String Attractors: Verification and Optimization

Dominik Kempa
Department of Computer Science, University of Helsinki, Finland
dkempa@cs.helsinki.fi
https://orcid.org/0000-0003-2286-7417

Alberto Policriti
Department of Computer Science, University of Udine, Italy
alberto.policriti@uniud.it
https://orcid.org/0000-0001-8502-5896

Nicola Prezza
Department of Computer Science, University of Pisa, Italy
nicola.prezza@di.unipi.it
https://orcid.org/0000-0003-3553-4953

Eva Rotenberg
DTU Compute, Technical University of Denmark, Denmark
erot@dtu.dk
https://orcid.org/0000-0001-5853-7909

Abstract

String attractors [STOC 2018] are combinatorial objects recently introduced to unify all known dictionary compression techniques in a single theory. A set $\Gamma \subseteq [1..n]$ is a k*-attractor* for a string $S \in \Sigma^n$ if and only if every distinct substring of S of length at most k has an occurrence crossing at least one of the positions in Γ. Finding the smallest k-attractor is NP-hard for $k \geq 3$, but polylogarithmic approximations can be found using reductions from dictionary compressors. It is easy to reduce the k-attractor problem to a set-cover instance where the string's positions are interpreted as sets of substrings. The main result of this paper is a much more powerful reduction based on the truncated suffix tree. Our new characterization of the problem leads to more efficient algorithms for string attractors: we show how to check the validity and minimality of a k-attractor in near-optimal time and how to quickly compute exact solutions. For example, we prove that a minimum 3-attractor can be found in $\mathcal{O}(n)$ time when $|\Sigma| \in \mathcal{O}(\sqrt[3+\epsilon]{\log n})$ for some constant $\epsilon > 0$, despite the problem being NP-hard for large Σ.

2012 ACM Subject Classification Theory of computation → Data compression

Keywords and phrases Dictionary compression, String attractors, Set cover

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.52

Related Version A full version of the paper is available at [15], `https://arxiv.org/abs/1803.01695`.

1 Introduction

The goal of dictionary compression is to reduce the size of an input string by exploiting its repetitiveness. In the last decades, several dictionary compression techniques – some more powerful than others – were developed to achieve this goal: Straight-Line programs [17] (context-free grammars generating the string), Macro schemes [23] (a set of substring equations having the string as unique solution), the run-length Burrows-Wheeler transform [4] (a string permutation whose number of equal-letter runs decreases as the string's repetitiveness

26th Annual European Symposium on Algorithms (ESA 2018).
Editors: Yossi Azar, Hannah Bast, and Grzegorz Herman; Article No. 52; pp. 52:1–52:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

increases), and the compact directed acyclic word graph [3, 6] (the minimization of the suffix tree). Each scheme from this family comes with its own set of algorithms and data structures to perform compressed-computation operations – e.g. random access – on the compressed representation. Despite being apparently unrelated, in [16] all these compression schemes were proven to fall under a common general scheme: they all induce a set $\Gamma \subseteq [1..n]$ whose cardinality is bounded by the compressed representation's size and with the property that each distinct substring has an occurrence crossing at least one position in Γ. A set with this property is called a *string attractor*. Intuitively, positions in a string attractor capture "interesting" regions of the string; a string of low complexity (that is, more compressible), will generate a smaller attractor. Surprisingly, given such a set one can build a data structure of size $\mathcal{O}(|\Gamma| \operatorname{polylog}(n))$ supporting random access queries in optimal time [16]: string attractors therefore provide a universal framework for performing compressed computation on top of *any* dictionary compressor (and even optimally for particular queries such as random access).

These premises suggest that an algorithm computing the smallest string attractor for a given string would be a valuable tool for designing better compressed data structures. Unfortunately, computing a minimum string attractor is NP-hard. The problem remains NP-hard even under the restriction that only substrings of length at most k are captured by Γ, for any $k \geq 3$ and on large alphabets. In this case, we refer to the problem as *k-attractor*. Not all hope is lost, though: as shown in [16], dictionary compressors are actually heuristics for computing a small n-attractor (with polylogarithmic approximation rate w.r.t. the minimum), and, more generally, k-attractor admits a $\mathcal{O}(\log k)$-approximation based on a reduction to set cover. It is actually easy to find such a reduction: choose as universe the set of distinct substrings and as set collection the string's positions (i.e., set s_i contains all substrings crossing position i). The main limitation of this approach is that the set of distinct substrings could be quadratic in size; this makes the strategy of little use in cases where the goal is to design usable (i.e., as close as possible to linear-time) algorithms on string attractors.

The core result of this paper is a much more powerful reduction from k-attractor to set-cover: the universe $\mathcal{U}$ of our instance is equal to the set of edges of the k-truncated suffix tree, while the size of the set collection $\mathcal{S} \subseteq 2^{\mathcal{U}}$ is bounded by the size of the $(2k-1)$-truncated suffix tree. First of all, we obtain a universe that is always at least k times smaller than the naive approach. Moreover, the size of our set-cover instance does not depend on the string's length n, unless σ and k do. This allows us to show that k-attractor is actually solvable in polynomial time for small values of k and σ, and leads us to efficient algorithms for a wide range of different problems on string attractors.

The paper is organized as follows. In Section 1.1 we describe the notation used throughout the paper and we report the main notions related to k-attractors. In Section 1.2 we give the main theorem stating our reduction to set-cover (Theorem 5) and briefly discuss the results that we obtain in the rest of the paper by applying it. Theorem 5 itself is proven in Section 2 and is used in Section 3 to provide fast algorithms on string attractors. Finally, in Section 3.4 we introduce and study the complexity of the closely-related *sharp-k-attractor* problem: to capture all distinct substrings of length exactly k. The full version [15] of this paper covers additional material related to the approximation of minimum k-attractors.

1.1 Notation and definitions

Throughout we consider a string $S[1..n]$ of n symbols. We assume the reader to be familiar with the notions of *suffix tree* [24], *suffix array* [18], and *wavelet tree* [10, 21]. By $ST^k(S)$ we denote a k-truncated suffix tree of S, i.e., a compact trie containing all substrings of S

of length at most k. $\mathcal{E}(\mathcal{T})$ denotes the set of edges of the compact trie $\mathcal{T}$. $\mathcal{L}(\mathcal{T})$ denotes the set of leaves at maximum string depth of the compact trie $\mathcal{T}$ (i.e., leaves whose string depth is equal to the maximum string depth among all leaves). Let $e = \langle u, v\rangle$ be an edge in the (truncated) suffix tree of S. With $s(e)$ we denote the string read from the suffix tree root to (and including) the first character in the label of e. $\lambda(e) = |s(e)|$ is the length of this string. We will also refer to $\lambda(e)$ as the *string depth* of e. Note that edges $e_1, \dots, e_t$ of the k-truncated suffix tree have precisely the same labels $s(e_1), \dots, s(e_t)$ of the suffix tree edges $e'_1, \dots, e'_t$ at string depth $\lambda(e'_i) \leq k$. It follows that we can use these two edge sets interchangeably when we are only interested in their labels (this will be the case in our results). Let $SA[1..n]$ denote the suffix array of S. $\langle l_e, r_e\rangle$, with $e \in \mathcal{E}(ST^k(S))$ being an edge in the k-truncated suffix tree, will denote the suffix array range corresponding to the string $s(e)$, i.e., $SA[l_e..r_e]$ contains all suffixes prefixed by $s(e)$.

Unless otherwise specified, we give the space of our data structures in words of $\Theta(\log n)$ bits each.

With the following definition we recall the notion of *k-attractor* of a string [16].

▶ **Definition 1.** A *k-attractor* of a string $S \in \Sigma^n$ is a set of positions $\Gamma \subseteq [1..n]$ such that every substring $S[i..j]$ with $i \leq j < i + k$ has at least one occurrence $S[i'..j'] = S[i..j]$ with $j'' \in [i'..j']$ for some $j'' \in \Gamma$.

When $k = n$, we simply call Γ an *attractor* of S.

▶ **Definition 2.** A *minimal k-attractor* of a string $S \in \Sigma^n$ is a k-attractor Γ such that $\Gamma - \{j\}$ is not a k-attractor of S for any $j \in \Gamma$.

▶ **Definition 3.** A *minimum k-attractor* of a string $S \in \Sigma^n$ is a k-attractor Γ^* such that, for any k-attractor Γ of S, $|\Gamma^*| \leq |\Gamma|$.

▶ **Theorem 4.** *[16, Thm. 4.2] The problem of deciding whether a string S admits a k-attractor of size at most t is NP-complete for $k \geq 3$.*

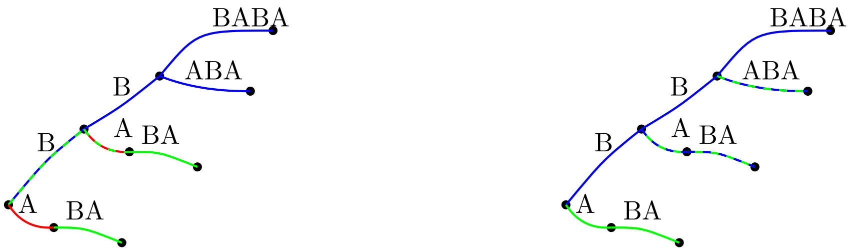
1.2 Overview of the contributions

Our main theorem is a reduction to set-cover based on the notion of truncated suffix tree:

▶ **Theorem 5.** *Let $S' = \#^{k-1}S\#^{k-1}$, with $\# \notin \Sigma$. An instance of k-attractor can be reduced to a set-cover instance with universe equal to the set of edges of the k-truncated suffix tree of S and set collection with one element for each leaf of the $(2k-1)$-truncated suffix tree of S'.*

Figure 1 depicts the main technique (Lemma 11) standing at the core of our reduction: a set Γ is a valid attractor if and only if it marks (or colors, in the picture), all suffix tree edges.

Using the reduction of Theorem 5, we obtain the following results. First, we present efficient algorithms to check the validity and minimality of a k-attractor. Note that it is trivial to perform these checks in cubic time (or quadratic, with little more care). In Theorem 18 we show that we can check whether a set $\Gamma \subseteq [1..n]$ is a valid k-attractor for S in $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ words of space. Using recent advances in compact data structures, we show how to further reduce this working space to $\mathcal{O}(n \log \sigma)$ bits without affecting query times when $k \leq \sigma^{\mathcal{O}(1)}$, or with a small time penalty in the general case. In particular, when k is polynomial in the alphabet size, we can always check the correctness of a k-attractor in $\mathcal{O}(n)$ time and $\mathcal{O}(n \log \sigma)$ bits of space. With similar techniques, in Theorem 22 we show how to verify that Γ is a *minimal* k-attractor for S in near-optimal $\mathcal{O}(n \log n)$ time. To conclude, in Theorem 24 we show that a minimum k-attractor can be found in $\mathcal{O}(n) + \exp\big(\mathcal{O}(\sigma^k \log \sigma^k)\big)$ time. In particular, this result yields the following corollaries:



(a) The suffix tree of the string BBBABA, colored corresponding to attractor positions $2, 5, 6$. This attractor is minimal: removing any position leaves some edge uncolored.

(b) The suffix tree of the string BBBABA, colored corresponding to attractor positions $3, 4$. This attractor is minimum: it is minimal and of minimum size.

Figure 1 A position i "marks" (or, here, *colors*) a suffix tree edge e if and only if it crosses an occurrence of the string read from the root to the first letter in the label of e. A set of positions forms a k-attractor if and only if they color all edges of the k-truncated suffix tree (in this figure, $k = 6$ and we color the whole suffix tree). Dashed lines indicate that the edge has multiple colors. The string terminator \$ (and edges labeled with \$) is omitted for simplicity.

▶ **Corollary 6.** *k-attractor is in P when $\sigma^k \log \sigma^k \in \mathcal{O}(\log n)$.*

▶ **Corollary 7.** *For constant k, a minimum k-attractor can be found in $\mathcal{O}(n)$ time if there exists $\epsilon > 0$ such that $\sigma^{k+\epsilon} \in \mathcal{O}(\log n)$.*

Proof. Let $\sigma^{k+\epsilon} = \sigma^{k(1+\epsilon')}$, where $\epsilon' = \epsilon/k > 0$ is a constant. For any constant $\epsilon' > 0$ we have that $\sigma^k \log \sigma^k \in o(\sigma^k \cdot (\sigma^k)^{\epsilon'}) = o(\sigma^{k(1+\epsilon')})$. It follows that $\sigma^k \log \sigma^k \in o(\log n)$, i.e., by Theorem 24 we can find a minimum k-attractor in linear time. ◀

With the above result we can, for example, find a minimum 3-attractor in $\mathcal{O}(n)$ time when $\sigma \in \mathcal{O}(\sqrt[3+\epsilon]{\log n})$, for some $\epsilon > 0$ (keep in mind that 3-attractor is NP-complete for large alphabets).

2 A better reduction to set-cover

In this section we give our main result: a reduction from k-attractor to set-cover using a universe smaller than the one used in [16]. We start with an alternative characterization of k-attractors based on the k-truncated suffix tree.

▶ **Definition 8** (Marker). $j \in \Gamma$ is a *marker* for a suffix tree edge e if and only if

$$\exists i \in SA[l_e..r_e] \ : \ i \leq j < i + \lambda(e)$$

Equivalently, we say that j *marks* e (see Figure 1).

▶ **Definition 9** (Edge marking). $\Gamma \subseteq [1..n]$ *marks* a suffix tree edge e if and only if there exists a $j \in \Gamma$ that marks e.

▶ **Definition 10** (Suffix tree k-marking). $\Gamma \subseteq [1..n]$ is a *suffix tree k-marking* if and only if it marks every edge e such that $\lambda(e) \leq k$ (equivalently, every $e \in \mathcal{E}(ST^k(S))$).

When $k = n$ we simply say *suffix tree marking* (since all edges satisfy $\lambda(e) \leq n$). We now show that the notions of k-attractor and suffix tree k-marking are equivalent.

▶ **Lemma 11.** *Γ is a k-attractor if and only if it is a suffix tree k-marking.*

Proof. ($\Rightarrow$) Let Γ be a k-attractor. Pick any suffix tree edge e such that $\lambda(e) \leq k$. Then, $\lambda(e) = |s(e)| \leq k$ and, by definition of k-attractor, there exists a $j \in \Gamma$ and an i such that $s(e) = S[i..i + |s(e)| - 1]$ and $i \leq j \leq i + |s(e)| - 1$. We also have that $i \in SA[l_e..r_e]$ (being $\langle l_e, r_e \rangle$ precisely the suffix array range of suffixes prefixed by $s(e)$). Putting these results together, we found an $i \in SA[l_e..r_e]$ such that $i \leq j \leq i + \lambda(e) - 1$ for some $j \in \Gamma$, which by Definition 9 means that Γ marks e. Since the argument works for any edge e at string depth at most k, we obtain that Γ is a suffix tree k-marking.

($\Leftarrow$) Let Γ be a suffix tree k-marking. Let, moreover, s be a substring of S of length at most k. Consider the lowest suffix tree edge e (i.e., the e with maximum $\lambda(e)$) such that $s(e)$ prefixes s. In particular, $\lambda(e) \leq k$. Note that, by definition of suffix tree, every occurrence $S[i..i + |s(e)| - 1] = s(e)$ of $s(e)$ in S prefixes an occurrence of s: $S[i..i + |s| - 1] = s$. By definition of k-marking, there exists a $j \in \Gamma$ such that j is a marker for e, which means (by Definition 8) that $\exists i \in SA[l_e..r_e] \ : \ i \leq j < i + \lambda(e)$. Since $i \in SA[l_e..r_e]$, $SA[i]$ is an occurrence of $s(e)$, and therefore of s. But then, we have that $i \leq j < i + \lambda(e) = i + |s(e)| \leq i + |s|$, i.e., $S[SA[i]..SA[i] + |s| - 1]$ is an occurrence of s crossing $j \in \Gamma$. Since the argument works for every substring s of S of length at most k, we obtain that Γ is a k-attractor. ◀

An equivalent formulation of Lemma 11 is that Γ is a k-attractor if and only if it marks all edges of the k-truncated suffix tree. In particular (case $k = n$), Γ is an attractor if and only if it is a suffix tree marking.

Lemma 11 will be used to obtain a smaller universe $\mathcal{U}$ in our set-cover reduction. With the following lemmas we show that also the size of the set collection $\mathcal{S}$ can be considerably reduced when k and σ are small.

▶ **Definition 12** (k-equivalence)**.** Two positions $i, j \in [1..n]$ are k-equivalent, indicated as $i \equiv_k j$, if and only if

$$S'[i - k + 1..i + k - 1] = S'[j - k + 1..j + k - 1]$$

where $S'[i] = \#$ if $i < 1$ or $i > n$ (note that we allow negative positions) and $S'[i] = S[i]$ otherwise, and $\# \notin \Sigma$ is a new character.

It is easy to see that k-equivalence is an equivalence relation. First, we bound the size of the distinct equivalence classes of $\equiv_k$ (i.e., the size of the quotient set $[1..n]/\equiv_k$).

▶ **Lemma 13.** $|[1..n]/\equiv_k| = |\mathcal{L}(ST^{2k-1}(S'))| \leq \min\{n, \sigma^{2k-1} + 2k - 2\}$

Proof. By definition of $\equiv_k$, the set $[1..n]/\equiv_k$ has one element per distinct substring of length $(2k - 1)$ in S', that is, per distinct path from the suffix tree root to each of the nodes in $\mathcal{L}(ST^{2k-1}(S'))$. Clearly, $|\mathcal{L}(ST^{2k-1}(S'))| \leq n$. On the other hand, there are at most σ^{2k-1} distinct substrings of length $2k - 1$ on Σ. Moreover, there are $2k - 2$ additional substrings to consider on the borders of S' (to include the runs of symbol $\#$). It follows that the cardinality of $\mathcal{L}(ST^{2k-1}(S'))$ is upper-bounded also by $\sigma^{2k-1} + 2k - 2$. ◀

We now show that any minimal k-attractor can have at most one element from each equivalence class of $\equiv_k$.

▶ **Lemma 14.** *If Γ is a minimal k-attractor, then for any $1 \leq i \leq n$ it holds $|\Gamma \cap [i]_{\equiv_k}| \leq 1$.*

Proof. Suppose, by contradiction, that $|\Gamma \cap [i]_{\equiv_k}| > 1$ for some i. Then, let $j, j' \in \Gamma \cap [i]_{\equiv_k}$, with $j \neq j'$. By definition of $\equiv_k$, $S'[j - k + 1..j + k - 1] = S'[j' - k + 1..j' + k - 1]$. This means that if a substring of S of length at most k has an occurrence crossing position j

in Γ then it has also one occurrence crossing position $j' \in (\Gamma - \{j\})$. On the other hand, any other substring occurrence crossing any position $j'' \neq j, j'$ is also captured by $\Gamma - \{j\}$ since j'' belongs to this set. This implies that $\Gamma - \{j\}$ is a k-attractor, which contradicts the minimality of Γ. ◀

Moreover, if we swap any element of a k-attractor with an equivalent element then the resulting set is still a k-attractor:

▶ **Lemma 15.** *Let Γ be a k-attractor. Then, $(\Gamma - \{j\}) \cup \{j'\}$ is a k-attractor for any $j \in \Gamma$ and any $j' \equiv_k j$.*

Proof. Pick any occurrence of a substring s, $|s| \leq k$, crossing position j. By definition of $\equiv_k$, since $j' \equiv_k j$ there is also an occurrence of s crossing j'. This implies that $\Gamma' = (\Gamma - \{j\}) \cup \{j'\}$ is a k-attractor. ◀

Lemmas 14 and 15 imply that we can reduce the set of candidate positions from $[1..n]$ to $\mathcal{C} = \{\min(I) \mid I \in [1..n]/\equiv_k\}$ (that is, an arbitrary representative – in this case, the minimum – from any class of $\equiv_k$), and still be able to find a minimal/minimum k-attractor. Note that, by Lemma 13, $|[1..n]/\equiv_k| \leq \min\{n, \sigma^{2k-1} + 2k - 2\}$.

We can now prove our main theorem.

Proof of Theorem 5. We build our set-cover instance $\langle \mathcal{U}, \mathcal{S} \rangle$ as follows. We choose $\mathcal{U} = \mathcal{E}(ST^k(S))$, i.e., the set of edges of the k-truncated suffix tree. The set collection $\mathcal{S}$ is defined as follows: let $s_i = \{e \in \mathcal{E}(ST^k(S)) \mid i \text{ marks } e\}$, $\mathcal{C} = \{\min(I) \mid I \in [1..n]/\equiv_k\}$ and put:

$$\mathcal{S} = \{s_i \mid i \in \mathcal{C}\}.$$

By definition of $\equiv_k$, each $I \in [1..n]/\equiv_k$ is unambiguously identified by a substring of length $2k - 1$ of the string $S' = \#^{k-1} S \#^{k-1}$. We therefore obtain $|\mathcal{S}| = |\mathcal{L}(ST^{2k-1}(S'))|$. We now prove the correctness and completeness of the reduction.

Correctness. By the definition of our reduction, a solution $\{s_{i_1}, \dots, s_{i_\gamma}\}$ to $\langle \mathcal{U}, \mathcal{S} \rangle$ yields a set $\Gamma = \{i_1, \dots, i_\gamma\}$ of positions marking every edge in $\mathcal{E}(ST^k(S))$. Then, Lemma 11 implies that Γ is a k-attractor.

Completeness. Let $\Gamma = \{i_1, \dots, i_\gamma\}$ be a minimal k-attractor. Then, Lemmas 14 and 15 imply that the following set is also a minimal k-attractor of the same size: $\Gamma' = \{j_1 = \min([i_1]_{\equiv_k}), \dots, j_\gamma = \min([i_\gamma]_{\equiv_k})\}$. Note that $\Gamma' \subseteq \{\min(I) \mid I \in [1..n]/\equiv_k\}$. By Lemma 11, Γ' marks every edge in $\mathcal{E}(ST^k(S))$. Then, by definition of our reduction the collection $\{s_{j_1}, \dots, s_{j_\gamma}\}$ covers $\mathcal{U} = \mathcal{E}(ST^k(S))$. ◀

In the rest of the paper, we use the notation $\mathcal{U} = \mathcal{E}(ST^k(S))$ and $\mathcal{C} = \{\min(I) \mid I \in [1..n]/\equiv_k\}$ to denote the universe to be covered (edges of the k-truncated suffix tree) and the candidate attractor positions, respectively. Recall, moreover, that $|\mathcal{U}| \leq \min\{n, \sigma^k\}$ and $|\mathcal{C}| \leq \min\{n, \sigma^{2k-1} + 2k - 2\}$.

3 Faster algorithms

In this section we use properties of our reduction to provide faster algorithms for a range of problems: (i) checking that a given set $\Gamma \subseteq [1..n]$ is a k-attractor, (ii) checking that a given set is a *minimal* k-attractor, and (iii) finding a minimum k-attractor. We note that problems (i)-(ii) admit naive cubic solutions, while problem (iii) is NP-hard for $k \geq 3$ [16].

We assume that the input string S is over the integers alphabet $[1..\sigma]$ where $\sigma \in n^{\mathcal{O}(1)}$. Note that suffix-sorting can be performed in linear time and space on such alphabets [14].

3.1 Checking the attractor property

Given a string S, a set $\Gamma \subseteq [1..n]$, and an integer $k \geq 1$, is Γ a k-attractor for S? We show that this question can be answered in $\mathcal{O}(n)$ time.

The main idea is to use Lemma 11 and check, for every suffix tree edge e at string depth at most k, if Γ marks e. Consider the suffix array $SA[1..n]$ of S and the array $D[1..n]$ defined as follows: $D[i] = \text{succ}(\Gamma, SA[i]) - SA[i]$, where $\text{succ}(X, x)$ returns the smallest element larger than or equal to x in the set X (i.e., $D[i]$ is the distance between $SA[i]$ and the element of Γ following – and possibly equal to – $SA[i]$). D can be built in linear time and space by creating a bit-vector $B[1, n]$ such that $B[i] = 1$ iff $i \in \Gamma$ and pre-processing B for constant-time successor queries [13, 5]. We build a range-minimum data structure (RMQ) on D ($\mathcal{O}(n)$ bits of space, constant query time [9]). Then for every suffix tree edge e such that $\lambda(e) \leq k$, we check (in constant time) that $\lambda(e) > \min(D[l_e..r_e])$. The following lemma ensures that this is equivalent to checking whether Γ marks e.

▶ **Lemma 16.** *$\lambda(e) > \min(D[l_e..r_e])$ if and only if Γ marks e.*

Proof. ($\Rightarrow$) Assume that $\lambda(e) > \min(D[l_e..r_e])$. By definition of D, this means that there exist an index $i' \in [l_e..r_e]$ and a $j \in \Gamma$, with $j \geq i = SA[i']$, such that $j - i = D[i'] < \lambda(e)$. Equivalently, $i \leq j < i + \lambda(e)$, i.e., Γ marks e.

($\Leftarrow$) Assume that Γ marks e. Then, by definition, there exist an index $i' \in [l_e..r_e]$ and a $j \in \Gamma$ such that $SA[i'] = i \leq j < i + \lambda(e)$. Then, $j - i < \lambda(e)$. Since $D[i']$ is computed taking the $j \in \Gamma$, $j \geq SA[i']$, minimizing $j - SA[i']$, it must be the case that $D[i'] \leq j - i < \lambda(e)$. Since $i' \in [l_e..r_e]$, this implies that $\min(D[l_e..r_e]) < \lambda(e)$. ◀

Together, Lemmas 11 and 16 imply that, if $\lambda(e) > \min(D[l_e..r_e])$ for every edge at string depth at most k, then Γ is a k-attractor for S. Since the suffix tree, as well as the other structures used by our algorithm, can be built in linear time and space on alphabet $[1..n]$ [7] and checking each edge takes constant time, we obtain that the problem of checking whether a set $\Gamma \subseteq [1..n]$ is a valid k-attractor can be solved in $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ words of space. We now show how to improve upon this working space by using recent results in the field of compact data structures. In the following result, we assume that the input string is packed in $\mathcal{O}(n \log \sigma)$ bits (that is, $\mathcal{O}(n/\log_\sigma n)$ words).

We first need the following Lemma from [2]:

▶ **Lemma 17.** *[2, Thm. 3] In $\mathcal{O}(n)$ time and $\mathcal{O}(n \log \sigma)$ bits of space we can enumerate the following information for each suffix tree edge e:*

- *The suffix array range $\langle l_e, r_e \rangle$ of the string $s(e)$, and*
- *the length $\lambda(e)$ of $s(e)$.*

Proof. In [2, Thm. 3] (see also [1]) the authors show how to enumerate the following information for each right-maximal substring W of S in $\mathcal{O}(n)$ time and $\mathcal{O}(n \log \sigma)$ bits of space: $|W|$ and the suffix array range $\mathit{range}(Wb)$ of the string Wb, for all $b \in \Sigma$ such that Wb is a substring of S. Since W is right-maximal, those Wb are equal to our strings $s(e)$ (for every edge e). It follows that our problem is solved by outputting all $\mathit{range}(Wb)$ and $|Wb|$ returned by the algorithm in [2, Thm. 3]. ◀

We can now prove our theorem. Note that the input set $\Gamma \subseteq [1..n]$ can be encoded in n bits, so also the input fits in $\mathcal{O}(n \log \sigma)$ bits.

▶ **Theorem 18.** *Given a string $S \in [1..\sigma]^n$, a set $\Gamma \subseteq [1..n]$, and an integer $k \geq 1$, we can check whether Γ is a k-attractor for S in:*

- *Optimal $\mathcal{O}(n \log \sigma)$ bits of space and $\mathcal{O}(n \log^\epsilon n)$ time, for any constant $\epsilon > 0$, or*
- *$\mathcal{O}(n(\log \sigma + \log k))$ bits of space and $\mathcal{O}(n)$ time.*

Proof. To achieve the first trade-off we will replace the D array (occupying $\mathcal{O}(n \log n)$ bits) with a smaller data structure supporting random access to D. We start by replacing the standard suffix array with a compressed suffix array (CSA) [8, 11]. Given a text stored in $\mathcal{O}(n \log \sigma)$ bits, the CSA can be built in deterministic $\mathcal{O}(n)$ time and optimal $\mathcal{O}(n \log \sigma)$ bits of space [20], and supports access queries to the suffix array SA in $\mathcal{O}(\log^\epsilon n)$ time [11], for any constant $\epsilon > 0$ chosen at construction time. Given that $D[i] = \mathrm{succ}(\Gamma, SA[i]) - SA[i]$ and we can compute the successor function in constant time using a $\mathcal{O}(n)$-bit data structure (array B), $D[i]$ can be computed in $\mathcal{O}(\log^\epsilon n)$ time. Using access to D, the RMQ data structure (occupying $\mathcal{O}(n)$ bits) can be built in $\mathcal{O}(n \log^\epsilon n)$ time and $\mathcal{O}(n)$ bits of space [9, Thm. 5.8]. At this point, observe that the order in which we visit suffix tree edges does not affect the correctness of our algorithm. By using Lemma 17 we can enumerate $\lambda(e)$ and $\langle l_e, r_e \rangle$ for every suffix tree edge e in linear time and compact space, and check $\lambda(e) > \min(D[l_e..r_e])$, whenever $\lambda(e) \leq k$ (Lemma 16).

To achieve the second trade-off we observe that in our algorithm we only explore the suffix tree up to depth k (i.e., we only perform the check of Lemma 16 when $\lambda(e) \leq k$), hence any $D[i] > k$ can be replaced with $D[i] = k + 1$ without affecting the correctness of the verification procedure. In this way, array D can be stored in just $\mathcal{O}(n \log k)$ bits. To compute the D array in $\mathcal{O}(n)$ time and compact space we observe that it suffices to access all pairs $\langle i, SA[i] \rangle$ in *any* order (not necessarily $\langle 1, SA[1] \rangle, \langle 2, SA[2] \rangle, \dots$). From [2, Thm. 10], in $\mathcal{O}(n)$ time and $\mathcal{O}(n \log \sigma)$ bits of space we can build a compressed suffix array supporting constant-time LF function computation. By repeatedly applying LF from the first suffix array position, we enumerate entries of the inverse suffix array ISA in right-to-left order in $\mathcal{O}(n)$ time [2, Lem. 1]. This yields the sequence of pairs $\langle ISA[i], i \rangle = \langle j, SA[j] \rangle$, for $i = n, \dots, 1$ and $j = ISA[i]$, which can be used to compute D in linear time and compact space. As in the first trade-off, we use Lemma 17 to enumerate $\lambda(e)$ and $\langle l_e, r_e \rangle$ for every suffix tree edge e, and check $\lambda(e) > \min(D[l_e..r_e])$, whenever $\lambda(e) \leq k$ (Lemma 16). ◀

Note that with the second trade-off of Theorem 18 we achieve $\mathcal{O}(n)$ time and optimal $\mathcal{O}(n \log \sigma)$ bits of space when $k \leq \sigma^{\mathcal{O}(1)}$ (in particular, this is always the case when k is constant). Note also that, since we now assume that the input string is packed in $\mathcal{O}(n / \log_\sigma n)$ words, the running time is not optimal (as $\Omega(n / \log_\sigma n)$ is a lower-bound in this model).

3.2 Checking minimality

Given a string S, a set $\Gamma \subseteq [1..n]$, and an integer $k \geq 1$, is Γ a *minimal* k-attractor for S? The main result of this section is that this question can be answered in near-optimal $\mathcal{O}(n \log n)$ time.

We first show that minimal k-attractors admit a convenient characterization based on the concept of suffix tree k-marking.

▶ **Definition 19** (k-necessary position)**.** $j \in \Gamma$ is *k-necessary* with respect to Γ if and only if there is at least one suffix tree edge e such that:

1. $\lambda(e) \leq k$,
2. j marks e, and
3. If $j' \in \Gamma$ marks e, then $j' = j$

▶ **Definition 20** (k-necessary set)**.** Γ is *k-necessary* if and only if all its elements are k-necessary with respect to Γ.

When Γ is clear from the context, we just say *k-necessary* (referring to some $j \in \Gamma$) instead of *k-necessary with respect to* Γ.

▶ **Lemma 21.** *Γ is a minimal k-attractor if and only if:*

1. *It is a k-attractor, and*
2. *it is k-necessary.*

Proof. ($\Rightarrow$) Let Γ be a minimal k-attractor. Let $j \in \Gamma$. Since Γ is minimal, $\Gamma - \{j\}$ is not a k-attractor. From Theorem 11, this implies that $\Gamma - \{j\}$ is not a k-marking, i.e., there exists a suffix tree edge e, with $\lambda(e) \leq k$, that is not marked by $\Gamma - \{j\}$. On the other hand, the fact that Γ is a k-attractor implies (Theorem 11) that Γ is a k-marking, i.e., it also marks edge e. This, in particular, implies that j marks e. Now, let $j' \in \Gamma$ be a position that marks e. Assume, by contradiction, that $j' \neq j$. Then, $j' \in \Gamma - \{j\}$, which implies that $\Gamma - \{j\}$ marks e. This is a contradiction, therefore it must be the case that $j' = j$, i.e., j is k-necessary. Since the argument works for any $j \in \Gamma$, we obtain that all $j \in \Gamma$ are k-necessary.

($\Leftarrow$) Assume that Γ is a k-attractor and all $j \in \Gamma$ are k-necessary. Then, choose an arbitrary $j \in \Gamma$. By Definition 19, there exists an edge e that is only marked by j, i.e., for every $j' \in \Gamma - \{j\}$, j' does not mark e. This implies (Theorem 11) that $\Gamma - \{j\}$ is not a k-attractor. Since the argument works for any $j \in \Gamma$, we obtain that Γ is a minimal k-attractor. ◀

A naive solution for the minimality-checking problem is to test the k-attractor property on $\Gamma - \{i\}$ for every $i \in \Gamma$ using Theorem 18. This solution, however, runs in quadratic time. Our efficient strategy relies on colored range reporting and consists in checking, for every suffix tree edge e, if there is only one $j \in \Gamma$ marking it. In this case, we flag j as necessary. If, in the end, all attractor positions are flagged as necessary, then the attractor is minimal by Lemma 21. Notice that it can turn out that a single suffix tree edge can be marked by more than one necessary $j \in \Gamma$.

▶ **Theorem 22.** *Given a string $S \in [1..\sigma]^n$, a set $\Gamma \subseteq [1..n]$, and an integer $k \geq 1$, we can check whether Γ is a minimal k-attractor for S in $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n \log |\Gamma|)$ space.*

Proof. We associate to each element in Γ a distinct color from the set $\Sigma_\Gamma = \{c_i \mid i \in \Gamma\}$, and we build a two-dimensional Σ_Γ-colored grid $L \subseteq [1..n]^2 \times \Sigma_\Gamma$ (i.e., each point $\langle i, j\rangle$ in L is associated with a color from Σ_Γ) defined as $L = \{\langle i, D[i], c_{SA[i]+D[i]}\rangle,\ i = 1, ..., n\}$, that is, at coordinates $\langle i, D[i]\rangle$ we insert a point "colored" with the color associated to the attractor position immediately following – and possibly equal to – $SA[i]$. Then, for every suffix tree edge e we check that $L \cap [l_e..r_e] \times [0..\lambda(e)-1]$ contains at least two distinct colors. If there are at least two distinct colors $c_k \neq c_{k'}$ in the range $[l_e..r_e] \times [0..\lambda(e) - 1]$, then we do not mark k and k' as necessary (note that they could be marked later by some other edge, though). However, even if there is only one color c_k in the range this may not be enough to mark k as necessary. The reason for this is that in array D we are tracking only the attractor position i' *immediately following* each text position i; it could well be that the attractor position $i'' > i'$ immediately following i' marks e, but we miss it because we track only i'. This problem can be easily solved inserting in L also a point corresponding to the *second* nearest attractor position following every text position (so the number of points only doubles). It is easy to see that this is sufficient to solve our problem, since we only aim at enumerating at most two distinct colors in a range.

At this point, we have reduced the problem to the so-called *three-sided colored orthogonal range reporting* problem in two dimensions: report the distinct colors inside a three-sided orthogonal range in a grid. For this problem, the fastest known data structure takes $\mathcal{O}(n \log n)$ space and answers queries in $\mathcal{O}(\log^2 n + i)$ time, where n is the number of points in the grid and i is the number of returned points [12]. This would result in an overall running time of $\mathcal{O}(n \log^2 n)$ for our algorithm. We note that our problem is, however, simpler than the general one. In our case, it is enough to list *two* distinct colors (if any); we are not interested in counting the total number of such colors or reporting an arbitrary number of them.

Our solution relies on wavelet trees [10]. First, we pre-process the set of $v \le 2n$ points so that they fit in a grid $[1..v] \times [1..v]$ such that every row and every column contain exactly one point. Mapping the original query on this grid can be easily done in constant time using well-established rank reduction techniques that we do not discuss here (see, e.g. [22]). We can view this grid as an integer vector $V \in [1..v]^v$, where each $V[i]$ is associated with a color $V[i].c \in \Sigma_\Gamma$. We build a wavelet tree $WT(V)$ on V. Let u_s denote the internal node of $WT(V)$ reached following the path $s \in \{0,1\}^*$. With V_s we denote the subsequence of V associated with u_s, i.e., the subsequence of V such that the binary representation of each $V_s[i]$ is prefixed by s. For each internal node u_s we store (i) the sequence of colors $C_s = V_s[1].c, \dots, V_s[|V_s|].c$, and (ii) a bitvector $B_s[1..|V_s|]$ such that $B_s[1] = 0$ and $B_s[i] \ne B_s[i-1]$ iff $C_s[i] \ne C_s[i-1]$. We pre-process each B_s for constant-time rank and select queries [13, 5]. Overall, our data structure takes $\mathcal{O}(n \log \Gamma)$ words of space (that is, $\mathcal{O}(n \log \Gamma \log n)$ bits: at each of the $\log n$ levels of the wavelet tree we store $v \le 2n$ colors).

To report two distinct colors in the range $[l, r] \times [l', r']$, we find in $\mathcal{O}(\log v)$ time the $\mathcal{O}(\log v)$ nodes of $WT(V)$ covering $[l', r']$ as usually done when solving orthogonal range queries on wavelet trees (see [21] for full details). For each such node u_s, let the range $V_s[l_s, r_s]$ contain the elements in common between V_s and $V[l..r]$ (i.e., the range obtained mapping $[l..r]$ on V_s). We check whether in $B_s[l_s..r_s]$ there are two distinct bits at adjacent positions. If this is the case, we locate their positions $B_s[i_0] = 0$ and $B_s[i_1] = 1$, with $l_s \le i_0 = i_1 - 1 \le r_s$ (the case $i_1 = i_0 - 1$ is symmetric), and return the colors $C_s[i_0]$ and $C_s[i_1]$. By construction of B_s, $C_s[i_0] \ne C_s[i_1]$ and therefore we are done. Note that i_0 and i_1 can be easily found in constant time using rank/select queries on B_s.

If, on the other hand, all sequences $B_s[l_s..r_s]$ are unary, then we just need to retrieve the $\mathcal{O}(\log v)$ colors $C_s[l_s]$, for all the u_s covering the interval $[l', r']$, and check if any two of them are distinct (e.g. radix-sort them in linear time). Also this step runs in $\mathcal{O}(\log v)$ time.

Overall, our solution uses $\mathcal{O}(n \log |\Gamma|)$ space and runs in $\mathcal{O}(n \log v) = \mathcal{O}(n \log n)$ time. ◀

3.3 Computing a minimum k-attractor

Computing a minimum k-attractor is NP-hard for $k \ge 3$ and large σ. In this section we show that the problem is actually polynomial-time-solvable for small k and σ. Our algorithm takes advantage of both our reduction to set-cover and the verification algorithm of Theorem 18.

First, we give an upper-bound to the cardinality of the set of all minimal k-attractors. This will speed up our procedure for finding a minimum k-attractor (which is, in particular, minimal). By Lemma 13, there are no more than $\exp\big(\mathcal{O}(\sigma^{2k})\big)$ k-attractors for S. With the following lemma, we give a better upper-bound to the number of *minimal* k-attractors.

▶ **Lemma 23.** *The number of minimal k-attractors is* $\exp\big(\mathcal{O}(\sigma^k \log \sigma^k)\big)$.

Proof. Let minimal(σ, k) denote the maximum number of minimal k-attractors on the alphabet $[1..\sigma]$ (independently of the string length n). Let Γ be a minimal k-attractor. By Lemma 21, for every $j \in \Gamma$ there is at least one edge $e \in \mathcal{U}$ marked by j only. Let

$\text{edge} : \Gamma \rightarrow \mathcal{U}$ be the function defined as follows: $\text{edge}(j) = e$ such that (i) e is marked by j only, and (ii) among all edges marked by j only, e is the one with the lexicographically smallest $s(e)$ (where, if $s(e')$ prefixes $s(e'')$, then we consider $s(e')$ smaller than $s(e'')$ in lexicographic order). Let $\mathcal{U}' = \text{edge}(\Gamma)$ be the image of Γ through edge function. By its definition, edge is a bijection between Γ and $\mathcal{U}'$. This implies that $|\Gamma| = |\mathcal{U}'| \leq |\mathcal{U}| \leq \sigma^k$: a minimal k-attractor is a set of cardinality at most σ^k chosen from a universe $\mathcal{C}$ of size at most $|\mathcal{C}| \leq \sigma^{2k-1} + 2k - 2 \leq \sigma^{2k}$, therefore: $\text{minimal}(\sigma, k) \leq \sum_{i=1}^{\sigma^k} \binom{\sigma^{2k}}{i}$. We now give an upper-bound to the function $f(N,t) = \sum_{i=1}^{t} \binom{N}{i}$, where we assume $t \geq 2$ for simplicity (the hypothesis holds in our case since $t = \sigma^k$). Then, we will plug our bound in the above inequality. Since $\binom{N}{i} < \frac{N^i}{i!}$, we have that

$$\begin{array}{rcl} f(N,t) & < & \sum_{i=1}^{t} \frac{N^i}{i!} \\ & = & \sum_{i=1}^{t} \left(\frac{N}{t}\right)^i \cdot \frac{t^i}{i!} \\ & \leq & \sum_{i=1}^{t} \left(\frac{N}{t}\right)^i \cdot \sum_{i=1}^{t} \frac{t^i}{i!} \\ & \in & \mathcal{O}\left(\left(\frac{N \cdot e}{t}\right)^t\right) \end{array}$$

We obtain our claim: $\text{minimal}(\sigma, k) \leq f(\sigma^{2k}, \sigma^k) \in \mathcal{O}(e^{\sigma^k} \cdot \sigma^{k\sigma^k}) \leq \exp\big(\mathcal{O}(\sigma^k \log \sigma^k)\big)$. ◀

Using the above lemma, we now provide a strategy to find a minimum k-attractor.

▶ **Theorem 24.** *A minimum k-attractor can be found in $\mathcal{O}(n) + \exp\big(\mathcal{O}(\sigma^k \log \sigma^k)\big)$ time.*

Proof. Let $c(i) = S'[i-k+1..i+k-1]$, where $S'[i] = \# \notin \Sigma$ if $i < 1$ or $i > n$ and $S'[i] = S[i]$ otherwise, be the context string associated to position i. Consider the string

$$C = c(i_1)\$c(i_2)\$\ldots\$c(i_t)$$

where $\{i_1, i_2, \ldots, i_t\} = \mathcal{C}$ and $\# \neq \$ \notin \Sigma$. By our choice of $\mathcal{C}$, the length of this string is $|C| = (|\mathcal{C}| \cdot 2k) - 1 \leq (\sigma^{2k-1} + 2k) \cdot 2k \in \mathcal{O}(\sigma^{2k} \cdot k) \leq \exp\big(\mathcal{O}(\log \sigma^k)\big)$. We can build C in $\mathcal{O}(n + |C|)$ time using the suffix tree of S (i.e., extracting all paths from the root to nodes at string depth at most $2k-1$).

Let now $\Gamma'' \subseteq \{k \cdot (2j+1) \mid j = 0, \ldots, t-1\}$. It is easy to see that $\Gamma' = \{i \mid C[i] = \$ \textit{ or } C[i] = \#\} \ \cup \Gamma''$ is a k-attractor for C if and only if the set $\Gamma = \{i_{(x-k)/(2k)} \mid x \in \Gamma''\}$ is a k-attractor for S. Suppose that Γ' is a k-attractor for C, and consider a substring s of S of length at most k. By construction of C, s is also a substring of C; in particular, there is an occurrence $C[i..i+|s|-1] = s$ crossing a position $k \cdot (2j+1) \in \Gamma''$, for some j. Then, $i_j \in \Gamma$ and, by the way we defined C, there is an occurrence of s crossing position i_j in S. Conversely, suppose that Γ is a k-attractor for S, and let s be a substring of C of length at most k. If s contains either \$ or #, then it must cross one of the positions in $\{i \mid C[i] = \$ \textit{ or } C[i] = \#\} \ \subseteq \Gamma'$. Otherwise, it appears inside one of the substrings $c(i_k)$, for some $k \in [1..t]$. But then, this means that s appears in S and, in particular, that it has some occurrence $s' = s$ crossing a position $i_j \in \Gamma$. From the way we constructed C, s has an occurrence in C crossing position $k \cdot (2j+1) \in \Gamma'' \subseteq \Gamma'$.

At this point, we check whether Γ' is a k-attractor for C for all possible Γ'', and return the smallest such set. Instead of trying all subsets of $\mathcal{C}$, we use Lemma 23 and generate only subsets of $\mathcal{C}$ of size at most σ^k; these subsets will include all minimal k-attractors and, in particular, all minimum k-attractors. By Lemma 23, there are at most $\exp\big(\mathcal{O}(\sigma^k \log \sigma^k)\big)$ such sets, and each verification takes linear $\mathcal{O}(|C|) = \exp\big(\mathcal{O}(\log \sigma^k)\big)$ time using Theorem 18. Overall, our algorithm for the minimum k-attractor runs in $\mathcal{O}(n) + \exp\big(\mathcal{O}(\log \sigma^k)\big) \cdot \exp\big(\mathcal{O}(\sigma^k \log \sigma^k)\big)$ time, which simplifies to $\mathcal{O}(n) + \exp\big(\mathcal{O}(\sigma^k \log \sigma^k)\big)$. ◀

3.4 Sharp attractors

The complexity of k-attractor has been fully characterized in [16], except for the particular case $k = 2$: for $k \geq 3$, the problem has been proven to be NP-complete, while for $k = 1$ it is clearly solvable in linear time. While we have not been able to settle the case $k = 2$, we show a polynomial-time solution under the additional constraint that only substrings of length *exactly* equal to 2 are captured by the attractor set. We denote with the name *k-sharp attractor* this variant. Formally, we define a *k-sharp* attractor of a string $S \in \Sigma^n$ to be a set of positions $\Gamma \subseteq [1..n]$ such that every substring $S[i..j]$ with $j-i+1 = k$ has an occurrence $S[i'..j'] = S[i..j]$ with $j'' \in [i'..j']$ for some $j'' \in \Gamma$. In other words, a k-sharp-attractor is a subset that covers all substrings of length exactly k. By MINIMUM-k-SHARP-ATTRACTOR we denote the optimization problem of finding the smallest k-sharp attractor of a given input string, and by k-SHARP-ATTRACTOR $= \{\langle T, p\rangle \mid$ String T has a k-sharp-attractor of size $\leq p\}$ we denote the corresponding decision problem. In the full version [15] of this paper we show that, for any constant $k \geq 3$, k-SHARP-ATTRACTOR is NP-complete. The proof is an adaptation of the original one proposed for k-attractors in [16], and is based on a reduction from set cover. Here we show that for $k = 2$ the problem can be solved in polynomial time (again, the case $k = 1$ is trivially solvable in linear time).

▶ **Theorem 25.** *Minimum 2-sharp-attractor is in P.*

Proof. It is easy to show that 2-sharp-attractor is in P by a reduction to edge cover. Given a string S, let $V \subseteq \Sigma^2$ be the set of strings of length 2 that occur at least once in S. For every substring of length 3 of the form xyz, add the edge (xy, yz) to the edge-set E, and add self-loops for the first and last pair.

A position $\gamma \in \Gamma$ thus corresponds to an edge, e_γ, and it is easy to see that Γ is a 2-sharp-attractor if and only if $\{e_\gamma | \gamma \in \Gamma\}$ is an edge cover.

The number of vertices and edges in this graph are both $\leq n$, so a minimum edge cover can be found in $\mathcal{O}(n\sqrt{n})$ time [19]. ◀

References

1 Djamal Belazzougui. Linear time construction of compressed text indices in compact space. In *Annual Symposium on Theory of Computing (STOC)*, pages 148–193. ACM, 2014.

2 Djamal Belazzougui, Fabio Cunial, Juha Kärkkäinen, and Veli Mäkinen. Linear-time string indexing and analysis in small space. *arXiv preprint 1609.06378*, 2016.

3 Anselm Blumer, Janet Blumer, David Haussler, Ross McConnell, and Andrzej Ehrenfeucht. Complete inverted files for efficient text retrieval and analysis. *Journal of the ACM*, 34(3):578–595, 1987.

4 Michael Burrows and David J. Wheeler. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994.

5 David Clark. *Compact Pat trees.* PhD thesis, University of Waterloo, 1998.

6 Maxime Crochemore and Renaud Vérin. Direct construction of compact directed acyclic word graphs. In *Combinatorial Pattern Matching (CPM)*, pages 116–129. Springer, 1997.

7 Martin Farach. Optimal suffix tree construction with large alphabets. In *Annual Symposium on Foundations of Computer Science (FOCS)*, pages 137–143. IEEE, 1997.

8 Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *Journal of the ACM*, 52(4):552–581, 2005.

9 Johannes Fischer and Volker Heun. Space-efficient preprocessing schemes for range minimum queries on static arrays. *SIAM Journal on Computing*, 40(2):465–492, 2011.

10 Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *Annual Symposium on Discrete Algorithms (SODA)*, pages 841–850. SIAM, 2003.

11 Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM Journal on Computing*, 35(2):378–407, 2005.

12 Prosenjit Gupta, Ravi Janardan, and Michiel Smid. Further results on generalized intersection searching problems: counting, reporting, and dynamization. *Journal of Algorithms*, 19(2):282–317, 1995.

13 Guy Joseph Jacobson. *Succinct static data structures.* PhD thesis, Carnegie Mellon University, 1988.

14 Juha Kärkkäinen, Peter Sanders, and Stefan Burkhardt. Linear work suffix array construction. *Journal of the ACM*, 53(6):918–936, 2006.

15 Dominik Kempa, Alberto Policriti, Nicola Prezza, and Eva Rotenberg. String attractors: Verification and optimization. *arXiv*, 2018. arXiv:1803.01695.

16 Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: String attractors. In *Annual Symposium on Theory of Computing (STOC)*, pages 827–840. ACM, 2018.

17 John C. Kieffer and En-Hui Yang. Grammar-based codes: A new class of universal lossless source codes. *IEEE Transactions on Information Theory*, 46(3):737–754, 2000.

18 Udi Manber and Gene Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.

19 Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs. In *Annual Symposium on Foundations of Computer Science (SFCS)*, pages 17–27. IEEE, 1980.

20 J. Ian Munro, Gonzalo Navarro, and Yakov Nekrich. Space-efficient construction of compressed indexes in deterministic linear time. In *Annual Symposium on Discrete Algorithms (SODA)*, pages 408–424. SIAM, 2017.

21 Gonzalo Navarro. Wavelet trees for all. *Journal of Discrete Algorithms*, 25:2–20, 2014.

22 Gonzalo Navarro. *Compact data structures: A practical approach.* Cambridge University Press, 2016.

23 James A. Storer and Thomas G. Szymanski. Data compression via textual substitution. *Journal of the ACM*, 29(4):928–951, 1982.

24 Peter Weiner. Linear pattern matching algorithms. In *Annual Symposium on Switching and Automata Theory (SWAT/FOCS)*, pages 1–11. IEEE, 1973.