suffix array) for b suffixes of a given text T of length n, using only O(b) words of space during construction. Attempts at breaking the naive bound of Ω(nb) time for this problem can be traced back to the origins of string indexing in 1968. First results were only obtained in 1996, but only for the case where the suffixes were evenly spaced in T. In this paper there is no constraint on the locations of the suffixes. We show that the sparse suffix tree can be constructed in O(nlog2 b) time. To achieve this we develop a technique, which may be of independent interest, that allows to efficiently answer b longest common prefix queries on suffixes of T, using only O(b) space. We expect that this technique will prove useful in many other applications in which space usage is a concern. Our first solution is Monte-Carlo and outputs the correct tree with high probability. We then give a Las-Vegas algorithm which also uses O(b) space and runs in the same time bounds with high probability when b = O(\sqrt{n}). Furthermore, additional tradeoffs between the space usage and the construction time for the Monte-Carlo algorithm are given.

General information
State: Published
Organisations: Department of Applied Mathematics and Computer Science, Algorithms and Logic, Karlsruhe Institute for Technology, Weizmann Institute of Science, University of Warwick
Contributors: Bille, P., Fischer, J., Gørtz, I. L., Kopelowitz, T., Sach, B., Vildhøj, H. W.
Pages: 148-159
Publication date: 2013

Host publication information
Title of host publication: Automata, Languages, and Programming: 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I
Publisher: Springer
ISBN (Print): 978-3-642-39205-4
ISBN (Electronic): 978-3-642-39206-1
(Lecture Notes in Computer Science, Vol. 7965).
Keywords: Algorithms, Automata theory, Query processing, Forestry
Tree compression with top trees

We introduce a new compression scheme for labeled trees based on top trees [3]. Our compression scheme is the first to simultaneously take advantage of internal repeats in the tree (as opposed to the classical DAG compression that only exploits rooted subtree repeats) while also supporting fast navigational queries directly on the compressed representation. We show that the new compression scheme achieves close to optimal worst-case compression, can compress exponentially better than DAG compression, is never much worse than DAG compression, and supports navigational queries in logarithmic time.

General information

State: Published
Organisations: Department of Applied Mathematics and Computer Science, Algorithms and Logic, University of Haifa
Pages: 160-171
Publication date: 2013

Host publication information

Title of host publication: Automata, Languages, and Programming: 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I
Publisher: Springer
ISBN (Print): 978-3-642-39205-4
ISBN (Electronic): 978-3-642-39206-1
(Lecture Notes in Computer Science, Vol. 7965).
DOIs:
10.1007/978-3-642-39206-1_13
Source: dtu
Source-ID: n::oai:DTIC-ART:compendex/389989334::35477
Research output: Research - peer-review – Article in proceedings – Annual report year: 2013

Fast Arc-Annotated Subsequence Matching in Linear Space

An arc-annotated string is a string of characters, called bases, augmented with a set of pairs, called arcs, each connecting two bases. Given arc-annotated strings P and Q the arc-preserving subsequence problem is to determine if P can be obtained from Q by deleting bases from Q. Whenever a base is deleted any arc with an endpoint in that base is also deleted. Arc-annotated strings where the arcs are "nested" are a natural model of RNA molecules that captures both the primary and secondary structure of these. The arc-preserving subsequence problem for nested arc-annotated strings is basic primitive for investigating the function of RNA molecules. Gramm et al. (ACM Trans. Algorithms 2(1): 44-65, 2006) gave an algorithm for this problem using $O(nm)$ time and space, where $m$ and $n$ are the lengths of P and Q, respectively. In this paper we present a new algorithm using $O(nm)$ time and $O(n+m)$ space, thereby matching the previous time bound while significantly reducing the space from a quadratic term to linear. This is essential to process large RNA molecules where the space is likely to be a bottleneck. To obtain our result we introduce several novel ideas which may be of independent interest for related problems on arc-annotated strings. © 2010 Springer Science+Business Media, LLC.

General information

State: Published
Organisations: Algorithms and Logic, Department of Informatics and Mathematical Modeling
Contributors: Bille, P., Gørtz, I. L.
Pages: 209-223
Publication date: 2012
Peer-reviewed: Yes

Publication information

Journal: Algorithmica
Volume: 62
Issue number: 1-2
ISSN (Print): 0178-4617
Ratings:
BFI (2019): BFI-level 2
Web of Science (2019): Indexed yes
Subsequence matching, Arc-annotated strings, Algorithms

BFI (2018): BFI-level 2  
Web of Science (2018): Indexed yes  

BFI (2017): BFI-level 2  
Scopus rating (2017): CiteScore 1.27 SJR 0.56 SNIP 1.354  
Web of Science (2017): Impact factor 0.667  
Web of Science (2017): Indexed yes  

BFI (2016): BFI-level 2  
Scopus rating (2016): CiteScore 1.11 SJR 0.648 SNIP 1.184  
Web of Science (2016): Impact factor 0.735  
Web of Science (2016): Indexed yes  

BFI (2015): BFI-level 2  
Scopus rating (2015): CiteScore 1.15 SJR 0.739 SNIP 1.16  
Web of Science (2015): Impact factor 0.795  
Web of Science (2015): Indexed yes  

BFI (2014): BFI-level 2  
Scopus rating (2014): CiteScore 1.2 SJR 0.854 SNIP 1.197  
Web of Science (2014): Impact factor 0.791  
Web of Science (2014): Indexed yes  

BFI (2013): BFI-level 2  
Scopus rating (2013): CiteScore 1.26 SJR 0.952 SNIP 1.395  
Web of Science (2013): Impact factor 0.567  
ISI indexed (2013): ISI indexed yes  

BFI (2012): BFI-level 2  
Scopus rating (2012): CiteScore 0.99 SJR 0.836 SNIP 1.204  
Web of Science (2012): Impact factor 0.488  
ISI indexed (2012): ISI indexed yes  
Web of Science (2012): Indexed yes  

BFI (2011): BFI-level 2  
Scopus rating (2011): CiteScore 0.91 SJR 0.848 SNIP 1.184  
Web of Science (2011): Impact factor 0.604  
ISI indexed (2011): ISI indexed yes  
Web of Science (2011): Indexed yes  

BFI (2010): BFI-level 2  
Scopus rating (2010): SJR 0.904 SNIP 1.317  
Web of Science (2010): Impact factor 1.239  
BFI (2009): BFI-level 2  
Scopus rating (2009): SJR 0.95 SNIP 1.351  

BFI (2008): BFI-level 1  
Scopus rating (2008): SJR 1.129 SNIP 1.218  
Scopus rating (2007): SJR 0.99 SNIP 1.43  
Scopus rating (2006): SJR 1.13 SNIP 1.757  
Scopus rating (2005): SJR 0.781 SNIP 1.387  
Scopus rating (2004): SJR 1.001 SNIP 1.472  
Scopus rating (2003): SJR 0.888 SNIP 1.279  
Scopus rating (2002): SJR 0.876 SNIP 1.428  
Scopus rating (2001): SJR 1.15 SNIP 1.093  
Scopus rating (2000): SJR 0.408 SNIP 1.055  
Scopus rating (1999): SJR 0.401 SNIP 1.253  

Original language: English  
Keywords: Subsequence matching, Arc-annotated strings, Algorithms  
DOIs:  
10.1007/s00453-010-9451-8  
Source: orbit  
Source-ID: 317303
Longest Common Extensions via Fingerprinting

The longest common extension (LCE) problem is to preprocess a string in order to allow for a large number of LCE queries, such that the queries are efficient. The LCE value, \( \text{LCE}_s(i,j) \), is the length of the longest common prefix of the pair of suffixes starting at index \( i \) and \( j \) in the string \( s \). The LCE problem can be solved in linear space with constant query time and a preprocessing of sorting complexity. There are two known approaches achieving these bounds, which use nearest common ancestors and range minimum queries, respectively. However, in practice a much simpler approach with linear query time, no extra space and no preprocessing achieves significantly better average case performance. We show a new algorithm, Fingerprint \( k \), which for a parameter \( k \), \( 1 \leq k \leq \log n \), on a string of length \( n \) and alphabet size \( \sigma \), gives \( O(k n^{1/k}) \) query time using \( O(k n) \) space and \( O(k n + \text{sort}(n,\sigma)) \) preprocessing time, where \( \text{sort}(n,\sigma) \) is the time it takes to sort \( n \) numbers from \( \sigma \). Though this solution is asymptotically strictly worse than the asymptotically best previously known algorithms, it outperforms them in practice in average case and is almost as fast as the simple linear time algorithm. On worst case input, this new algorithm is significantly faster in practice compared to the simple linear time algorithm. We also look at cache performance of the new algorithm, and we show that for \( k = 2 \), cache optimization can improve practical query time.

General information
State: Published
Organisations: Department of Informatics and Mathematical Modeling, Algorithms and Logic, Computer Science and Engineering, Technical University of Denmark
Contributors: Bille, P., Gørtz, I. L., Kristensen, J.
Pages: 119-130
Publication date: 2012

Host publication information
Title of host publication: Language and Automata Theory and Applications : 6th International Conference, LATA 2012 A Coruña, Spain, March 5-9, 2012 Proceedings
Publisher: Springer
ISBN (Print): 978-3-642-28331-4
ISBN (Electronic): 978-3-642-28332-1
(Lecture Notes in Computer Science, Vol. 7183).
DOIs: 10.1007/978-3-642-28332-1_11
URLs: http://grammars.grfmc.com/lata2012/
Source: dtu
Source-ID: n:oat:DTIC-ART:inspec/363418467::15475
Research output: Research - peer-review \ Article in proceedings – Annual report year: 2012

Stochastic vehicle routing with recourse
We study the classic Vehicle Routing Problem in the setting of stochastic optimization with recourse. StochVRP is a two-stage problem, where demand is satisfied using two routes: fixed and recourse. The fixed route is computed using only a demand distribution. Then after observing the demand instantiations, a recourse route is computed - but costs here become more expensive by a factor \( \lambda \). We present an \( O(\log n \cdot \log(n\lambda)) \)-approximation algorithm for this stochastic routing problem, under arbitrary distributions. The main idea in this result is relating StochVRP to a special case of submodular orienteering, called knapsack rank-function orienteering. We also give a better approximation ratio for knapsack rank-function orienteering than what follows from prior work. Finally, we provide a Unique Games Conjecture based \( \omega(1) \) hardness of approximation for StochVRP, even on star-like metrics on which our algorithm achieves a logarithmic approximation.

General information
State: Published
Organisations: Department of Informatics and Mathematical Modeling, Algorithms and Logic, Computer Science and Engineering, IBM Research
Contributors: Gørtz, I. L., Nagarajan, V., Saket, R.
Pages: 411-423
Publication date: 2012

Host publication information
Title of host publication: Automata, Languages, and Programming : 39th International Colloquium, ICALP 2012
Publisher: Springer
Editors: Czumaj, A., Mehlhorn , K., Pitts, A., Wattenhofer, R.
ISBN (Print): 978-3-642-31593-0
String Indexing for Patterns With Wildcards
We consider the problem of indexing a string $t$ of length $n$ to report the occurrences of a query pattern $p$ containing $m$ characters and $j$ wildcards. Let $occ$ be the number of occurrences of $p$ in $t$, and $\sigma$ the size of the alphabet. We obtain the following results. - A linear space index with query time $O(m + \sigma j \log \log n + occ)$. This significantly improves the previously best known linear space index by Lam et al. [ISAAC 2007], which requires query time $\Theta(jn)$ in the worst case. - An index with query time $O(m + j + occ)$ using space $O(\sigma k^2 n \log k \log n)$, where $k$ is the maximum number of wildcards allowed in the pattern. This is the first non-trivial bound with this query time. - A time-space trade-off, generalizing the index by Cole et al. [STOC 2004]. Our results are obtained using a novel combination of well-known and new techniques, which could be of independent interest.

General information
State: Published
Organisations: Department of Informatics and Mathematical Modeling, Algorithms and Logic, Computer Science and Engineering
Contributors: Bille, P., Gørtz, I. L., Vildhøj, H. W., Vind, S. J.
Pages: 283-294
Publication date: 2012

Host publication information
Title of host publication: Algorithm Theory – SWAT 2012 : 13th Scandinavian Symposium and Workshops, Helsinki, Finland, July 4-6, 2012. Proceedings
Publisher: Springer
Editors: Fornin, F. V., Kaski, P.
ISBN (Print): 978-3-642-31154-3
ISBN (Electronic): 978-3-642-31155-0
(Lecture Notes in Computer Science, Vol. 7357).
Electronic versions:
1110.5236v2.pdf
DOIs:
10.1007/978-3-642-31155-0_25
Source: dtu
Source-ID: n::oai:DTIC-ART:inspec/367285895::18311
Research output: Research - peer-review › Article in proceedings – Annual report year: 2012

String matching with variable length gaps
We consider string matching with variable length gaps. Given a string $T$ and a pattern $P$ consisting of strings separated by variable length gaps (arbitrary strings of length in a specified range), the problem is to find all ending positions of substrings in $T$ that match $P$. This problem is a basic primitive in computational biology applications. Let $m$ and $n$ be the lengths of $P$ and $T$, respectively, and let $k$ be the number of strings in $P$. We present a new algorithm achieving time $O(n \log k + m + \alpha)$ and space $O(m + A)$, where $A$ is the sum of the lower bounds of the lengths of the gaps in $P$ and $\alpha$ is the total number of occurrences of the strings in $P$ within $T$. Compared to the previous results this bound essentially achieves the best known time and space complexities simultaneously. Consequently, our algorithm obtains the best known bounds for almost all combinations of $m$, $n$, $k$, $A$, and $\alpha$. Our algorithm is surprisingly simple and straightforward to implement. We also present algorithms for finding and encoding the positions of all strings in $P$ for every match of the pattern.

General information
State: Published
Organisations: Department of Informatics and Mathematical Modeling, Algorithms and Logic, Computer Science and Engineering, Department of Applied Mathematics and Computer Science, Cognitive Systems
Pages: 25-34
Publication date: 2012
Peer-reviewed: Yes

Publication information
Journal: Theoretical Computer Science
Volume: 443
Time-Space Trade-offs for Longest Common Extensions

We revisit the longest common extension (LCE) problem, that is, preprocess a string $T$ into a compact data structure that supports fast LCE queries. An LCE query takes a pair $(i, j)$ of indices in $T$ and returns the length of the longest common prefix of the suffixes of $T$ starting at positions $i$ and $j$. We study the time-space trade-offs for the problem, that is, the space used for the data structure vs. the worst-case time for answering an LCE query. Let $n$ be the length of $T$. Given a parameter $\tau$, $1 \leq \tau \leq n$, we show how to achieve either $O(n/\sqrt{\tau})$ space and $O(\tau)$ query time, or $O(n/\tau)$ space and $O(\tau \log(|LCE(i,j)|/\tau))$ query time, where $|LCE(i,j)|$ denotes the length of the LCE returned by the query. These bounds provide the first smooth trade-offs for the LCE problem and almost match the previously known bounds at the extremes when $\tau = 1$ or $\tau = n$. We apply the result to obtain improved bounds for several applications where the LCE problem is the computational bottleneck, including approximate string matching and computing palindromes. Finally, we also present an efficient technique to reduce LCE queries on two strings to one string.
Capacitated Vehicle Routing with Non-Uniform Speeds

The capacitated vehicle routing problem (CVRP) [21] involves distributing (identical) items from a depot to a set of demand locations in the shortest possible time, using a single capacitated vehicle. We study a generalization of this problem to the setting of multiple vehicles having non-uniform speeds (that we call Heterogenous CVRP), and present a constant-factor approximation algorithm. The technical heart of our result lies in achieving a constant approximation to the following TSP variant (called Heterogenous TSP). Given a metric denoting distances between vertices, a depot r containing k vehicles having speeds \( \lambda_i \) \( i = 1 \cdots k \), the goal is to find a tour for each vehicle (starting and ending at r), so that every vertex is covered in some tour and the maximum completion time is minimized. This problem is precisely Heterogenous CVRP when vehicles are uncapacitated. The presence of non-uniform speeds introduces difficulties for employing standard tour-splitting techniques. In order to get a better understanding of this technique in our context, we appeal to ideas from the 2-approximation for minimum makespan scheduling in unrelated parallel machines of Lenstra et al. [19]. This motivates the introduction of a new approximate MST construction called Level-Prim, which is related to Light Approximate Shortest-Path Trees [18]. The last component of our algorithm involves partitioning the Level-Prim tree and matching the resulting parts to vehicles. This decomposition is more subtle than usual since now we need to enforce correlation between the lengths of the parts and their distances to the depot.

General information

State: Published
Organisations: Algorithms and Logic, Department of Informatics and Mathematical Modeling, Carnegie Mellon University, IBM Research
Contributors: Gørtz, I. L., Molinaro, M., Nagarajan, V., Ravi, R.
Pages: 235-247
Publication date: 2011

Host publication information
Title of host publication: Lecture Notes in Computer Science : The 15th Conference on Integer Programming and Combinatorial Optimization
Volume: 6655/2011
Place of publication: Berlin
Publisher: Springer
ISBN (Print): 9783642208065
(Lecture Notes in Computer Science).
Electronic versions:
hkr_Capacitated_Vehicle_Routing_with_Non_uniform_Speeds.pdf
DOIs:
10.1007/978-3-642-20807-2_19

Bibliographical note
This Conference Proceeding is brought to you for free and open access by Research Showcase @ CMU. It has been accepted for inclusion in Tepper School of Business by an authorized administrator of Research Showcase @ CMU. For more information, please contact researchshowcase@andrew.cmu.edu.
Source: orbit
Source-ID: 274706
Research output: Research - peer-review : Article in proceedings – Annual report year: 2011

Locating Depots for Capacitated Vehicle Routing

We study a location-routing problem in the context of capacitated vehicle routing. The input to k-LocVRP is a set of demand locations in a metric space and a fleet of k vehicles each of capacity Q. The objective is to locate k depots, one for each vehicle, and compute routes for the vehicles so that all demands are satisfied and the total cost is minimized. Our main result is a constant-factor approximation algorithm for k-LocVRP. To achieve this result, we reduce k-LocVRP to the following generalization of k median, which might be of independent interest. Given a metric \( (V, d) \), bound k and parameter \( \rho \in \mathbb{R}^+ \), the goal in the k median forest problem is to find \( S \subseteq V \) with \( |S| = k \) minimizing: \( E \in V d(u, S) + \rho \cdot d(MST(V/S)) \), where \( d(u, S) = \min_{w \in S} d(u, w) \) and MST(V/S) is a minimum spanning tree in the graph obtained by contracting S to a single vertex. We give a \((3+\varepsilon)\)-approximation algorithm for k median forest, which leads to a \((12+\varepsilon)\)-approximation algorithm for k-LocVRP, for any constant \( \varepsilon > 0 \). The algorithm for k median forest is t-swap local search, and we prove that it has locality gap \( 3 + 2 \tau \); this generalizes the corresponding result for k median [3]. Finally we consider the k median
forest problem when there is a different (unrelated) cost function $c$ for the MST part, i.e. the objective is $\sum_{u \in V} d(u, S) + c(\text{MST}(V \setminus S))$. We show that the locality gap for this problem is unbounded even under multi-swaps, which contrasts with the $c = d$ case. Nevertheless, we obtain a constant-factor approximation algorithm, using an LP based approach along the lines of [12].

**General information**
State: Published
Organisations: Algorithms and Logic, Department of Informatics and Mathematical Modeling, IBM Research
Contributors: Gørtz, I. L., Nagarajan, V.
Pages: 230-241
Publication date: 2011

**Host publication information**
Volume: 6845
Publisher: Springer
ISBN (Print): 978-3-642-22934-3
ISBN (Electronic): 978-3-642-22935-0
(Lecture Notes in Computer Science).
DOIs: 10.1007/978-3-642-22935-0_20
Source: orbit
Source-ID: 312498
Research output: Research - peer-review › Article in proceedings – Annual report year: 2011

---

**Substring Range Reporting**
We revisit various string indexing problems with range reporting features, namely, position-restricted substring searching, indexing substrings with gaps, and indexing substrings with intervals. We obtain the following main results. – We give efficient reductions for each of the above problems to a new problem, which we call substring range reporting. Hence, we unify the previous work by showing that we may restrict our attention to a single problem rather than studying each of the above problems individually. – We show how to solve substring range reporting with optimal query time and little space. Combined with our reductions this leads to significantly improved time-space trade-offs for the above problems. In particular, for each problem we obtain the first solutions with optimal time query and $O(n \log(1)n)$ space, where $n$ is the length of the indexed string. Our bounds for substring range reporting are based on a novel combination of suffix trees and range reporting data structures. The reductions are simple and general and may apply to other combinations of string indexing with range reporting.

**General information**
State: Published
Organisations: Algorithms and Logic, Department of Informatics and Mathematical Modeling
Contributors: Bille, P., Gørtz, I. L.
Pages: 299-308
Publication date: 2011

**Host publication information**
Title of host publication: Combinatorial Pattern Matching: 22nd Annual Symposium, CPM 2011 Palermo, Italy, June 27-29, 2011 Proceedings
Volume: 6661
Publisher: Springer
ISBN (Print): 978-3-642-21457-8
ISBN (Electronic): 978-3-642-21458-5
(Lecture Notes in Computer Science).
Electronic versions: 1108.3683.pdf
DOIs: 10.1007/978-3-642-21458-5_26
URLs: http://cpm2011.unipa.it/
Source: orbit
Source-ID: 312501
Research output: Research - peer-review › Article in proceedings – Annual report year: 2011
The Tree Inclusion Problem: In Linear Space and Faster

Given two rooted, ordered, and labeled trees P and T the tree inclusion problem is to determine if P can be obtained from T by deleting nodes in T. This problem has recently been recognized as an important query primitive in XML databases. Kilpeläinen and Mannila [1995] presented the first polynomial-time algorithm using quadratic time and space. Since then several improved results have been obtained for special cases when P and T have a small number of leaves or small depth. However, in the worst case these algorithms still use quadratic time and space. Let nS, lS, and dS denote the number of nodes, the number of leaves, and the depth of a tree S ∈ {P, T}. In this article we show that the tree inclusion problem can be solved in space O(nT ) and time: Equation Presented ∑. This improves or matches the best known time complexities while using only linear space instead of quadratic. This is particularly important in practical applications, such as XML databases, where the space is likely to be a bottleneck. © 2011 ACM.

General information
State: Published
Organisations: Algorithms and Logic, Department of Informatics and Mathematical Modeling
Contributors: Bille, P., Gørtz, I. L.
Pages: 38
Publication date: 2011
Peer-reviewed: Yes

Publication information
Journal: A C M Transactions on Algorithms
Volume: 7
Issue number: 3
ISSN (Print): 1549-6325
Ratings:
BFI (2019): BFI-level 2
Web of Science (2019): Indexed yes
BFI (2018): BFI-level 2
Web of Science (2018): Indexed yes
BFI (2017): BFI-level 2
Scopus rating (2017): CiteScore 2.11 SJR 0.804 SNIP 1.701
Web of Science (2017): Impact factor 0.652
Web of Science (2017): Indexed yes
BFI (2016): BFI-level 2
Scopus rating (2016): CiteScore 2.1 SJR 1.32 SNIP 1.951
Web of Science (2016): Impact factor 1.458
Web of Science (2016): Indexed yes
BFI (2015): BFI-level 2
Scopus rating (2015): CiteScore 1.94 SJR 1.443 SNIP 1.77
Web of Science (2015): Impact factor 0.776
Web of Science (2015): Indexed yes
BFI (2014): BFI-level 2
Scopus rating (2014): CiteScore 1.61 SJR 1.325 SNIP 1.471
Web of Science (2014): Impact factor 0.895
Web of Science (2014): Indexed yes
BFI (2013): BFI-level 2
Scopus rating (2013): CiteScore 1.4 SJR 1.148 SNIP 1.334
Web of Science (2013): Impact factor 0.4
ISI indexed (2013): ISI indexed yes
BFI (2012): BFI-level 2
Scopus rating (2012): CiteScore 1.21 SJR 1.072 SNIP 1.296
Web of Science (2012): Impact factor 0.54
ISI indexed (2012): ISI indexed yes
BFI (2011): BFI-level 2
Scopus rating (2011): CiteScore 1.11 SJR 1.015 SNIP 1.164
Web of Science (2011): Impact factor 0.646
Fast Arc-Annotated Subsequence Matching in Linear Space

An arc-annotated string is a string of characters, called bases, augmented with a set of pairs, called arcs, each connecting two bases. Given arc-annotated strings P and Q the arc-preserving subsequence problem is to determine if P can be obtained from Q by deleting bases from Q. Whenever a base is deleted any arc with an endpoint in that base is also deleted. Arc-annotated strings where the arcs are "nested" are a natural model of RNA molecules that captures both the primary and secondary structure of these. The arc-preserving subsequence problem for nested arc-annotated strings is basic primitive for investigating the function of RNA molecules. Gramm et al. [ACM Trans. Algorithms 2006] gave an algorithm for this problem using $O(nm)$ time and space, where $m$ and $n$ are the lengths of P and Q, respectively. In this paper we present: a new algorithm using $O(nm)$ time and $O(n+m)$ space, thereby matching the previous time bound while significantly reducing the space from a quadratic term to linear. This is essential to process large RNA molecules where the space is a likely to be a bottleneck. To obtain our result we introduce several novel ideas which may be of independent interest for related problems on arc-annotated strings.
Improved Approximate String Matching and Regular Expression Matching on Ziv-Lempel Compressed Texts

We study the approximate string matching and regular expression matching problem for the case when the text to be searched is compressed with the Ziv-Lempel adaptive dictionary compression schemes. We present a time-space trade-off that leads to algorithms improving the previously known complexities for both problems. In particular, we significantly improve the space bounds, which in practical applications are likely to be a bottleneck.

General information
State: Published
Organisations: Algorithms and Logic, Department of Informatics and Mathematical Modeling, IT University of Copenhagen, University of Southern Denmark
Contributors: Bille, P., Fagerberg, R., Gørtz, I. L.
Number of pages: 14
Pages: article number 3
Publication date: 2009
Peer-reviewed: Yes

Publication information
Journal: A C M Transactions on Algorithms
Volume: 6
Issue number: 1
ISSN (Print): 1549-6325
Ratings:
BFI (2019): BFI-level 2
Web of Science (2019): Indexed yes
BFI (2018): BFI-level 2
Web of Science (2018): Indexed yes
BFI (2017): BFI-level 2
Scopus rating (2017): CiteScore 2.11 SJR 0.804 SNIP 1.701
Web of Science (2017): Impact factor 0.652
Web of Science (2017): Indexed yes
BFI (2016): BFI-level 2
Scopus rating (2016): CiteScore 2.1 SJR 1.32 SNIP 1.951
Web of Science (2016): Impact factor 1.458
Web of Science (2016): Indexed yes
BFI (2015): BFI-level 2
Scopus rating (2015): CiteScore 1.94 SJR 1.443 SNIP 1.77
Web of Science (2015): Impact factor 0.776
Web of Science (2015): Indexed yes
BFI (2014): BFI-level 2
Scopus rating (2014): CiteScore 1.61 SJR 1.325 SNIP 1.471
Web of Science (2014): Impact factor 0.895
Web of Science (2014): Indexed yes
BFI (2013): BFI-level 2
Scopus rating (2013): CiteScore 1.4 SJR 1.148 SNIP 1.334
Web of Science (2013): Impact factor 0.4
ISI indexed (2013): ISI indexed yes
BFI (2012): BFI-level 2
Scopus rating (2012): CiteScore 1.21 SJR 1.072 SNIP 1.296
Web of Science (2012): Impact factor 0.54
ISI indexed (2012): ISI indexed yes
BFI (2011): BFI-level 2
Scopus rating (2011): CiteScore 1.11 SJR 1.015 SNIP 1.164
Matching Subsequences In Trees
Given two rooted, labeled trees $P$ and $T$ the tree path subsequence problem is to determine which paths in $P$ are subsequences of which paths in $T$. Here a path begins at the root and ends at a leaf. In this paper we propose this problem as a useful query primitive for XML data, and provide new algorithms improving the previously best known time and space bounds.

General information
State: Published
Organisations: Algorithms and Logic, Department of Informatics and Mathematical Modeling, IT University of Copenhagen
Contributors: Bille, P., Gørtz, I. L.
Pages: 306-314
Publication date: 2009
Peer-reviewed: Yes

Publication information
Journal: Journal of Discrete Algorithms
Volume: 7
Issue number: 3
ISSN (Print): 1570-8667
Ratings:
BFI (2019): BFI-level 1
Web of Science (2019): Indexed yes
BFI (2018): BFI-level 1
Web of Science (2018): Indexed yes
BFI (2017): BFI-level 1
Scopus rating (2017): CiteScore 0.97 SJR 0.549 SNIP 1.156
Web of Science (2017): Indexed yes
BFI (2016): BFI-level 1
Scopus rating (2016): CiteScore 0.76 SJR 0.472 SNIP 0.843
BFI (2015): BFI-level 1
Scopus rating (2015): CiteScore 0.81 SJR 0.619 SNIP 0.997
Web of Science (2015): Indexed yes
BFI (2014): BFI-level 1
Scopus rating (2014): CiteScore 0.75 SJR 0.517 SNIP 0.71
BFI (2013): BFI-level 1
Scopus rating (2013): CiteScore 0.97 SJR 0.748 SNIP 1.033
BFI (2012): BFI-level 1
Scopus rating (2012): CiteScore 1.25 SJR 0.748 SNIP 1.388
Minimum Makespan Multi-vehicle Dial-a-Ride

Dial-a-Ride problems consist of a set $V$ of $n$ vertices in a metric space (denoting travel time between vertices) and a set of $m$ objects represented as source-destination pairs $\{(s_i, t_i)\}_{i=1}^m$, where each object requires to be moved from its source to destination vertex. In the multi-vehicle Dial-a-Ride problem, there are $q$ vehicles each having capacity $k$ and where each vehicle $j \in [q]$ has its own depot-vertex $r_j \in V$. A feasible schedule consists of a capacitated route for each vehicle (where vehicle $j$ originates and ends at its depot $r_j$) that together move all objects from their sources to destinations. The objective is to find a feasible schedule that minimizes the maximum completion time (i.e. makespan) of vehicles, where the completion time of vehicle $j$ is the time when it returns to its depot $r_j$ at the end of its route. We consider the preemptive version of multi-vehicle Dial-a-Ride, where an object may be left at intermediate vertices and transported by more than one vehicle, while being moved from source to destination. Approximation algorithms for the single vehicle Dial-a-Ride problem ($q=1$) have been considered in [3,10]. Our main results are an $O(\log^3 n)$-approximation algorithm for preemptive multi-vehicle Dial-a-Ride, and an improved $O(\log t)$-approximation for its special case when there is no capacity constraint (here $t \leq n$ is the number of distinct depot-vertices). There is an $\Omega(\log^{1/4} n)$ hardness of approximation known [9] even for single vehicle capacitated preemptive Dial-a-Ride.

We also obtain an improved constant factor approximation algorithm for the uncapacitated multi-vehicle problem on metrics induced by graphs excluding any fixed minor (e.g. planar metrics). In this case, we obtain improved guarantees of $O(\log n)$ for capacitated multi-vehicle Dial-a-Ride, and $O(1)$ for the uncapacitated problem.

Asymmetric k-Center with Minimum Coverage

In this paper we give approximation algorithms and inapproximability results for various asymmetric k-center with minimum coverage problems. In the k-center with minimum coverage problem, each center is required to serve a minimum number of clients. These problems have been studied by Lim et al. [A. Lim, B. Rodrigues, F. Wang, Z. Xu, k-center problems with minimum coverage, Theoret. Comput. Sci. 332 (1–3) (2005) 1–17] in the symmetric setting.
Finding Well-Balanced Pairs of Edge-Disjoint Trees in Edge-Weighted Graphs

The well-known number partition problem is NP-hard even in the following version: Given a set $S$ of $n$ non-negative integers; partition $S$ into two sets $X$ and $Y$ such that $|X| = |Y|$ and the sum of the elements in $X$ is as close as possible to the sum of the elements in $Y$ (or equivalently, minimize the maximum of the two sums). In this paper we study the following generalization of the partition problem: given an edge-weighted graph $G$ containing two edge-disjoint spanning trees. Find a pair of edge-disjoint spanning trees such that the maximum weight of these two trees is as small as possible. In the case when $G$ is precisely the union of two trees this problem may be seen as a generalization of the partition problem in which we have added a graph structure to the numbers (through the edges) and the extra restriction that only the sets $X$ and $Y$ which correspond to trees in $G$ are valid partitions. We first show how to obtain a 2-approximation via an algorithm for weighted matroid partition. Then we describe a simple heuristic which when applied to the 2-approximation above will result in a solution whose value is no more than $3/2$ times the value of an optimal solution. We also show that the approach above may sometimes exclude all the optimal solutions. Both the partition problem and its generalization to the problem above on edge-disjoint spanning trees are special cases of the problem of finding, in a weighted matroid with two disjoint bases, a pair of disjoint bases which minimize the maximum of their weights. In the last part of the paper we give some results on this problem for transversal matroids which turn out to be analogous to those for graphs.
Improved Approximate String Matching and Regular Expression Matching on Ziv-Lempel Compressed Texts

We study the approximate string matching and regular expression matching problem for the case when the text to be searched is compressed with the Ziv-Lempel adaptive dictionary compression schemes. We present a time-space trade-off that leads to algorithms improving the previously known complexities for both problems. In particular, we significantly improve the space bounds. In practical applications the space is likely to be a bottleneck and therefore this is of crucial importance.

General information
State: Published
Organisations: Algorithms and Logic, Department of Informatics and Mathematical Modeling
Contributors: Bille, P., Fagerberg, R., Gørtz, I. L.
Pages: 52-62
Publication date: 2007
Asymmetry in k-Center Variants

General information
State: Published
Organisations: Computer Science and Engineering, Department of Informatics and Mathematical Modeling
Contributors: Gørtz, I. L., Wirth, A.
Pages: 188-199
Publication date: 2006
Peer-reviewed: Yes

Publication information
Journal: Theoretical Computer Science
Volume: 361
Issue number: 2-3
ISSN (Print): 0304-3975
Ratings:
BFI (2019): BFI-level 2
Web of Science (2019): Indexed yes
BFI (2018): BFI-level 2
Web of Science (2018): Indexed yes
BFI (2017): BFI-level 2
Scopus rating (2017): CiteScore 1.08 SJR 0.488 SNIP 0.996
Web of Science (2017): Impact factor 0.772
Web of Science (2017): Indexed yes
BFI (2016): BFI-level 2
Scopus rating (2016): CiteScore 0.97 SJR 0.547 SNIP 0.996
Web of Science (2016): Impact factor 0.698
Web of Science (2016): Indexed yes
BFI (2015): BFI-level 2
Scopus rating (2015): CiteScore 1 SJR 0.592 SNIP 1.123
Web of Science (2015): Impact factor 0.643
Web of Science (2015): Indexed yes
BFI (2014): BFI-level 2
Scopus rating (2014): CiteScore 1.08 SJR 0.669 SNIP 1.148
Web of Science (2014): Impact factor 0.657
Web of Science (2014): Indexed yes
BFI (2013): BFI-level 2
Scopus rating (2013): CiteScore 1.17 SJR 0.717 SNIP 1.297
Web of Science (2013): Impact factor 0.516
ISI indexed (2013): ISI indexed yes
BFI (2012): BFI-level 2
Scopus rating (2012): CiteScore 1.16 SJR 0.78 SNIP 1.21
Web of Science (2012): Impact factor 0.489
ISI indexed (2012): ISI indexed yes
Web of Science (2012): Indexed yes
BFI (2011): BFI-level 2
Scopus rating (2011): CiteScore 1.17 SJR 0.747 SNIP 1.246
Web of Science (2011): Impact factor 0.665
ISI indexed (2011): ISI indexed yes
Hardness of Preemptive Finite Capacity Dial-a-Ride

General information
State: Published
Organisations: Computer Science and Engineering, Department of Informatics and Mathematical Modeling
Contributors: Gørtz, I. L.
Pages: 200-211
Publication date: 2006

Host publication information
Title of host publication: Proc. 9th Intl. Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2006) : Lecture Notes in Computer Science
Volume: 4110

Matching Subsequences in Trees

General information
State: Published
Organisations: Computer Science and Engineering, Department of Informatics and Mathematical Modeling, IT University of Copenhagen
Contributors: Bille, P., Gørtz, I. L.
Pages: 248-259
Publication date: 2006
Projects:

**Differential Privacy and Approximation Algorithms**
Steiner, T. A., PhD Student, Department of Applied Mathematics and Computer Science
Gartz, I. L., Main Supervisor, Department of Applied Mathematics and Computer Science
Bille, P., Supervisor, Department of Applied Mathematics and Computer Science
Rotenberg, E., Supervisor, Department of Applied Mathematics and Computer Science
15/02/2019 → 14/02/2022
Project: PhD

**Flows and Colourings**
Langhede, R. M., PhD Student, Department of Applied Mathematics and Computer Science
Thomassen, C., Main Supervisor, Department of Applied Mathematics and Computer Science
Gartz, I. L., Supervisor, Department of Applied Mathematics and Computer Science
Rotenberg, E., Supervisor, Department of Applied Mathematics and Computer Science
Technical University of Denmark
01/01/2018 → 31/12/2020
Award relations: Flows and Colourings
Project: PhD

**Graph Coloring and Decomposition**
Lyngsie, K. S., PhD Student, Department of Applied Mathematics and Computer Science
Thomassen, C., Main Supervisor, Department of Applied Mathematics and Computer Science
Gartz, I. L., Supervisor, Department of Applied Mathematics and Computer Science
Technical University of Denmark
01/08/2016 → 31/07/2019
Award relations: Graph Coloring and Decomposition
Project: PhD

**Compressed Computation on Structured Data**
Ettienne, M. B., PhD Student, Department of Applied Mathematics and Computer Science
Bille, P., Main Supervisor, Department of Applied Mathematics and Computer Science
Gartz, I. L., Supervisor, Department of Applied Mathematics and Computer Science
Witt, C., Examiner, Department of Applied Mathematics and Computer Science
Fischer, J. C., Examiner
Munro, J. I., Examiner
Samfinansieret - Andet
01/05/2015 → 14/11/2018
Award relations: Compressed Computation on Structured Data
Project: PhD

**Algorithms for Compressed Computation**
Christiansen, A. R., PhD Student, Department of Applied Mathematics and Computer Science
Bille, P., Main Supervisor, Department of Applied Mathematics and Computer Science
Gartz, I. L., Supervisor, Department of Applied Mathematics and Computer Science
Witt, C., Examiner, Department of Applied Mathematics and Computer Science
Grossi, R., Examiner
Puglisi, S. J., Examiner
Samfinansieret - Andet
01/11/2014 → 13/03/2018
Award relations: Algorithms for Compressed Computation
Matching and Compression of Strings with Automata and Word Packing
Skjoldjensen, F. R., PhD Student, Department of Applied Mathematics and Computer Science
Bille, P., Main Supervisor, Department of Applied Mathematics and Computer Science
Gørtz, I. L., Supervisor, Department of Applied Mathematics and Computer Science
Thomassen, C., Supervisor, Department of Applied Mathematics and Computer Science
Witt, C., Examiner, Department of Applied Mathematics and Computer Science
Landau, G. M., Examiner
Pagh, R., Examiner
Forskningsrådsfinansiering
01/03/2014 → 14/06/2017
Award relations: Matching and Compression of Strings with Automata and Word Packing
Project: PhD

Decomposition of Graphs
Merker, M., PhD Student, Department of Applied Mathematics and Computer Science
Thomassen, C., Main Supervisor, Department of Applied Mathematics and Computer Science
Gørtz, I. L., Examiner, Department of Applied Mathematics and Computer Science
Bang-Jensen, J., Examiner
Barat, J., Examiner, Department of Applied Mathematics and Computer Science
Anden EU-finansiering
01/09/2013 → 23/11/2016
Award relations: Decomposition of Graphs
Project: PhD

Algorithms for Metadata
Vind, S. J., PhD Student, Department of Informatics and Mathematical Modeling
Gørtz, I. L., Main Supervisor, Department of Informatics and Mathematical Modeling
Bille, P., Supervisor, Department of Informatics and Mathematical Modeling
Witt, C., Examiner, Department of Informatics and Mathematical Modeling
Italiano, G. F., Examiner
Clifford, R., Examiner
Technical University of Denmark
01/10/2012 → 20/05/2015
Award relations: Algorithms for Metadata
Project: PhD

Fast Approximate Tree Pattern Matching
Cording, P. H., PhD Student, Department of Informatics and Mathematical Modeling
Bille, P., Main Supervisor, Department of Informatics and Mathematical Modeling
Gørtz, I. L., Supervisor, Department of Informatics and Mathematical Modeling
Fischer, P., Examiner, Department of Informatics and Mathematical Modeling
Grossi, R., Examiner
Landau, G. M., Examiner
Technical University of Denmark
15/12/2011 → 04/03/2015
Award relations: Fast Approximate Tree Pattern Matching
Project: PhD

Approximate Text Indexing and String Matching Algorithms
Vildhøj, H. W., PhD Student, Department of Informatics and Mathematical Modeling
Gørtz, I. L., Main Supervisor, Department of Informatics and Mathematical Modeling
Bille, P., Supervisor, Department of Informatics and Mathematical Modeling
Witt, C., Examiner, Department of Informatics and Mathematical Modeling
Farach-Colton, M., Examiner
Lewenstein, M., Examiner
Technical University of Denmark
15/09/2011 → 19/12/2014
Award relations: Approximate Text Indexing and String Matching Algorithms
Project: PhD