Czech Technical University in Prague
Faculty of Information Technology
Department of Theoretical Computer Science

Searching Regularities in Strings using Finite Automata

by

Ondřej Guth

A dissertation thesis submitted to
the Faculty of Information Technology, Czech Technical University in Prague,
in partial fulfilment of the requirements for the degree of Doctor.

Dissertation degree study programme: Informatics

Prague, February 2014

Supervisor:
Bořivoj Melichar
Department of Theoretical Computer Science
Faculty of Information Technology
Czech Technical University in Prague
Thákurova 9
160 00 Prague 6
Czech Republic

Abstract

This dissertation thesis deals with particular types of string regularities, covers and seeds. We say that factor (possibly approximate) of a string is its cover if the string may be constructed just using superpositions of the cover. Seed of a string is a generalization of cover such that seed is a cover of an extension of that string. In this dissertation thesis we focus on algorithms for computation of all approximate covers or seeds of a string. As a measure of approximation, the Hamming distance is used. The main contributions of this thesis are algorithms for computing all approximate covers or seeds of a string based on formalism of finite automata, therefore they are not hard to understand, similar to each other (they solve similar problems), easy to implement (there are no complex data structures) and it is straightforward to extend them to similar problems.

Keywords:

cover, seed, Hamming distance, string regularity

Acknowledgements

This work could not have been completed without support of a number of people. First of all, I would like to thank my thesis supervisor, Bořivoj Melichar – for his kindness, support and patience not only with reading my papers repeatedly numerous times. He has given me a lot of helpful advice. His enthusiasm for idea of using finite automata as unifying basic theory for algorithms in stringology and his way of presenting and explaining the automata-based principles as simply understandable, those have been a big inspiration for me. It has been an honour for me to be his Ph.D. student.

I would also like to thank my colleague, Jan Holub, for his encouragement during my work and also for helping me with funding as a head of our research group. I shall also acknowledge Michal Voráček for his advice in the very beginning of my research. I want to mention Miroslav Balík for contribution to my paper (and to Chapter 4).

I am grateful to my friends and people who have been close to me and expressing me their support when I have needed it.

And finally, my greatest thanks to my parents for their support and care.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

Cover, and its generalization, seed, are kinds of repetitive structure of a string and examples of typical regularities of strings. Searching regularities in strings is an important problem in many areas of science. In molecular biology, repetitive elements in chromosomes determine the likelihood of certain diseases. In computer science, repetitive elements in strings are important in data compression, speech recognition, coding etc.

String regularities may be seen as a description of repetitive structure of a string. One example is period. Every string $\boldsymbol{w}$ may be written as $\boldsymbol{w} = \boldsymbol{p}^i\boldsymbol{s}$, where $\boldsymbol{s}$ is a prefix of $\boldsymbol{p}$ and length of $\boldsymbol{p}$ is called period (so-called normal form of a string [45]). Although period describes structure of a string in an interesting way, for many strings, $|\boldsymbol{p}| = |\boldsymbol{w}|$ (i.e. the regularity is too strict for such a string). Therefore, the notion of quasiperiod (so-called cover, can be seen as a generalization of period) and even less strict notion, seed, have been introduced.

Searching regularities have been intensively studied for many years and a lot of algorithms have been proposed. Many of the algorithms are not similar to any other, though they solve similar problems. It is also difficult to understand many of them. Moreover, many of the algorithms are difficult to extend, they solve only one specific problem.

In this work, the notion of finite automata is used to solve covers- and seeds-related problems. It is motivated by intuitive and elegant solutions to various problems in stringology. The algorithms presented here directly follow from basic principles and properties of covers and seeds that are modelled using the formalism of finite automata. Therefore, the algorithms are similar to each other, straightforward, and simple to understand, to implement and to extend.

1.2 Problem statement

In this dissertation thesis, problems related to computation of all covers or seeds, exact or approximate, are considered. For approximation, Hamming distance is used. When com-

puting k-approximate regularities, the fixed maximal number of errors given by constant k for the whole string is allowed.

Three notions concerning covers are studied and three algorithms are proposed. The algorithms for computing exact and restricted approximate covers with Hamming distance are alternatives to already existing ones. The algorithm for computing approximate covers with Hamming distance is the first known solution of this problem.

An *exact cover* $\boldsymbol{u}$ of string $\boldsymbol{w}$ is a factor of $\boldsymbol{w}$ such that every position of $\boldsymbol{w}$ lies within some occurrence of $\boldsymbol{u}$ in $\boldsymbol{w}$. A *restricted approximate cover* $\boldsymbol{u}$ of string $\boldsymbol{w}$ is a factor of $\boldsymbol{w}$ such that every position of $\boldsymbol{w}$ lies within some approximate occurrence of $\boldsymbol{u}$ in $\boldsymbol{w}$. An *approximate cover* $\boldsymbol{u}$ of string $\boldsymbol{w}$ is a string such that every position of $\boldsymbol{w}$ lies within some approximate occurrence of $\boldsymbol{u}$ in $\boldsymbol{w}$.

Similarly, notions concerning seeds are studied.

An *exact seed* $\boldsymbol{v}$ of string $\boldsymbol{w}$ is a factor of $\boldsymbol{w}$ and $\boldsymbol{v}$ covers some superstring of $\boldsymbol{w}$ and $|\boldsymbol{v}| \leq |\boldsymbol{w}|$. An *approximate seed* $\boldsymbol{v}$ of string $\boldsymbol{w}$ is an approximate cover of some superstring of $\boldsymbol{w}$. A *restricted approximate seed* $\boldsymbol{v}$ of string $\boldsymbol{w}$ is an approximate seed of $\boldsymbol{w}$ a factor of $\boldsymbol{w}$. A *left seed* of string $\boldsymbol{w}$ is a seed of $\boldsymbol{w}$ and also a prefix of $\boldsymbol{w}$.

In this dissertation thesis, the problems of computing *all covers* or *seeds* of string, *all restricted smallest distance approximate covers* or *seeds* of string, and *all smallest distance approximate covers* or *seeds* of string are studied and algorithms for solving the problems are proposed.

The *smallest distance* means that for every cover or seed found, its real (i.e. the smallest) distance is computed with respect to selected distance function (e.g., the Hamming distance).

1.3 Related work and previous results

1.3.1 Approaches not using finite automata

The idea of quasiperiodic string (i.e. coverable by its proper factor) was introduced by Apostolico and Ehrenfeucht in 1990 [3], they also presented an $\mathcal{O}(n \log^2 n)$-time algorithm for computing *all maximal quasiperiodic factors* of a string of length n using suffix trees, later revisited [4]. A loglinear-time ($\mathcal{O}(n \cdot \log n)$) algorithm, based on Crochemore's partitioning technique [18], was given by Iliopoulos and Mouchard [31]. Another loglinear-time algorithm for the same problem, utilizing suffix trees, was later introduced by Brodal and Pedersen [11].

Apostolico, Farach, and Iliopoulos introduced a linear-time algorithm to find the *minimum* cover of a string in 1991 [5]. A linear-time algorithm solving the same problem for all prefixes (so-called cover array) of given string was presented by Breslauer [9]. A parallel algorithm for the same problem was introduced by Breslauer [10].

In 1994 Moore and Smyth [39, 40, 41] gave a linear-time algorithm for computing all covers of a given string. An algorithm for parallel computation of all covers of a string was introduced by Iliopoulos and Park [32]. Algorithms for searching all periods or covers of

a given string or a circular string with don't care symbols was introduced by Iliopoulos, Mohamed, Mouchard, Perdikuri, Smyth, and Tsakalidis [29].

A linear-time algorithm for computing maximum cover array of a given string (i.e. lengths of maximum cover of every prefix of given string and in fact lengths of all covers of every prefix of that string) was introduced by Li and Smyth [36].

In 2002 Sim, Park, Kim, and Lee [44] introduced searching approximate covers (however, the paper is in Korean). Christodoulakis, Iliopoulos, Park, and Sim [14] implemented the algorithm and showed its practical time complexity for Hamming, Levenshtein, and weighted Levenshtein distance for computation the distance of given cover of given string and for solution of the restricted smallest distance approximate cover.

The notion of seeds was introduced by Iliopoulos, Moore, and Park in 1993 [30] (it was corrected many years later [17]), where an algorithm for searching all seeds of a given string is presented. This algorithm works in $\mathcal{O}(n \log n)$ time for given string of length n. This is achieved by reporting only groups of seeds, instead of each found seed (number of seeds is $\mathcal{O}(n^2)$). In 2012 Kociumaka, Kubica, Rodoszewksi, Rytter and Waleń [34] introduced a linear-time algorithm to compute all seeds of a string. A parallel approach to searching seeds was introduced by Berkman, Iliopoulos, and Park [6].

In 2011 Christou et al. [17] presented an algorithm for computing the shortest seed array of a string executing in $\mathcal{O}(n^2)$ time for string of length n and they also introduced linear-time algorithms for computing the shortest- and the longest-left-seed array of a string. In 2011 Christou, Crochemore, Guth, Iliopoulos and Pissis [15, 16] introduced a loglinear-time algorithm that computes the shortest-right-seed array of a string and a linear-time algorithm for computing the longest-right-seed array of a string.

Searching approximate seeds was solved in 2003 by Christodoulakis, Iliopoulos, Park, and Sim [12, 13] using dynamic programming. They consider Hamming, Levenshtein, and weighted Levenshtein distance and the computing the distance of given seed of given string and the problem of the restricted smallest distance approximate seed. Moreover, they proved that general problem of the smallest distance approximate seed is NP-complete.

1.3.2 Finite automata approaches

Aho–Corasick automaton [1] was used by Antoniou, Crochemore, Iliopoulos, Jayasekera, and Landau [2] to search repetitions, covers, and seeds of constant-limited length in a generalized string (so-called degenerate or indeterminate) with constant-limited number of generalized symbols.

Deterministic factor automaton was firstly studied by Blumer, Ehrenfeucht, Haussler, and McConnell [7]. It was proved that the deterministic factor and suffix automaton has linear-state complexity and a linear-time algorithm for its construction directly from input string was given by Blumer, Haussler, Ehrenfeucht, Chen, and Seiferas [8]. An algorithm for constructing deterministic factor automaton and deterministic suffix automaton directly from input string was also given by Crochemore [19]. An algorithm for direct construction of a deterministic suffix automaton for maximum Hamming distance k was introduced by

Crochemore, Epifanio, Gabriele, and Mignosi [20]. There is an attempt for a proof of state-complexity of the automaton.

An algorithm using construction of a nondeterministic suffix or factor automaton with ε-transitions and subsequent subset construction of deterministic one was introduced by Melichar [37]. An extension of this algorithm to construction of approximate suffix or factor automaton was presented by Melichar, Holub, and Polcar [38]. In contrast to this approach, the algorithm presented in [20] is based on construction of equivalence classes that are not directly connected with end sets and approximation.

An algorithm for searching all repetitions, borders, and covers in generalized strings that is based on analysis of d-subsets of deterministic suffix automaton was introduced by Voráček and Melichar [47]. The analysis of d-subsets of deterministic suffix automaton was by Voráček and Melichar [51] used also for computing seeds in generalized strings. These algorithms and the one for computing cover array were correctly defined and their time complexity was derived by Voráček [48], they were implemented and further improved by Flouri [25].

1.4 Goals of the dissertation thesis

The goal of this dissertation thesis is to improve the up-to-date knowledge of computing particular types of string regularities, covers and seeds, especially their approximate version. The goals are summarized as follows:

- Improve existing or introduce new algorithms for problems of computing covers and seeds.
- Base the algorithms on formalism of finite automata as a unifying "framework" that makes it easier to understand and implement them and also to extend them to solve similar problems. It is to be shown that as for many problems in stringology, searching all covers and seeds, especially the approximate one, is solvable using finite automata.

1.5 Structure of the dissertation thesis

This thesis consists of 6 chapters. Apart from this introductory chapter, Chapter 2 contains preliminaries (mathematical and notational) necessary for reading the rest of this thesis. This chapter may be skipped and referred back as needed. In Chapter 3, theoretical background is presented, we summarize there already-known properties of covers and seeds, distance metrics, and exact and approximate indexing of strings using finite automata. This chapter does not contain any new results, however, many of properties, lemmas and algorithms presented there are necessary for algorithms and proofs of main results given in this thesis. Chapter 4 presents new algorithms for problems of computing all covers – exact and approximate with Hamming distance. It starts with properties of covers needed for the algorithms, then the algorithms are described, their correctness and time and space

complexities are proved and finally, experimental results are presented. The solution of problem of searching all restricted k-approximate covers has been published in [A.1] and together with algorithm for searching all k-approximate covers has been presented in [A.4]. Chapter 5 contains new algorithms for problems of computing all seeds with Hamming distance; structure of this chapter is similar to one of Chapter 4. The solution of problem of searching all restricted k-approximate seeds has been published in [A.2]. Finally, in Chapter 6, an overview of contribution of this thesis is given, and some open problems and directions for future research is presented.

Chapter 2

Preliminaries

In this chapter, mathematical and notational preliminaries are given. All the notions used through this thesis are defined along with the problems being solved. In the end, list of commonly used abbreviations is given.

2.1 Notations

We adopt the following general naming conventions:

- A for an alphabet,
- a, b, c for alphabet symbols,
- $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ for strings,
- $\boldsymbol{p}$ for a prefix of a string,
- $\boldsymbol{s}$ for a suffix of a string,
- ε for empty string,
- $h, i, j, l, m, n, \lambda$ for integer variables,
- k for maximum allowed (considered) distance,
- D for distance function,
- H for Hamming distance,
- q for a state,
- q_0 for initial state,
- e for a state of a nondeterministic finite automaton and for an element of a d-subset,

- Q for set of states,
- F for set of final states,
- δ for automaton transition function,
- $\mathcal{M}$ for a finite automaton,
- L for a language,
- B for arbitrary set,
- C for arbitrary list.

2.2 Basic definitions

Definition 2.2.1 (Alphabet)**.** An *alphabet* A is a finite nonempty set of symbols.

Notation 2.2.2 (Cardinality). *Cardinality* (also called size) of a set B is denoted by $|B|$.

Notation 2.2.3 (Powerset). For any set B, the set of all subsets of B is denoted by $\mathcal{P}(B)$. The finite set of all subsets of B is denoted by $\mathcal{P}_F(B)$.

2.3 Strings

Definition 2.3.1 (String)**.** A *string* over an alphabet A is a finite sequence of symbols of A.

Notation 2.3.2. The i-th symbol a of string $\boldsymbol{w} = a_1 a_2 \dots a_{\boldsymbol{w}}$ is denoted either by subscript as a_i or as $\boldsymbol{w}[i]$, i.e. string $\boldsymbol{w}$ may be written as $\boldsymbol{w}[1]\boldsymbol{w}[2]\dots\boldsymbol{w}[|\boldsymbol{w}|]$.

Definition 2.3.3 (Circular string)**.** A string $\boldsymbol{w}$ is said to be a *circular string* if it is formed by concatenating $\boldsymbol{w}[1]$ to the right of $\boldsymbol{w}[|\boldsymbol{w}|]$.

Definition 2.3.4 (String with don't care symbols)**.** Symbol $\circ \notin A$ is a *don't care symbol* if it matches any symbol of some alphabet A. Then string $\boldsymbol{w}$ is said to be a *string with don't care symbols* over alphabet A if $\boldsymbol{w} \in (A \cup \{\circ\})^*$.

Definition 2.3.5 (Generalized symbol)**.** Assuming an alphabet A, a *generalized symbol* $\tilde{a} \notin A$ is a symbol equal to a subset of symbols of A, it may be substituted by any symbol from the subset.

Definition 2.3.6 (Generalized string)**.** A *generalized string* $\tilde{\boldsymbol{w}}$ may contain one or more generalized symbols.

Note 2.3.7. Generalized symbol or string, is also called inteterminate or degenerate.

Definition 2.3.8 (Length of string)**.** A *length of string* $\boldsymbol{w}$ is the number of its symbols and is denoted by $|\boldsymbol{w}|$.

Definition 2.3.9 (Reversed string)**.** Having string $\boldsymbol{w} = a_1 a_2 \dots a_{|\boldsymbol{w}|}$, the *reversed string* is $a_{|\boldsymbol{w}|} a_{|\boldsymbol{w}|-1} \dots a_1$ and it is denoted by $\boldsymbol{w}^{\mathrm{R}}$.

Definition 2.3.10 (Empty string)**.** An *empty string* ε is a string containing no symbol and its length is 0.

Notation 2.3.11 (Set of all strings). A *set of all strings* over alphabet A is denoted by A^*.

Notation 2.3.12 (Set of all nonempty strings). A *set of all nonempty strings* over alphabet A is denoted by A^+.

Definition 2.3.13 (Effective alphabet)**.** An *effective alphabet* of a string $\boldsymbol{w}$ is denoted by A_w and it is the minimum set of symbols such that $\boldsymbol{w} \in A_w^*$ holds.

Definition 2.3.14 (Concatenation of strings)**.** A *concatenation* of two strings $\boldsymbol{v}, \boldsymbol{w} \in A^*$ is string obtained by writing sequentially symbols of $\boldsymbol{v}$ and then symbols of $\boldsymbol{w}$. Is is denoted by $\boldsymbol{vw}$ or $\boldsymbol{v}.\boldsymbol{w}$.

Definition 2.3.15 (Iteration of string)**.** An *iteration* of string $\boldsymbol{w} \in A^*$ is denoted as $\boldsymbol{w}^i$ and it is defined in this way: $\boldsymbol{w}^0 = \varepsilon, \boldsymbol{w}^1 = \boldsymbol{w}, \boldsymbol{w}^i = \boldsymbol{w}^{i-1}\boldsymbol{w}$.

Definition 2.3.16 (Superposition of strings)**.** A *superposition* of two strings $\boldsymbol{v}, \boldsymbol{w} \in A^*, \boldsymbol{v} = \boldsymbol{pu}, \boldsymbol{w} = \boldsymbol{us}$ is any string that may be written as $\boldsymbol{pus}$. Strings $\boldsymbol{v}$ and $\boldsymbol{w}$ have an *overlap* in string $\boldsymbol{pus}$.

Note 2.3.17. When string $\boldsymbol{u}$ from the definition of superposition is empty, the operation is equivalent to concatenation.

Definition 2.3.18 (Language)**.** A *language* $L \subseteq A^*$ is a set of strings over alphabet A.

Definition 2.3.19 (Symbol size of language)**.** *Symbol size of a language* L is denoted by $\|L\|$ and it is equal to $\sum_{\boldsymbol{w} \in L} |\boldsymbol{w}|$.

Definition 2.3.20 (Editing operation)**.** Assuming strings $\boldsymbol{u}, \boldsymbol{v} \in A^*$ such that $\boldsymbol{u}$ is not equal to $\boldsymbol{v}$, string $\boldsymbol{u}$ may be converted to string $\boldsymbol{v}$ using one or more *editing operations*. Every editing operation is atomic and it may be applied to empty string (in case of operation insert of single symbol), to one symbol, or to a pair of subsequent symbols (operation transpose of two symbols).

Definition 2.3.21 (Editing operations)**.** Particular editing operations are for strings $\boldsymbol{u} \in A^*; \boldsymbol{v}, \boldsymbol{w} \in A^+; |\boldsymbol{v}| = |\boldsymbol{w}|; |\boldsymbol{u}| = |\boldsymbol{w}| - 1$ defined as follows:

replace string $\boldsymbol{v}$ is converted to string $\boldsymbol{w}$ using single operation *replace* if

- $\boldsymbol{v}[j] \neq \boldsymbol{w}[j]$ for some (one) $1 \leq j \leq |\boldsymbol{w}|$, and

- $\forall i, 1 \leq i \leq |\boldsymbol{w}|, i \neq j : \boldsymbol{v}[i] = \boldsymbol{w}[i]$;

insert string $\boldsymbol{u}$ is converted to string $\boldsymbol{w}$ using single operation *insert* if

- $\boldsymbol{w}[j] \in A$ for some (one) $1 \leq j \leq |\boldsymbol{w}|$, and
- $\forall i, 1 \leq i < j, \boldsymbol{u}[i] = \boldsymbol{w}[i]$, and
- $\forall i, j < i \leq |\boldsymbol{u}| : \boldsymbol{u}[i] = \boldsymbol{w}[i+1]$;

delete string $\boldsymbol{w}$ is converted to string $\boldsymbol{u}$ using single operation *delete* if

- $\boldsymbol{w}[j] \in A$ for some (one) $1 \leq j \leq |\boldsymbol{w}|$, and
- $\forall i, 1 \leq i < j, \boldsymbol{u}[i] = \boldsymbol{w}[i]$, and
- $\forall i, j < i \leq |\boldsymbol{u}| : \boldsymbol{u}[i] = \boldsymbol{w}[i+1]$;

transpose string $\boldsymbol{v}$ is converted to string $\boldsymbol{w}$ using single operation *transpose* if

- $\boldsymbol{v}[j] = \boldsymbol{w}[j+1], \boldsymbol{v}[j+1] = \boldsymbol{w}[j]$ for some (one) $1 \leq j < |\boldsymbol{w}|$, and
- $\forall i, 1 \leq i \leq |\boldsymbol{w}|, i \neq j, i \neq j+1 : \boldsymbol{v}[i] = \boldsymbol{w}[i]$.

Definition 2.3.22 (Distance)**.** *Distance* $D(\boldsymbol{x}, \boldsymbol{y})$ between two strings $\boldsymbol{x}, \boldsymbol{y} \in A^*$ is the minimum number of editing operations that are necessary to convert one string $\boldsymbol{x}$ to the other string $\boldsymbol{y}$. Its value is determined by a *distance function*.

Definition 2.3.23 (Hamming distance)**.** The *Hamming distance* between strings $\boldsymbol{x}$ and $\boldsymbol{y}$ is equal to the minimum number of editing operations *replace* that are necessary to convert $\boldsymbol{x}$ into $\boldsymbol{y}$. The Hamming distance function is denoted by H.

Definition 2.3.24 (Levenshtein distance)**.** The *Levenshtein distance* between strings $\boldsymbol{x}$ and $\boldsymbol{y}$ is equal to the minimum number of editing operations *replace*, *insert*, and *delete* that are necessary to convert $\boldsymbol{x}$ into $\boldsymbol{y}$. The Levenshtein distance function is denoted by D_L.

Note 2.3.25. The Levenshtein distance is also known as *edit distace*.

Definition 2.3.26 (Damerau distance)**.** The *Damerau distance* between strings $\boldsymbol{x}$ and $\boldsymbol{y}$ is equal to the minimum number of editing operations *replace*, *insert*, *delete*, and *transpose* (of two subsequent symbols) that are necessary to convert $\boldsymbol{x}$ into $\boldsymbol{y}$. The Damerau distance is denoted by D_G.

Note 2.3.27. The Damerau distance is also known as the Levenshtein–Damerau or generalized Levenshtein distance.

Definition 2.3.28 (Prefix)**.** Assuming strings $\boldsymbol{p}, \boldsymbol{u}, \boldsymbol{w} \in A^*$, string $\boldsymbol{p}$ is a *prefix* of string $\boldsymbol{w}$ if $\boldsymbol{w}$ may be written as $\boldsymbol{pu}$.

Definition 2.3.29 (k-approximate prefix)**.** Assuming strings $\boldsymbol{p}, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in A^*$ and integer $k \geq 0$, string $\boldsymbol{v}$ is a *k-approximate prefix* of string $\boldsymbol{w}$ under some distance function D if $\boldsymbol{w}$ may be written as $\boldsymbol{pu}$ and $D(\boldsymbol{p}, \boldsymbol{v}) \leq k$.

Notation 2.3.30 (Set of all k-approximate prefixes). *Set of all k-approximate prefixes* of string $\boldsymbol{w}$ under some distance function D is denoted by $\mathsf{pref}_D^k(\boldsymbol{w})$.

Definition 2.3.31 (Suffix)**.** Assuming strings $\boldsymbol{s}, \boldsymbol{u}, \boldsymbol{w} \in A^*$, string $\boldsymbol{s}$ is a *suffix* of string $\boldsymbol{w}$ if $\boldsymbol{w}$ may be written as $\boldsymbol{us}$.

Definition 2.3.32 (k-approximate suffix)**.** Assuming strings $\boldsymbol{s}, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in A^*$ and integer $k \geq 0$, string $\boldsymbol{v}$ is a *k-approximate suffix* of string $\boldsymbol{w}$ under some distance function D if $\boldsymbol{w}$ may be written as $\boldsymbol{us}$ and $D(\boldsymbol{s}, \boldsymbol{v}) \leq k$.

Notation 2.3.33 (Set of all k-approximate suffixes). *Set of all k-approximate suffixes* of string $\boldsymbol{w}$ under some distance function D is denoted by $\mathsf{suff}_D^k(\boldsymbol{w})$.

Definition 2.3.34 (Factor)**.** Assuming strings $\boldsymbol{p}, \boldsymbol{s}, \boldsymbol{u}, \boldsymbol{w} \in A^*$, string $\boldsymbol{u}$ is a *factor* of string $\boldsymbol{w}$ if $\boldsymbol{w}$ may be written as $\boldsymbol{pus}$.

Note 2.3.35. Factor is also known as *substring*.

Notation 2.3.36. Factor $\boldsymbol{u}$ of string $\boldsymbol{w}$ may be also written as $\boldsymbol{u} = \boldsymbol{w}[i..j]$ if $j - i + 1 = |\boldsymbol{u}|$ and for all $l = 1, \ldots, |\boldsymbol{u}|$: $\boldsymbol{u}[l] = \boldsymbol{w}[i + l - 1]$.

Definition 2.3.37 (k-approximate factor of string)**.** Assuming strings $\boldsymbol{p}, \boldsymbol{s}, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in A^*$ and integer $k \geq 0$, string $\boldsymbol{v}$ is a *k-approximate factor* of string $\boldsymbol{w}$ under some distance function D if $\boldsymbol{w}$ may be written as $\boldsymbol{pus}$ and $D(\boldsymbol{u}, \boldsymbol{v}) \leq k$.

Notation 2.3.38 (Set of all k-approximate factors). *Set of all k-approximate factors* of string $\boldsymbol{w}$ under some distance function D is denoted by $\mathsf{fact}_D^k(\boldsymbol{w})$.

Definition 2.3.39 (Extension)**.** Assuming strings $\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y} \in A^*$, string $\boldsymbol{u} = \boldsymbol{xwy}$ is called an *extension* of $\boldsymbol{w}$, string $\boldsymbol{xw}$ is a *left extension* of $\boldsymbol{w}$, string $\boldsymbol{wy}$ is a *right extension* of $\boldsymbol{w}$.

Note 2.3.40 (Superstring). An extension of a string is also called *superstring*.

Definition 2.3.41 (Occurrence)**.** Assuming strings $\boldsymbol{u}, \boldsymbol{w} \in A^*$, string $\boldsymbol{u}$ *has an occurrence* in string $\boldsymbol{w}$ if $\boldsymbol{u}$ is a factor of $\boldsymbol{w}$. We also say that string $\boldsymbol{u}$ *occurs* in string $\boldsymbol{w}$. Factor $\boldsymbol{u}$ of $\boldsymbol{w}$ *occurs at position i* (that is also called *end position*) in string $\boldsymbol{w}$ if for all $j \in \{1, \ldots, |\boldsymbol{u}|\}$ it holds that $\boldsymbol{u}[j] = \boldsymbol{w}[i - |\boldsymbol{u}| + j]$. We also say that $\boldsymbol{u}$ has an *occurrence at position i* in string $\boldsymbol{w}$.

Definition 2.3.42 (End set)**.** Assuming strings $\boldsymbol{u}, \boldsymbol{w}$, an *end set* is a totally ordered set of all i such that string $\boldsymbol{u}$ occurs at position i in string $\boldsymbol{w}$, denoted by $\mathsf{endset}_{\boldsymbol{w}}(\boldsymbol{u})$. (Ordering of an end set is equal to ordering of natural numbers. The smallest $i \in \mathsf{endset}_{\boldsymbol{w}}(\boldsymbol{u})$ corresponds to the leftmost end position of $\boldsymbol{u}$ in $\boldsymbol{w}$.)

Definition 2.3.43 (k-approximate occurrence)**.** Assuming strings $\boldsymbol{u}, \boldsymbol{w} \in A^*$, string $\boldsymbol{u}$ *has a k-approximate occurrence* in string $\boldsymbol{w}$ under some distance function D if $\boldsymbol{u} \in \mathsf{fact}_D^k(\boldsymbol{w})$. k-approximate factor $\boldsymbol{u}$ of $\boldsymbol{w}$ *k-approximately occurs at position i* (that is also called *k-approximate end position*) under some distance function D if there exists factor $\boldsymbol{v}$ of $\boldsymbol{w}$ that occurs at position i in $\boldsymbol{w}$ and $D(\boldsymbol{x}, \boldsymbol{v}) \leq k$.

Definition 2.3.44 (k-approximate end set)**.** Assuming string $\boldsymbol{u}, \boldsymbol{w}$ and maximum distance $k \geq 0$ under some distance function D, a *k-approximate end set* is a totally ordered set of all i such that string $\boldsymbol{u}$ k-approximately occurs at position i in string $\boldsymbol{w}$ under D. (Ordering of a k-approximate end set is equal to ordering of natural numbers regardless approximation of corresponding occurrence. The smallest i from the end set corresponds to the leftmost approximate end position of $\boldsymbol{u}$ in $\boldsymbol{w}$.)

2.4 Regularities of strings

Definition 2.4.1 (Border)**.** Assuming strings $\boldsymbol{u}, \boldsymbol{w} \in A^*$, string $\boldsymbol{u}$ is a *border* of string $\boldsymbol{w}$ if $\boldsymbol{u}$ is a prefix and a suffix of $\boldsymbol{w}$.

Definition 2.4.2 (k-approximate border)**.** String $\boldsymbol{v}$ is a *k-approximate border* of string $\boldsymbol{w}$ with maximum distance $k \geq 0$ under some distance function D if $\boldsymbol{w} = \boldsymbol{pu} = \boldsymbol{xs}$ and both $D(\boldsymbol{v}, \boldsymbol{p}) \leq k$ and $D(\boldsymbol{v}, \boldsymbol{s}) \leq k$.

Definition 2.4.3 (Restricted k-approximate border)**.** String $\boldsymbol{v}$ is a *restricted k-approximate border* of string $\boldsymbol{w}$ with maximum distance $k \geq 0$ under some distance function D if $\boldsymbol{v}$ is a k-approximate border of $\boldsymbol{w}$ under D and $\boldsymbol{v}$ is either a prefix or a suffix of $\boldsymbol{w}$.

Definition 2.4.4 (Period)**.** Assuming strings $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in A^*$ and an integer $i \geq 1$, every string $\boldsymbol{w}$ may be written as $\boldsymbol{u}^i\boldsymbol{v}$ where string $\boldsymbol{v}$ is a prefix of $\boldsymbol{u}$ and the length of string $\boldsymbol{u}$ is called *period*. String $\boldsymbol{u}$ is called *generator* of $\boldsymbol{w}$, number i is *exponent*. String $\boldsymbol{w}$ such that the maximum possible exponent is 1 is called *primitive*, otherwise $\boldsymbol{w}$ is called *periodic*.

Note 2.4.5. In many publications, by *period* is meant *generator* as stated in Definition 2.4.4, not its length.

Definition 2.4.6 (Repeating factor)**.** Assuming strings $\boldsymbol{u}, \boldsymbol{w} \in A^*$, string $\boldsymbol{u}$ is a *repeating factor* of string $\boldsymbol{w}$ if there exists at least one integer $j \neq 0$ such that $\boldsymbol{u}[i] = \boldsymbol{w}[i + h] = \boldsymbol{w}[i + h + j]$ holds for some $1 \leq h < |\boldsymbol{w}|$ and for all $1 \leq i \leq |\boldsymbol{u}|$.

Definition 2.4.7 (Repetition)**.** Assuming a repeating factor $\boldsymbol{u}$ of string $\boldsymbol{w}$ with an end set $E = \{i_1, i_2, \ldots, i_{|E|}\}$. At the position i_1 is the *first occurrence* (also called the *leftmost occurrence*) of $\boldsymbol{u}$ in $\boldsymbol{w}$, at the other positions there are *repetitions* of $\boldsymbol{u}$ in $\boldsymbol{w}$.

Note 2.4.8. Repetition given in Definition 2.4.7 is also called *repeat*. There exists another definition of repetition in literature, which constraints occurrences of the repeating factor to be adjacent. Definition 2.4.7 used in this dissertation thesis allows consecutive occurrences to overlap or to have gaps between them.

Definition 2.4.9 (Square)**.** Assuming strings $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in A^*$, factor $\boldsymbol{u}$ of string $\boldsymbol{w}$ is a *square* if it can be written as $\boldsymbol{u} = \boldsymbol{v}\boldsymbol{v}$. Then string $\boldsymbol{v}$ is a *root of square* $\boldsymbol{u}$.

Definition 2.4.10 (k-approximately repeating factor)**.** Assuming strings $\boldsymbol{u}, \boldsymbol{w} \in A^*$ and integer $k \geq 0$, string $\boldsymbol{u}$ is a *k-approximately repeating factor* of string $\boldsymbol{w}$ under some distance function D if $\boldsymbol{u}$ (exactly) occurs in string $\boldsymbol{w}$ at position i and there exists at least one such $j \neq i$ that $\boldsymbol{u}$ k-approximately occurs in $\boldsymbol{w}$ under D with distance less or equal to k at position j.

Definition 2.4.11 (k-approximate repetition)**.** Assuming a k-approximately repeating factor $\boldsymbol{u}$ of string $\boldsymbol{w}$ with maximum distance $k \geq 0$ under some distance function D and with k-approximate end set $E = \{i_1, \ldots, i_{|E|}\}$ under D, at the position i_1 is the *first k-approximate occurrence* of $\boldsymbol{u}$ in $\boldsymbol{w}$, at the other positions there are *k-approximate repetitions* of $\boldsymbol{u}$ in $\boldsymbol{w}$.

Definition 2.4.12 (Number of k-approximate repetitions)**.** Assuming strings $\boldsymbol{u}, \boldsymbol{w} \in A^*$ such that $\boldsymbol{u}$ is a factor of $\boldsymbol{w}$, and maximum distance $k \geq 0$ under some distance function D. When $\boldsymbol{u}$ k-approximately occurs r times in $\boldsymbol{w}$ under D, we say that *number of k-approximate repetitions* of $\boldsymbol{u}$ in $\boldsymbol{w}$ under D, denoted by $R^k_{D,\boldsymbol{u}}(\boldsymbol{w})$, is equal to $r - 1$. Then *number of k-approximate repetitions of all factors* of $\boldsymbol{w}$ under D, denoted by $R^k_D(\boldsymbol{w})$, is defined as

$$R^k_D(\boldsymbol{w}) = \sum_{\boldsymbol{w} \in \mathsf{fact}^k_D(\boldsymbol{w})} R^k_{D,\boldsymbol{u}}(\boldsymbol{w})$$

Definition 2.4.13 (Cover)**.** Assuming strings $\boldsymbol{v}, \boldsymbol{w} \in A^*$, string $\boldsymbol{v}$ is a *cover* of string $\boldsymbol{w}$ if $\boldsymbol{w}$ may be constructed using just superpositions (or concatenations) of copies of $\boldsymbol{v}$. We also say that $\boldsymbol{v}$ *covers* $\boldsymbol{w}$ or $\boldsymbol{w}$ is *covered* by $\boldsymbol{v}$. As each string $\boldsymbol{w}$ is covered by itself, such cover $\boldsymbol{w}$ of $\boldsymbol{w}$ is called *trivial*, all the other covers of $\boldsymbol{w}$ are *nontrivial* or *proper*. String having only trivial cover is *superprimitive*. String having at least one nontrivial cover is *quasiperiodic*.

Notation 2.4.14 (Set of all covers). Set of all covers of string $\boldsymbol{w}$ is denoted by $\mathsf{L}^{\mathrm{c}}(\boldsymbol{w})$.

Definition 2.4.15 (Cover array)**.** Assuming string $\boldsymbol{w} \in A^*$, a *cover array* of $\boldsymbol{w}$ is an array of $|\boldsymbol{w}|$ elements, where each i-th element, $1 \leq i \leq |\boldsymbol{w}|$, is equal to the longest proper cover of the prefix of $\boldsymbol{w}$ of length i.

Definition 2.4.16 (k-approximate cover)**.** String $\boldsymbol{v}$ is a *k-approximate cover* of string $\boldsymbol{w}$ with maximum distance k under some distance function D if there exists a set of strings $B = \{\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_{|B|}\}$ such that:

1. $D(\boldsymbol{v}, \boldsymbol{u}_i) \leq k$ for all i such that $1 \leq i \leq |B|$,
2. $\boldsymbol{w}$ can be constructed by superpositions (or concatenations) of copies of strings from B.

Notation 2.4.17 (Set of all k-approximate covers). Set of all covers of string $\boldsymbol{w}$ with maximum distance k under some distance function D is denoted by $\mathsf{L}^{\mathrm{c}}_{D^k}(\boldsymbol{w})$.

Definition 2.4.18 (k-approximate cover)**.** String $\boldsymbol{v}$ is a *restricted k-approximate cover* of string $\boldsymbol{w}$ with maximum distance $k \geq 0$ under some distance function D if $\boldsymbol{v}$ is a k-approximate cover of $\boldsymbol{w}$ under D and $\boldsymbol{v}$ is a *factor* of $\boldsymbol{w}$.

Definition 2.4.19 (Smallest distance k-approximate cover)**.** Assuming a k-approximate cover $\boldsymbol{v}$ of string $\boldsymbol{w}$ with maximum distance $k \geq 0$ under some distance function D, distance $l \geq 0$ is the *smallest distance* of k-approximate cover $\boldsymbol{v}$ of $\boldsymbol{w}$ if $\boldsymbol{v}$ is an l-approximate cover of $\boldsymbol{w}$ and there exists no $i < l$ such that $\boldsymbol{v}$ is an i-approximate cover of $\boldsymbol{w}$.

Definition 2.4.20 (Seed)**.** Assuming strings $\boldsymbol{v}, \boldsymbol{w} \in A^*$, string $\boldsymbol{v}$ is a *seed* of string $\boldsymbol{w}$ if $\boldsymbol{v}$ is a factor of $\boldsymbol{w}$ and $\boldsymbol{v}$ is a cover of an extension (superstring) of $\boldsymbol{w}$ and $|\boldsymbol{v}| \leq |\boldsymbol{w}|$.

Definition 2.4.21 (Left seed)**.** Assuming strings $\boldsymbol{v}, \boldsymbol{w} \in A^*$, string $\boldsymbol{v}$ is a *left seed* of string $\boldsymbol{w}$ if $\boldsymbol{v}$ is a cover of some right extension of $\boldsymbol{w}$.

Note 2.4.22. A left seed a string is its prefix, it is left-aligned, therefore it is called "left".

Definition 2.4.23 (Right seed)**.** Assuming strings $\boldsymbol{v}, \boldsymbol{w} \in A^*$, string $\boldsymbol{v}$ is a *right seed* of string $\boldsymbol{w}$ if $\boldsymbol{v}$ is a cover of some left extension of $\boldsymbol{w}$.

Definition 2.4.24 (Shortest-seed array)**.** Assuming string $\boldsymbol{w} \in A^*$, a *shortest-seed array* of $\boldsymbol{w}$ is an array of $|\boldsymbol{w}|$ elements, where each i-th element, $1 \leq i \leq |\boldsymbol{w}|$, is equal to the shortest nonempty seed of the prefix of $\boldsymbol{w}$ of length i.

Definition 2.4.25 (Shortest-left-seed array)**.** Assuming string $\boldsymbol{w} \in A^*$, a *shortest-left-seed array* of $\boldsymbol{w}$ is an array of $|\boldsymbol{w}|$ elements, where each i-th element, $1 \leq i \leq |\boldsymbol{w}|$, is equal to the shortest nonempty left seed of the prefix of $\boldsymbol{w}$ of length i.

Note 2.4.26. Notions of longest-left-seed array, shortest-right-seed array and longest-right-seed array are defined accordingly.

Definition 2.4.27 (k-approximate seed)**.** String $\boldsymbol{v}$ is a *k-approximate seed* of string $\boldsymbol{w}$ with maximum distance $k \geq 0$ under some distance function D if $\boldsymbol{v}$ is a k-approximate cover of some extension of $\boldsymbol{w}$ under D.

Definition 2.4.28 (Restricted k-approximate seed)**.** String $\boldsymbol{v}$ is a *restricted k-approximate seed* of string $\boldsymbol{w}$ with maximum distance $k \geq 0$ under some distance function D if $\boldsymbol{v}$ is a k-approximate seed of $\boldsymbol{w}$ under D and $\boldsymbol{v}$ is a *factor* of $\boldsymbol{w}$.

2.5 Finite automata

Definition 2.5.1 (Deterministic finite automaton)**.** A *totally defined deterministic finite automaton (DFA) automaton* $\mathcal{M}$ is a quintuple (Q, A, δ, q_0, F) where

- Q is a nonempty finite set of states,
- A is a nonempty finite input alphabet,
- $\delta : Q \times A \mapsto Q$ is a transition function,
- $q_0 \in Q$ is initial state,
- $F \subseteq Q$ is a set of final states.

Note 2.5.2. Totally defined DFA $\mathcal{M} = (Q, A, \delta, q_0, F)$ is also called complete DFA. For every possible pair $(q, a), q \in Q, a \in A$ is $\delta(q, a)$ defined.

Definition 2.5.3 (Partially defined DFA)**.** A DFA $\mathcal{M} = (Q, A, \delta, q_0, F)$ is *partially defined* if for some possible pair $(q, a), q \in Q, a \in A$ is $\delta(q, a)$ undefined.

Definition 2.5.4 (Successor of state of DFA)**.** Assuming a DFA $\mathcal{M} = (Q, A, \delta, q_0, F)$, a state $q_1 \in Q$, and a symbol $a \in A$, state $q_2 \in Q$ is a *successor* of q_1 for a if $\delta(q_1, a) = q_2$.

Definition 2.5.5 (Predecessor of state of DFA)**.** Assuming a deterministic finite automaton $\mathcal{M} = (Q, A, \delta, q_0, F)$ and a state $q_2 \in Q$, every state $q_1 \in Q$ is a *predecessor* of q_2 if $\delta(q_1, a) = q_2$ holds for some symbol $a \in A$.

Definition 2.5.6 (Trie-like automaton)**.** A *trie-like automaton* $\mathcal{M} = (Q, A, \delta, q_0, F)$ is a DFA where each state $q \in Q \setminus \{q_0\}$ has exactly one predecessor.

Definition 2.5.7 (Extended transition function)**.** An *extended transition function* of a DFA $\mathcal{M} = (Q, A, \delta, q_0, F)$ is denoted by δ^* and it is defined for $q \in Q, a \in A, \boldsymbol{u} \in A^*$ inductively:

1. $\delta^*(q, \varepsilon) = q$,
2. $\delta^*(q, \boldsymbol{u}a) = \delta(\delta^*(q, \boldsymbol{u}), a)$.

Definition 2.5.8 (Acyclic DFA)**.** A DFA $\mathcal{M} = (Q, A, \delta, q_0, F)$ is *acyclic* if for any state $q \in Q$ there exists no nonempty sequence of transition to the same state, i.e. $\forall q \in Q, \boldsymbol{w} \in A^+ : \delta^*(q, \boldsymbol{w}) \neq q$.

Definition 2.5.9 (String accepted by DFA)**.** Assuming a DFA $\mathcal{M} = (Q, A, \delta, q_0, F)$, string $\boldsymbol{w}$ is *accepted* by $\mathcal{M}$ if and only if $\delta^*(q_0, \boldsymbol{w}) \in F$.

Definition 2.5.10 (Language accepted by DFA)**.** Assuming a DFA $\mathcal{M} = (Q, A, \delta, q_0, F)$, language L is the *language accepted* by $\mathcal{M}$ if and only if for each $\boldsymbol{w} \in L$ holds that $\boldsymbol{w}$ is accepted by $\mathcal{M}$.

Definition 2.5.11 (Left language of state of DFA)**.** Assuming a deterministic finite automaton $\mathcal{M} = (Q, A, \delta, q_0, F)$ and its state $q \in Q$, language L_q is *left language* of state q defined as $L_q = \{\boldsymbol{w} : \boldsymbol{w} \in A^* \wedge \delta^*(q_0, \boldsymbol{w}) = q\}$.

Notation 2.5.12. Assuming state q of a trie-like automaton, the string of left language of q is denoted by $\mathsf{lfactor}(q)$.

Definition 2.5.13 (Depth of state of DFA)**.** Assuming an acyclic deterministic finite automaton $\mathcal{M} = (Q, A, \delta, q_0, F)$ and its state $q \in Q$, the *depth* of q is the length of the longest string from left language of q.

Note 2.5.14. Depth for an automaton that is not acyclic is undefined.

Definition 2.5.15 (Nondeterministic finite automaton)**.** A *nondeterministic finite automaton (NFA)* $\mathcal{M}$ is a quintuple (Q, A, δ, q_0, F) where

- Q is a nonempty finite set of states,
- A is a nonempty finite input alphabet,
- $\delta : Q \times A \mapsto \mathcal{P}(Q)$ is a transition function,
- $q_0 \in Q$ is initial state,
- $F \subseteq Q$ is a set of final states.

Notation 2.5.16. For sake of simplicity, a *label* of state q_i of a NFA, where i is a number, is the number i.

Definition 2.5.17 (Extended transition function of NFA)**.** An *extended transition function* of a NFA $\mathcal{M} = (Q, A, \delta, q_0, F)$ is denoted by δ^* and it is defined for $q_1, q_2 \in Q, a \in A, \boldsymbol{u} \in A^*$ inductively:

1. $\delta^*(q_1, \varepsilon) = \{q_1\}$,
2. $\delta^*(q_1, \boldsymbol{u}a) = \bigcup_{q_2 \in \delta^*(q_1, \boldsymbol{u})} \delta(q_2, a)$.

Definition 2.5.18 (Acyclic NFA)**.** A NFA $\mathcal{M} = (Q, A, \delta, q_0, F)$ is *acyclic* if for any state $q \in Q$ there exists no nonempty sequence of transition containing same state q, i.e. $\forall q \in Q, \boldsymbol{w} \in A^+ : q \notin \delta^*(q, \boldsymbol{w})$.

Definition 2.5.19 (NFA with ε transitions)**.** A *NFA with ε transitions* $\mathcal{M}$ is a quintuple (Q, A, δ, q_0, F) where

- Q is a nonempty finite set of states,
- A is a nonempty finite input alphabet,
- $\delta : Q \times (A \cup \{\varepsilon\}) \mapsto \mathcal{P}(Q)$ is a transition function,

- $q_0 \in Q$ is initial state,
- $F \subseteq Q$ is a set of final states.

This type of automaton may be abbreviated by ε-NFA.

Definition 2.5.20 (ε-closure)**.** [28, 38] Assuming a nondeterministic finite automaton with ε transitions $\mathcal{M} = (Q, A, \delta, q_0, F)$ and its states $q_1, q_2 \in Q$, the function ε-closure is defined as follows:

1. $q_1 \in \varepsilon\text{-closure}(q_1)$,
2. $q_2 \in \varepsilon\text{-closure}(q_1)$ if $q_2 \in \delta^*(q_1, \varepsilon)$.

Definition 2.5.21 (Extended transition function of ε-NFA)**.** An *extended transition function* of a NFA with ε transitions $\mathcal{M} = (Q, A, \delta, q_0, F)$ is denoted by δ^* and it is defined for $q_1, q_2 \in Q, a \in A, \boldsymbol{u} \in A^*$ inductively:

1. $\delta^*(q_1, \varepsilon) = \varepsilon\text{-closure}(q_1)$,
2. $\delta^*(q_1, \boldsymbol{u}a) = \bigcup_{q_2 \in \delta^*(q_1, \boldsymbol{u})} (\delta(q_2, a) \cup \varepsilon\text{-closure}(q_2))$.

Definition 2.5.22 (Language accepted by NFA)**.** Assuming a nondeterministic finite automaton $\mathcal{M} = (Q, A, \delta, q_0, F)$ either with or without ε transitions, language L is *accepted* by $\mathcal{M}$ if and only if $L = \{\boldsymbol{w} : q \in \delta^*(q_0, \boldsymbol{w}) \wedge q \in F\}$.

Definition 2.5.23 (Left language of state of NFA)**.** Assuming a nondeterministic finite automaton $\mathcal{M} = (Q, A, \delta, q_0, F)$ and its state $q \in Q$, language L is *left language* of q if and only if $L = \{\boldsymbol{w} : \boldsymbol{w} \in A^* \wedge q \in \delta^*(q_0, \boldsymbol{w})\}$.

Definition 2.5.24 (Finite automaton)**.** Finite automaton is either a DFA, NFA, or NFA with ε transitions.

Notation 2.5.25 (Language accepted by FA). Language L accepted by a finite automaton (FA) $\mathcal{M}$ is denoted by $L(\mathcal{M})$. Finite automaton $\mathcal{M}$ accepting a language L is denoted by $\mathcal{M}(L)$.

Definition 2.5.26 (Equivalence of FA)**.** Finite automata $\mathcal{M}_1, \mathcal{M}_2$ are said to be *equivalent* if $L(\mathcal{M}_1) = L(\mathcal{M}_2)$ (the accept the same language).

Definition 2.5.27 (Subset construction)**.** [42, 38] Assuming a nondeterministic finite automaton $\mathcal{M}_N = (Q_N, A, \delta_N, q_0^N, F_N)$, *subset construction* of DFA $\mathcal{M} = (Q, A, \delta, q_0, F)$ equivalent to $\mathcal{M}_N$ is defined as follows:

1. Set $Q = \{q_0\}$ will be defined, state $q_0 = \{q_0^N\}$ will be treated as unmarked.
2. If each state in Q is marked then continue with step 4.

3. Arbitrary unmarked state q will be chosen from Q and the following operations will be executed:
   a) $\delta(q, a) \leftarrow \bigcup \delta_N(q_N, a)$ for all $q_N \in q$ and for all $a \in A$,
   b) $Q \leftarrow Q \cup \delta(q, a)$ for all $a \in A$,
   c) state $q \in Q$ will be marked,
   d) continue with step 2.
4. $F = \{q : q \in Q, q_N \cap F_N \neq \emptyset, q_N \in q\}$.

Note 2.5.28. A FA $\mathcal{M}$ created using subset construction from an automaton $\mathcal{M}_N$ is equivalent to $\mathcal{M}_N$ and it is deterministic. [42]

Definition 2.5.29 (d-subset)**.** Assuming a FA $\mathcal{M} = (Q, A, \delta, q_0, F)$ created using subset construction from a NFA $\mathcal{M}_N$, each state $q \in Q$ of $\mathcal{M}$ is a subset of states of $\mathcal{M}_N$ called *d-subset*, denoted by $\mathsf{d}(q)$. The d-subset is a totally ordered set, the ordering is equal to ordering of depths of states of $\mathcal{M}_N$ (Definition 2.5.39) as natural numbers.

Definition 2.5.30 (Prefix automaton)**.** Assuming string $\boldsymbol{w} \in A^*$, a FA accepting the set of all prefixes of $\boldsymbol{w}$ is called *prefix automaton* for $\boldsymbol{w}$.

Definition 2.5.31 (Suffix automaton)**.** Assuming string $\boldsymbol{w} \in A^*$, a FA accepting the set of all suffixes of $\boldsymbol{w}$ is called *suffix automaton* for $\boldsymbol{w}$.

Notation 2.5.32. Nondeterministic suffix automaton for string $\boldsymbol{w}$ is denoted by $\mathcal{M}_{\text{S}^\text{N}}(\boldsymbol{w})$.

Definition 2.5.33 (Factor automaton)**.** Assuming string $\boldsymbol{w} \in A^*$, a FA accepting the set of all factors of $\boldsymbol{w}$ is called *factor automaton* for $\boldsymbol{w}$.

Definition 2.5.34 (k-approximate prefix automaton)**.** Assuming string $\boldsymbol{w} \in A^*$, integer $k \geq 0$ and some distance function D, a FA accepting the set of all k-approximate prefixes of $\boldsymbol{w}$ under D is called *k-approximate prefix automaton* for $\boldsymbol{w}$, D, and k.

Definition 2.5.35 (k-approximate suffix automaton)**.** Assuming string $\boldsymbol{w} \in A^*$, integer $k \geq 0$ and some distance function D, a FA accepting the set of all k-approximate suffixes of $\boldsymbol{w}$ under D is called *k-approximate suffix automaton* for $\boldsymbol{w}$, D, and k.

Notation 2.5.36. Nondeterministic k-approximate suffix automaton for string $\boldsymbol{w}$ and distance D is denoted by $\mathcal{M}^{D,k}_{\text{S}^\text{N}}(\boldsymbol{w})$, the deterministic one is denoted by $\mathcal{M}^{D,k}_{\text{S}^\text{D}}(\boldsymbol{w})$.

Definition 2.5.37 (k-approximate factor automaton)**.** Assuming string $\boldsymbol{w} \in A^*$, integer $k \geq 0$ and some distance function D, a FA accepting the set of all k-approximate factors of $\boldsymbol{w}$ under D is called *k-approximate factor automaton* for $\boldsymbol{w}$, D, and k.

Definition 2.5.38 (Backbone)**.** Assume a deterministic factor or suffix automaton $\mathcal{M} = (Q, A, \delta, q_0, F)$ (either exact or k-approximate one) for string $\boldsymbol{w} \in A$. A *backbone* of $\mathcal{M}$ is a deterministic automaton $\mathcal{M}_B = (Q_B, A, \delta_B, q_0, F_B)$ such that for all $0 \leq i \leq |\boldsymbol{w}|$ holds $Q_B = \{q_i : q_i \in Q\}$, $\delta_B(q_{i-1}, \boldsymbol{w}[i]) = q_i$, and $F_B = \{q : q \in Q_B \cap F\}$.

Definition 2.5.39 (Depth of state of suffix automaton)**.** Assuming a nondeterministic suffix automaton $\mathcal{M}$, for any state q_i from Algorithm 3.3 the number i is its *depth*.

Definition 2.5.40 (Depth and level of a state)**.** Assume a nondeterministic k-approximate suffix automaton with ε transitions $\mathcal{M}$, for any state q_i^j from Algorithm 3.7 the number i is its *depth*, denoted by $\mathsf{depth}(q_i^j)$ and the number j is its *level*, denoted by $\mathsf{level}(q_i^j)$.

Definition 2.5.41 (Cover candidate automaton)**.** Assume string $\boldsymbol{w}$ and maximum distance k under some distance function D. The DFA accepting all strings that are k-approximate prefixes and simultaneously k-approximate suffixes of $\boldsymbol{w}$ is called a *cover-candidate automaton* and is denoted by $\mathcal{M}^{D,k}_{\mathrm{C^D}}(\boldsymbol{w})$.

Notation 2.5.42. Cover-candidate automaton for string $\boldsymbol{w}$ accepting strings that are exact prefixes and simultaneously suffixes of $\boldsymbol{w}$ is denoted by $\mathcal{M}_{\mathrm{C^D}}(\boldsymbol{w})$.

Notation 2.5.43. Assume maximum part of deterministic k-approximate suffix automaton for string $\boldsymbol{w}$ and a distance D such that d-subset of each of its state contains at least one element with level equal to 0. Such an automaton is denoted by $\tilde{\mathcal{M}}^{D,k}_{\mathrm{S^D}}(\boldsymbol{w})$.

2.6 Problems

Definition 2.6.1. Let $\boldsymbol{w}$ be given string. The *all covers problem* is to find the set $\mathsf{L}^{\mathrm{c}}(\boldsymbol{w})$ such that for any string $\boldsymbol{u}$ holds: $\boldsymbol{u} \in \mathsf{L}^{\mathrm{c}}(\boldsymbol{w})$ if and only if $\boldsymbol{u}$ is a cover of $\boldsymbol{w}$.

Definition 2.6.2. Let $\boldsymbol{w}$ be given string, D be given distance function and k be maximum allowed distance. The *all smallest distance k-approximate covers problem* is to find the set $\mathsf{L}^{\mathrm{c}}_{D^k}(\boldsymbol{w})$ of pairs $(\boldsymbol{u}, l)$ such that

1. string $\boldsymbol{u}$ is in the pair from $\mathsf{L}^{\mathrm{c}}_{D^k}(\boldsymbol{w})$ if and only if $\boldsymbol{u}$ is a k-approximate cover of $\boldsymbol{w}$ under distance function D, and
2. l is the smallest distance such that $\boldsymbol{u}$ is an l-approximate cover of $\boldsymbol{w}$.

Definition 2.6.3. Let $\boldsymbol{w}$ be given string, D be given distance function and k be maximum allowed distance. The *all restricted smallest distance k-approximate covers problem* is to find the set $\mathsf{L}^{\mathrm{c}}_{D^k}(\boldsymbol{w})$ of pairs $(\boldsymbol{u}, l)$ such that

1. string $\boldsymbol{u}$ is in a pair from $\mathsf{L}^{\mathrm{c}}_{D^k}(\boldsymbol{w})$ if and only if $\boldsymbol{u}$ is a restricted k-approximate cover of $\boldsymbol{w}$ under distance function D, and
2. l is the smallest distance such that $\boldsymbol{u}$ is a restricted l-approximate cover of $\boldsymbol{w}$.

Definition 2.6.4. Let $\boldsymbol{w}$ be given string. The *all seeds problem* is to find the set $\mathsf{L}^{\mathrm{s}}(\boldsymbol{w})$ such that for any string $\boldsymbol{u}$ holds: $\boldsymbol{u} \in \mathsf{L}^{\mathrm{s}}(\boldsymbol{w})$ if and only if $\boldsymbol{u}$ is a seed of $\boldsymbol{w}$.

Definition 2.6.5. Let $\boldsymbol{w}$ be given string, D be given distance function and k be maximum allowed distance. The *all smallest distance k-approximate seeds problem* is to find the set $\mathsf{L}^{\mathrm{s}}_{D^k}(\boldsymbol{w})$ of pairs $(\boldsymbol{u}, l)$ such that

1. $\boldsymbol{u}$ is in a pair from $\boldsymbol{u} \in \mathsf{L}^{\mathrm{s}}_{D^k}(\boldsymbol{w})$ if and only if $\boldsymbol{u}$ is a k-approximate seed of $\boldsymbol{w}$ under distance function D, and
2. l is the smallest distance such that $\boldsymbol{u}$ is an l-approximate seed of $\boldsymbol{w}$.

Definition 2.6.6. Let $\boldsymbol{w}$ be given string, D be given distance function and k be maximum allowed distance. The *all restricted smallest distance k-approximate seeds problem* is to find the set $\mathsf{L}^{\mathrm{s}}_{D^k}(\boldsymbol{w})$ of pairs $(\boldsymbol{u}, l)$ such that

1. $\boldsymbol{u}$ is in a pair from $\boldsymbol{u} \in \mathsf{L}^{\mathrm{s}}_{D^k}(\boldsymbol{w})$ if and only if $\boldsymbol{u}$ is a restricted k-approximate seed of $\boldsymbol{w}$ under distance function D, and
2. l is the smallest distance such that $\boldsymbol{u}$ is a restricted l-approximate seed of $\boldsymbol{w}$.

Chapter 3

Background and state-of-the-art

To compute covers and seeds, their properties are needed. Many of them have already been known and are summarised in this section. We explore properties of exact covers and seeds. To compute approximate ones, that are generalizations of exact ones and thus have some uncommon properties, we need formal way of approximation – distance metrics. The ways to work with strings within some distance, including searching and extracting some patterns, are also summarized. Because this dissertation thesis is focused on use the formalism of finite automata, contribution of their usage in this area is also mentioned.

3.1 Theoretical background

3.1.1 Properties of exact covers

Recalling Definition 2.4.13, having cover $\boldsymbol{u}$ of string $\boldsymbol{w}$, every position in $\boldsymbol{w}$ must lie within some occurrence of $\boldsymbol{u}$ in $\boldsymbol{w}$.

Property 3.1.1. Cover $\boldsymbol{u}$ of $\boldsymbol{w}$ is both a prefix and a suffix of $\boldsymbol{w}$, i.e $\boldsymbol{u}$ is a border of $\boldsymbol{w}$. [9]

Property 3.1.2. Looking at Definitions 2.4.6 and 2.4.13, it is obvious that any nontrivial cover of string $\boldsymbol{w}$ is a repeating factor of $\boldsymbol{w}$. [5]

Property 3.1.3. Assume cover $\boldsymbol{u}$ of $\boldsymbol{w}$ and positions of its consecutive occurrences i, j. For their overlap $l = i + |u| - j$ always holds $l \geq 0$. [47]

Note 3.1.4. Cover $\boldsymbol{u}$ of $\boldsymbol{w}$ must be long enough to cover positions of $\boldsymbol{w}$ between *any* two consecutive occurrences i, j of $\boldsymbol{u}$ within $\boldsymbol{w}$, i.e. $j - i \leq |u|$. I.e., for any nontrivial cover $\boldsymbol{u}$ of string $\boldsymbol{w}$ and for its end set $C = \mathsf{endset}_{\boldsymbol{w}}(\boldsymbol{u}) = [i_1, i_2, \ldots, i_{|C|}]$ of $\boldsymbol{u}$ in $\boldsymbol{w}$ holds: $j = 2, 3, \ldots, |C| : i_j - i_{j-1} \leq |\boldsymbol{u}|$. [5]

Lemma 3.1.5. Any cover $\boldsymbol{u}$ of string $\boldsymbol{w}$ covers also any border $\boldsymbol{y}$ of $\boldsymbol{w}$ such that $|\boldsymbol{y}| \geq |\boldsymbol{u}|$. [5, 9]

Recalling Definition 2.4.4, every string $\boldsymbol{w}$ may be written as $\boldsymbol{w} = \boldsymbol{u}^i\boldsymbol{v}$, where $\boldsymbol{v}$ is a prefix of $\boldsymbol{u}$. Periodic strings have covers related to their generator:

Lemma 3.1.6. Assuming $\boldsymbol{w} = \boldsymbol{u}^i\boldsymbol{v}$, $\boldsymbol{uv}$ covers $\boldsymbol{w}$. [5]

Proof. (Idea [5]) $\boldsymbol{uv}$ is a border of $\boldsymbol{w}$, also $\boldsymbol{v}$ is a prefix of $\boldsymbol{u}$, thus occurrences of $\boldsymbol{uv}$ can overlap to cover $\boldsymbol{w}$. □

Regarding the shortest cover, the following lemma holds:

Lemma 3.1.7. Any string has exactly one superprimitive cover (which is also the shortest one). [4, Lemma 3.1], [9]

Proof. (Idea [4]) $\boldsymbol{w}$ cannot have two superprimitive covers $\boldsymbol{u}, \boldsymbol{v}; |\boldsymbol{u}| < |\boldsymbol{v}|$. As both $\boldsymbol{u}, \boldsymbol{v}$ must be prefixes and suffixes of $\boldsymbol{w}$, $\boldsymbol{u}$ is also prefix and suffix of $\boldsymbol{v}$, respectively. Assume string $\boldsymbol{y}$ as the longest common prefix of $\boldsymbol{w}$ and $\boldsymbol{v}$ such that $\boldsymbol{y}$ is covered by $\boldsymbol{u}$. There must exist such $\boldsymbol{y}, |\boldsymbol{y}| \geq |\boldsymbol{u}|, |\boldsymbol{y}| < |\boldsymbol{v}|$ (every string is covered by itself). It holds that $|\boldsymbol{y}| \geq |\boldsymbol{v}| - |\boldsymbol{u}|$, because $\boldsymbol{u}$ covers $\boldsymbol{w}$ and thus also a right extension of $\boldsymbol{v}$. Because $\boldsymbol{u}$ is also a suffix of $\boldsymbol{v}$, it covers $\boldsymbol{v}$, therefore $\boldsymbol{v}$ cannot be superprimitive. □

Two consecutive occurrences of a cover have also special properties:

Lemma 3.1.8. Assuming string $\boldsymbol{w}$ and its cover $\boldsymbol{u}$ having consecutive occurrences (that may also overlap) with start positions i, j, factor $\boldsymbol{w}[i..j-1]$ is always root of a square in $\boldsymbol{w}$. [4]

Note 3.1.9. It is obvious that similar property holds when regarding end positions of consecutive occurrences of a cover.

Lemma 3.1.10. Assume string $\boldsymbol{w}$ and its factors $\boldsymbol{x}$, $\boldsymbol{y}$ such that $\boldsymbol{x}$ covers $\boldsymbol{w}$ and $|\boldsymbol{x}| \geq |\boldsymbol{y}|$. Then $\boldsymbol{y}$ is a cover of $\boldsymbol{w}$ if and only if $\boldsymbol{y}$ is a cover of $\boldsymbol{x}$. [40]

It is possible to write every string $\boldsymbol{w}$ as $\boldsymbol{w} = (\boldsymbol{xy})^i\boldsymbol{x}, i = 1, 2, \ldots, l$, where $\boldsymbol{x}$ has maximum possible length. Then the following characterization of every cover holds:

Lemma 3.1.11. ([40]) Assuming strings $\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y}$ such that $\boldsymbol{w} = (\boldsymbol{xy})^i\boldsymbol{x}, i = 1, 2, \ldots, l$ and $\boldsymbol{x}$ is of maximum possible length, then each cover $\boldsymbol{u}$ of string $\boldsymbol{w}$ is either:

- $(\boldsymbol{xy})^i\boldsymbol{x}, i = 1, 2, \ldots, l$;
- a proper cover of $\boldsymbol{xyx}$ when $l > 1$;
- a cover of $\boldsymbol{x}$ which also covers $\boldsymbol{xyx}$ when $l = 1$.

Property 3.1.12. Assume string $\boldsymbol{w}$ having the shortest period equal to i. Then for the shortest cover $\boldsymbol{u}$ of $\boldsymbol{w}$ holds $i > \frac{|\boldsymbol{u}|}{2}$. [17]

3.1.2 Properties of exact seeds

Since notion of seed is a generalization of cover, not all properties of covers hold for seeds. For example, assuming seed $\boldsymbol{u}$ of string $\boldsymbol{w}$, Property 3.1.3 holds obviously for seeds, as they may be no gap between consecutive occurrences of a seed $\boldsymbol{u}$ of $\boldsymbol{w}$, on the other hand, Property 3.1.1 does not hold for seeds in general, because seed $\boldsymbol{u}$ covers an extension of $\boldsymbol{w}$ and therefore $\boldsymbol{u}$ is not necessarily a prefix or suffix of $\boldsymbol{w}$.

Property 3.1.13. ([25], adopted for strings) Assume string $\boldsymbol{w}$ and its factor $\boldsymbol{u}$, then $\boldsymbol{u}$ is a seed of $\boldsymbol{w}$ if and only if there exist factors $\boldsymbol{p}_1, \boldsymbol{v}_1, \boldsymbol{s}_1, \boldsymbol{p}_2, \boldsymbol{v}_2, \boldsymbol{s}_2$ of $\boldsymbol{w}$ such that

1. $\boldsymbol{w} = \boldsymbol{p}_1\boldsymbol{v}_1\boldsymbol{s}_1; 0 \leq |\boldsymbol{p}_1| < |\boldsymbol{u}|; 0 \leq |\boldsymbol{s}_1| < |\boldsymbol{u}|; |\boldsymbol{u}| \leq |\boldsymbol{v}_1|$ and $\boldsymbol{u}$ is a cover of $\boldsymbol{v}_1$;
2. $\boldsymbol{w} = \boldsymbol{p}_2\boldsymbol{v}_2\boldsymbol{s}_2; |\boldsymbol{p}_1| \leq |\boldsymbol{p}_2| < |\boldsymbol{u}|; |\boldsymbol{s}_1| \leq |\boldsymbol{s}_2| < |\boldsymbol{u}|$ and $\boldsymbol{p}_2$ is a suffix of $\boldsymbol{u}$ and and $\boldsymbol{s}_2$ is a prefix of $\boldsymbol{u}$.

Note 3.1.14 (Length and end-positions of seed [48]). Another way to write part 1 of the previous Property is the following: Assume string $\boldsymbol{w}$ and its seed $\boldsymbol{u}$ having more than one occurrence in $\boldsymbol{w}$. For its end set $C = \{r_1, r_2, \ldots, r_{|C|}\}$ and for $j = 2, 3, \ldots, |C|$ holds: $r_j - r_{j-1} \leq |\boldsymbol{u}|$.

Lemma 3.1.15 (Prefix and suffix of seed [48])**.** Writing the previous Property in other words: Assuming string $\boldsymbol{w}$ and its seed $\boldsymbol{u}$, having an end set $C = \{i_1, i_2, \ldots, i_{|C|}\}$ of $\boldsymbol{u}$ in $\boldsymbol{w}$, there must exist some prefix of $\boldsymbol{u}$ of length at least $|\boldsymbol{w}| - i_{|C|}$ equal to suffix of $\boldsymbol{w}$ of the same length and there must exist some suffix of $\boldsymbol{u}$ of length at least $i_1 - |\boldsymbol{u}|$ equal to prefix of $\boldsymbol{w}$ of the same length.

Proof. (Idea) Assuming factor $\boldsymbol{v}_1$ of $\boldsymbol{w}$ covered by $\boldsymbol{u}$, string $\boldsymbol{w}$ may be written as $\boldsymbol{p}_1\boldsymbol{v}_1\boldsymbol{s}_1$. Assuming an extension $\boldsymbol{z}$ of $\boldsymbol{w}$ covered by $\boldsymbol{u}$, string $\boldsymbol{z}$ may be written as $\boldsymbol{xwy} = \boldsymbol{xp}_1\boldsymbol{v}_1\boldsymbol{s}_1\boldsymbol{y}$. Suppose that $\boldsymbol{z}$ is of minimal possible length. To $\boldsymbol{u}$ be a cover of $\boldsymbol{z}$, $\boldsymbol{xp}_1$ must be either equal to $\boldsymbol{u}$ or a prefix of $\boldsymbol{u}$; $\boldsymbol{s}_1\boldsymbol{y}$ must be either equal to $\boldsymbol{u}$ or a suffix of $\boldsymbol{u}$. The end set C contains at least one element, as $\boldsymbol{u}$ is a factor of $\boldsymbol{w}$. The factor of $\boldsymbol{w}$ determined by C is equal to the factor $\boldsymbol{v}_1$, it is the longest factor of $\boldsymbol{w}$ covered by $\boldsymbol{u}$. Therefore, the suffix of $\boldsymbol{w}$ that is necessary to be equal to the prefix of $\boldsymbol{u}$ has to be of length at least $|\boldsymbol{s}_1| = |\boldsymbol{w}| - i_{|C|}$ (the length of the prefix of $\boldsymbol{u}$ is not necessarily equal to $|\boldsymbol{s}_1|$ as overlap is possible). The prefix of $\boldsymbol{w}$ that is necessary to be equal to the suffix of $\boldsymbol{u}$ has to be of length at least $|\boldsymbol{p}_1| = i_1 - |\boldsymbol{u}|$. □

There is a special case of a seed and for such the following holds:

Lemma 3.1.16. ([30]) Assume seed $\boldsymbol{u}$ of string $\boldsymbol{w}$ such that it is possible to use occurrences of $\boldsymbol{u}$ in $\boldsymbol{w}$ without gaps or overlaps (only concatenations). Then:

1. every cyclic rotation of $\boldsymbol{u}$ is a seed of $\boldsymbol{w}$;
2. any extension of $\boldsymbol{u}$ which is a factor of $\boldsymbol{w}$ is a seed of $\boldsymbol{w}$.

Lemma 3.1.17. ([30]) Assume strings $\boldsymbol{u}, \boldsymbol{w}$ such that $|\boldsymbol{u}| \leq |\boldsymbol{w}|$, $\boldsymbol{u}$ is not a factor of $\boldsymbol{w}$ and $\boldsymbol{u}$ is a cover of an extension of $\boldsymbol{w}$. Then there exists a seed of $\boldsymbol{w}$ that is either:

- a factor of $\boldsymbol{u}$,
- a cyclic rotation of $\boldsymbol{u}$, or
- a cyclic rotation of a factor of $\boldsymbol{u}$.

For the purpose of the next two lemmas, the notion of easy seed [30] is introduced.

Definition 3.1.18 (Easy seed)**.** Assume seed $\boldsymbol{u}$ of string $\boldsymbol{w}$. String $\boldsymbol{u}$ is an *easy seed* if there exists a factor $\boldsymbol{v}$ of $\boldsymbol{u}$ such that $\boldsymbol{v}$ is also a seed of $\boldsymbol{w}$ and it is possible to use occurrences of $\boldsymbol{v}$ in $\boldsymbol{w}$ without gaps or overlaps (only concatenations) to cover an extension of $\boldsymbol{w}$.

Lemma 3.1.19. Assume string $\boldsymbol{w}$ having the shortest period equal to i. A seed $\boldsymbol{u}$ of $\boldsymbol{w}$ is an easy seed if and only if $|\boldsymbol{u}| \geq i$. [30]

Property 3.1.20. Assume string $\boldsymbol{w}$ having the shortest period equal to i. Then for the shortest seed $\boldsymbol{u}$ of $\boldsymbol{w}$ holds $i > \frac{|\boldsymbol{u}|}{2}$. [17]

Lemma 3.1.21. Assume string $\boldsymbol{w}$, its prefix $\boldsymbol{p}$ of length equal to its shortest period, and a factor $\boldsymbol{u}$ of $\boldsymbol{w}$. Then $\boldsymbol{u}$ is an easy seed of $\boldsymbol{w}$ if and only if $\boldsymbol{u}$ is a right extension of a cyclic rotation of $\boldsymbol{p}$. [30]

Property 3.1.22. Assume string $\boldsymbol{w}$ and its seed $\boldsymbol{u}$. A cover of $\boldsymbol{u}$ is also a seed of $\boldsymbol{w}$. [17]

There exists another characterization of every seed of a string based on the following notion.

Definition 3.1.23 (Border seed)**.** Assume string $\boldsymbol{w}$, its factor $\boldsymbol{u}$ and its decomposition $\boldsymbol{w} = \boldsymbol{pys}$, where $\boldsymbol{y}$ is the longest possible string such that $\boldsymbol{u}$ is a border of $\boldsymbol{y}$. Then $\boldsymbol{u}$ is a *border seed* of $\boldsymbol{w}$ if $\boldsymbol{u}$ is a seed of $\boldsymbol{pus}$. [17]

Property 3.1.24. String $\boldsymbol{u}$ is a seed of string $\boldsymbol{w}$ if and only if $\boldsymbol{u}$ is a border seed of $\boldsymbol{w}$ and for any two consecutive positions i, j of $\boldsymbol{u}$ in $\boldsymbol{w}$ holds $|\boldsymbol{u}| \geq |j - i|$. [17]

3.1.3 Exact indexing strings

Finite automata as a formalism for string algorithms is considered in this dissertation thesis, therefore just algorithms for indexing and searching strings based on finite automata are mentioned in this section.

Based on the previously mentioned properties, techniques for searching factors and their positions are needed. Those can be provided by factor or suffix automata.

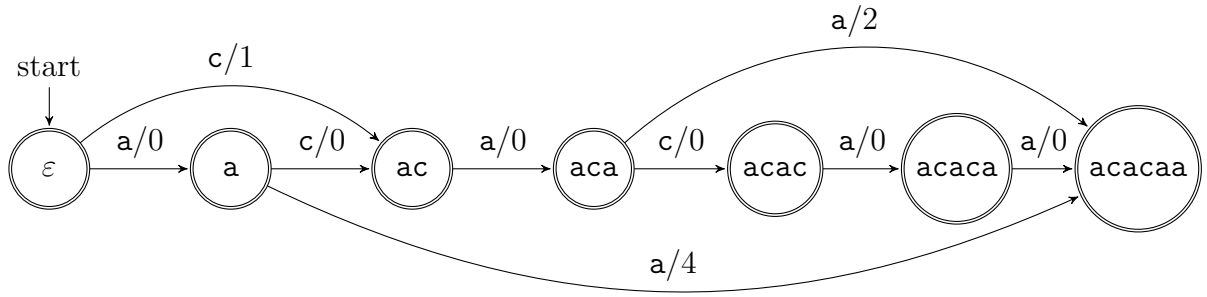

Figure 3.1: Factor transducer for string acacaa (Example 3.1.25)

3.1.3.1 Online construction of indexing automata

A factor automaton, i.e. automaton accepting the set of all factors of given string $\boldsymbol{w} \in A^*$ which is linear in the length of $\boldsymbol{w}$, can be constructed in linear time and space in the length of $\boldsymbol{w}$ (the time needed for its construction is $\mathcal{O}(|\boldsymbol{w}| \lg |A|)$). Crochemore [19] also gave an algorithm to construct in linear time and space factor transducer, i.e. automaton that outputs start position of the leftmost occurrence of given factor of $\boldsymbol{w}$ (starting by 0, in fact, it outputs the length of prefix of $\boldsymbol{w}$ not included in the given factor, sum of the output needs to be done to get the length). The construction is done online, i.e. the automaton is being constructed incrementally, symbol by symbol, in that sense that it is correct after each symbol being processed. It is explained using Example 3.1.25.

Example 3.1.25 (Factor transducer)**.** Given string $\boldsymbol{w} =$ acacaa, transition diagram of factor transducer for $\boldsymbol{w}$ is depicted at Figure 3.1. For factor caa, its output is $1+0+2 = 3$, as as length of prefix of $\boldsymbol{w}$ preceding the factor is 3. Note that output for factor ac is $0 + 0 = 0$ and there is no way to detect the other occurrence.

Note 3.1.26. Following some properties of covers and seeds, e.g., 3.1.3, 3.1.24, information about just the leftmost occurrence is insufficient.

It is also possible to construct suffix automaton (i.e. automaton accepting the set of all suffixes of given string $\boldsymbol{w}$) and suffix transducer as described in [19]. It is also able to read start position of given suffix of $\boldsymbol{w}$ as a sum of output of the suffix transducer. And as with factor transducer, it is also possible to get just that one position of the suffix.

3.1.3.2 Nondeterministic indexing automata

Algorithm to construct deterministic suffix or factor automaton from a nondeterministic one using subset construction was presented in [37, 38].

The basic idea is to construct simple nondeterministic automaton accepting for given string $\boldsymbol{w}$ all parts of $\boldsymbol{w}$ of some type (all the prefixes, suffixes, or factors of $\boldsymbol{w}$). Each state q_i is labelled by number i that way that labels of states on a path corresponding to $\boldsymbol{w}$ are growing sequence starting from 0 – label of state q_i correspond to i-th symbol of $\boldsymbol{w}$

Algorithm 3.1 Construction of a prefix automaton

Input: String $\boldsymbol{w} \in A^*_{\boldsymbol{w}}$.
Output: Prefix automaton $\mathcal{M} = (Q, A_{\boldsymbol{w}}, \delta, q_0, F)$ for $\boldsymbol{w}$.

1: $q_0 \leftarrow$ new state
2: **for all** $i, 1 \leq i \leq |\boldsymbol{w}|$ **do**
3: $\quad q_i \leftarrow$ new state
4: $\quad \delta(q_{i-1}, \boldsymbol{w}[i]) \leftarrow q_i$
5: **end for**
6: $Q \leftarrow \bigcup_{j=0}^{|\boldsymbol{w}|} \{q_j\}$
7: $F \leftarrow Q$

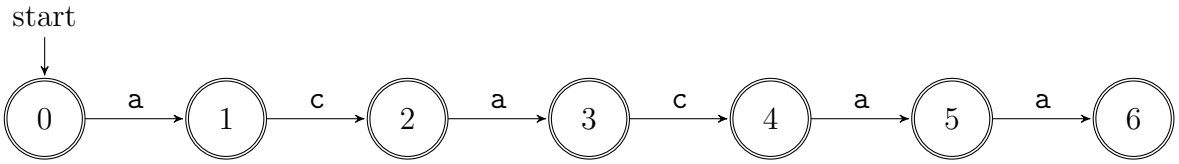

Figure 3.2: Prefix automaton for string `acacaa`

(i.e. to position i within $\boldsymbol{w}$). Such a nondeterministic automaton is not used as an index of $\boldsymbol{w}$ directly, instead, it is transformed to an equivalent deterministic automaton using subset construction. It contains labels of states of the nondeterministic automaton in its deterministic subsets, therefore for any string $\boldsymbol{u}$ accepted by the deterministic automaton, end positions of all occurrences of $\boldsymbol{u}$ in $\boldsymbol{w}$ may be extracted from d-subset of corresponding state.

Observation 3.1.27. Assuming deterministic suffix automaton $\mathcal{M} = (Q, A, \delta, q_0, F)$ constructed using subset construction from nondeterministic suffix automaton $\mathcal{M}_S$ for string $\boldsymbol{w} \in A^*$, for any state $q \in Q$ and for any string $\boldsymbol{u}$ of left language of q holds: position of $\boldsymbol{u}$ in $\boldsymbol{w}$ is i if and only if $\mathsf{depth}(e) = i$ when $e \in \mathsf{d}(q)$.[38]

Construction of nondeterministic prefix automaton is described in Alg. 3.1. See also Fig. 3.2. Number of each state equals to the length (and also end position) of corresponding prefix.

To construct nondeterministic suffix automaton, prefix automaton may be used, because prefixes are also factors. To accept the other factors, ε transitions must be added starting from initial state to the other states, in order to skip any prefix. As with prefix automaton, every state of factor automaton is final. The algorithm is described in Alg. 3.2.

Example 3.1.28 (Construction of factor automaton)**.** To construct factor automaton for string $\boldsymbol{w} =$ `acacaa` using Alg. 3.2, prefix automaton for the same string using Alg. 3.1 is constructed. Then ε transitions are added. As the numbering is preserved, after reading any factor of $\boldsymbol{w}$, label of accepting state corresponds to end position of the factor. See Fig. 3.3 for transition diagram of resulting automaton.

Algorithm 3.2 Construction of a nondeterministic factor automaton with ε transitions

Input: String $\boldsymbol{w} \in A_{\boldsymbol{w}}^*$.

Output: Factor automaton $\mathcal{M} = (Q, A_{\boldsymbol{w}}, \delta, q_0, F)$ for $\boldsymbol{w}$.

1: construct prefix automaton $\mathcal{M}_P = (Q_P, A_{\boldsymbol{w}}, \delta_P, q_0^P, F_P)$ using Alg. 3.1
2: $Q \leftarrow Q_P$
3: $\delta \leftarrow \delta_P$
4: $q_0 \leftarrow q_0^P$
5: $F \leftarrow F_P$
6: **for all** $i, 1 \leq i \leq |\boldsymbol{w}|$ **do**
7: $\quad \delta(q_0, \varepsilon) \leftarrow \{q_i\}$
8: **end for**

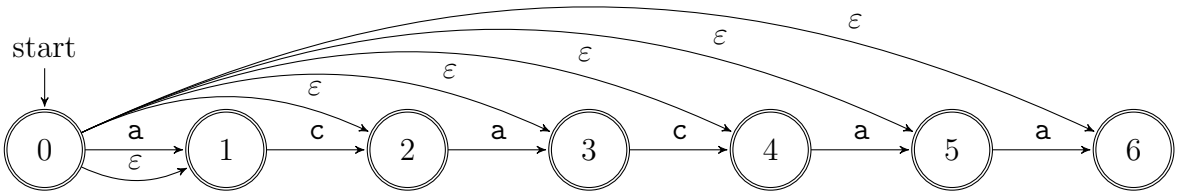

Figure 3.3: Factor automaton for string `acacaa` (Example 3.1.28)

Every suffix of string $\boldsymbol{w}$ is its factor having end position equal to the length of $\boldsymbol{w}$. Therefore, to construct nondeterministic suffix automaton, construction of nondeterministic factor automaton is used. The difference is that the only final state is that one corresponding to string $\boldsymbol{w}$. The algorithm for construction of nondeterministic suffix automaton is described in Alg. 3.3. See also Fig. 3.4.

Algorithm 3.3 Construction of a nondeterministic suffix automaton with ε transitions

Input: String $\boldsymbol{w} \in A_{\boldsymbol{w}}^*$.

Output: Suffix automaton $\mathcal{M} = (Q, A_{\boldsymbol{w}}, \delta, q_0, F)$ for $\boldsymbol{w}$.

1: construct $\mathcal{M}_P = (Q_F, A_{\boldsymbol{w}}, \delta_F, q_0^F, F_F)$ using Alg. 3.1
2: $Q \leftarrow Q_F$
3: $\delta \leftarrow \delta_F$
4: $q_0 \leftarrow q_0^F$
5: **for all** $i, 1 \leq i \leq |\boldsymbol{w}|$ **do**
6: $\quad \delta(q_0, \varepsilon) \leftarrow \{q_i\}$
7: **end for**
8: $F \leftarrow \{q_{|\boldsymbol{w}|}\}$

To be able to use the above indexing automata to retrieve all the positions of string $\boldsymbol{u}$ in string $\boldsymbol{w}$ in linear time in length of $\boldsymbol{u}$, deterministic automata are needed.

Prefix automaton is deterministic when constructed using Alg. 3.1. Factor or suffix

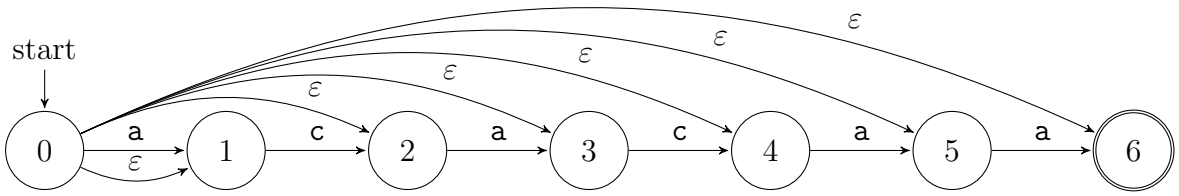

Figure 3.4: Suffix automaton for string acacaa (Example 3.1.29)

Algorithm 3.4 Elimination of ε transitions [28, 38]

Input: NFA with ε transitions $\mathcal{M}_E = (Q_E, A, \delta_E, q_0^E, F_E)$.
Output: NFA without ε transitions $\mathcal{M} = (Q, A, \delta, q_0, F)$ equivalent to $\mathcal{M}_E$.

1: $Q \leftarrow Q_E$
2: $q_0 \leftarrow q_0^E$
3: **for all** $q_1 \in Q, a \in A$ **do**
4: $\quad \delta(q_1, a) \leftarrow \bigcup_{q_2 \in \varepsilon\text{-}\mathsf{closure}(q_1)} \delta_E(q_2, a)$
5: **end for**
6: $F \leftarrow \{q : \varepsilon\text{-}\mathsf{closure}(q) \cap F_E \neq \emptyset, q \in Q\}$

automaton must be transformed to one without ε transitions (Alg. 3.4), then subset construction should be made (Def. 2.5.27).

Example 3.1.29. This example shows construction of deterministic suffix automaton for string $\boldsymbol{w}$ = acacaa. The first step is to construct nondeterministic suffix automaton $\mathcal{M}_E = (Q_N, \{\mathsf{a}, \mathsf{c}\}, \delta_E, q_0, F_E)$ with ε transitions using Alg. 3.3. Its transition diagram is depicted at Fig. 3.4.

As the next step, ε transitions should be removed using Alg. 3.4 and equivalent automaton $\mathcal{M}_N = (Q_N, \{\mathsf{a}, \mathsf{c}\}, \delta_N, q_0, F_N)$ is constructed. Epsilon closure of each state but the initial one (i.e. $q_i \in Q_N \setminus \{q_0\}$) contains just q_i. Therefore, outgoing transitions from states q_i remain unchanged. Epsilon closure of the initial state contains every other state, including final state. The initial state becomes final and outgoing non-ε transitions are added from q_0 to every other state but q_1. Transition diagram of automaton $\mathcal{M}_N$ is depicted at Fig. 3.5.

To transform M_N to equivalent deterministic suffix automaton $\mathcal{M} = (Q, \{\mathsf{a}, \mathsf{c}\}, \delta, q_0, F)$, subset construction (Def. 2.5.27) is used. Every state in Q contains states of Q_N in its d-subset. Every state of $\mathcal{M}$ that contains state from F_N in its d-subset (and therefore corresponds to a suffix of $\boldsymbol{w}$) becomes final. Transition diagram of $\mathcal{M}$ is depicted at Fig. 3.6.

Proposition 3.1.30 (Border detection [38])**.** Assuming string $\boldsymbol{w}$, all strings that are simultaneously prefixes and suffixes of $\boldsymbol{w}$ are accepted by a backbone of deterministic suffix automaton constructed for string $\boldsymbol{w}$.

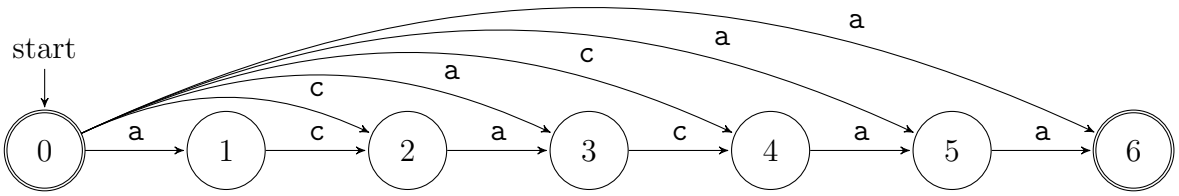

Figure 3.5: Nondeterministic suffix automaton without ε transitions for string acacaa (Example 3.1.29)

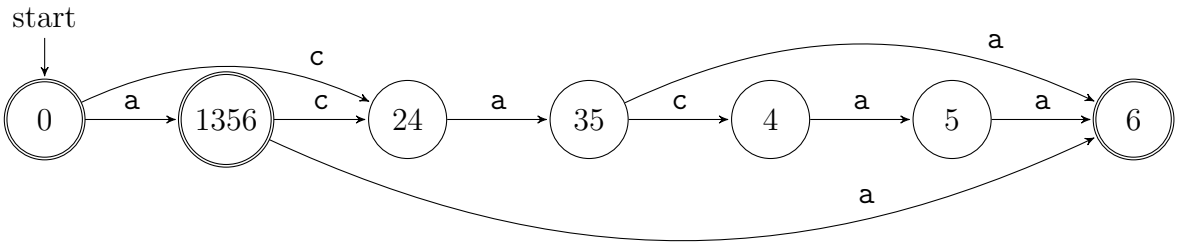

Figure 3.6: Deterministic suffix automaton for string acacaa (Example 3.1.29)

Proof. Following the Definition 2.5.38, in the backbone $\mathcal{M}_B = (Q_B, A, \delta_B, q_0, F_B)$ of a suffix automaton $\mathcal{M} = (Q, A, \delta, q_0, F)$, for every position i of string $\boldsymbol{w}$ there exists state $q_i \in Q_B$. Moreover, by $\delta_B(q_{i-1}, \boldsymbol{w}[i]) = q_i$ follows that for every string $\boldsymbol{u} \in A^*$ holds $\delta^*(q_0, \boldsymbol{u}) = q_{|\boldsymbol{u}|}$ if and only if $\boldsymbol{u}$ is a prefix of $\boldsymbol{w}$ (i.e. if $\boldsymbol{u}$ is not a prefix of $\boldsymbol{w}$, $\delta^*(q_0, \boldsymbol{u})$ is undefined). As the set of final states of $\mathcal{M}_B$ is defined as $F_B = \{q : q \in Q_B \cap F\}$, the string accepted by $\mathcal{M}_B$ must be a suffix of $\boldsymbol{w}$. Therefore, every string accepted by $\mathcal{M}_B$ is simultaneously prefix and suffix of $\boldsymbol{w}$. □

Example 3.1.31 (Border detection)**.** This example shows detection of borders of string $\boldsymbol{w}$ = acacaa. The first step is to construct a nondeterministic suffix automaton (its transition diagram is depicted at Figure 3.4). Then, equivalent deterministic suffix automaton is created using subset construction (its transition diagram is shown at Figure 3.6). Using just its backbone (i.e. part that corresponds to prefixes of string $\boldsymbol{w}$), final states are 1356 and 6. State 1356 corresponds to string a, which is a prefix (part of the backbone) and a suffix (the state is final), therefore a is a border of $\boldsymbol{w}$. The other state 6 corresponds to the string $\boldsymbol{w}$.

3.1.4 Distance metrics

Approximate string regularities are considered in this dissertation thesis. Some distance metrics are needed to measure distance between two strings. Moreover, there exist more approaches on considering some limit of a level of approximation.

One of existing approaches considers a given maximum percentage of errors. Another one allows up to given fixed k errors when comparing two strings. There also exists a hybrid one that allows given k errors in every window of r symbols. In this dissertation thesis, the fixed k number of errors for the whole string is considered as the maximum allowed distance.

To work with approximate regularities, level of similarity or difference of two strings must be known. The measure of difference between two strings is distance (Def. 2.3.22) determined by distance function.

As a measure of approximation in this dissertation thesis, the Hamming distance (introduced in [27]) is used. Besides Def. 2.3.23, it may be defined for strings $\boldsymbol{u}, \boldsymbol{v}; |\boldsymbol{u}| = |\boldsymbol{v}|$ as number of positions at which the strings differ. The Hamming distance is defined for strings of equal length only.

However, there exist other distance functions. In contrast to the Hamming distance, the Levenshtein and Damerau distance are defined for strings of different length. These are defined for strings $\boldsymbol{u}, \boldsymbol{v}$ as the minimal number of particular edit operations in $\boldsymbol{u}$ to make it equal to $\boldsymbol{v}$. The edit operations for the Levenshtein distance (introduced in [35]) are:

- symbol replace,
- symbol insert,
- symbol delete.

The edit operations for the Damerau distance (the idea was introduced in [22]) are:

- symbol replace,
- symbol insert,
- symbol delete,
- transposition of two adjacent symbols.

In fact, the Levenshtein distance is a special case of the Damerau one (transposition is not allowed) and the Hamming distance is a special case of the Levenshtein one (insert and delete not allowed).

Each edit operation may have associated some cost, or cost may be associated with particular symbols. Then the distance of two strings is defined as the minimal cost of edit operations needed to transform one string into another. Example of definition and application can be found in [43], another definition and algorithm to compute the distance can be found in [21].

The above distances consider edit operations for just single symbols. There exist distances related to change of a factor as an edit operation. Motivated by problems in computational biology, parts of chromosomes of two organisms are considered to be similar if factor of one chromosome if a reversion of a factor of the other chromosome. The related

edit operation is *reversal* of a factor and the distance is called *reversal distance* [33]. To measure distance between strings $\boldsymbol{u}, \boldsymbol{v}$, another possible distance may be defined as the minimal number of operations insert and block (i.e. factor) move; a block move represents common factor in strings $\boldsymbol{u}, \boldsymbol{v}$ [46]. Another measure of approximation may be based on function $\mathsf{diff}(\boldsymbol{u}, \boldsymbol{v})$ [23]; this function is defined as the minimal number of symbol positions in $\boldsymbol{u}$ such that, if ignored, every factor of $\boldsymbol{u}$ is also a factor of $\boldsymbol{v}$. To make it a metric, the distance must be computed as symmetric and as satisfying the triangle inequality, such as ([23])

$$\mathsf{d} = \log_2((\mathsf{diff}(\boldsymbol{u}, \boldsymbol{v}) + 1)(\mathsf{diff}(\boldsymbol{v}, \boldsymbol{u}) + 1))$$

However, distances considering edit operations with strings instead of single symbols are not subject of this dissertation thesis.

3.1.5 Approximate regularities

Unlike for exact covers, there are no already known properties of approximate covers (besides properties that directly follow from definitions of approximate cover) to use as a base of an algorithm for searching them.

3.1.5.1 Approximate seeds

Property 3.1.32. When searching for k-approximate seed $\boldsymbol{u}$ of string $\boldsymbol{w}$, covering a left extension of $\boldsymbol{w}$ can be done as matching of a suffix $\boldsymbol{u}[i..|\boldsymbol{u}|]$ with a prefix $\boldsymbol{w}[1..j]$ ($j = |\boldsymbol{u}| - i + 1$) with Hamming distance at most k. Similar property holds for covering a right extension of $\boldsymbol{w}$.[13, 14]

Observation 3.1.33. Lemma 3.1.15 holds similarly for k-approximate seed $\boldsymbol{u}$ of $\boldsymbol{w}$ with Hamming distance (idea in [13]): Having a k-approximate end set $C = \{i_1, i_2, \ldots, i_{|C|}\}$ of $\boldsymbol{u}$ in $\boldsymbol{w}$, there must exist some prefix of $\boldsymbol{u}$ of length at least $|\boldsymbol{w}| - i_{|C|}$ equal to some k-approximate suffix of $\boldsymbol{w}$ of the same length and there must exist some suffix of $\boldsymbol{u}$ of length at least $i_1 - |v|$ equal to some k-approximate prefix of $\boldsymbol{w}$ of the same length.

Proof. (Idea) Assuming k-approximate factor $\boldsymbol{v}$ of $\boldsymbol{w}$ having $\boldsymbol{u}$ as its k-approximate cover, string $\boldsymbol{w}$ may be written as $\boldsymbol{pvs}$. Assuming extension $\boldsymbol{z}$ of $\boldsymbol{w}$ such that $\boldsymbol{u}$ is a k-approximate cover of $\boldsymbol{z}$, string $\boldsymbol{z}$ may be written as $\boldsymbol{xwy} = \boldsymbol{xpvsy}$. Suppose that $\boldsymbol{z}$ is of minimal possible length. To $\boldsymbol{u}$ be a k-approximate cover of $\boldsymbol{z}$, there must exist some strings $\boldsymbol{x}', \boldsymbol{p}'$ such that $\mathsf{H}(\boldsymbol{x}'\boldsymbol{p}', \boldsymbol{xp}) \leq k$ and $\boldsymbol{x}'\boldsymbol{p}'$ must be either equal to $\boldsymbol{u}$ or a prefix of $\boldsymbol{u}$; there must also exist some strings $\boldsymbol{s}', \boldsymbol{y}'$ such that $\mathsf{H}(\boldsymbol{s}'\boldsymbol{y}', \boldsymbol{sy}) \leq k$ and $\boldsymbol{s}'\boldsymbol{y}'$ must be either equal to $\boldsymbol{u}$ or a suffix of $\boldsymbol{u}$. The end set C of k-approximate end-positions contains at least one element, as $\boldsymbol{u}$ is a factor of $\boldsymbol{w}$. The factor of $\boldsymbol{w}$ determined by the end set C is equal to the factor $\boldsymbol{v}$, it is the longest factor of $\boldsymbol{w}$ for which holds that $\boldsymbol{u}$ is its k-approximate cover. Therefore, the k-approximate suffix of $\boldsymbol{w}$ that is necessary to be equal to the prefix of $\boldsymbol{u}$ has to be of length at least $|\boldsymbol{s}| = |\boldsymbol{w}| - i_{|C|}$ and the k-approximate prefix of $\boldsymbol{w}$ that is necessary to be equal to the suffix of $\boldsymbol{u}$ has to be of length at least $|\boldsymbol{p}| = i_1 - |v|$. □

3.1.6 Approximate indexing strings

Finite automata as a formalism for string algorithms is considered in this dissertation thesis, therefore just algorithms for approximate indexing and searching strings based on finite automata are mentioned in this section. Computation of approximate string regularities in this dissertation thesis is focused on Hamming distance.

Like for exact covers and seeds, searching factors, prefixes and suffixes, and their positions is needed for computing approximate covers and seeds. Those can be done with factor or suffix automata for particular distance function.

The minimal deterministic suffix automaton that accepts the set of all k-approximate suffixes of given string with Hamming distance was defined by Crochemore, Epifanio, Gabriele, and Mignosi [20]. However, this automaton is not capable of giving the list of all occurrences of some k-approximate suffix in indexed string.

Way to obtain an automaton for indexing all k-approximate suffixes of given string $\boldsymbol{w}$ including their positions is using construction of nondeterministic k-approximate suffix automaton [38]. The idea consists of the following steps:

1. Construct a FA $\mathcal{M}$ that accepts all strings similar to $\boldsymbol{w}$ under some particular distance function D (i.e. every string $\boldsymbol{u}$ such that $D(\boldsymbol{u}, \boldsymbol{w}) \leq k$). This construction is formalised in Algorithm 3.5.

2. Based on the previous step, construct an NFA $\mathcal{M}^k_{S^{\mathrm{N}}}$ that accepts all k-approximate suffixes of $\boldsymbol{w}$ under D. This step is formalised in Algorithm 3.7.

Left language of each final state of $\mathcal{M}^k_{S^{\mathrm{N}}}(\boldsymbol{w})$ contains just k-approximate suffixes of $\boldsymbol{w}$, left language of every state of $\mathcal{M}^k_{S^{\mathrm{N}}}(\boldsymbol{w})$ contains just k-approximate factors of $\boldsymbol{w}$. Moreover, due to regular structure of $\mathcal{M}^k_{S^{\mathrm{N}}}(\boldsymbol{w})$, depth of each state q corresponds to a position of all strings of left language of q in $\boldsymbol{w}$, level of q corresponds to distance $D(\boldsymbol{u}, \boldsymbol{w})$ of all strings $\boldsymbol{u}$ of left language of q.

To obtain indexing structure, $\mathcal{M}^k_{S^{\mathrm{N}}}(\boldsymbol{w})$ can be transformed to deterministic finite automaton $\mathcal{M}^k_{S^{\mathrm{D}}}(\boldsymbol{w} = (Q_{\mathrm{D}}, A_{\boldsymbol{w}}, \delta_{\mathrm{D}}, q_{0\mathrm{D}}, F_{\mathrm{D}})$ using subset construction. As a property of any DFA, after reading a string $\boldsymbol{u}$, just one state $q \in Q_{\mathrm{D}}$ is reached if $\boldsymbol{u}$ is a k-approximate factor of $\boldsymbol{w}$, i.e. $\delta^*_{\mathrm{D}}(q_{0\mathrm{D}}, \boldsymbol{u}) = q$ (and no state is reached if $\boldsymbol{u}$ is not a k-approximate factor of $\boldsymbol{w}$). As d-subset of q is constructed as a subset of states of $\mathcal{M}^k_{S^{\mathrm{N}}}(\boldsymbol{w})$, each element of $r \in \mathsf{d}(q)$ corresponds to all strings $\boldsymbol{u}$ of left language of q: depths of all such elements correspond to end set, level of each element r is equal to l such that $\boldsymbol{u}$ l-approximately occurs at position $\mathsf{depth}(r)$.

Lemma 3.1.34. For a deterministic suffix automaton $\mathcal{M}^k_{S^{\mathrm{D}}}(\boldsymbol{w}) = (Q_{\mathrm{D}}, A_{\boldsymbol{w}}, \delta_{\mathrm{D}}, q_{0\mathrm{D}}, F_{\mathrm{D}})$ created using subset construction from nondeterministic k-approximate suffix automaton $\mathcal{M}^k_{S^{\mathrm{N}}}(\boldsymbol{w})$ (created using Alg. 3.3 and 3.4) and for its state $q_i \in Q_{\mathrm{D}}$ such that $\forall r \in \mathsf{d}(q_i)$: $\mathsf{level}(r) > 0$ holds that any successor of q_i cannot contain element r^j such that $\mathsf{level}(r^j) = 0$.

Proof. It holds from property of transitions of $\mathcal{M}^k_{S^{\mathrm{N}}}(\boldsymbol{w}) = (Q_{\mathrm{N}}, A_{\boldsymbol{w}}, \delta_{\mathrm{N}}, q_{0\mathrm{N}}, F_{\mathrm{N}})$: for all successors $r^j \in Q_{\mathrm{N}}$ of $r^i \in Q_{\mathrm{N}}$ holds that $\mathsf{level}(r^i) \leq \mathsf{level}(r^j)$. □

Algorithm 3.5 Construction of an NFA $\mathcal{M}$ with possible ε transitions accepting all strings with distance at most k under some distance function D from string $\boldsymbol{w}$ (idea found in [38])

Input: Input string $\boldsymbol{w} \in A^+$, k, and D.
Output: Finite automaton $\mathcal{M} = (Q, A_w, \delta, q_0^0, F)$.

1. Define $Q = \{q_i^j : 0 \leq i \leq |\boldsymbol{w}|, 0 \leq j \leq k\}$.
2. Define $F = \{q_{|\boldsymbol{w}|}^j : 0 \leq j \leq k\}$.
3. Define $\delta(q_{i-1}^j, \boldsymbol{w}[i]) = \{q_i^j\}$ for all i, j such that $1 \leq i \leq |\boldsymbol{w}|, 0 \leq j \leq k$.
4. The rest depends on the distance function D:
   a) Hamming distance: for all a, i, j such that $1 \leq i \leq |\boldsymbol{w}|, 1 \leq j \leq k, a \in A \backslash \{\boldsymbol{w}[i]\}$:
   $$\delta(q_{i-1}^{j-1}, a) = \{q_i^j\}$$
   b) Levenshtein distance: take Step 4a and for all a, i, j such that $1 \leq i \leq |\boldsymbol{w}|, 1 \leq j \leq k, a \in A\}$ add:
   $$\delta(q_{i-1}^{j-1}, \varepsilon) = \{q_i^j\}$$
   $$\delta(q_0^{j-1}, a) = \{q_0^j\}$$
   $$\delta(q_i^{j-1}, a) = \{q_i^j\}$$
   c) Damerau distance: take Step 4b and for all i, j such that $1 \leq i \leq |\boldsymbol{w}| - 1, 1 \leq j \leq k$ add:
   $$Q \leftarrow Q \cup \{q_{i-1,i+1}^j\}$$
   $$\delta(q_{i-1}^{j-1}, \boldsymbol{w}[i+1]) = \{q_{i-1,i+1}^j\}$$
   $$\delta(q_{i-1,i+1}^j, \boldsymbol{w}[i]) = \{q_i^j\}$$
5. Remove states q_i^j from Q such that there exists no string $\boldsymbol{u} \in A^*$ such that $q_i^j \in \delta^*(q_0^0, \boldsymbol{u})$.

Algorithm 3.6 Construction of a nondeterministic k-approximate prefix automaton $\mathcal{M}$ for string $\boldsymbol{w}$ under some distance function D [38]

Input: Input string $\boldsymbol{w} \in A^+$, k, and D.
Output: k-approximate prefix automaton $\mathcal{M} = (Q, A, \delta, q_0, F)$ for $\boldsymbol{w}$, D, and k.

1. Construct FA $\mathcal{M}_B = (Q_B, A, \delta_B, q_0^B, F_B)$ using Algorithm 3.5.
2. Define $Q = Q_B, \delta = \delta_B, q_0 = q_0^B$.
3. Define $F = Q$.

Algorithm 3.7 Construction of a nondeterministic k-approximate suffix automaton $\mathcal{M}$ with ε transitions for string $\boldsymbol{w}$ under some distance function D [38]

Input: Input string $\boldsymbol{w} \in A^+$, k, and D.
Input: k-approximate suffix automaton $\mathcal{M} = (Q, A, \delta, q_0, F)$ for $\boldsymbol{w}$, D, and k.

1. Construct k-approximate prefix automaton $\mathcal{M}_B = (Q_P, A, \delta_P, q_0^P, F_P)$ for $\boldsymbol{w}$, D, and k using Algorithm 3.6.
2. Define $Q = Q_P, \delta = \delta_P, q_0 = q_0^P$.
3. For each i such that $1 \leq i \leq |\boldsymbol{w}|$: having states q_i^0 as defined in Algorithm 3.5, define $\delta(q_0, \varepsilon) = q_i^0$.
4. For each j such that $0 \leq j \leq k$: having states $q_{|\boldsymbol{w}|}^j$ as defined in Algorithm 3.5, define $F = \{q_{|\boldsymbol{w}|}^j\}$.

Lemma 3.1.35 (Size of nondeterministic suffix FA)**.** A k-approximate nondeterministic suffix automaton $\mathcal{M}_S = (Q, A, \delta, q_0, F)$ for string $\boldsymbol{w}$ and maximum Hamming distance k contains

$$(|\boldsymbol{w}| + 1) \cdot (k + 1) - \frac{k^2 + k}{2}$$

states and

$$|A| \cdot \left(|\boldsymbol{w}| \cdot (k + 1) - 1 + \frac{k - k^2}{2} \right) + |\boldsymbol{w}| - k + 1$$

transitions.

Proof. The automaton consists of layers of states $q^{(i)}$ for each level i. Layer 0 contains $|\boldsymbol{w}| + 1$ states. Each layer i contains one state less in comparison with layer $i - 1$, thus it contains $|\boldsymbol{w}| - i + 1$ and layer k contains $|\boldsymbol{w}| - k + 1$ states. Therefore, the number of states of $\mathcal{M}_S$ is $(|\boldsymbol{w}| + 1) \cdot (k + 1) - \frac{k^2+k}{2}$.

The automaton contains $|A|$ transitions from each state, with some exceptions. There are $k+1$ final states having no successor. In layer k, each state has only one successor. From

the initial state, there are $|\boldsymbol{w}|$ transitions defined to states in layer 0 having $\mathsf{level}(q^{(0)}) = 0$ and $|\boldsymbol{w}| \cdot (|A| - 1)$ transitions to states in layer 1 having $\mathsf{level}(q^{(1)}) = 1$. Thus in $\mathcal{M}_S$ there are $|Q| \cdot |A| + |\boldsymbol{w}| \cdot |A| - (k+1) \cdot |A| - (|\boldsymbol{w}| - k + 1) \cdot (|A| - 1) = |A| \cdot (|Q| - 2) + |\boldsymbol{w}| - k + 1$ transitions, where $|Q| = (|\boldsymbol{w}| + 1) \cdot (k+1) - \frac{k^2+k}{2}$. □

3.2 Previous results and related work

In this section, already-known algorithms for computing covers and seeds are given. Some of the ideas are also used in our contribution in Chapters 4 and 5.

3.2.1 Exact covers

Voráček et al. [49, 47, 50, 25] introduced finite-automata-based algorithm for computing all covers of a generalized string (Definition 2.3.6). This algorithm is applicable for searching covers in a string, as any string is a special case of generalized string. To compute all covers of a generalized string $\tilde{\boldsymbol{w}}$, firstly they construct a deterministic automaton accepting all the borders of $\tilde{\boldsymbol{w}}$. The generalized border automaton is constructed from a deterministic generalized suffix automaton for $\tilde{\boldsymbol{w}}$: state to the generalized border automaton is added from the deterministic generalized suffix automaton if it is a successor of an existing state q of the border automaton for symbol $a \in \tilde{\boldsymbol{w}}[i]$ where i is depth of q. Therefore, left language of q contains only prefixes of $\tilde{\boldsymbol{w}}$. After complete construction of the generalized border automaton, the following is done for each final state q (that represents generalized border of $\tilde{\boldsymbol{w}}$):

1. d-subset $\mathsf{d}(q)$ (defined as a set without order in this case) is converted to a sorted list C,
2. C is checked whether property stated in Note 3.1.4 holds,
    - if true, left language of q is extracted and added to the partially computed set of all covers of $\tilde{\boldsymbol{w}}$.

3.2.2 Approximate covers

Sim, Park, Kim and Lee [44] gave polynomial-time algorithms for the problem of searching all the approximate covers. They consider Levenshtein and weighted Levenshtein distance. Unlike for Levenshtein distance, for the weighted one different costs of each operation and for each particular symbol are considered, e.g., given by penalty matrix. (However, the paper is in Korean.) The algorithms were described, implemented and experimental results presented by Christodoulakis, Iliopoulos, Park and Sim [14]. They included the Hamming distance and consider two problems:

1. Given strings $\boldsymbol{w}$ and $\boldsymbol{u}$, and distance function D. Find the minimum l such that $\boldsymbol{u}$ is an l-approximate cover of $\boldsymbol{w}$ (smallest distance of a cover).

2. Given string $\boldsymbol{w}$, find a factor $\boldsymbol{u}$ of $\boldsymbol{w}$ such that $\boldsymbol{u}$ is a k-approximate cover of $\boldsymbol{w}$ under given distance D and there is no factor of $\boldsymbol{w}$ that is an l-approximate cover of $\boldsymbol{w}$ and $l < k$ (restricted k-approximate cover).

Problem 1 is solved in two steps:

1. compute distance between $\boldsymbol{u}$ and every factor of $\boldsymbol{w}$; for Hamming distance this means character-by character comparison of $|\boldsymbol{w}| - |\boldsymbol{u}| + 1$ factors of length $|\boldsymbol{u}|$;

2. using previously computed distances, incrementally get the minimum distance l such that $|\boldsymbol{u}|$ is an l-approximate cover of each prefix of $\boldsymbol{w}$; it is computed using sliding window of length $|\boldsymbol{u}|$.

Time complexity of the solution is $\mathcal{O}(|\boldsymbol{w}| \cdot |\boldsymbol{u}|)$ for Hamming and Levenshtein distance.

Problem 2 is considered for relative distance function, i.e., weighted Levenshtein distance, or Levenshtein or Hamming distance divided by string length (so-called relative distance). This problem can be solved applying solution for Problem 1 for every factor of $\boldsymbol{w}$ in $\mathcal{O}(|\boldsymbol{w}|^4)$ time for Levenshtein and weighted Levenshtein distance, and $\mathcal{O}(|\boldsymbol{w}|^3)$ for Hamming distance.

3.2.3 Exact seeds

Voráček et al. [50, 47, 51, 25] presented algorithm for searching all seeds of a generalized string based on finite automata. As in case of covers, this algorithm is applicable for searching all seeds in a string. To compute all covers of a generalized string $\tilde{\boldsymbol{w}}$, deterministic suffix automaton $\mathcal{M}(Q, A, \delta, q_0, F)$ for $\tilde{\boldsymbol{w}}$ is constructed. For each state $q \in Q$, the following is done:

1. d-subset $\mathsf{d}(q)$ (defined as a set without order in this case) is converted to a sorted list C corresponding to end set of strings from left language of q,

2. based on $C[1]$ and $C[|C|]$, minimum length of possible corresponding seed $\boldsymbol{u}$ is computed as maximum of:

   - for any $2 \leq j \leq |C|$ the maximum of $C[j] - C[j-1]$ (consecutive end positions);
   - to match prefix of $\tilde{\boldsymbol{w}}$ before the leftmost position of $\boldsymbol{u}$, its length of seed is at least $\lfloor \frac{C[1]}{2} \rfloor + 1$, because the leftmost start position of $\boldsymbol{u}$ in $\tilde{\boldsymbol{w}}$ is $C[1] - |\boldsymbol{u}| + 1$;
   - to match suffix of $\tilde{\boldsymbol{w}}$ behind the rightmost position of $\boldsymbol{u}$, its length of seed is at least $|\tilde{\boldsymbol{w}}| - C[|C|] + 1$;

3. left language extraction of q is done and for each string $\boldsymbol{u}$ longer than the computed minimum length, it is checked whether both of the following holds:

   - if exists a prefix $\boldsymbol{v}$ of $\boldsymbol{u}$ long enough such that it is a suffix of $\tilde{\boldsymbol{w}}$ (checked using transitions of $\mathcal{M}$, the condition is satisfied $\delta^*(q_0, \boldsymbol{v}) = q' \in F$),

- if exists a suffix $\boldsymbol{v}'$ of $\boldsymbol{u}$ long enough such that it is a prefix of $\tilde{\boldsymbol{w}}$ and there exists sequence of transitions $\delta^*(q_0, \boldsymbol{v}') = q''$ such that some element of $\mathsf{d}(q'')$ has depth equal to $C[1] - |\boldsymbol{u}|$ (the author also proposed to use a deterministic suffix automaton for reversion of input string);

4. if the previous is true, $\boldsymbol{u}$ is a seed of $\tilde{\boldsymbol{w}}$.

3.2.4 Approximate seeds

Christodoulakis, Iliopoulos, Park and Sim [13] presented algorithms for computing the smallest distance of a seed and for searching a restricted approximate seed with minimum possible distance – for Hamming, Levenshtein and weighted Levenshtein distance. They consider three problems:

1. Given strings $\boldsymbol{w}$ and $\boldsymbol{u}$, and distance function D. Find the minimum l such that $\boldsymbol{u}$ is an l-approximate seed of $\boldsymbol{w}$ (smallest distance of a seed).

2. Given string $\boldsymbol{w}$, find a factor $\boldsymbol{u}$ of $\boldsymbol{w}$ such that $\boldsymbol{u}$ is a k-approximate seed of $\boldsymbol{w}$ under given distance D and there is no factor of $\boldsymbol{w}$ that is an l-approximate seed of $\boldsymbol{w}$ and $l < k$ (restricted k-approximate seed).

3. Given string $\boldsymbol{w}$, find a string $\boldsymbol{u}$ such that $\boldsymbol{u}$ is a k-approximate seed of $\boldsymbol{w}$ under given distance D and there is no factor of $\boldsymbol{w}$ that is an l-approximate seed of $\boldsymbol{w}$ and $l < k$ (k-approximate seed).

Problem 1 is solved in two steps:

1. compute distance between $\boldsymbol{u}$ and every factor of $\boldsymbol{w}$; for Hamming distance this means character-by character comparison of $|\boldsymbol{w}| - |\boldsymbol{u}| + 1$ factors of length $|\boldsymbol{u}|$ plus distance of every prefix of $\boldsymbol{u}$ and suffix of $\boldsymbol{w}$, and vice versa;

2. using previously computed distances, incrementally get the minimum distance l such that $|\boldsymbol{u}|$ is an l-approximate seed of each prefix of $\boldsymbol{w}$; it is computed using sliding window of length $|\boldsymbol{u}|$. When considering prefix of $\boldsymbol{w}$ shorter than $|\boldsymbol{u}|$, the previously computed distance between a suffix of $\boldsymbol{u}$ and the prefix of $\boldsymbol{w}$ is used.

Time complexity of the solution is $\mathcal{O}(|\boldsymbol{w}| \cdot |\boldsymbol{u}| + |\boldsymbol{u}|^2)$ [14] for Hamming and Levenshtein distance.

Problem 2 is considered for relative distance function. This problem can be solved applying solution for Problem 1 for every factor of $\boldsymbol{w}$ in $\mathcal{O}(|\boldsymbol{w}|^4)$ time for Levenshtein and weighted Levenshtein distance, and $\mathcal{O}(|\boldsymbol{w}|^3)$ for Hamming distance.

It is proved that Problem 3 for weighted Levenshtein distance is NP-hard. It is done by reduction from the shortest common supersequence problem.

Chapter 4

Searching covers of strings

In computation of covers, two main problems have been considered in literature: one cover with some property (the shortest one or approximate one having the smallest distance) or all covers. Here, the more general problem is considered. It is shown how to solve the problems of all covers (Definition 2.6.1), all restricted smallest distance k-approximate covers (Definition 2.6.3), and all smallest distance k-approximate covers (Definition 2.6.2) – given some string and maximum distance under Hamming distance. It is then possible to compute and select one cover with particular property.

4.1 Basic facts and properties

Lemma 4.1.1. Assuming strings $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in A^*$ such that $\boldsymbol{u}$ is a factor of $\boldsymbol{v}$ and $\boldsymbol{u}$ is not a factor of $\boldsymbol{w}$ then $\boldsymbol{v}$ cannot be a factor of $\boldsymbol{w}$.

Proof. When $\boldsymbol{u}$ is not a factor of $\boldsymbol{w}$ then any string $\boldsymbol{v}$ containing $\boldsymbol{u}$ as its factor cannot be a factor of $\boldsymbol{w}$. □

Lemma 4.1.2. Assuming Hamming distance for k-approximate factors $\boldsymbol{p}, \boldsymbol{v}$ of $\boldsymbol{w}$ having the leftmost k-approximate positions in $\boldsymbol{w}$: $i_{\boldsymbol{p}}, i_{\boldsymbol{v}}$, respectively, holds: if $\boldsymbol{p}$ is a prefix of $\boldsymbol{v}$ and $|\boldsymbol{v}| > |\boldsymbol{p}|$ then for positions $i_{\boldsymbol{p}}, i_{\boldsymbol{v}}$ holds

$$i_{\boldsymbol{v}} > i_{\boldsymbol{p}}$$

Proof. There occurs a factor $\boldsymbol{y}$ of $\boldsymbol{w}$ at position $i_{\boldsymbol{p}}$ such that $\mathsf{H}(\boldsymbol{p}, \boldsymbol{y}) \leq k$, i.e., there are at most k operations replace needed to convert $\boldsymbol{p}$ to $\boldsymbol{y}$ and there exists no such factor $\boldsymbol{y}'$ of $\boldsymbol{w}$ that $\mathsf{H}(\boldsymbol{p}, \boldsymbol{y}') \leq k$ having its position left of $i_{\boldsymbol{p}}$. As $\boldsymbol{p}$ is a prefix of $\boldsymbol{v}$, $i_{\boldsymbol{v}} \geq i_{\boldsymbol{u}}$ must hold. Moreover, as $|\boldsymbol{v}| > |\boldsymbol{p}|$, $i_{\boldsymbol{v}} > i_{\boldsymbol{p}}$ must hold. □

4.1.1 Approximate covers

Some properties of exact covers need to be modified to fit approximate covers. There are also some properties that hold for k-approximate covers but not for restricted k-approximate covers.

Lemma 4.1.3. Given some distance function D and integer k, a k-approximate cover $\boldsymbol{u}$ of string $\boldsymbol{w}$ with respect to D is a k-approximate border of $\boldsymbol{w}$ with respect to D.

Proof. When $\boldsymbol{v}$ is an approximate cover of $\boldsymbol{w}$ with respect to k and D, there must exist some string $\boldsymbol{p}$ in a pair from $\mathsf{L}^{\mathrm{c}}_{\mathsf{H}^k}(\boldsymbol{w})$ (the set $\mathsf{L}^{\mathrm{c}}_{\mathsf{H}^k}(\boldsymbol{w})$ as defined in Definition 2.4.16) that is a prefix of $\boldsymbol{w}$ and for that $D(\boldsymbol{p}, \boldsymbol{v}) \leq k$ holds. There must also exist some string $\boldsymbol{s}$ in a pair from $\mathsf{L}^{\mathrm{c}}_{\mathsf{H}^k}(\boldsymbol{w})$ that is a suffix of $\boldsymbol{w}$ and for that $D(\boldsymbol{s}, \boldsymbol{v}) \leq k$ holds. Therefore, $\boldsymbol{v}$ is a k-approximate border of $\boldsymbol{w}$ with respect to D. □

Observation 4.1.4. Given maximum distance k under some distance function D, restricted k-approximate cover $\boldsymbol{v}$ of string $\boldsymbol{w}$ with respect to D is not necessarily a restricted k-approximate border of $\boldsymbol{w}$ with respect to D.

Example 4.1.5. Let us have string $\boldsymbol{w} = \mathtt{bbababb}$ and let us consider its restricted 1-approximate cover $\boldsymbol{v} = \mathtt{aba}$ under maximum Hamming distance $k = 1$. $\mathtt{bba}$ is the prefix of $\boldsymbol{w}$, $\mathtt{abb}$ is the suffix of $\boldsymbol{w}$. $\mathsf{H}(\mathtt{aba}, \mathtt{bba}) = 1$ and $\mathsf{H}(\mathtt{aba}, \mathtt{abb}) = 1$, thus $\boldsymbol{v}$ is a 1-approximate border of w. As $\mathsf{H}(\mathtt{bba}, \mathtt{abb}) = 2 > k = 1$, $\boldsymbol{v}$ is not a restricted 1-approximate border of $\boldsymbol{w}$.

Observation 4.1.6. Given maximum distance k under some distance function D, every k-approximate cover $\boldsymbol{v}$ of string $\boldsymbol{w}$ is simultaneously a k-approximate prefix and a k-approximate suffix of $\boldsymbol{w}$.

Proof. It follows from Definition 2.4.16. Having the set $\mathsf{L}^{\mathrm{c}}_{\mathsf{H}^k}(\boldsymbol{w})$ from the definition, in order to construct $\boldsymbol{w}$ by concatenations and superpositions of copies of strings from $\mathsf{L}^{\mathrm{c}}_{\mathsf{H}^k}(\boldsymbol{w})$, it is necessary to "cover" a prefix of $\boldsymbol{w}$, therefore there must exist a prefix $\boldsymbol{p}$ of $\boldsymbol{w}$ such that $\boldsymbol{p}$ is in pair from $\mathsf{L}^{\mathrm{c}}_{\mathsf{H}^k}(\boldsymbol{w})$, so $D(\boldsymbol{v}, \boldsymbol{p}) \leq k$, i.e., $\boldsymbol{v}$ is a k-approximate prefix of $\boldsymbol{w}$. Similar property holds for suffix of $\boldsymbol{w}$. □

Lemma 4.1.7. Given maximum distance k under Hamming distance function H, string $\boldsymbol{v}$ is a nontrivial restricted k-approximate cover of string $\boldsymbol{w}$ if and only if

1. $\boldsymbol{v}$ is an exact factor of $\boldsymbol{w}$, and
2. $\boldsymbol{v}$ is simultaneously a k-approximate prefix and a k-approximate suffix of $\boldsymbol{w}$, and
3. for a k-approximate end set $C = \{i_1, i_2, \ldots, i_{|C|}\}$ of $\boldsymbol{v}$ holds:

$$\forall j = 2, 3, \ldots, |C| : i_j - i_{j-1} \leq |\boldsymbol{v}| \tag{4.1}$$

Proof. (if) Assuming $\boldsymbol{v}$ is a restricted k-approximate cover of $\boldsymbol{w}$ under Hamming distance and $\boldsymbol{v} \neq \boldsymbol{w}$, there exists the set $\mathsf{L}^{\mathrm{c}}_{\mathsf{H}^k}(\boldsymbol{w})$ from Definition 2.4.16. Without loss of generality, instead of the set $\mathsf{L}^{\mathrm{c}}_{\mathsf{H}^k}(\boldsymbol{w})$, list B' may be considered – the list $B' = [\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_{|B'|}]$ contains all the strings from $\mathsf{L}^{\mathrm{c}}_{\mathsf{H}^k}(\boldsymbol{w})$ ordered by their end positions in $\boldsymbol{w}$ from the leftmost to the rightmost one. Following Observation 4.1.6, $\boldsymbol{v}$ must be a k-approximate prefix and a k-approximate suffix of $\boldsymbol{w}$. As for every string $\boldsymbol{u} \in B'$ holds $D(\boldsymbol{v}, \boldsymbol{u}) \leq k$, ordered set $C' = \{j_1, j_2, \ldots, j_{|C'|}\}$, where $j_i \in C'$ is equal to position of $\boldsymbol{u}_i \in B'$ in $\boldsymbol{w}$, is a k-approximate end set of $\boldsymbol{v}$, i.e., $C' = C$ in the lemma. Because D is defined for strings of equal length, $\forall \boldsymbol{u} \in B' : |\boldsymbol{u}| = |\boldsymbol{v}|$. In order to construct $\boldsymbol{w}$ just by concatenations and superpositions of strings from B', there must be no gaps between subsequent strings of B', i.e., (4.1) must hold.

(only if) Assume a factor $\boldsymbol{v}$ of $\boldsymbol{w}$ ($\boldsymbol{v} \neq \boldsymbol{w}$), and that for a k-approximate end set $C = \{i_1, i_2, \ldots, i_{|C|}\}$ of $\boldsymbol{v}$ holds (4.1) and that $\boldsymbol{v}$ is a k-approximate prefix and a k-approximate suffix of $\boldsymbol{w}$. Let $B' = [\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_{|B'|}]$ be a list of strings $\boldsymbol{u}$ such that $\forall \boldsymbol{u} \in B' : D(\boldsymbol{v}, \boldsymbol{u}) \leq k$ and $\forall i : 1 \leq i \leq |B'| : \boldsymbol{u}_i$ occurs at position $i \in C$ in $\boldsymbol{w}$. Because D is defined for strings of equal lengths, $\forall i : 2 \leq i \leq |B'| : |\boldsymbol{u}_i| = |\boldsymbol{u}_{i-1}| = |\boldsymbol{v}|$. It clearly holds that i_1 is a position of a prefix $\boldsymbol{p} = \boldsymbol{u}_1 \in B'$ such that $D(\boldsymbol{p}, \boldsymbol{v}) \leq k$ and that $i_{|C|}$ is a position of a suffix $\boldsymbol{s} = \boldsymbol{u}_{|B'|} \in B'$ such that $D(\boldsymbol{s}, \boldsymbol{v}) \leq k$. By (4.1) holds that there is no gap between any two subsequent strings of B', therefore it is possible to construct $\boldsymbol{w}$ by concatenations or superpositions of copies of strings from B', i.e., B' meets the definition of the set $\mathsf{L}^{\mathrm{c}}_{\mathsf{H}^k}(\boldsymbol{w})$ from Definition 2.4.16. Because $\boldsymbol{v}$ is a factor of $\boldsymbol{w}$, it is also a restricted k-approximate cover of $\boldsymbol{w}$. □

Lemma 4.1.8. Given maximum distance k under Hamming distance function H, string $\boldsymbol{v}$ is a nontrivial k-approximate cover of string $\boldsymbol{w}$ if and only if

1. $\boldsymbol{v}$ is simultaneously a k-approximate prefix and a k-approximate suffix of $\boldsymbol{w}$, and
2. for a k-approximate end set $C = \{i_1, i_2, \ldots, i_{|C|}\}$ of $\boldsymbol{v}$ holds

$$\forall j = 2, 3, \ldots, |C| : i_j - i_{j-1} \leq |\boldsymbol{v}|$$

Proof. (if) The if part of proof of Lemma 4.1.7 holds here as well, because there is no assumption about any (exact) position of string $\boldsymbol{v}$ in $\boldsymbol{w}$.

(only if) Assuming any string $\boldsymbol{v} \neq \boldsymbol{w}$ such that for a k-approximate end set $C = \{i_1, i_2, \ldots, i_{|C|}\}$ of $\boldsymbol{v}$ holds (4.1) and that $\boldsymbol{v}$ is a k-approximate prefix and a k-approximate suffix of $\boldsymbol{w}$, by the rest of the only-if part of proof of Lemma 4.1.7 follows that $\boldsymbol{v}$ meets the Definition 2.4.16. As there is no requirement for $\boldsymbol{v}$ being a factor of $\boldsymbol{w}$), $\boldsymbol{v}$ is a k-approximate cover of $\boldsymbol{w}$. □

Note 4.1.9. Lemma 4.1.8 does not hold for Levenshtein distance and generally for any distance function defined for strings of not equal lengths. See Figure 4.1 for example. String $\boldsymbol{v} = \texttt{abc}$ is a 1-approximate cover of string $\boldsymbol{w} = \texttt{abcabbabbc}$ under Levenshtein distance. The 1-approximate end set of $\boldsymbol{v}$ is $C = \{3, 6, 10\}$, because $D_L(\boldsymbol{v}, \texttt{abb}) = 1$ and

1	2	3	4	5	6	7	8	9	10
a	b	c	a	b	b	a	b	b	c
a	b	c							
			a	b	R				
						a	b	I	c

Figure 4.1: 1-approximate covering of string abcabbabbc under Levenshtein distance

$D_L(\boldsymbol{v}, \texttt{abbc}) = 1$. The set C contains subsequent elements $6, 10$ for which holds $10 - 6 = 4$, but the length of $\boldsymbol{v}$ is 3, therefore (4.1) does not hold.

4.2 Finite-automata-based solution

Voráček [48] has already solved the problem of finding all covers of generalized strings using finite automata. As string is special case of generalized string, his algorithm is applicable to searching all covers of strings. Simpler, easier to understand, and less time and space complex algorithm for strings is presented below. The main contribution, an algorithm for computing all covers with Hamming distance is shown, too.

Let us begin with principles common for searching exact and approximate covers of strings.

Observation 4.2.1. Assuming string $\boldsymbol{w}$ and maximum distance $k \geq 0$ under some distance function D, each k-approximate cover of $\boldsymbol{w}$ is from language L equal to intersection of set of all k-approximate prefixes and set of all k-approximate suffixes of $\boldsymbol{w}$. In case of exact covers, k is equal to 0. Therefore, each such cover is accepted by FA that accepts L.

Proof. Following Property 3.1.1, Lemma 4.1.3 and Observation 4.1.6, each k-approximate cover of $\boldsymbol{w}$ is simultaneously a k-approximate prefix and a k-approximate suffix of $\boldsymbol{w}$. Therefore, an automaton accepting just such prefixes and suffixes must accept all the covers. □

Note 4.2.2. In the Observation 4.2.1, there is no assumption about the distance used.

Based on idea presented by Voráček [48], all the covers of a string (either exact, approximate, or restricted approximate) may be found within two steps: Firstly, find a set of strings that may be covers (or filter out strings not being covers for sure) using cover-candidate automaton. Secondly, for each candidate string, verify if it is a cover with given constraint.

Proposition 4.2.3. Following the Observation 4.2.1, Property 3.1.3 and the Lemmas 4.1.7 and 4.1.8, all the covers (either exact, approximate, or restricted approximate) of string $\boldsymbol{w}$ with given D and k may be computed using a two-phase algorithm:

1. Construct automata for $\boldsymbol{w}$, D, and k-approximate prefix (Algorithm 3.1 or 3.6) and k-approximate suffix (Algorithm 3.3 or 3.7), construct equivalent deterministic automata $\mathcal{M}_{\mathrm{P}}, \mathcal{M}_{\mathrm{S}}$. Construct an automaton $\mathcal{M}_{\mathrm{C^D}}(\boldsymbol{w})$ accepting language $L(\mathcal{M}_{\mathrm{C^D}}(\boldsymbol{w})) = L(\mathcal{M}_{\mathrm{P}}) \cap L(\mathcal{M}_{\mathrm{S}})$ using some standard algorithm (e.g., [38]).

2. For each final state q of $\mathcal{M}_{\mathrm{C^D}}(\boldsymbol{w})$ do: for each string of left language of q do the check whether it is able to cover $\boldsymbol{w}$ (Property 3.1.3 or one of Lemmas 4.1.7 or 4.1.8).

Example 4.2.4 (Phase 2 of Proposition 4.2.3)**.** Looking at state $2'4'$ at transition diagram in Figure 4.6, the numbers 2 and 4 represent 1-approximate positions with Hamming distance of string `cb` in string `abcc` (see the figure and the following sections). As $4 - 2 \leq |\mathtt{cb}|$, string `cb` is a 1-approximate cover of string `abcc`.

Observation 4.2.5. Assuming any state of a trie-like automaton, left language of the state contains exactly one string.

4.2.1 Exact covers

Algorithm 4.1 for computation of all covers of given string $\boldsymbol{w}$ follows idea stated in Proposition 4.2.3. As the first step, nondeterministic suffix automaton for $\boldsymbol{w}$ is constructed. Then cover-candidate automaton $\mathcal{M}_{\mathrm{C^D}}(\boldsymbol{w})$ is constructed and string of left language of each of its states is checked whether it is a cover. As any cover of $\boldsymbol{w}$ must be a prefix of $\boldsymbol{w}$ (Property 3.1.1), only prefixes are considered (lines 6 and 7). Note that in case of exact covers, cover-candidate automaton is always a backbone (recall Definition 2.5.38). For each state q_i, its d-subset $\mathsf{d}(q_i)$ (Definition 2.5.29) is constructed (lines 9–13). If the state represents a suffix of $\boldsymbol{w}$ (line 15), i.e., string $\boldsymbol{u}$ in its left language is a border of $\boldsymbol{w}$, it is checked whether property stated in Note 3.1.4 holds, i.e., whether it is possible to construct $\boldsymbol{w}$ using superpositions of $\boldsymbol{u}$.

Lemma 4.2.6. Using Algorithm 4.1 for construction states of a cover-candidate automaton $\mathcal{M} = (Q, A, \delta, q_0, F)$ from a nondeterministic suffix automaton $\mathcal{M}_{\mathrm{N}} = (Q_{\mathrm{N}}, A, \delta_{\mathrm{N}}, q_{0\mathrm{N}}, F_{\mathrm{N}})$, all d-subsets are constructed as sorted in ascending order by depths within $\mathcal{M}_{\mathrm{N}}$, i.e., sorting takes no extra time.

Proof. Assume $q_{i-1}, q_i \in Q \setminus \{q_0\}$ such that q_i is a successor of q_{i-1}. It holds that for any $q_1^S, q_2^S \in Q_{\mathrm{N}} \setminus \{q_{0\mathrm{N}}\}$ such that q_2^S is a successor of q_1^S if and only if

$$\mathsf{depth}(q_2^S) = \mathsf{depth}(q_1^S) + 1 \tag{4.2}$$

(it is a property of nondeterministic suffix automaton constructed using Algorithm 3.2). Line 9 of the algorithm processes elements of $\mathsf{d}(q_{i-1})$ in order by depth in $\mathcal{M}_{\mathrm{N}}$. There exists at most one e_l for any $e_j \in Q_{\mathrm{N}} \setminus \{q_{0\mathrm{N}}\}$ (line 10) and because of (4.2) it always holds: $e_l = e_j + 1$. Therefore, line 10 always processes such the elements in order by depth in $\mathcal{M}_{\mathrm{N}}$.

Algorithm 4.1 Computation of all covers of string $\boldsymbol{w}$

Input: String $\boldsymbol{w} \in A_{\boldsymbol{w}}^*$.
Output: Set $\mathsf{L}^{\mathrm{c}}(\boldsymbol{w})$ of all covers of string $\boldsymbol{w}$.

1: $\mathsf{L}^{\mathrm{c}}(\boldsymbol{w}) \leftarrow \emptyset$
2: $\boldsymbol{u} \leftarrow \varepsilon$
3: $\mathcal{M}_{\mathrm{S^N}}(\boldsymbol{w}) = (Q_{\mathrm{N}}, A_{\boldsymbol{w}}, \delta_{\mathrm{N}}, q_{0\mathrm{N}}, F_{\mathrm{N}}) \leftarrow$ nondeterministic suffix automaton with removed ε transitions for string $\boldsymbol{w}$ (Alg. 3.3 and 3.4)
4: define q_0 to be initial state of $\mathcal{M}_{\mathrm{C^D}}(\boldsymbol{w}) = (Q, A_{\boldsymbol{w}}, \delta, q_0, F)$
5: $\mathsf{d}(q_0) \leftarrow \{q_{0\mathrm{N}}\}$
6: **for** $i \leftarrow 1, |\boldsymbol{w}|$ **do**
7: $\quad \boldsymbol{u} \leftarrow \boldsymbol{u}.\boldsymbol{w}[i]$
8: $\quad$ create new state q_i
9: $\quad$ **for all** $e_j \in \mathsf{d}(q_{i-1})$ **do**
10: $\quad\quad$ **for all** $e_l \in \delta_{\mathrm{N}}(e_j, \boldsymbol{w}[i])$ in order by $\mathsf{depth}(e_l)$ **do**
11: $\quad\quad\quad$ append all e_l to $\mathsf{d}(q_i)$
12: $\quad\quad$ **end for**
13: $\quad$ **end for**
14: $\quad$ discard q_{i-1}
15: $\quad$ **if** for the last element $e_{|\mathsf{d}(q_i)|}$ of $\mathsf{d}(q_i)$ holds: $\mathsf{depth}(e_{|\mathsf{d}(q_i)|}) = |\boldsymbol{w}|$ **then** $\quad \triangleright\ q_i$ is final
16: $\quad\quad$ **if** for all $j = 2, 3, \ldots, |\mathsf{d}(q_i)| : \mathsf{depth}(e_j) - \mathsf{depth}(e_{j-1}) \leq |\boldsymbol{u}|$ **then**
17: $\quad\quad\quad$ $\mathsf{L}^{\mathrm{c}}(\boldsymbol{w}) \leftarrow \mathsf{L}^{\mathrm{c}}(\boldsymbol{w}) \cup \{\boldsymbol{u}\}$
18: $\quad\quad$ **end if**
19: $\quad$ **end if**
20: **end for**

In case of q_0, there exists just single element $q_{0\mathrm{N}} \in \mathsf{d}(q_0)$ (line 9). Line 10 may process multiple elements e_l and δ_{N} is expected to be implemented in order to retrieve successors of $q_{0\mathrm{N}}$ in order by depth in $\mathcal{M}_{\mathrm{N}}$. □

Theorem 4.2.7 (Correctness of Algorithm 4.1)**.** Assuming string $\boldsymbol{w}$ as input string for Algorithm 4.1 and set of strings $\mathsf{L}^{\mathrm{c}}(\boldsymbol{w})$ as output of the algorithm, the set $\mathsf{L}^{\mathrm{c}}(\boldsymbol{w})$ is the set of all covers $\mathsf{L}^{\mathrm{c}}(\boldsymbol{w})$ of string $\boldsymbol{w}$.

Proof. In the algorithm, in the inner for loops (lines 9–13), there is d-subset $\mathsf{d}(q_i)$ constructed as a set of successors of all elements of $\mathsf{d}(q_{i-1})$ using transition function of $\mathcal{M}_{\mathrm{S^N}}(\boldsymbol{w})$. As just $\boldsymbol{w}[i]$ is used as input symbol for the transition function, the constructed states correspond to backbone of deterministic suffix automaton for string $\boldsymbol{w}$ (only prefix of $\boldsymbol{w}[i]$ is used to construct $\mathsf{d}(q_i)$). By Proposition 3.1.30 follows that just final states of the backbone contain in their left languages strings being simultaneously prefixes and suffixes (Property 3.1.1). State q_{i-1} may be discarded after construction of $\mathsf{d}(q_i)$ as it is needed no more. If the depth of the last element $e_{|\mathsf{d}(q_i)|}$ of $\mathsf{d}(q_i)$ is equal to length of input string (line 15) then q_i is final state of the backbone as $e_{|\mathsf{d}(q_i)|}$ is final state of $\mathcal{M}_{\mathrm{S^N}}(\boldsymbol{w})$ (property

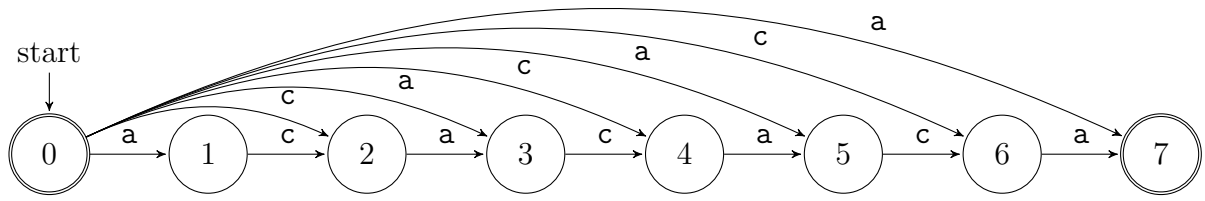

Figure 4.2: Transition diagram of nondeterministic suffix automaton for string acacaca (Example 4.2.8)

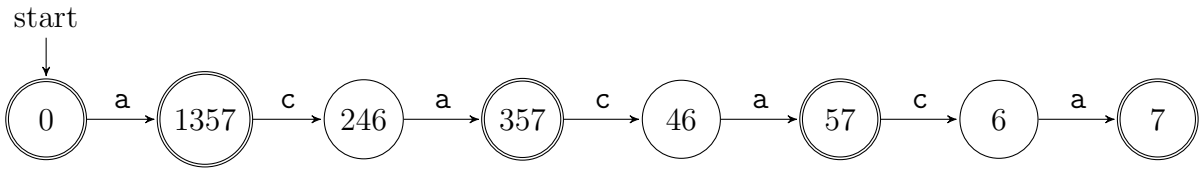

Figure 4.3: Transition diagram of backbone of deterministic suffix automaton for string acacaca (Example 4.2.8)

of a nondeterministic suffix automaton). The subsequent condition (line 16) for the prefix and suffix $\boldsymbol{u}$ to be a cover of $\boldsymbol{w}$ follows by Property 3.1.3. □

Example 4.2.8. Let us have string $\boldsymbol{w} = \mathtt{acacaca}$ and let us compute the set of all covers $\mathsf{L}^{\mathrm{c}}(\boldsymbol{w})$ of $\boldsymbol{w}$. Firstly, nondeterministic suffix automaton is constructed using Algorithm 3.3, equivalent nondeterministic suffix automaton $\mathcal{M}_{\mathrm{S^N}}(\boldsymbol{w})$ without ε transitions is constructed using Algorithm 3.4. Its transition diagram is depicted at Figure 4.2. Then backbone $\mathcal{M}_{\mathrm{C^D}}(\boldsymbol{w})$ of deterministic suffix automaton is constructed, its transition diagram is depicted at Figure 4.3. For each prefix $\boldsymbol{u}$ of $\boldsymbol{w}$ state of $\mathcal{M}_{\mathrm{C^D}}(\boldsymbol{w})$ and its d-subset (i.e., end set of $\boldsymbol{u}$) is created. Each state that contains 7 in its d-subset represents border of $\boldsymbol{w}$. State 1357 represents border a that is not a cover of $\boldsymbol{w}$, because $3 - 1 = 2 > |\mathtt{a}|$, i.e., there are gaps between its consecutive occurrences. State 357 represents border aca and because there are no gaps between its consecutive occurrences ($7 - 5 = 5 - 3 = 2 \leq |\mathtt{aca}|$), it is a cover of $\boldsymbol{w}$. Similar holds for state 57. Therefore, $\mathsf{L}^{\mathrm{c}}(\boldsymbol{w}) = \{\mathtt{aca}, \mathtt{acaca}, \mathtt{acacaca}\}$.

4.2.1.1 Time and space complexity

Lemma 4.2.9. For every state q of cover-candidate automaton $\mathcal{M}$ constructed by Algorithm 4.1 holds: line 11 never appends elements having the same depth in $\mathsf{d}(q)$.

Proof. It holds from properties of transition function of nondeterministic suffix automaton $\mathcal{M}_{\mathrm{N}} = (Q_{\mathrm{N}}, A, \delta_{\mathrm{N}}, q_{0\mathrm{N}}, F_{\mathrm{N}})$ for string $\boldsymbol{w}$: for successors e_1, e_2 of the initial state $q_{0\mathrm{N}}$ holds:

$$e_1 \neq e_2 \Rightarrow \mathsf{depth}(e_1) \neq \mathsf{depth}(e_2)$$

Therefore, all the e_l processed by line 10 for $e_j = q_{0\mathrm{N}}$ have distinct depths. For all successors e_l of all states $e_j \in Q_\mathrm{N} \setminus \{q_{0\mathrm{N}}\}$ holds:

$$\mathsf{depth}(e_l) = \mathsf{depth}(e_j) + 1$$

(there is at most single successor for any e_j for all symbols, which is a property of a nondeterministic suffix automaton). Let us use induction. For direct successors of initial state q_0, line 11 never appends elements having the same depth (already proved). Let us consider any state $q_{i-1} \in Q \setminus \{q_0\}$. For any successor q_j of such state q_{i-1}, any element e_l of $\mathsf{d}(q_j)$ is constructed from element $e_j \in \mathsf{d}(q_{i-1})$ this way: $e_l \in \delta_\mathrm{N}(e_j, a), a \in A$ and thus $\mathsf{depth}(e_l) = \mathsf{depth}(e_j)+1$ (line 10). Therefore, the Lemma holds for all d-subsets of $\mathcal{M}$. □

Theorem 4.2.10 (Space complexity of Algorithm 4.1)**.** Assuming string $\boldsymbol{w}$ as input string of Algorithm 4.1, space complexity of the algorithm is

$$\mathcal{O}(|\boldsymbol{w}|)$$

Proof. Space complexity of construction of nondeterministic suffix automaton is $\mathcal{O}(|\boldsymbol{w}|)$ (Algorithm 3.3). String $\boldsymbol{u}$ cannot be longer than $|\boldsymbol{w}|$. The set $\mathsf{L}^\mathrm{c}(\boldsymbol{w})$ contains at most $|\mathsf{L}^\mathrm{c}(\boldsymbol{w})|$ strings, thus it needs $|\mathsf{L}^\mathrm{c}(\boldsymbol{w})|$ space that is $\mathcal{O}(|\boldsymbol{w}|)$ (as number of prefixes of $\boldsymbol{w}$). The algorithm never stores more than two states in memory at a time (line 14). Following Lemma 4.2.9, number of elements of any d-subset cannot be greater than number of states of $\mathcal{M}_{\mathrm{S}^\mathrm{N}}(\boldsymbol{w})$ that is $|\boldsymbol{w}| + 1$. Therefore, there at most $\mathcal{O}(|\boldsymbol{w}|)$ elements stored in memory at a time. There is nothing else stored in memory by the algorithm. □

Theorem 4.2.11 (Time complexity of Algorithm 4.1)**.** Assuming string $\boldsymbol{w} \in A^*_{\boldsymbol{w}}$ as input string of Algorithm 4.1, time complexity of the algorithm is

$$\mathcal{O}(|\boldsymbol{w}|^2)$$

Proof. Time complexity of construction of $\mathcal{M}_{\mathrm{S}^\mathrm{N}}(\boldsymbol{w})$ without ε-transitions $\mathcal{O}(|\boldsymbol{w}| \cdot |A_{\boldsymbol{w}}|)$ (recall that $|A_{\boldsymbol{w}}| \leq |\boldsymbol{w}|$, number of states is $|\boldsymbol{w}| + 1$, number of transitions is $\boldsymbol{w}$). For each state, the d-subset is constructed in $\mathcal{O}(|\boldsymbol{w}|)$ time (lines 9–13, initial state q_0 has $|\boldsymbol{w}|$ elements in $\mathsf{d}(q_0)$, every other state has at most 1 element in its d-subset). The check whether the state is final is done in constant time (line 15), and the check whether string $\boldsymbol{u}$ covers $\boldsymbol{w}$ is done in $\mathcal{O}(|\boldsymbol{w}|)$ time 16. As the number of processed states is $|\boldsymbol{w}| + 1$, the time complexity is $\mathcal{O}(|\boldsymbol{w}|^2)$. □

4.2.2 Approximate covers

Since Lemmas 4.1.7 and 4.1.8 are proved for Hamming distance only, only Hamming distance is considered in this Section for computing all approximate and restricted approximate covers.

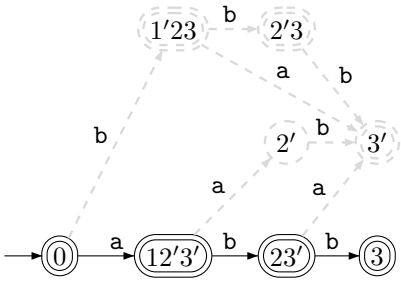

Figure 4.4: Transition diagram of deterministic suffix automaton for string $\boldsymbol{w} = \mathtt{abb}$ over alphabet $A = \{\mathtt{a}, \mathtt{b}\}$ for Hamming distance $k = 1$ from Example 4.2.13 (parts outside backbone are dashed); note that not every state of the backbone is necessarily final in general

Observation 4.2.12. Assuming a backbone of a deterministic k-approximate suffix automaton (Definition 2.5.38) for string $\boldsymbol{w}$ and maximum distance k under some distance function D, final states have only exact prefixes in their left languages, but they contain no approximate ones.

Example 4.2.13. Let us consider string $\boldsymbol{w} = \mathtt{abb}$ and maximum Hamming distance $k = 1$. Backbone of deterministic suffix automaton for $\boldsymbol{w}$ and k accepts all exact prefixes of $\boldsymbol{w}$. It does not accept any k-approximate prefix of $\boldsymbol{w}$, e.g., $\mathtt{bb}$. See Figure 4.4.

Note 4.2.14. Recall Definition 2.5.38. Backbone is defined in this work to accept only exact prefixes of given string. No definition of any approximate backbone is given here.

Observation 4.2.15 (Depth and string length)**.** Assuming k-approximate deterministic suffix automaton $\mathcal{M} = (Q, A, \delta, q_0, F)$ constructed using subset construction from nondeterministic k-approximate suffix automaton $\mathcal{M}^{\mathsf{H},k}_{\mathrm{S^N}}(\boldsymbol{w})$ for string $\boldsymbol{w} \in A^*$ and maximum Hamming distance k, and any state $q \in Q$. For any string $\boldsymbol{u}$ from left language of state q holds: there exists no element e of $\mathsf{d}(q)$ such that $\mathsf{depth}(e) < |\boldsymbol{u}|$.

Proof. It clearly holds from the transition function of $\mathcal{M}^{\mathsf{H},k}_{\mathrm{S^N}}(\boldsymbol{w}) = (Q_{\mathrm{N}}, A, \delta_{\mathrm{N}}, q_{0\mathrm{N}}, F_{\mathrm{N}})$: if $e \in Q_{\mathrm{N}}, e \in \delta^*_S(q_{0\mathrm{N}}, \boldsymbol{u})$ then $\mathsf{depth}(e) \geq |\boldsymbol{u}|$. □

Lemma 4.2.16. Having a state q of a deterministic k-approximate suffix automaton $\mathcal{M}$ constructed using subset construction from a nondeterministic k-approximate suffix automaton for string $\boldsymbol{w}$ and maximum Hamming distance k, the following properties hold:

1. for each string $\boldsymbol{u}$ from left language of state q: $|\boldsymbol{u}|$ is equal to the minimum depth of elements of $\mathsf{d}(q)$ if and only if $\boldsymbol{u}$ is a k-approximate prefix of $\boldsymbol{w}$,

2. depth of state q is less than the minimum depth of elements of $\mathsf{d}(q)$ if and only if any string of left language of q cannot be a k-approximate prefix of $\boldsymbol{w}$.

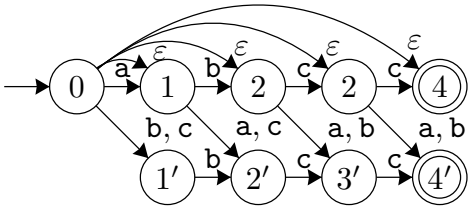

Figure 4.5: Transition diagram of nondeterministic 1-approximate suffix automaton with ε transitions for string $\boldsymbol{w} = \mathtt{abcc}$ and Hamming distance from Example 4.2.17

Proof. Recall that $\mathcal{M}$ is constructed using subset construction (Definition 2.5.27) from a nondeterministic k-approximate suffix automaton (Algorithm 3.7) for string $\boldsymbol{w}$ with maximum Hamming distance k. Proof of the properties of the Lemma is the following:

1. As the minimum depth of elements of the d-subset is equal to the leftmost k-approximate (end) position of $\boldsymbol{u}$ in $\boldsymbol{w}$, when the length of $\boldsymbol{u}$ is equal to that depth, it must be a k-approximate prefix of $\boldsymbol{w}$ under Hamming distance k (its start position is equal to the start of $\boldsymbol{w}$); a string of other length cannot be its k-approximate prefix.

2. Clearly holds from the property 1: if depth of state q is less than the minimum depth of elements of $\mathsf{d}(q)$ then the longest string of left language of q (Definition 2.5.13) is not long enough to be a k-approximate prefix of $\boldsymbol{w}$; if the depth of q is not less than the minimum depth of the elements then any longest string from left language of q has its leftmost (end) position equal to its length and thus it is a k-approximate prefix.

Therefore, the Lemma holds. □

Example 4.2.17 (Property 2 of Lemma 4.2.16)**.** Let us consider string $\boldsymbol{w} = \mathtt{abcc}$ and maximum Hamming distance $k = 1$. Transition diagram of nondeterministic suffix automaton for $\boldsymbol{w}$ and k is shown in Figure 4.5, transition diagram of deterministic suffix automaton for $\boldsymbol{w}$ and k is depicted in Figure 4.6. For state $3'4$ holds that its depth is equal to 2 as the longest string $\mathtt{cc}$ of its left language is of length 2. This string is not a 1-approximate prefix of $\boldsymbol{w}$ with Hamming distance 1, as its leftmost 1-approximate position is 3.

Corollary 4.2.18. Assuming string $\boldsymbol{w} \in A^*$ and maximum Hamming distance k, from Lemma 4.2.16 follows that if construction of the deterministic k-approximate suffix automaton is used as a base of construction of a cover-candidate automaton $\mathcal{M}^{\mathsf{H},k}_{\mathrm{C^D}}(\boldsymbol{w})$ for Hamming distance, for each state q of $\mathcal{M}^{\mathsf{H},k}_{\mathrm{C^D}}(\boldsymbol{w})$ holds that determination whether string $\boldsymbol{u}$ from left language of q is a k-approximate prefix of $\boldsymbol{w}$ may be based on the comparison of length of string $\boldsymbol{u}$ and the minimum depth of elements of $\mathsf{d}(q)$. Therefore, construction of intersection is not needed for searching k-approximate covers with Hamming distance.

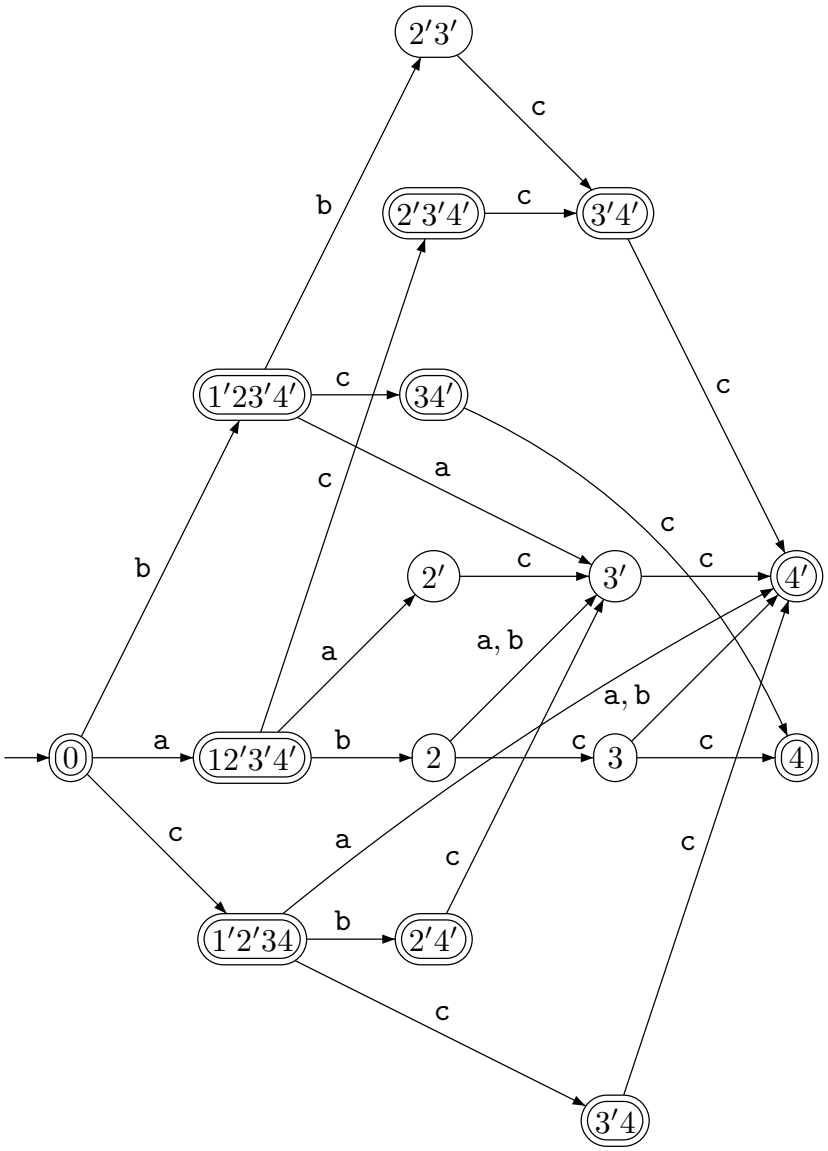

Figure 4.6: Transition diagram of deterministic 1-approximate suffix automaton for string $\boldsymbol{w} = \texttt{abcc}$ and Hamming distance from Example 4.2.17

Example 4.2.19. Let us consider string $\boldsymbol{w} = \texttt{abcc}$ and maximum Hamming distance $k = 1$. Transition diagram of nondeterministic 1-approximate suffix automaton with ε transitions for $\boldsymbol{w}$ is depicted in Figure 4.5, transition diagram of deterministic 1-approximate suffix automaton for $\boldsymbol{w}$ is shown in Figure 4.6. Looking at state $4'$ and string $\texttt{ca}$ from its left language, it holds that as the minimum depth of elements of the d-subset $4'$ is equal to 4 and length of $\texttt{ca}$ is 2, $\texttt{ca}$ is not a 1-approximate prefix of $\boldsymbol{w}$.

Lemma 4.2.20. Assume string $\boldsymbol{w} \in A^*$, maximum Hamming distance k and any string $\boldsymbol{v} \in A^*$ that is not a k-approximate prefix of $\boldsymbol{w}$ with respect to k. Then for any string $\boldsymbol{u} \in A^*$ holds that $\boldsymbol{v}.\boldsymbol{u}$ is not a k-approximate prefix of $\boldsymbol{w}$ with respect to k.

Proof. As string $\boldsymbol{v}$ is not a k-approximate prefix of $\boldsymbol{w}$, it clearly holds that the leftmost position $i_{\boldsymbol{v}}$ of $\boldsymbol{v}$ is $i_{\boldsymbol{v}} > |\boldsymbol{v}|$ (by Observation 4.2.15).

1. For $\boldsymbol{u} = \varepsilon$ it clearly holds.
2. For $\boldsymbol{u} = a, a \in A$: by Lemma 4.1.2, for a k-approximate position $i_{\boldsymbol{u}}$ of $\boldsymbol{v}.a$ holds $i_{\boldsymbol{u}} \geq i_{\boldsymbol{v}} + 1$. As $|\boldsymbol{v}a| = |\boldsymbol{v} + 1|$, $\boldsymbol{v}.\boldsymbol{u}$ is not long enough to be a k-approximate prefix of $\boldsymbol{w}$.
3. Suppose the Lemma holds for $\boldsymbol{u} = \boldsymbol{u}' \in A^+$. It is to be proved that the Lemma also holds for any $\boldsymbol{u} = \boldsymbol{u}'' = \boldsymbol{u}'.a, a \in A$: as $\boldsymbol{v}\boldsymbol{u}'$ is not a k-approximate prefix of $\boldsymbol{w}$, for the leftmost k-approximate position $i_{\boldsymbol{u}'}$ holds $i_{\boldsymbol{u}'} > |\boldsymbol{v}\boldsymbol{u}'|$. As $|\boldsymbol{u}''| = |\boldsymbol{u}'| + 1$, by Lemma 4.1.2 for the leftmost k-approximate position $i_{\boldsymbol{u}''}$ holds of $\boldsymbol{v}\boldsymbol{u}''$ holds $i_{\boldsymbol{u}''} > i_{\boldsymbol{u}'}$ and therefore $\boldsymbol{v}\boldsymbol{u}''$ is not long enough to be a k-approximate prefix of $\boldsymbol{w}$.

Therefore, the Lemma holds. □

Corollary 4.2.21. From Lemma 4.2.20 follows: during construction of a cover-candidate automaton (Definition 2.5.41) for Hamming distance, just states containing at least one k-approximate prefix in their left languages are necessary to be constructed.

Proof. Assume cover-candidate automaton $\mathcal{M}^{\mathsf{H},k}_{\mathrm{C^D}}(\boldsymbol{w})$ constructed using subset construction from a nondeterministic k-approximate suffix automaton for string $\boldsymbol{w} \in A^*$, maximum Hamming distance k, and state q of $\mathcal{M}^{\mathsf{H},k}_{\mathrm{C^D}}(\boldsymbol{w})$ having no k-approximate prefix of $\boldsymbol{w}$ in its left language. From Lemma 4.2.20 follows that for any string $\boldsymbol{v}$ from left language of q and for any string $\boldsymbol{u} \in A^*$ holds that $\boldsymbol{v}.\boldsymbol{u}$ cannot be a k-approximate prefix of $\boldsymbol{w}$. Therefore, by removing state q (and by not constructing its possible successors), the number of k-approximate prefixes of $\boldsymbol{w}$ accepted by $\mathcal{M}^{\mathsf{H},k}_{\mathrm{C^D}}(\boldsymbol{w})$ cannot decrease. □

Proposition 4.2.22. Assuming string $\boldsymbol{w}$ and some distance function D, the deterministic automaton accepting only k-approximate prefixes and suffixes of $\boldsymbol{w}$ with respect to D may be constructed as a trie-like automaton.

Proposition 4.2.23. When constructing a cover-candidate automaton $\mathcal{M}$ for string $\boldsymbol{w}$, maximum distance k under some distance function D, and restricted k-approximate covers

of $\boldsymbol{w}$ only, it is sufficient for $\mathcal{M}$ to accept (exact) factors only. Recall that $\mathcal{M}$ is constructed using subset construction from a nondeterministic k-approximate suffix automaton for $\boldsymbol{w}$ and D. Such factors are accepted by such states q for that holds that $\mathsf{d}(q)$ contains at least one element with level equal to zero (see Algorithm 3.7 and Definition 2.5.40). Also recall Lemma 4.1.1. Therefore, construction of only states having at least one zero-level element in d-subset to accept restricted k-approximate cover is necessary. See Proposition 4.2.26.

Example 4.2.24. Let us consider string $\boldsymbol{w} = \mathtt{abcc}$ and maximum Hamming distance $k = 1$. Transition diagram of nondeterministic 1-approximate suffix automaton with ε transitions for $\boldsymbol{w}$ is shown in Figure 4.5, transition diagram of deterministic 1-approximate suffix automaton for $\boldsymbol{w}$ is depicted in Figure 4.6. Looking at state $34'$, its left language contains only string $\mathtt{bc}$ that is exact factor of $\boldsymbol{w}$. Looking at state $2'4'$, its left language contains only string $\mathtt{cb}$ that is not a factor of $\boldsymbol{w}$. It is also obvious that by removing this state, the number of exact factors accepted by the automaton does not decrease (due to Lemma 3.1.34). The same holds for the other states that contain only non-zero-level elements in d-subsets, i.e., $2'3'$ (left language $\{\mathtt{bb}\}$), $2'3'4'$ (left language $\{\mathtt{ac}\}$), $3'4'$ (left language $\{\mathtt{acc}, \mathtt{bbc}\}$), $2'$ (left language $\{\mathtt{aa}\}$), $3'$ (left language $\{\mathtt{aac}, \mathtt{aba}, \mathtt{abb}, \mathtt{cbc}\}$), and $4'$ (left language $\{\mathtt{aacc}, \mathtt{abac}, \mathtt{abbc}, \mathtt{cbcc}, \mathtt{ca}, \mathtt{ccc}\}$).

Proposition 4.2.25. Assume searching all k-approximate covers of string $\boldsymbol{w}$ with maximum Hamming distance k. Following Proposition 4.2.3, the step 1 may be replaced by a construction of a k-approximate cover-candidate automaton for Hamming distance with respect to the ideas proposed in this Section. For the subsequent computation from the step 2, d-subsets with elements corresponding to states of nondeterministic suffix automaton for $\boldsymbol{w}$ and k must be preserved. The cover-candidate automaton may be constructed as a trie-like automaton. The states for purpose of the step 2 may be selected in order given by depth-first search of the cover-candidate automaton (see Algorithms 4.2 and 4.3). Note that there is no need to explicitly store transitions (Algorithm 4.3).

Proposition 4.2.26. Having Proposition 4.2.25, the following Algorithm 4.3 may be extended for computing restricted covers in this way: modify the statement on line 11 to:
11: **if** exists e in $\mathsf{d}(q)$ such that $\mathsf{level}(e) = 0$ and $|\mathsf{d}(q)| > 1$ **then**

Lemma 4.2.27. Assuming string $\boldsymbol{w}$ and maximum Hamming distance k, for the set $\mathsf{L}^{\mathrm{c}}_{\mathsf{H}^k}(\boldsymbol{w})$ computed using Algorithm 4.2 holds: $\boldsymbol{u}$ is in a pair from $\mathsf{L}^{\mathrm{c}}_{\mathsf{H}^k}(\boldsymbol{w})$ if and only if $\boldsymbol{u}$ is a k-approximate cover of string $\boldsymbol{w}$.

Proof. A nondeterministic k-approximate suffix automaton $\mathcal{M}^{\mathsf{H},k}_{\mathrm{S^N}}(\boldsymbol{w})$ for Hamming distance accepts all k-approximate covers of $\boldsymbol{w}$ as it accepts all k-approximate suffixes of $\boldsymbol{w}$. Using Algorithm 4.3, there is constructed a successor q of each existing state q_i of $\mathcal{M}^{\mathsf{H},k}_{\mathrm{C^D}}(\boldsymbol{w})$ for each symbol a of input alphabet A. The state q has depth defined as 1 plus depth of its predecessor and it is correct, as depth of initial state is equal to 0. The string of left language of the state is defined as string of left language of its predecessor concatenated with symbol a and it is correct. The d-subset $\mathsf{d}(q)$ contains all the successors of states

Algorithm 4.2 Computation of all smallest distance k-approximate covers of string with Hamming distance

Input: String $\boldsymbol{w} \in A$, the maximum Hamming distance k.
Output: Set $\mathsf{L}^{\mathrm{c}}_{\mathsf{H}^k}(\boldsymbol{w})$ of all smallest distance k-approximate covers of $\boldsymbol{w}$ with the smallest distances with maximum Hamming distance k.

1: construct nondeterministic k-approximate suffix automaton $\mathcal{M}'_{\mathrm{N}}$ for $\boldsymbol{w}$ and Hamming distace using Alg. 3.7
2: construct $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SN}}(\boldsymbol{w}) = (Q_{\mathrm{N}}, A, \delta_{\mathrm{N}}, q_{0\mathrm{N}}, F_{\mathrm{N}})$ from $\mathcal{M}'_{\mathrm{N}}$ using Algorithm 3.4 (removing ε transitions)
3: create q_0 of cover-candidate automaton $\mathcal{M}^{\mathsf{H},k}_{\mathrm{CD}}(\boldsymbol{w}) = (Q, A, \delta, q_0, F)$
4: $\mathsf{lfactor}(q_0) \leftarrow \varepsilon$
5: $\mathsf{depth}(q_0) \leftarrow 0$
6: $B' \leftarrow$ ProcessState$(q_0, A, k, \mathcal{M}^{\mathsf{H},k}_{\mathrm{SN}}(\boldsymbol{w}))$ (Algorithm 4.3)
7: $\mathsf{L}^{\mathrm{c}}_{\mathsf{H}^k}(\boldsymbol{w}) \leftarrow B'$ (computed using the previous step)

contained in $\mathsf{d}(q_i)$, no one is missing. The string $\mathsf{lfactor}(q)$ is not further processed just in the following cases:

- $e_1 \neq \mathsf{depth}(q) = |\mathsf{lfactor}(q)|$ which means that $\mathsf{lfactor}(q)$ is not long enough to be a k-approximate prefix of $\boldsymbol{w}$ (Corollary 4.2.21),
- the last element of $\mathsf{d}(q)$ is not a final state in $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SN}}(\boldsymbol{w})$, i.e., $\mathsf{lfactor}(q)$ is not a k-approximate suffix of $\boldsymbol{w}$,
- $\mathsf{d}(q)$ does not contain elements that represent k-approximate positions to approximately cover $\boldsymbol{w}$ (by Lemma 4.1.8 – recall that depth of each element corresponds to a k-approximate position),
- when $\mathsf{lfactor}(q)$ is not longer than k and even than l then it is trivially a k-approximate cover of $\boldsymbol{w}$ (redundant information).

Therefore, there is no k-approximate cover of $\boldsymbol{w}$ missing in B. And also only such string $\mathsf{lfactor}(q)$ is a k-approximate cover of $\boldsymbol{w}$. □

Example 4.2.28. Let us compute set $\mathsf{L}^{\mathrm{c}}_{\mathsf{H}^2}(\boldsymbol{w})$ of all restricted 2-approximate covers of string $\boldsymbol{w} =$ `acacca` using Algorithms 4.2 and 4.3 modified by Proposition 4.2.26. Transition diagram of nondeterministic 2-approximate suffix automaton without ε transitions for $\boldsymbol{w}$ is depicted at Figure 4.7. Then Algorithm 4.3 is used to construct (some) states of cover-candidate automaton. For each state q, its d-subset is constructed. Transition diagram of cover-candidate automaton as constructed using the Algorithm is depicted at Figure 4.8. State $2'3'4'5''6'$ is constructed and immediately discarded, because it does not meet the condition given by Proposition 4.2.26, i.e., it does not represent exact factor of $\boldsymbol{w}$. State 5 is also constructed and immediately discarded, because its d-subset contains only 1 element, i.e., it does not represent repeating k-approximate factor of $\boldsymbol{w}$. State

Algorithm 4.3 Process state of cover-candidate automaton for Hamming distance

Input: State q_i of partially constructed $\mathcal{M}^{\mathsf{H},k}_{\mathrm{C^D}}(\boldsymbol{w}) = (Q, A, \delta, q_0, F)$ having depth i, input alphabet A, maximum Hamming distance k, nondeterministic k-approximate suffix automaton $\mathcal{M}^{\mathsf{H},k}_{\mathrm{S^N}}(\boldsymbol{w}) = (Q_\mathrm{N}, A, \delta_\mathrm{N}, q_{0\mathrm{N}}, F_\mathrm{N})$ constructed using Alg. 3.7 and 3.4 for $\boldsymbol{w}$ and Hamming distance.

Output: The temporary (partial) set B' of smallest distance k-approximate covers of string $\boldsymbol{w}$.

1: **procedure** ProcessState($q_i, A, k, \mathcal{M}^{\mathsf{H},k}_{\mathrm{S^N}}(\boldsymbol{w})$)
2: $B' \leftarrow \emptyset$
3: **for all** $a \in A$ **do**
4: create new state q
5: $\mathsf{depth}(q) \leftarrow \mathsf{depth}(q_i) + 1$
6: **for all** $e_j \in \mathsf{d}(q_i)$ (in order by $\mathsf{d}(q_i)$) **do**
7: **for all** $e_l \in \delta_\mathrm{N}(e_j, a)$ in order by $\mathsf{depth}(e_l)$ **do**
8: append e_l to $\mathsf{d}(q)$
9: **end for**
10: **end for**
11: **if** $|\mathsf{d}(q)| > 1$ **then** ▷ $\mathsf{lfactor}(q)$ is a k-approximately repeating factor
12: $Q \leftarrow Q \cup \{q\}$
13: $\delta(q_i, a) \leftarrow q$
14: **if** $e_1 = \mathsf{depth}(q_i)$ for the first $e_1 \in \mathsf{d}(q)$ **then** ▷ q – k-approximate prefix
15: $u \leftarrow \mathsf{lfactor}(q) \leftarrow \mathsf{lfactor}(q_i).a$
16: **if** $\mathsf{depth}(e_{|\mathsf{d}(q)|}) = |\boldsymbol{w}|$ for the last $e_{|\mathsf{d}(q)|} \in \mathsf{d}(q)$ **then** ▷ q is final
17: **if** for all $i = 2, 3, \ldots, |\mathsf{d}(q)| : \mathsf{depth}(e_i) - \mathsf{depth}(e_{i-1}) \leq |\boldsymbol{u}|$ **then**
18: $l \leftarrow$ SmallestDistanceCover($\mathsf{d}(q)$) (Algorithm 4.4)
19: **if** $|\boldsymbol{u}| > l$ **then** ▷ $k \geq l$ always holds
20: $B' \leftarrow B' \cup (\boldsymbol{u}, l)$
21: **end if**
22: **end if**
23: **end if**
24: $B'' \leftarrow$ ProcessState($q, A, k, \mathcal{M}^{\mathsf{H},k}_{\mathrm{S^N}}(\boldsymbol{w})$)
25: $B' \leftarrow B' \cup B''$
26: **end if**
27: **end if**
28: discard state q from memory (correctly regarding $\mathcal{M}^{\mathsf{H},k}_{\mathrm{C^D}}(\boldsymbol{w})$)
29: **end for**
30: **return** B'
31: **end procedure**

Algorithm 4.4 Smallest distance of a cover of string $\boldsymbol{w}$ with Hamming distance.

Input: d-subset $\mathsf{d}(q) = \{e_1, e_2, \ldots, e_{|C|}\}$ of partially constructed cover-candidate automaton for string $\boldsymbol{w}$ and maximum Hamming distance k provided that $\boldsymbol{u} = \mathsf{lfactor}(q)$ is a k-approximate cover of $\boldsymbol{w}$.
Output: The smallest distance l of cover $\boldsymbol{u}$.

```
1: procedure SmallestDistanceCover(d(q), u)
2:     C ← d(q)
3:     l_min ← max{level(e_1), level(e_|C|)}      ▷ u is a k-approximate prefix and suffix of w
4:     l_max ← max_{e∈C}{level(e)}
5:     l ← l_max
6:     repeat
7:         for all e in C such that level(e) = l, e ≠ e_1, e ≠ e_|C| do
8:             remove e from C
9:         end for
10:        l ← l − 1
11:    until l ≥ l_min and when for all i = 2, 3, ..., |C| : depth(e_i) − depth(e_{i−1}) ≤ |u|
12:    l ← l + 1
13:    return l
14: end procedure
```

$45''6''$ is also constructed and immediately discarded, because depth of the first element of its d-subset is greater than string $\boldsymbol{u} = \mathtt{cac}$ of its left language, i.e., $\boldsymbol{u}$ is not a 2-approximate prefix of $\boldsymbol{w}$. State $q = 3'4''5''6$ represents string $\mathsf{lfactor}(q) = \mathtt{cca}$. Because depth of the first element of its d-subset is $3 = |\boldsymbol{u}|$ and because the last element of its d-subset is 6 and $\mathsf{depth}(6) = |\boldsymbol{w}|$, it represents 2-approximate prefix and 2-approximate suffix of $\boldsymbol{w}$. Because differences between depths of each two elements of d-subsets of q are $1 \leq |\boldsymbol{u}|$, $\boldsymbol{u}$ is a 2-approximate cover of $\boldsymbol{w}$. Then the smallest distance of $\boldsymbol{u}$ is computed (see Example 4.2.30). Some 2-approximate covers are not considered to be results due to relation between their length and smallest distance, e.g., state $23''45'6''$ represents 2-approximate suffix and prefix $\mathtt{ac}$ such that $\boldsymbol{w}$ can be constructed using superpositions of its 2-approximate occurrences in $\boldsymbol{w}$. However, its smallest distance is 2 and it is equal to its length, i.e., it 2-approximately occurs in every position in $\boldsymbol{w}$. Another example is state $2'3'4'56'$ that represents 1-approximate cover $\mathtt{cc}$. As its smallest distance is smaller than its length, it is considered to be a result. See the complete list of results in Table 4.1.

Lemma 4.2.29. Assuming string $\boldsymbol{w}$ and maximum Hamming distance k, every smallest distance of each found k-approximate cover is computed correctly using Algorithm 4.4.

Proof. Assume k-approximate cover $\boldsymbol{u}$ of $\boldsymbol{w}$ and k-approximate end set C as used in Algorithm 4.4. As $\boldsymbol{u}$ is a k-approximate suffix of $\boldsymbol{w}$, its rightmost k-approximate position must be considered and therefore $\boldsymbol{u}$ is a k-approximate cover of $\boldsymbol{w}$ with distance at least $\mathsf{level}(e_{|C|})$. As $\boldsymbol{u}$ is a k-approximate prefix of $\boldsymbol{w}$, its leftmost k-approximate position must be considered and therefore $\boldsymbol{u}$ is a k-approximate cover of $\boldsymbol{w}$ with distance at least $\mathsf{level}(e_1)$.

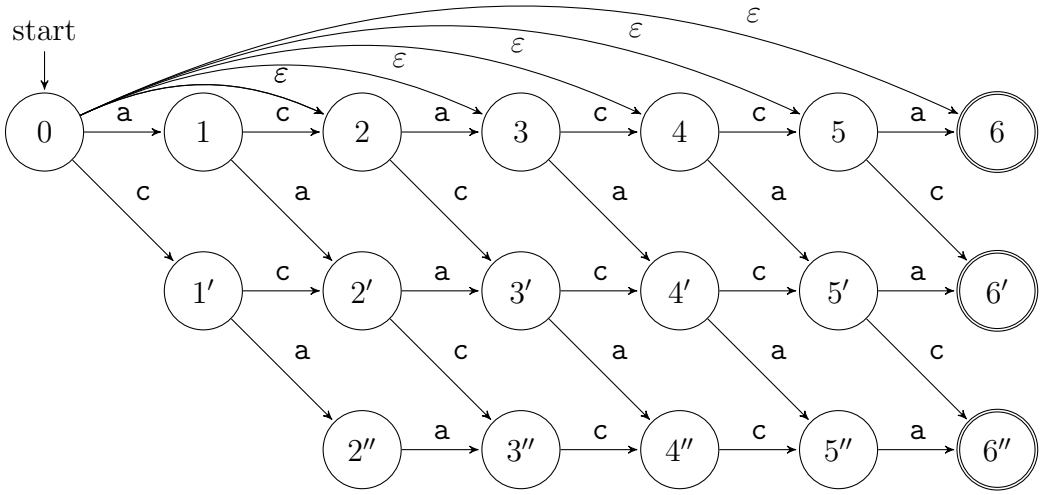

Figure 4.7: Transition diagram of nondeterministic 2-approximate suffix automaton for string `acacca`, Hamming distance and effective alphabet of `acacca`

Table 4.1: Restricted 2-approximate covers of `acacca` with Hamming distance

cover	d-subset	smallest distance
`cc`	$4''6$	1
`aca`	$35'6'$	1
`acc`	$3'4''56''$	2
`acac`	$46''$	2
`acca`	$4''6$	2
`cca`	$3'4''5''6$	1

In the inner loop in Algorithm 4.4 (lines 7–9), there the positions with the maximum approximation are removed, so they are no longer considered. The removing is being done until the condition (4.1) by Lemma 4.1.8 holds (line 11). The loop is terminated when the condition is not satisfied, therefore 1 must be added to the computed l. □

Example 4.2.30. Assume string $\boldsymbol{w} =$ `acacca` and let us compute the smallest Hamming distance (Algorithm 4.4) of its 2-approximate cover $\boldsymbol{u} =$ `cca` when d-subset of state q such that $\mathsf{lfactor}(q) = \boldsymbol{u}$ (corresponding to 2-approximate end set of $\boldsymbol{u}$ in $\boldsymbol{w}$, see Figure 4.9) is $\mathsf{d}(q) = C = \{3', 4'', 5'', 6\}$ (recall that for each element, its depth is denoted by numeric value, its level by number of primes). Initial values in the algorithm are computed as follows: $l_{\min} = 1, l_{\max} = 2$. During the first repeat-until loop iteration, elements of C having level equal to 2 are removed, i.e., $4''$, $5''$ and l is decremented to 1. Because for remaining consecutive elements of C holds: $\mathsf{depth}(6) - \mathsf{depth}(3') = 3 \leq |\boldsymbol{u}|$, the loop continues. During the second iteration of the loop, there is nothing to remove, just l is decremented. Now the set C is equal to $\{3', 6\}$, i.e., it is possible to construct $\boldsymbol{w}$ with

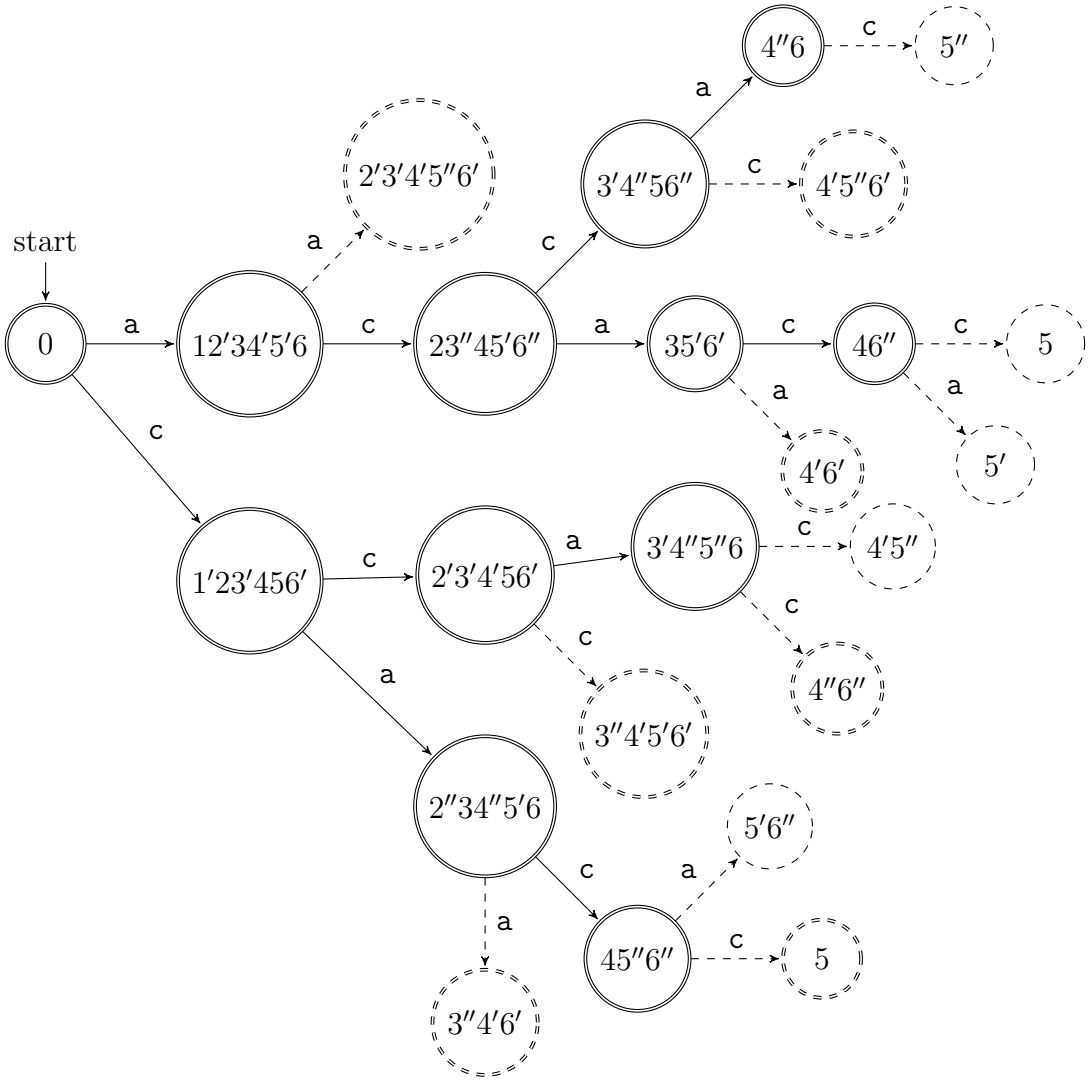

Figure 4.8: Deterministic 2-approximate trie-like suffix automaton for string `acacca` and Hamming distance (parts not constructed by Algorithm 4.3 for computation of restricted 2-approximate covers are dashed)

```
a c a c c a
c c a
  c c a
    c c a
      c c a
```

Figure 4.9: 2-approximate occurrences of string `cca` in string `acacca`

superposition of just 1-approximate occurrences of $\boldsymbol{u}$. Because $0 \not\geq l_{\min}$, the loop breaks and $l + 1 = 1$ is the computed smallest distance of approximate cover $\boldsymbol{u}$ of $\boldsymbol{w}$.

Theorem 4.2.31. Assuming string $\boldsymbol{w}$ and Hamming distance, the Algorithm 4.2 correctly computes set of all k-approximate covers of $\boldsymbol{w}$ and it computes the smallest distance for each found k-approximate cover.

Proof. Holds by Lemmas 4.2.27 and 4.2.29. □

Theorem 4.2.32 (Restricted k-approximate covers computation)**.** Assuming string $\boldsymbol{w}$ and maximum Hamming distance k, if Algorithm 4.3 does not process states q for that holds that there is no non-zero-level element in $\mathsf{d}(q)$, it computes just all restricted k-approximate covers of $\boldsymbol{w}$.

Proof. Such modification of the Algorithm means that only factors of $\boldsymbol{w}$ are processed (see Proposition 4.2.23). □

4.2.2.1 Time and space complexity

Note 4.2.33. As restricted k-approximate covers of string $\boldsymbol{w}$ are exact factors of $\boldsymbol{w}$, it is meaningful to consider only effective alphabet $A_{\boldsymbol{w}}$ for restricted k-approximate covers ($|A_{\boldsymbol{w}}| \leq |\boldsymbol{w}|$ always holds).

Note 4.2.34. It is meaningless to consider large k, because every factor of $\boldsymbol{w}$ having length less or equal to k is trivially k-approximate cover of $\boldsymbol{w}$. Thus $k \leq |\boldsymbol{w}|$ must always hold. Usually, $k \ll |\boldsymbol{w}|$ and $|A| \ll |\boldsymbol{w}|$ (e.g., in DNA analysis, $A = \{\mathtt{a}, \mathtt{c}, \mathtt{g}, \mathtt{t}\}$). Therefore k and $|A|$ may be considered as small constants independent on $|\boldsymbol{w}|$ for a variety of problems.

Lemma 4.2.35. Left languages of states of deterministic cover-candidate automaton $\mathcal{M}^{\mathsf{H},k}_{\mathrm{C^D}}(\boldsymbol{w}) = (Q_D, A_{\boldsymbol{w}}, \delta_D, q_0^D, F_D)$ constructed as a trie-like automaton (Algorithm 4.2) for computation of all restricted k-approximate covers of string $\boldsymbol{w}$ with Hamming distance are distinct, i.e.,

$$q_1, q_2 \in Q_D; q_1 \neq q_2 \Rightarrow \mathsf{lfactor}(q_1) \neq \mathsf{lfactor}(q_2)$$

By contradiction. Let us have following consideration: if there exist two states $q_1, q_2 \in Q_D, q_1 \neq q_2$ and $\mathsf{lfactor}(q_1) = \mathsf{lfactor}(q_2) = \boldsymbol{w}$, it means existence of two distinct sequences of transitions: $\delta_D^*(q_0^D, \boldsymbol{w}) = q_1$ and $\delta_D^*(q_0^D, \boldsymbol{w}) = q_2$. As Algorithm 4.2 creates new state for every $a \in A$, the resulting automaton $\mathcal{M}^{\mathsf{H},k}_{\mathrm{C^D}}(\boldsymbol{w})$ is deterministic and without cycles,

so such distinct sequences of transitions for the same string are not possible, thus either $q_1 = q_2$ or $\mathsf{lfactor}(q_1) \neq \mathsf{lfactor}(q_2)$. □

Note 4.2.36. Lemma 4.2.35 may be extended to deterministic cover-candidate automata for computation of all k-approximate (not only restricted) covers with Hamming distance without loss of generality. It holds by proof of the Lemma.

Lemma 4.2.37. In the cover-candidate automaton $\mathcal{M}^{\mathsf{H},k}_{\mathrm{C^D}}(\boldsymbol{w})$ for string $\boldsymbol{w}$ and the Hamming distance partially constructed using Algorithm 4.3 modified by Proposition 4.2.26 for computation of all restricted k-approximate covers of $\boldsymbol{w}$, having the number $R^k_{\mathsf{H}}(\boldsymbol{w})$ of all k-approximate repetitions of all factors in $\boldsymbol{w}$ (Definition 2.4.12), there are no more states assigned to set Q than

$$\frac{1}{2} \cdot (|\boldsymbol{w}|^2 + |\boldsymbol{w}|) - R^k_{\mathsf{H}}(\boldsymbol{w}) + 1$$

Proof. By Theorem 4.2.32, only states having factor of $\boldsymbol{w}$ in their left languages are considered to be states of Q. Number of factors of $\boldsymbol{w}$ is $\frac{1}{2} \cdot (|\boldsymbol{w}|^2 + |\boldsymbol{w}|)$. As left language of each state of Q contains exactly one string by Observation 4.2.5, the number of states of Q cannot be greater. By Lemma 4.2.35, a factor of $\boldsymbol{w}$ is contained in left language of exactly one state independent of number of its k-approximate repetitions, therefore $R^k_{\mathsf{H}}(\boldsymbol{w})$ is subtracted. One is added because of the initial state. □

Lemma 4.2.38. In the cover-candidate automaton $\mathcal{M}^{\mathsf{H},k}_{\mathrm{C^D}}(\boldsymbol{w})$ constructed using Alg. 4.2 for string $\boldsymbol{w}$, input alphabet A, and the maximum Hamming distance k for computation of all k-approximate covers of $\boldsymbol{w}$ there are no more states assigned to set Q than

$$\mathcal{O}(|\boldsymbol{w}|^{2 \cdot k} \cdot |A|^k)$$

Proof. Looking at Algorithm 4.3 and by proof of Lemma 4.2.27, there are processed states that have a k-approximate prefix of $\boldsymbol{w}$ in their left language. Therefore, the number of states of $\mathcal{M}^{\mathsf{H},k}_{\mathrm{C^D}}(\boldsymbol{w})$ cannot be greater. Considering a prefix $\boldsymbol{p}$ of $\boldsymbol{w}$, number of strings within Hamming distance at most k from $\boldsymbol{p}$ is $\mathcal{O}(|\boldsymbol{p}|^k \cdot |A|^k)$ (proved in [38]). All such strings are k-approximate prefixes of $\boldsymbol{w}$ of length equal to $|\boldsymbol{p}|$. Thus, number of all k-approximate prefixes of $\boldsymbol{w}$ is

$$\mathcal{O}(\sum_{i=1}^{|\boldsymbol{w}|} i^k \cdot |A|^k) = \mathcal{O}(|\boldsymbol{w}|^{2 \cdot k} \cdot |A|^k)$$

□

Lemma 4.2.39. Total number of states created using Algorithm 4.2 is at most

$$|A| \cdot |Q|$$

where Q is set of states assigned using Algorithm 4.3 of cover-candidate trie-like automaton $\mathcal{M}^{\mathsf{H},k}_{\mathrm{C^D}}(\boldsymbol{w})$ and A is input alphabet.

Proof. Using Algorithm 4.3 there are constructed not only states that are assigned to Q, but also states that have strings not being k-approximate prefix of $\boldsymbol{w}$ in their left languages. For every state q_i there is constructed a successor q for each $a \in A$, but not every such q is then assigned to Q. Number of such successors varies from 0 to $|A|$ for each state of Q. □

Note 4.2.40. Recall that for computation of all restricted k-approximate covers of string $\boldsymbol{w}$, $A = A_{\boldsymbol{w}}$ is considered and thus $|A| \leq |\boldsymbol{w}|$.

Lemma 4.2.41. For every state q of cover-candidate automaton $\mathcal{M}$ constructed for the Hamming distance by Algorithm 4.2 holds: there are no two elements of $\mathsf{d}(q)$ having the same depth.

Proof. It holds from properties of transition function of nondeterministic suffix automaton $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SN}}(\boldsymbol{w}) = (Q_{\mathrm{N}}, A, \delta_{\mathrm{N}}, q_{0\mathrm{N}}, F_{\mathrm{N}})$ for string $\boldsymbol{w}$ and maximum Hamming distance k: for any two direct successors of the initial state $q_{0\mathrm{N}}$ holds:

$$\mathsf{depth}(e_i) \neq \mathsf{depth}(e_j)$$

Therefore, d-subsets of successors of initial state of $\mathcal{M} = (Q, A, \delta, q_0, F)$ contain elements with distinct depths only. For any successors e_j of all states e_i of $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SN}}(\boldsymbol{w})$ but the initial one holds:

$$\forall a \in A, \forall e_i \in Q_{\mathrm{N}} \setminus \{q_{0\mathrm{N}}\} : \forall e_j \in \delta_{\mathrm{N}}(e_j, a) : \mathsf{depth}(e_j) = \mathsf{depth}(e_i) + 1$$

Let us use induction. Successors of initial state q_0 of $\mathcal{M}$ have no two elements with the same depth in its d-subset. Let us consider any state $q_i \in Q \setminus \{q_0\}$ having no two elements with the same depth in its d-subset. Any successor q_j of such state q_i cannot have d-subset having some elements with the same depth, as any element e^s of $\mathsf{d}(q_j)$ is constructed from element $e^i \in \mathsf{d}(q_i)$ this way (lines 7–9 of Algorithm 4.3): $e^s \in \delta_{\mathrm{N}}(e^i, a), a \in A$ and thus $\mathsf{depth}(e^s) = \mathsf{depth}(e^i) + 1$. Therefore, the lemma holds for all d-subsets of $\mathcal{M}$. □

Lemma 4.2.42. Number of all elements of all d-subsets of part of cover-candidate automaton $\mathcal{M} = (Q, A, \delta, q_0, F)$ constructed using Algorithm 4.2 for string $\boldsymbol{w}$ is not greater than

$$|\boldsymbol{w}| \cdot |Q|$$

Proof. Assuming a k-approximate nondeterministic suffix automaton $(Q_{\mathrm{N}}, A, \delta_{\mathrm{N}}, q_{0\mathrm{N}}, F_{\mathrm{N}})$, for its transition function holds:

$$\forall a \in A : |\delta_{\mathrm{N}}(q_{0\mathrm{N}}, a)| = |\boldsymbol{w}|$$

and

$$\forall a \in A, \forall r \in Q_{\mathrm{N}} \setminus \{q_{0\mathrm{N}}\} : |\delta_{\mathrm{N}}(r, a)| \leq 1$$

By Lemma 4.2.41, it is obvious that for all states q of $\mathcal{M}$ holds $|\mathsf{d}(q)| \leq |\boldsymbol{w}|$ and moreover for initial state q_0 of $\mathcal{M}$ holds $|\mathsf{d}(q_0)| = 1$. Therefore, number of elements of all d-subsets cannot be greater than $|\boldsymbol{w}|$-times number of states but the initial one. □

Lemma 4.2.43. Number of elements of all d-subsets constructed using Algorithm 4.2 is at most

$$|A| \cdot |\boldsymbol{w}| \cdot |Q|$$

where Q is set of all states of cover-candidate automaton $\mathcal{M}$ assigned by Algorithm 4.2 for string $\boldsymbol{w}$ and input alphabet A.

Proof. It clearly holds by Lemmas 4.2.39 and 4.2.42. □

Lemma 4.2.44. Time complexity of Algorithm 4.4 (the computation of the smallest distance of k-approximate cover $\boldsymbol{u} = \mathsf{lfactor}(q)$ of string $\boldsymbol{w}$ with maximum Hamming distance k) is at most

$$5 + k \cdot (2 \cdot |\mathsf{d}(q)| - 5)$$

that is

$$\mathcal{O}(k \cdot |\boldsymbol{w}|)$$

Proof. Value $l_{\min}$ is retrieved in constant time, retrieving $l_{\max}$ takes $|\mathsf{d}(q)|$ time that is $\mathcal{O}(|\boldsymbol{w}|)$ (follows by proof of Lemma 4.2.42). Then there are at most $l \leq k$ iterations in Algorithm 4.4 (lines 6–11). Each iteration means removal of some elements of a position list ($|\mathsf{d}(q)| - 2$ for the first iteration, during that at least one element is removed) and check whether $\boldsymbol{u}$ is still a k-approximate cover of $\boldsymbol{w}$ (takes $|\mathsf{d}(q)| - 1$ for the first iteration). □

Theorem 4.2.45. Time complexity of computation of all restricted k-approximate covers with their smallest distance for string $\boldsymbol{w}$ and alphabet A with maximum Hamming distance k (Algorithm 4.2 modified according to Proposition 4.2.26) is no more than

$$\mathcal{O}(|\boldsymbol{w}|^3 \cdot (|A| + k))$$

Time complexity of computation of all k-approximate covers with their smallest distance for string $\boldsymbol{w}$ and alphabet A with maximum Hamming distance k (Algorithm 4.2) is

$$\mathcal{O}(|\boldsymbol{w}|^{2 \cdot k + 1} \cdot |A|^k \cdot (|A| + k))$$

Proof. Construction of $\mathcal{M}^{\mathsf{H},k}_{\mathrm{S^N}}(\boldsymbol{w}) = (Q_{\mathrm{N}}, A_{\boldsymbol{w}}, \delta_{\mathrm{N}}, q_{0\mathrm{N}}, F_{\mathrm{N}})$ for $\boldsymbol{w}$ and k takes $|A| \cdot (|\boldsymbol{w}| \cdot (k + 1) - 1 + \frac{k - k^2}{2}) + |\boldsymbol{w}| - k + 1$ by Lemma 3.1.35 and Algorithm 3.7. For each already constructed state of cover-candidate automaton $\mathcal{M}^{\mathsf{H},k}_{\mathrm{C^D}}(\boldsymbol{w}) = (Q, A, \delta, q_0, F)$ and for each symbol of A, new d-subset is constructed. As each element of any d-subset may be constructed in constant time (just using already known δ_{N}) and the elements are naturally ordered (holds by Lemma 4.2.6, because the d-subsets are computed in the same way in Algorithm 4.3 as in Algorithm 4.1), all d-subsets are constructed in at most $|A| \cdot |Q| \cdot |\boldsymbol{w}|$ time (by Lemma 4.2.43). The left language extraction of state takes constant time. The computation of the smallest distance takes at most $|\boldsymbol{w}| + k \cdot (3 \cdot |\boldsymbol{w}| - 2)$ for each state of $\mathcal{M}^{\mathsf{H},k}_{\mathrm{C^D}}(\boldsymbol{w})$ (by Lemma 4.2.44). Therefore, the time complexity is no more than

$$|Q| \cdot (|\boldsymbol{w}| \cdot (|A| + 3 \cdot k + 1) - 2 \cdot k)$$

For computation of all k-approximate covers with Hamming distance, number of states of $\mathcal{M}^{\mathsf{H},k}_{\mathrm{C^D}}(\boldsymbol{w})$ is $|Q| = \mathcal{O}(|\boldsymbol{w}|^{2\cdot k} \cdot |A|^k)$ (by Lemma 4.2.38).

For computation of all restricted k-approximate covers with Hamming distance, there are at most $|\boldsymbol{w}|$ more operations for each state, because zero-level element is searched in each constructed d-subset. Number of states of $\mathcal{M}^{\mathsf{H},k}_{\mathrm{C^D}}(\boldsymbol{w})$ in this case is $\frac{1}{2} \cdot (|\boldsymbol{w}|^2 + |\boldsymbol{w}|) - R^k_{\mathsf{H}}(\boldsymbol{w}) + 1$ (by Lemma 4.2.37). Therefore, the time complexity of computing all restricted k-approximate covers with Hamming distance is no more than

$$\frac{1}{2} \cdot |\boldsymbol{w}|^3 \cdot (|A| + 3 \cdot k + 1) + \frac{1}{2} \cdot |\boldsymbol{w}|^2 \cdot (|A| + k + 1) + \\ + |\boldsymbol{w}| \cdot \left(|A| + \frac{5}{2} \cdot k + 1 - |A| \cdot R^k_{\mathsf{H}}(\boldsymbol{w}) - 3 \cdot k \cdot R^k_{\mathsf{H}}(\boldsymbol{w}) - R^k_{\mathsf{H}}(\boldsymbol{w}) \right) + \\ + 2 \cdot k \cdot (R^k_{\mathsf{H}}(\boldsymbol{w}) - 1)$$

□

Note 4.2.46. Number of all k-approximate covers of string $\boldsymbol{w}$ for input alphabet A and maximum Hamming distance k is $\mathcal{O}(|\boldsymbol{w}|^{2\cdot k} \cdot |A|^k)$ (like number of k-approximate prefixes of $\boldsymbol{w}$), thus sum of their lengths is

$$\|\mathsf{L}^{\mathrm{c}}_{\mathsf{H}^k}(\boldsymbol{w})\| = \mathcal{O}(|\boldsymbol{w}|^{2\cdot k+1} \cdot |A|^k)$$

Number of all restricted k-approximate covers of string $\boldsymbol{w}$ for maximum Hamming distance k is $\mathcal{O}(|\boldsymbol{w}|^2)$ (like number of factors of $\boldsymbol{w}$). Thus, the sum of their lengths is

$$\|\mathsf{L}^{\mathrm{c}}_{\mathsf{H}^k}(\boldsymbol{w})\| = \mathcal{O}(|\boldsymbol{w}|^3)$$

Lemma 4.2.47. During construction of cover-candidate automaton $\mathcal{M}^{\mathsf{H},k}_{\mathrm{C^D}}(\boldsymbol{w})$ using Algorithm 4.2), there are $\mathcal{O}(|\boldsymbol{w}|^2)$ elements of d-subsets stored in memory at a time.

Proof. From Algorithm 4.3 follows that for each constructed and in memory preserved state, at most one successor is constructed. Therefore, at most $|\boldsymbol{w}|$ states are stored in memory at a time. By proof of Lemma 4.2.42, each d-subset contains at most $|\boldsymbol{w}|$ elements. □

Theorem 4.2.48. Space complexity of Algorithm 4.2 for string $\boldsymbol{w}$, alphabet A, and maximum Hamming distance k is

$$\mathcal{O}(|\boldsymbol{w}| \cdot (|\boldsymbol{w}| + |A|))$$

Proof. It clearly holds that for construction of the nondeterministic suffix automaton $\mathcal{M}^{\mathsf{H},k}_{\mathrm{S^N}}(\boldsymbol{w}) = (Q_{\mathrm{N}}, A, \delta_{\mathrm{N}}, q_{0\mathrm{N}}, F_{\mathrm{N}})$, there is no need for any additional data structures. For the purpose of the construction of the cover-candidate automaton $\mathcal{M}^{\mathsf{H},k}_{\mathrm{C^D}}(\boldsymbol{w})$, only the set of states and transitions from $q_{0\mathrm{N}}$ need to be preserved, because the rest may be computed later in $\mathcal{O}(1)$ time and space using knowledge of a depth and a level of a state, k, and $\boldsymbol{w}$. Thus the space complexity of this construction is $\mathcal{O}((k + |A|) \cdot |\boldsymbol{w}|)$.

During the computation of the smallest distance (Algorithm 4.4), only $\mathcal{O}(1)$ additional space is needed. During the processing of states of $\mathcal{M}^{\mathsf{H},k}_{\mathrm{C^D}}(\boldsymbol{w})$ (Algorithm 4.3), the needed space is limited by the number of elements of all d-subsets preserved in a memory at a time, that is $\mathcal{O}(|\boldsymbol{w}|^2)$ by Lemma 4.2.47.

Therefore, the space complexity of Algorithm 4.2 is $\mathcal{O}(|\boldsymbol{w}| \cdot (|\boldsymbol{w}| + |A|))$, as $|A|$ may be greater than $|\boldsymbol{w}|$ in general. □

Note 4.2.49. The space needed by Algorithm 4.2 is also determined by the number of all k-approximate covers (the result, limited by the number of all k-approximate prefixes of $\boldsymbol{w}$ – $\mathcal{O}(|\boldsymbol{w}|^{2 \cdot k} \cdot |A|^k)$) for k-approximate covers and moreover by the number of all factors of $\boldsymbol{w}$ that is $\mathcal{O}(|\boldsymbol{w}|^2)$ – see Note 4.2.46.

4.3 Experimental results

The algorithms for searching covers with Hamming distance were implemented in C++ using STL. The implementation was compiled using GCC version 4.7.3 with O3 optimisation level. The experiments were run on a PC with Intel Core i5 CPU (2.5 GHz) and 4 GB RAM under Linux (Gentoo hardened kernel version 3.12.5-r1) with swap disabled. As input data, Saccharomyces cerevisiae S288c chromosome IV[1] was used. For various input lengths, input string consists of first n characters of the chromosome. For each input length n and maximum Hamming distance k, the following values were measured using GNU time utility:

- elapsed time as total number of CPU-seconds that the implemented program spent in user mode,
- memory consumption as maximum resident set size of the implemented program.[2]

Moreover, the following values were recorded:

- number of computed results (regularities found),
- maximum (peak) sum of sizes of all d-subsets stored in memory at a time,
- total number of states of a deterministic (trie-like) automaton that were examined,
- maximum (peak) number of states stored in memory at a time.

The implementation computes all the covers of its input and it does not store them in memory, instead, each found cover is reported to standard output as soon as it is found.

[1] The sequence was downloaded from ncbi.nlm.nih.gov.

[2] Due to bug [26] in the used GNU time utility version 1.7, the peak memory usage presented in this dissertation thesis is the maximum resident set size reported by GNU time utility divided by 4.

4.3.1 Restricted approximate covers

In this Section, results of experimental run of implementation of Algorithm 4.2 modified according to Proposition 4.2.26 are presented. Some properties of the input are shown at Figures 4.10 and 4.11 – number of found restricted k-approximate covers with Hamming distance in a prefix of input sequence. As expected, the number of found restricted k-approximate covers does not necessarily grow with length of input; for the same input it grows with k as more factors are k-approximate covers with higher k. It is also visible that for particular input, considering high values of maximum distance k (e.g., 9) does not produce meaningful results, as their number is greater than length of input.

Time needed to run the computation depending on input length and for a few different values of maximum Hamming distance is shown in Figure 4.12. Another view is shown in Figure 4.11 where the elapsed time together with number of found restricted k-approximate covers is depicted depending on maximum Hamming distance k for particular input length n. Even another view is provided by Figure 4.13 where the elapsed time together with number of all states being examined is depicted depending on maximum Hamming distance k for particular input length n. Elapsed time grows similarly to the number of processed states. As expected by Theorem 4.2.45, elapsed time grows with both n and k.

Memory needed for computation depending on input length and for particular value of maximum Hamming distance k is shown in Figure 4.14; the same figure shows how sum of sizes of all d-subsets stored in memory at a time depends on input length compared to real memory consumption. The memory consumption and the sum of sizes of all stored d-subsets grow similarly depending on input length. Similar view for number of states stored in memory at a time is depicted in Figure 4.15. Similar relation as between sum of sizes of stored d-subsets and memory consumption is not visible here. The of dependence of peak number of states (stored in memory at a time) on input length as shown in Figure 4.15 is result of property of input data. The number of states stored in memory at a time is equal to the longest k-approximately repeating factor of prefix of length n (depth-first search) and for particular input data, for any prefix longer than n there may exist no longer k-approximately repeating factor. Another view is shown in Figure 4.16 where the memory needed compared to the sum of sizes of all d-subsets stored in memory at a time is depicted depending on maximum Hamming distance k for particular input length n. It may seem to be in contrast to Theorem 4.2.48, as the memory consumption and the sum of sizes of all stored d-subsets grow with k. Due to line 11 of Algorithm 4.3, only states representing k-approximately repeating factors are processed and their number grows with k. In Theorem 4.2.48, processing of all k-approximate prefixes is considered, even of those that are not repeating.

The complete data set from experimental run is shown in Table B.1.

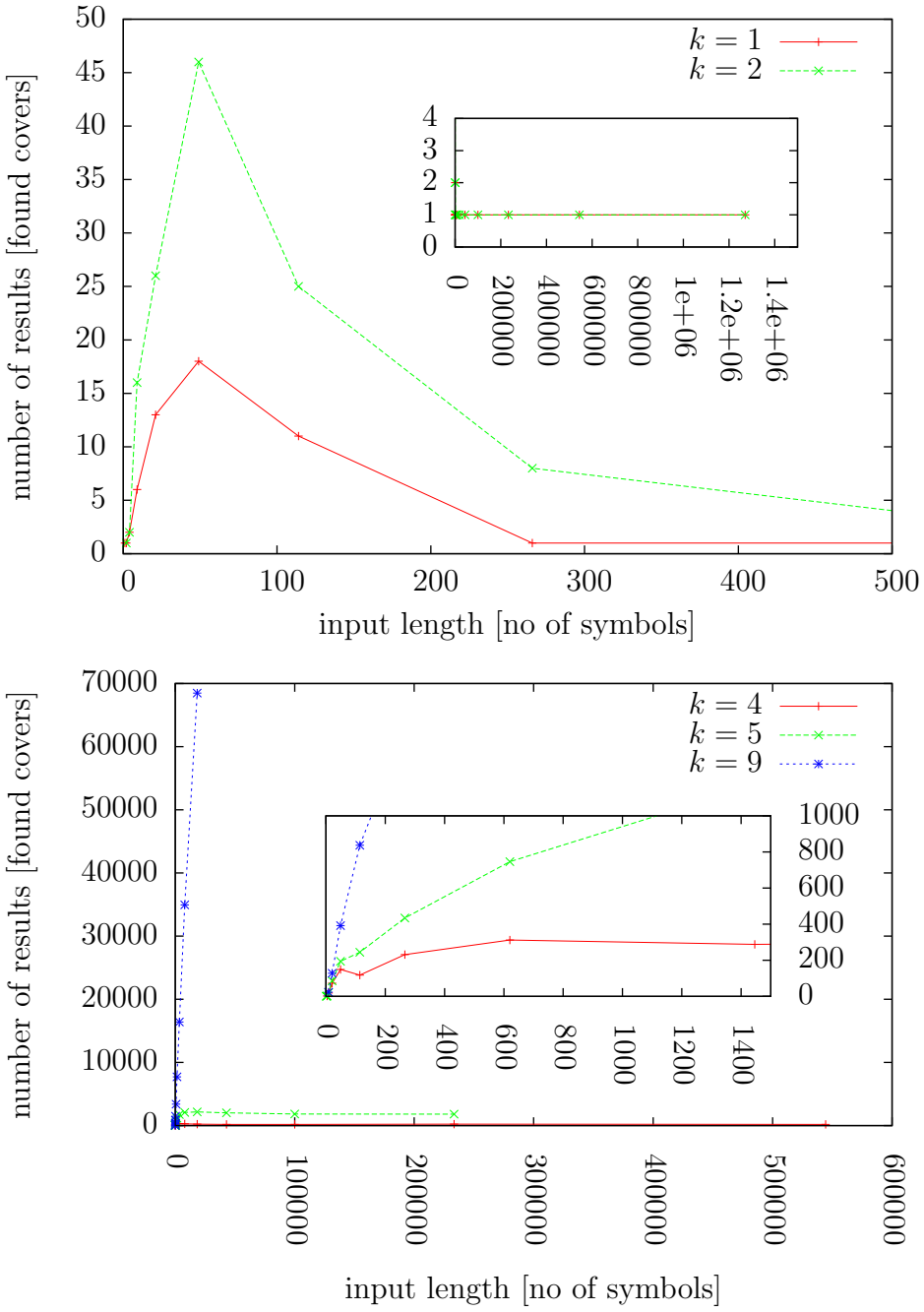

Figure 4.10: Number of all restricted k-approximate covers for particular maximum Hamming distance

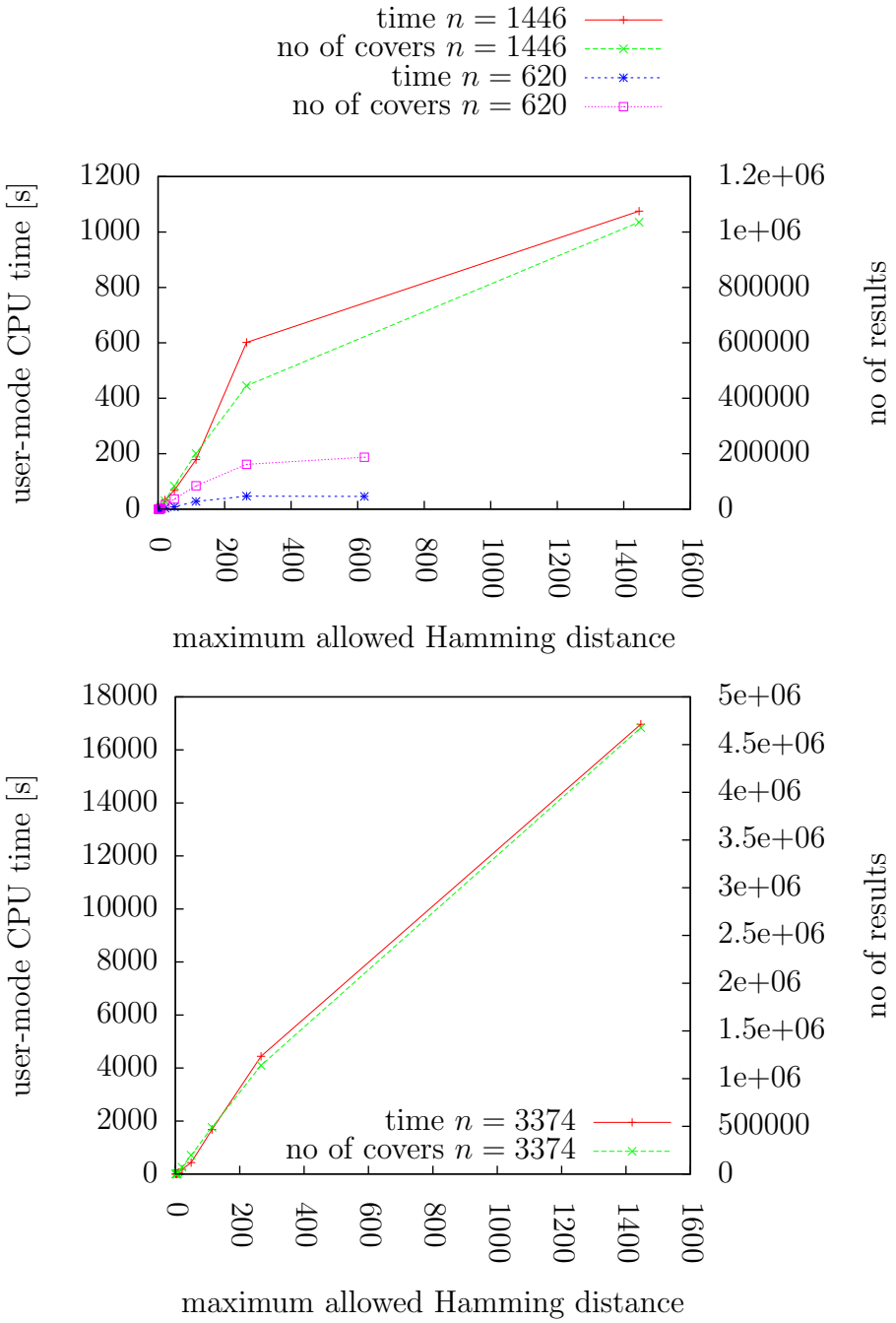

Figure 4.11: Elapsed time for computation of all restricted k-approximate covers for particular input length n compared to number of found results

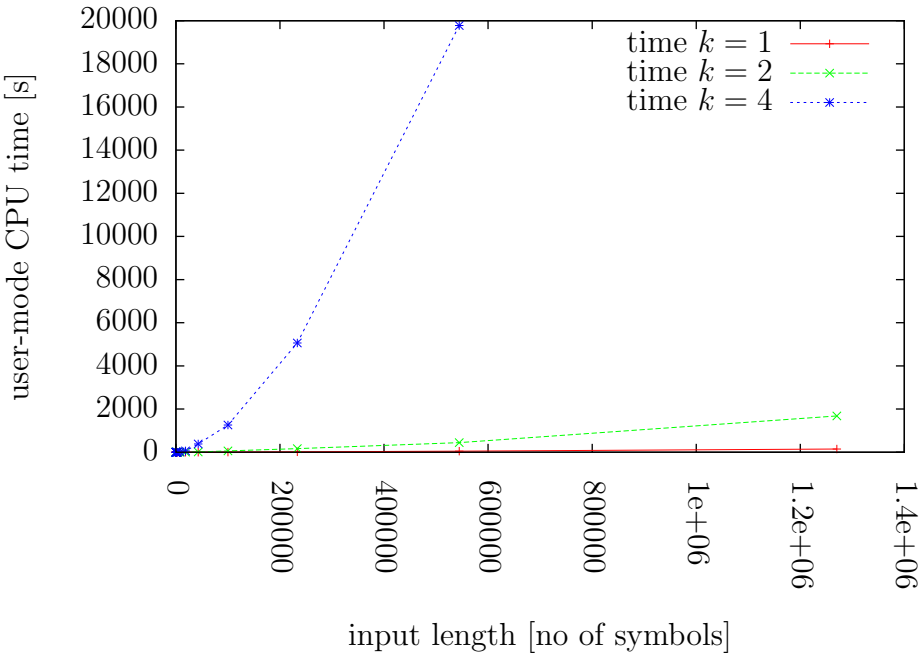

Figure 4.12: Elapsed time for computation of all restricted k-approximate covers for particular maximum Hamming distance k

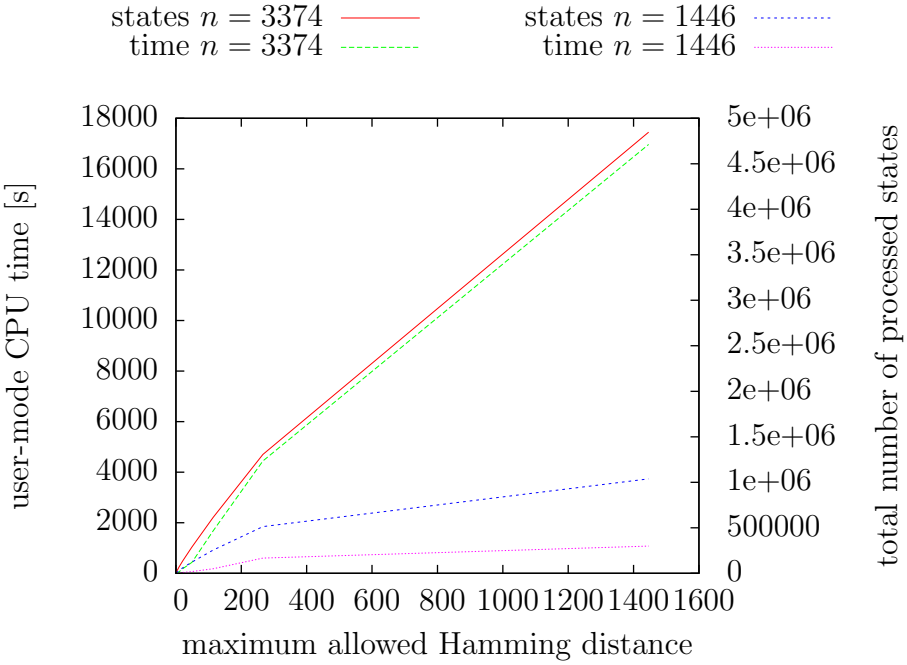

Figure 4.13: Number of states processed during computation of restricted k-approximate covers for particular input length n

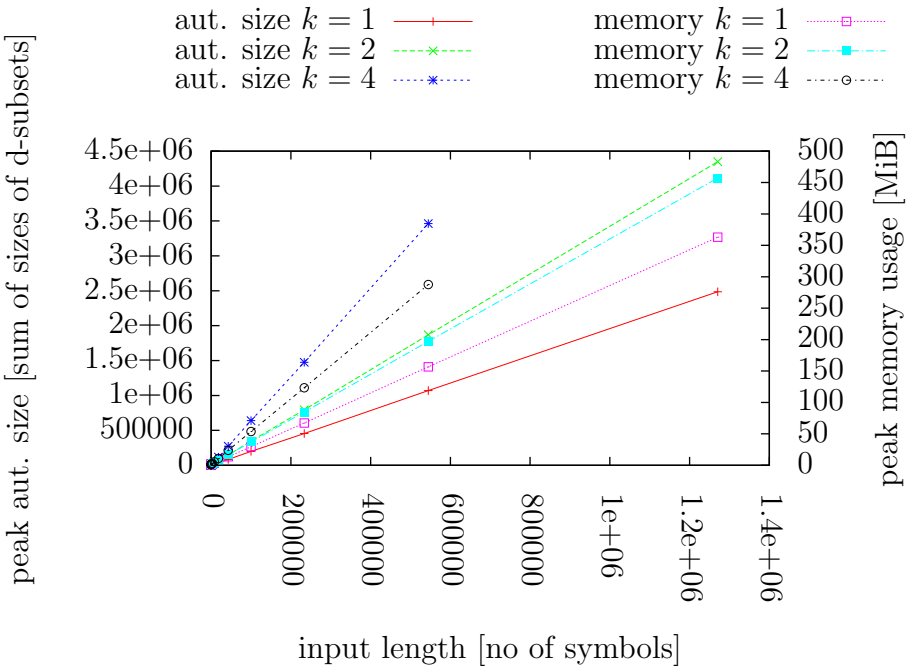

Figure 4.14: Memory needed for computation of all restricted k-approximate covers for particular maximum Hamming distance compared to sum of sizes of all d-subsets stored in memory at a time

4.3.2 Approximate covers

In this Section, results of experimental run of implementation of Algorithm 4.2 for computation of all k-approximate covers with Hamming distance are presented. As alphabet A, effective alphabet of particular prefix of length n of input data is used. Some properties of the input are shown at Figures 4.17 and 4.18 – number of found k-approximate covers with Hamming distance in a prefix of input sequence. As expected, the number of found k-approximate covers does not necessarily grow with length of input; for the same input it grows with k as more strings are k-approximate covers with higher k. Similarly as for computing restricted k-approximate covers, for particular input and with high values of k (e.g., 9), number of results is too high (compared to input length n) to consider them meaningful.

Time needed to run the computation depending on input length and for a few different values of maximum Hamming distance is shown in Figure 4.19. As expected, the elapsed time grows similarly to the number of processed states of constructed deterministic automaton, i.e., similarly to number of all processed k-approximately repeating factors of the input that need to be processed and checked. The dependence of the number of processed states on k is depicted in Figure 4.20. Another view is shown in Figure 4.18 where the elapsed time together with number of found k-approximate covers is depicted depending on maximum Hamming distance k for particular input length n. As expected

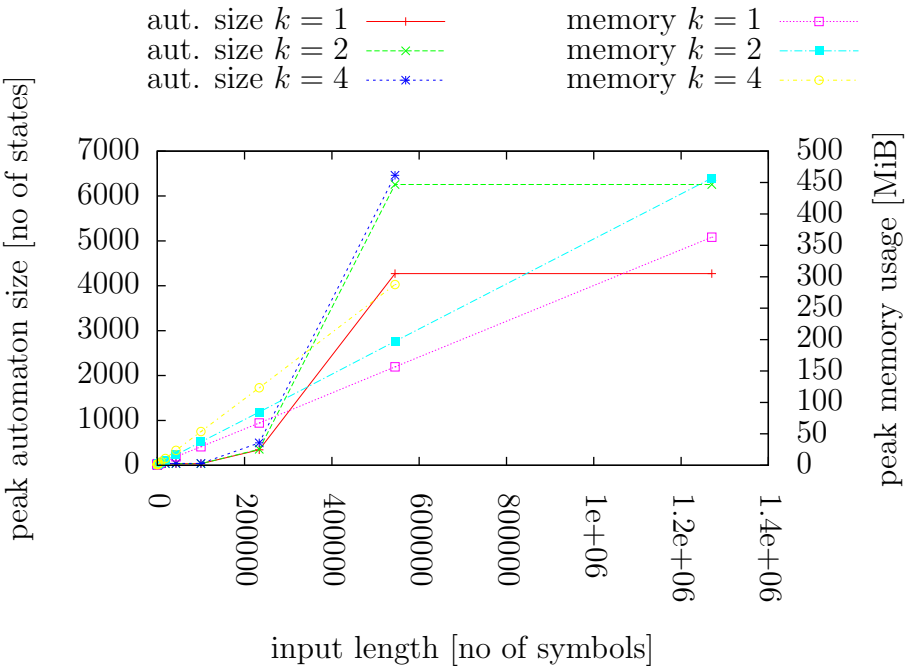

Figure 4.15: Memory needed for computation of all restricted k-approximate covers for particular maximum Hamming distance compared to maximum number of states stored during computation in memory at a time

by Theorem 4.2.45, elapsed time grows with both n and k.

Memory consumption of computation depending on input length and for particular value of maximum Hamming distance k is shown in Figure 4.22; the same figure shows how sum of sizes of all d-subsets stored in memory at a time depends on input length compared to real memory consumption. Similar view for number of states stored in memory at a time is depicted in Figure 4.21. The memory consumption and the sum of sizes of all stored d-subsets grow similarly depending on input length. Similar relation between memory consumption and maximal number of states stored in memory at a time is not visible (Figure 4.21). The latter corresponds to the length of the longest string that is a k-approximately repeating factor of prefix of input of length n and this is a property of particular input data. Another view is shown in Figure 4.23 where the memory needed compared to the sum of sizes of all d-subsets stored in memory at a time is depicted depending on maximum Hamming distance k for particular input length n. It may seem as in contrast to Theorem 4.2.48, as the memory consumption and the sum of sizes of all stored d-subsets grow with k. The reason is the same as for restricted k-approximate covers (see the previous section).

The complete data set from experimental run is shown in Table B.2.

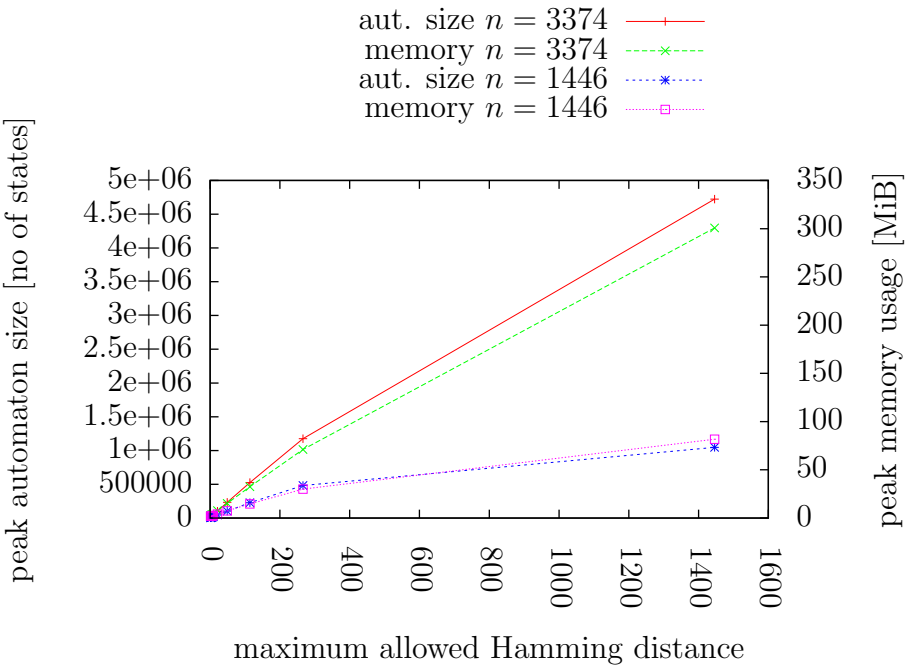

Figure 4.16: Memory needed for computation of all restricted k-approximate covers for particular input length n compared to maximum number of states stored during computation in memory at a time

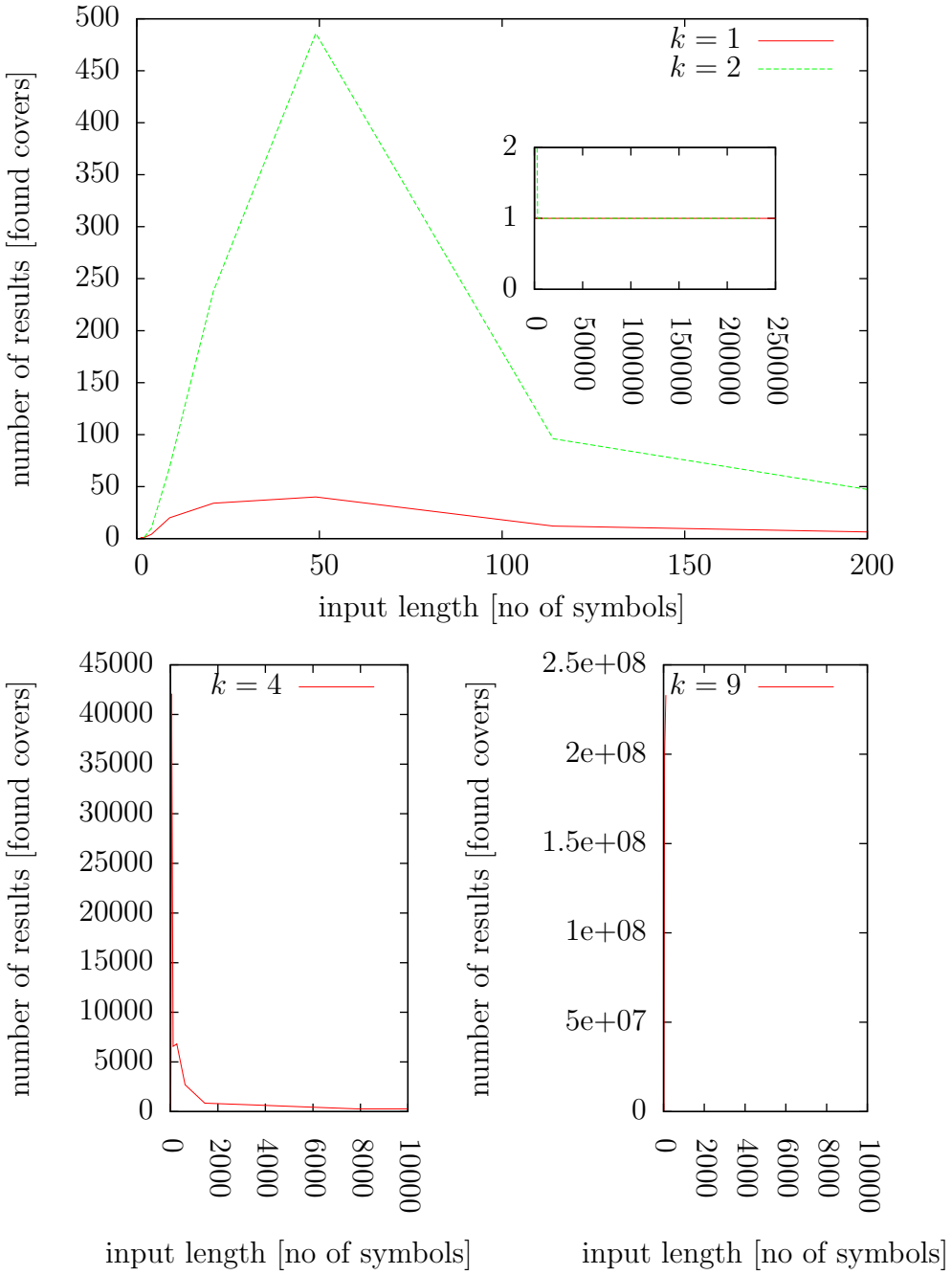

Figure 4.17: Number of all k-approximate covers for particular maximum Hamming distance

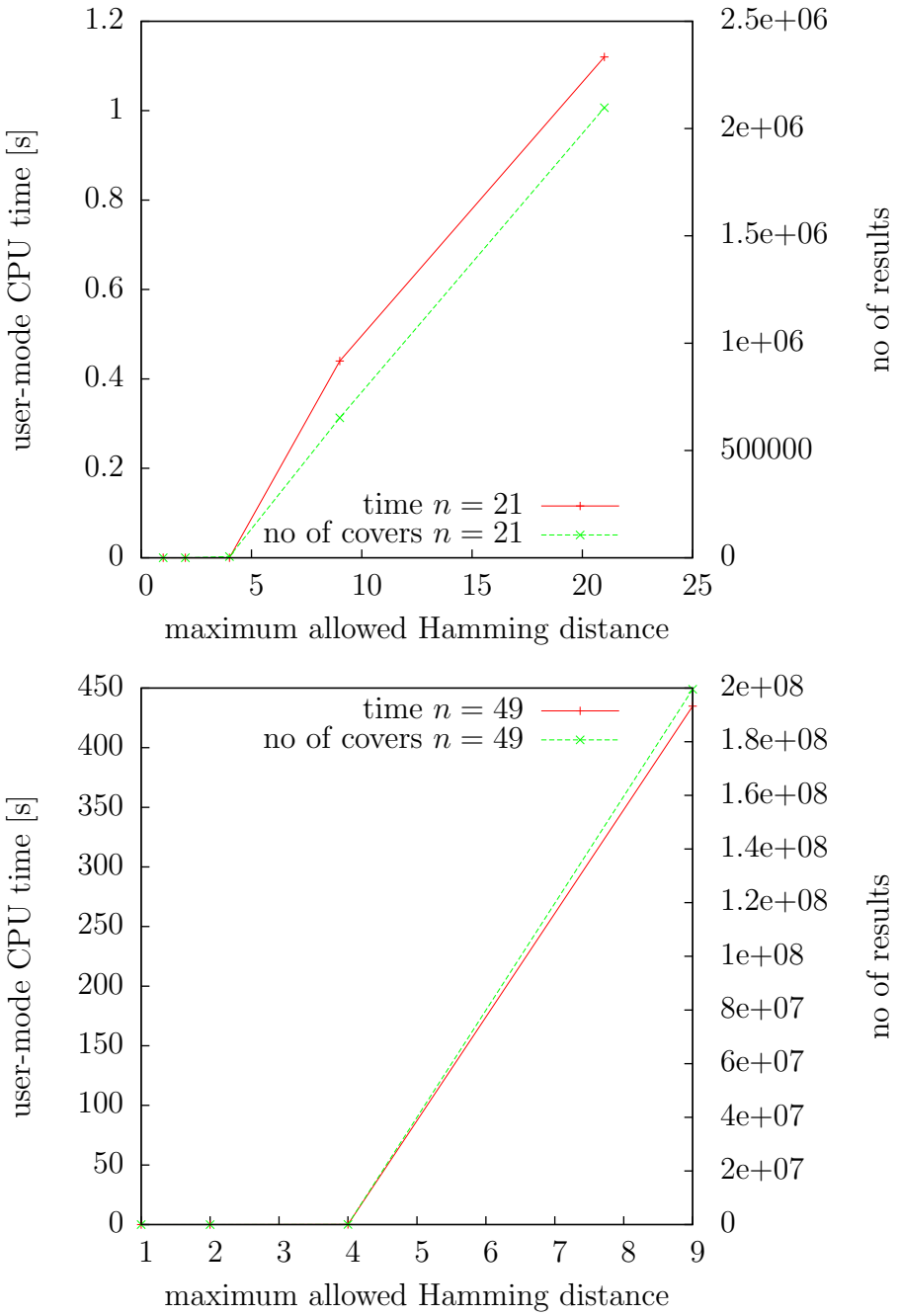

Figure 4.18: Elapsed time for computation of all k-approximate covers for particular input length n compared to number of found results

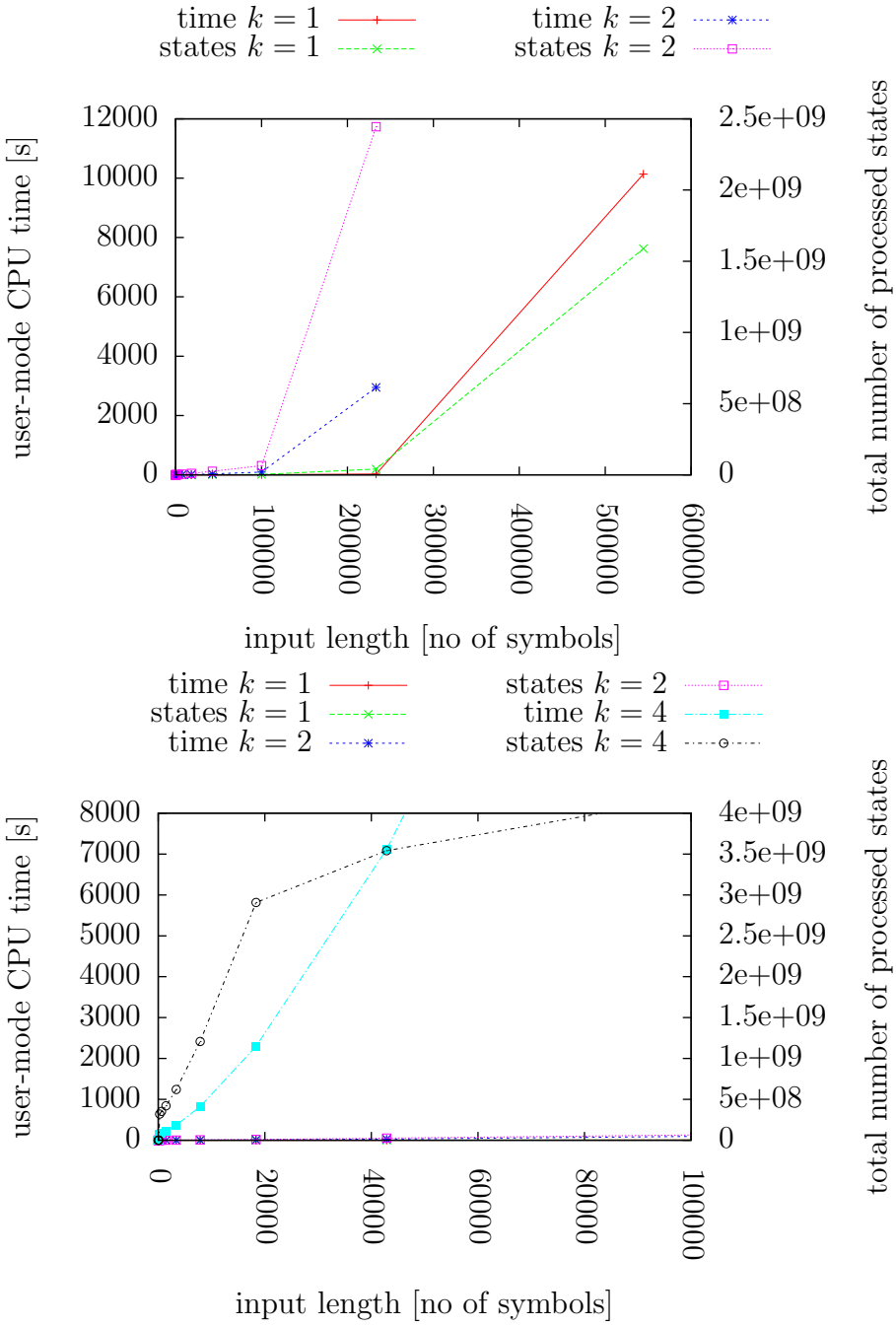

Figure 4.19: Elapsed time for computation of all k-approximate covers for particular maximum Hamming distance k compared to number of processed states

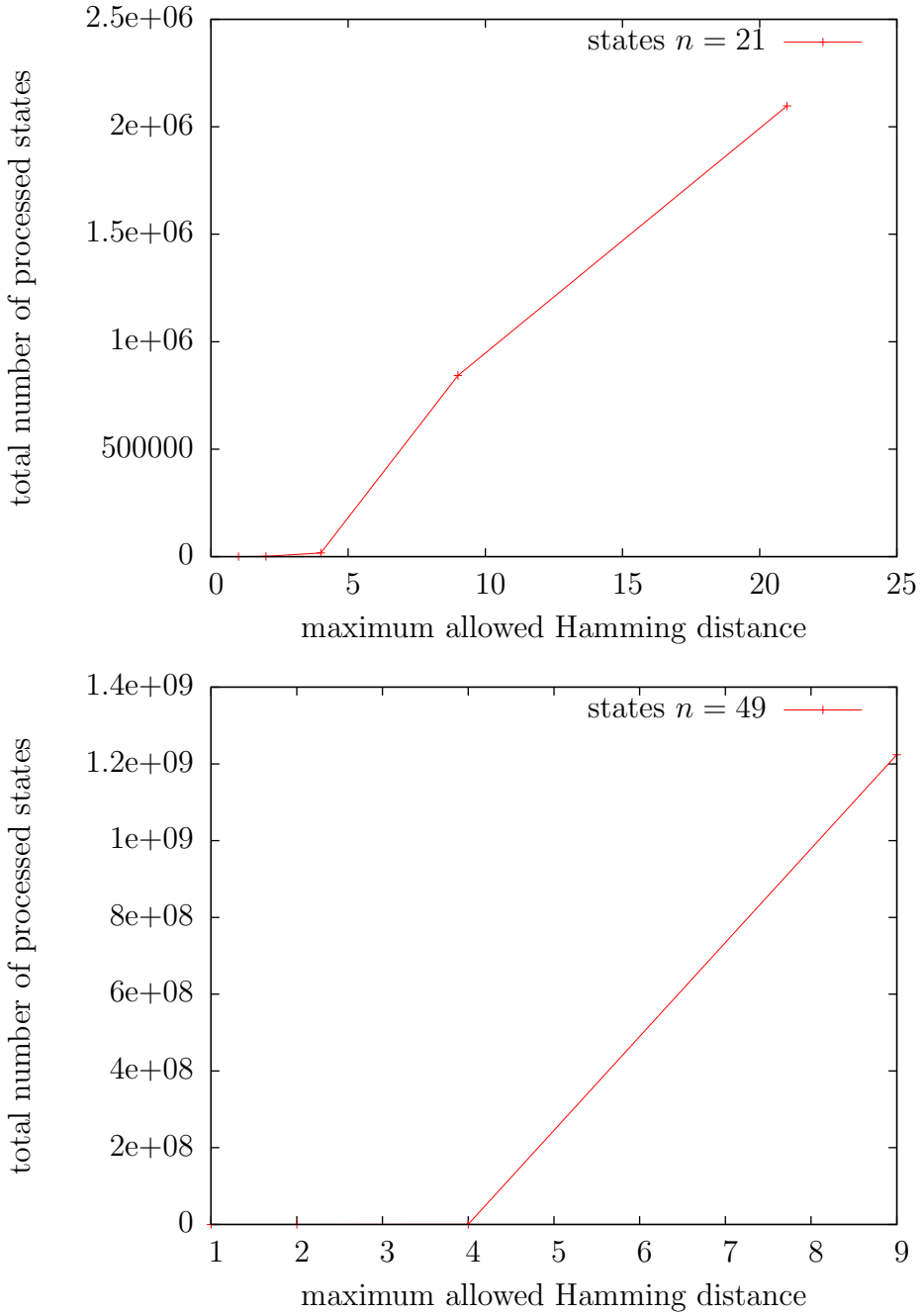

Figure 4.20: Number of states processed during computation of k-approximate covers for particular input length n

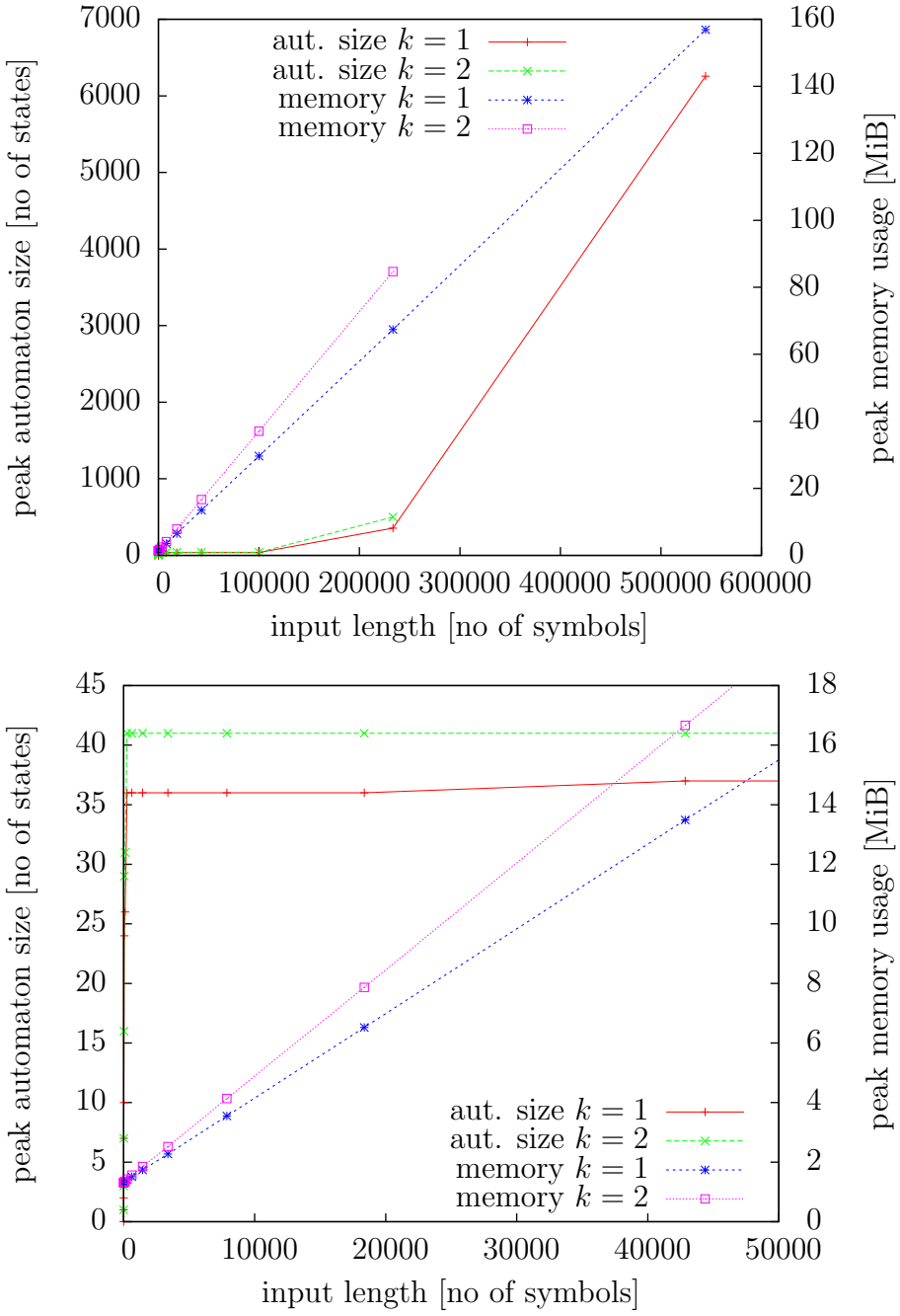

Figure 4.21: Memory needed for computation of all k-approximate covers for particular maximum Hamming distance compared to number of states stored in memory at a time

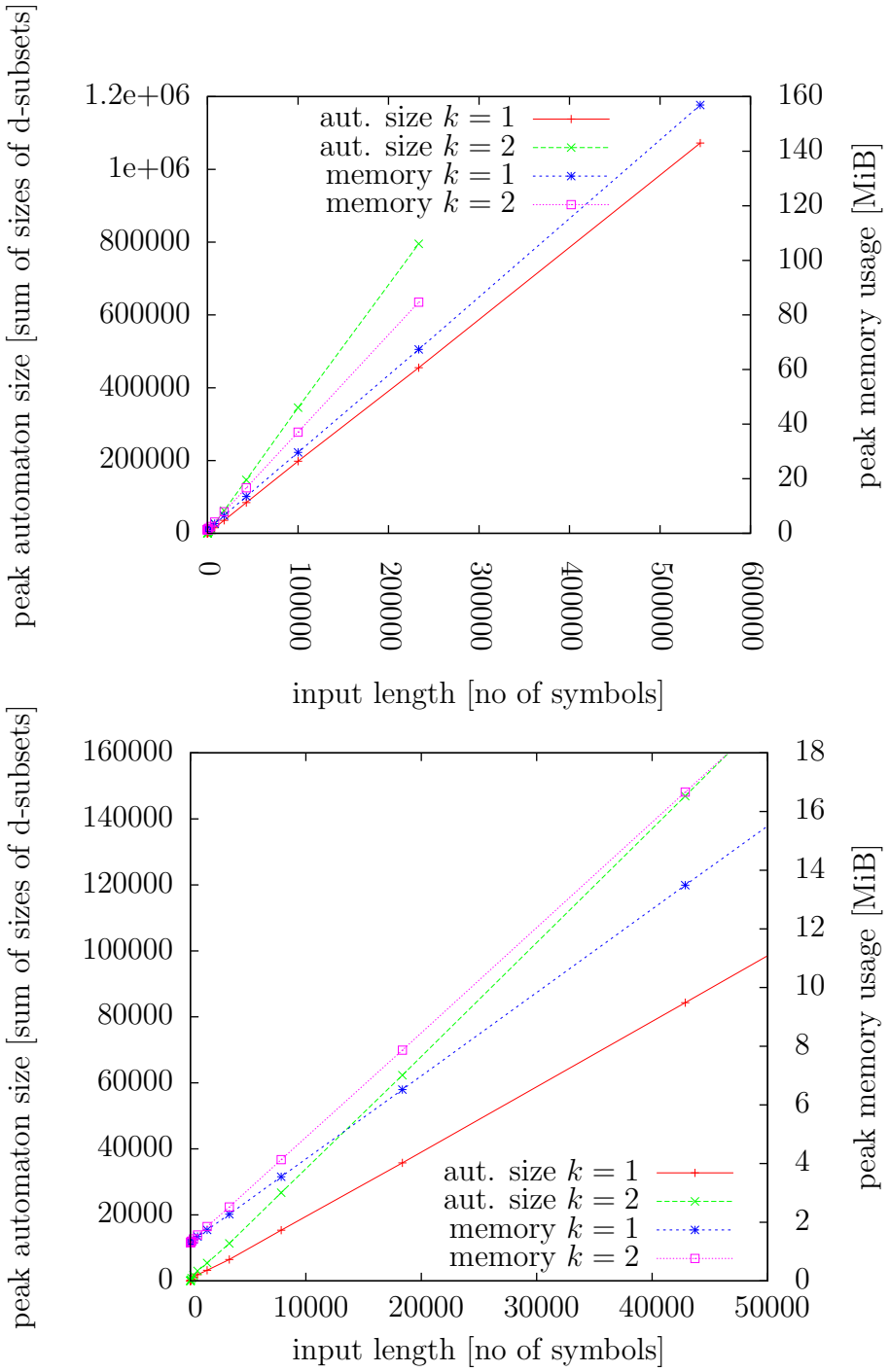

Figure 4.22: Memory needed for computation of all k-approximate covers for particular maximum Hamming distance compared to sum of sizes of all d-subsets stored in memory at a time

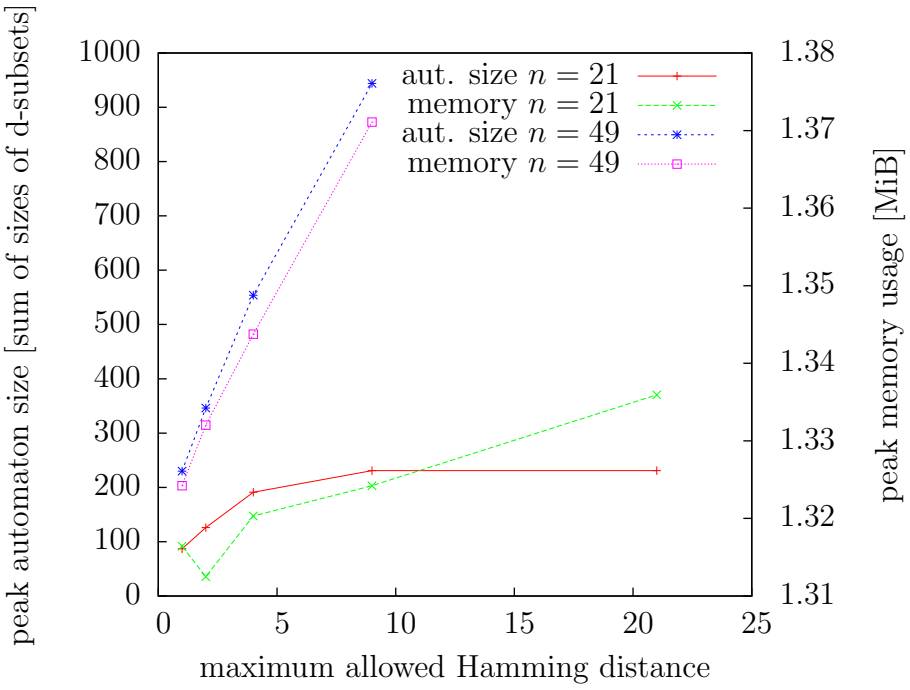

Figure 4.23: Memory needed for computation of all k-approximate covers for particular input length n compared to sum of sizes of all d-subsets stored in memory at a time

Chapter 5

Searching seeds of strings

Seed of a string is a generalization of the notion of cover. In this chapter, it is shown how to solve the problems of all restricted smallest distance k-approximate seeds (Definition 2.6.6), and all smallest distance k-approximate seeds (Definition 2.6.5) – given some string and maximum distance under given distance function.

5.1 Basic facts and properties

5.1.1 Approximate seeds

Some properties are common for exact and k-approximate seeds with Hamming distance. Hence the algorithm presented in [51] is used as a base of algorithm for the problem presented in this chapter, using some (but not all) techniques for searching covers in Chapter 4 for further improvements.

Observation 5.1.1. Assuming string $\boldsymbol{w}$, maximum distance k under some distance function D, and its k-approximate restricted seed $\boldsymbol{v}$, there exists some extension $\boldsymbol{z}$ of $\boldsymbol{w}$ such that $\boldsymbol{v}$ is a restricted k-approximate cover of $\boldsymbol{z}$.

Proof. As $\boldsymbol{v}$ is a restricted k-approximate seed of $\boldsymbol{w}$, it must be a k-approximate cover of some extension of $\boldsymbol{w}$ (Definition 2.4.27). Suppose $\boldsymbol{z}$ is that extension. Being a restricted k-approximate seed of $\boldsymbol{w}$, string $\boldsymbol{v}$ is a factor of $\boldsymbol{w}$ and thus $\boldsymbol{v}$ is a factor of $\boldsymbol{z}$. Therefore, $\boldsymbol{v}$ is a restricted k-approximate cover of $\boldsymbol{z}$. □

Observation 5.1.2. Assuming restricted k-approximate seed $\boldsymbol{v}$ of string $\boldsymbol{w}$ with maximum distance k under some distance function D, $\boldsymbol{v}$ is not necessarily a k-approximately repeating factor.

Example 5.1.3. Let us consider string $\boldsymbol{w}$ = `babca` and maximum Hamming distance $k = 1$. String $\boldsymbol{v}$ = `abc` is a restricted 1-approximate seed of $\boldsymbol{w}$ as it is a 1-approximate cover of extension $\boldsymbol{z}$ = `abbabcabc`. String $\boldsymbol{v}$ is not a 1-approximately repeating factor of $\boldsymbol{w}$.

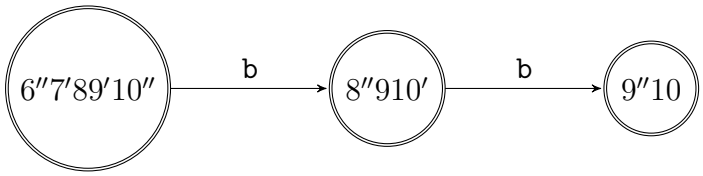

Figure 5.1: Part of deterministic 2-approximate suffix automaton used in Example 5.1.6

Observation 5.1.4. Note 3.1.14 holds also for k-approximate seed $\boldsymbol{u}$ of $\boldsymbol{w}$ with Hamming distance.

Observation 5.1.5. When searching for covers with Hamming distance, it is possible to remove all states q of deterministic suffix automaton that do not represent prefix, i.e., such q that $|\mathsf{lfactor}(q)| < \mathsf{depth}(r_1)$, where $\mathsf{d}(q) = \{r_1, \ldots, r_{|\mathsf{d}(q)|}\}$. Similar property between the first position and length of a factor may be used for searching seeds:

$$|\mathsf{lfactor}(q)| \leq \frac{1}{2} \cdot \mathsf{depth}(r_1)$$

Unlike in computing covers, this condition cannot be used for removing states q and their successors.

Example 5.1.6. Let us consider suffix automaton for string $\boldsymbol{w}$ = `bbbbbaaabb` and maximum Hamming distance $k = 2$. Part of the automaton used in this example is depicted at Figure 5.1. Factor `aaa` having 2-approximate end set $\{6, 7, 8, 9, 10\}$ cannot be a seed of $\boldsymbol{w}$ as its first 2-approximate position within $\boldsymbol{w}$ is 6. Factor `aaabb` having 2-approximate end set $\{9, 10\}$ is a 2-approximate seed of $\boldsymbol{w}$. It is obvious that for states q_1 and q_2 of the automaton, where $\mathsf{lfactor}(q_1)$ = `aaa` and $\mathsf{lfactor}(q_2)$ = `aaabb`, and $\mathsf{d}(q_1) = \{6'', 7', 8, 9', 10''\}$ and $\mathsf{d}(q_2) = \{9'', 10\}$, holds: q_2 is a successor of a state that is a successor of q_1. Therefore, q_1 must not be removed to be able to find 2-approximate seed `aaabb` of $\boldsymbol{w}$.

5.2 Finite-automata-based solution

5.2.1 Restricted approximate seeds

Every restricted k-approximate seed of string $\boldsymbol{w}$ is necessarily an exact factor of $\boldsymbol{w}$ (recall Definition 2.4.28) with possible k-approximate repetitions. Suffix automaton constructed for string $\boldsymbol{w}$ and maximum Hamming distance k has extended transitions defined for all k-approximate factors of $\boldsymbol{w}$. When deterministic suffix automaton $\mathcal{M}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w})$ is constructed using subset construction from nondeterministic one $\mathcal{M}^{\mathsf{H},k}_{\mathrm{S^N}}(\boldsymbol{w})$, each element of any d-subset of any state of $\mathcal{M}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w})$ contains information not only about position (depth within $\mathcal{M}^{\mathsf{H},k}_{\mathrm{S^N}}(\boldsymbol{w})$) but also about approximation (level within $\mathcal{M}^{\mathsf{H},k}_{\mathrm{S^N}}(\boldsymbol{w})$). Therefore, it may be easily determined whether string from left language of any state of $\mathcal{M}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w})$ is an exact factor of string $\boldsymbol{w}$.

```
1  2  3  4  5  6  7  8
b  b  b  b  b  a  a  a
b  b  b  a
   b  b  b  a
      b  b  b  a
         b  b  b  a
            b  b  b  a
```

Figure 5.2: Possible covering of string bbbbbaaa with string bbba and Hamming distance 2 from Example 5.2.3

Corollary 5.2.1 (of Lemma 3.1.34)**.** As only exact factor of $\boldsymbol{w}$ may be a restricted seed of $\boldsymbol{w}$, there is no need to construct any state of a k-approximate deterministic suffix automaton $\mathcal{M}_{\mathrm{SD}}^{\mathsf{H},k}(\boldsymbol{w})$ having only non-zero-level elements in its d-subset, as such state contains no exact factor of $\boldsymbol{w}$ in its left language. Therefore, when such state is created during construction, it may be removed and any of its successors need not be constructed. Such deterministic suffix automaton that contains only states having at least one zero-level element in its d-subset is denoted by $\tilde{\mathcal{M}}_{\mathrm{SD}}^{\mathsf{H},k}(\boldsymbol{w})$.

Note 5.2.2. As for computation of k-approximate covers, a trie-like suffix automaton is considered for computation of k-approximate seeds. Construction of the trie and left language extraction is simpler than for general suffix automaton. As left language of any state of the trie contains exactly one string, extraction of left language of any state takes linear time with respect to length of the string (e.g. using inverted transition function). See Figure 5.6 for transition diagram of such an automaton.

For computation of the smallest distance l_m of each seed (Algorithm 5.3), the idea used for searching covers (Chapter 4) may be used for searching seeds. Unlike searching covers, any position may be removed, including the first and the last one, thus the only lower bound of l_m is equal to 0. Determination whether to continue to decrement l is for seeds more complex than for covers, as computation of covering of central part of $\boldsymbol{w}$ is not sufficient condition for seeds. For factor $\boldsymbol{u}$ of $\boldsymbol{w}$, not only positions and their approximation need to be considered, but also distance of the uncovered prefix, suffix, of $\boldsymbol{w}$, and some suffix, prefix, of $\boldsymbol{u}$, respectively.

Example 5.2.3. Let us have string $\boldsymbol{w}$ = bbbbbaaa and maximum Hamming distance $k = 2$. One 2-approximate seed of $\boldsymbol{w}$ is bbba. It may be a 2-approximate seed of $\boldsymbol{w}$, because its positions in $\boldsymbol{w}$ are $4, 5, 6, 7, 8$ (see Figures 5.2, 5.6). When the position 8 is removed, bbba is still a seed of $\boldsymbol{w}$ with positions $4, 5, 6, 7$, all with approximation at most 1 (see Figure 5.3).

The cover-candidate automaton is needed not only to determine positions of each factor $\boldsymbol{u}$ of $\boldsymbol{w}$, but also for checking whether $\boldsymbol{u}$ is able to cover uncovered prefix and suffix of $\boldsymbol{w}$ (Algorithm 5.4). Thus, the automaton must be able to accept all the k-approximate

```
1  2  3  4  5  6  7  8
b  b  b  b  b  a  a  a
b  b  b  a
   b  b  b  a
      b  b  b  a
         b  b  b  a
                  b
```

Figure 5.3: Possible covering of an extension of bbbbbaaa with bbba and Hamming distance 1 from Example 5.2.3

prefixes of $\boldsymbol{w}$ of length at least $|\boldsymbol{u}| - 1$. Therefore, the depth-first search with removing states used in Chapter 4 for finding covers cannot be used. By contrast, only elements of d-subset $\mathsf{d}(q)$ may be removed after construction of all successors of q, transitions must be preserved. Thus, breadth-first search in the automaton is used (see Algorithm 5.1 and usage of queues L, L^{R}).

Like an exact seed, a k-approximate one must also cover the uncovered prefix and suffix of string $\boldsymbol{w}$ (i.e., some prefix of seed $\boldsymbol{u}$ must be a k-approximate suffix of $\boldsymbol{w}$ and some suffix of $\boldsymbol{u}$ must be a k-approximate prefix of $\boldsymbol{w}$). Similar technique as for exact seeds ([51]) is used (Algorithm 5.4), but with trie-like automata for string $\boldsymbol{w}$ $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w})$ and its reversion $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w}^{\mathrm{R}})$. When some suffix of seed $\boldsymbol{u}$ of $\boldsymbol{w}$ (i.e., some prefix of reversed $\boldsymbol{u}$) is accepted by $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w}^{\mathrm{R}})$, i.e., $\boldsymbol{u} = \mathsf{lfactor}(q)$ for some final state q of $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w}^{\mathrm{R}})$, $\boldsymbol{u}$ covers the uncovered prefix of $\boldsymbol{w}$ with approximation equal to level of the last element $r_{|\mathsf{d}(q)|}$ of $\mathsf{d}(q)$. Similarly for a prefix of $\boldsymbol{u}$, the uncovered suffix of $\boldsymbol{w}$ and $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w})$. The solution is stated in Algorithm 5.1.

Example 5.2.4. Let us compute set of all restricted k-approximate seeds with maximum Hamming distance $k = 2$ for string $\boldsymbol{w} =$ bbbbbaaa. Nondeterministic suffix automata $\mathcal{M}^{\mathsf{H},k}_{\mathrm{S^N}}(\boldsymbol{w})$ (see Figure 5.4) and $\mathcal{M}^{\mathsf{H},k}_{\mathrm{S^N}}(\boldsymbol{w}^{\mathrm{R}})$ (see Figure 5.5) are constructed. Next, subset construction of deterministic suffix automaton $\mathcal{M}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w})$ from $\mathcal{M}^{\mathsf{H},k}_{\mathrm{S^N}}(\boldsymbol{w})$ starts state-by-state (see transition diagram of $\mathcal{M}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w})$ with all states, that need to be constructed, at Figure 5.6), the same is done with automaton $\mathcal{M}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w}^{\mathrm{R}})$ from $\mathcal{M}^{\mathsf{H},k}_{\mathrm{S^N}}(\boldsymbol{w}^{\mathrm{R}})$. Some states may have only elements with non-zero level in its d-subset (for example $7''8'$). Such states are removed and their successors are not constructed as strings from their left languages (e.g. aaaa) are not factors of $\boldsymbol{w}$ (follows by Corollary 5.2.1).

All other states need to be checked whether their left languages contain some seeds. For example, state with d-subset $6''7'8$ contains string aaa in its left language. The string occurs approximately at positions $6, 7$ in $\boldsymbol{w}$ and exactly at position 8 in $\boldsymbol{w}$, thus it cannot be seed of $\boldsymbol{w}$, as its leftmost occurrence within $\boldsymbol{w}$ ends at position 6 and its length is 3 (i.e., the occurrence starts at position 4, so any proper suffix of aaa cannot cover the uncovered prefix (positions 1 to 3) of $\boldsymbol{w}$).

Other example is state with d-subset $4'5'67'8''$, which contains factor bbba of $\boldsymbol{w}$ in its

Algorithm 5.1 Compute set of restricted k-approximate seeds of $\boldsymbol{w}$ with the smallest Hamming distances.

Input: String $\boldsymbol{w}$, maximum Hamming distance k.
Output: Set $\mathsf{L}^{\mathrm{s}}_{\mathsf{H}^k}(\boldsymbol{w})$ of all seeds of $\boldsymbol{w}$.

1: $\mathsf{L}^{\mathrm{s}}_{\mathsf{H}^k}(\boldsymbol{w}) \leftarrow \emptyset$
2: construct $\mathcal{M}^{\mathsf{H},k}_{\mathrm{S^N}}(\boldsymbol{w}) = (Q_{\mathrm{N}}, A_{\boldsymbol{w}}, \delta_{\mathrm{N}}, q_{0\mathrm{N}}, F_{\mathrm{N}})$ (Algorithm 3.7)
3: construct $\mathcal{M}^{\mathsf{H},k}_{\mathrm{S^N}}(\boldsymbol{w}^{\mathrm{R}}) = (Q^{\mathrm{R}}_{\mathrm{N}}, A_{\boldsymbol{w}}, \delta^{\mathrm{R}}_{\mathrm{N}}, q^{\mathrm{R}}_{0\mathrm{N}}, F^{\mathrm{R}}_{\mathrm{N}})$ (Algorithm 3.7)
4: create new state $q_{0\mathrm{D}}$ for $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w}) = (Q_{\mathrm{D}}, A_{\boldsymbol{w}}, \delta_{\mathrm{D}}, q_{0\mathrm{D}}, F_{\mathrm{D}})$
5: create new state $q^{\mathrm{R}}_{0\mathrm{D}}$ for $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w}^{\mathrm{R}}) = (Q^{\mathrm{R}}_{\mathrm{D}}, A_{\boldsymbol{w}}, \delta^{\mathrm{R}}_{\mathrm{D}}, q^{\mathrm{R}}_{0\mathrm{D}}, F^{\mathrm{R}}_{\mathrm{D}})$
6: $\mathsf{lfactor}(q_{0\mathrm{D}}) \leftarrow \varepsilon$
7: $\mathsf{depth}(q_{0\mathrm{D}}) \leftarrow 0$, $\mathsf{depth}(q^{\mathrm{R}}_{0\mathrm{D}}) \leftarrow 0$
8: create L, L^{R} new empty queues of states
9: $\mathsf{enqueue}(L, q_{0\mathrm{D}})$, $\mathsf{enqueue}(L^{\mathrm{R}}, q^{\mathrm{R}}_{0\mathrm{D}})$
10: **while** L^{R} is not empty **do**
11: $\quad q^{\mathrm{R}}_t \leftarrow \mathsf{dequeue}(L^{\mathrm{R}})$
12: $\quad$ **if** $\mathsf{depth}(q_t) < |\boldsymbol{w}|$ **then**
13: $\quad\quad$ **for all** $a \in A_{\boldsymbol{w}}$ **do**
14: $\quad\quad\quad$ use Algorithm 5.2 to compute new state q^{R}_u, modify $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w}^{\mathrm{R}})$ and L^{R}
15: $\quad\quad$ **end for**
16: $\quad$ **end if**
17: $\quad$ discard all elements of $\mathsf{d}(q^{\mathrm{R}}_t)$ but the last one
18: **end while**
19: **while** L is not empty **do** ▷ construct $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w})$ and compute seeds in this loop
20: $\quad q_t \leftarrow \mathsf{dequeue}(L)$
21: $\quad$ **if** $\mathsf{depth}(q_t) < |\boldsymbol{w}|$ **then**
22: $\quad\quad$ **for all** $a \in A_{\boldsymbol{w}}$ **do**
23: $\quad\quad\quad$ use Algorithm 5.2 to compute new state q_u, modify $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w})$ and L
24: $\quad\quad\quad$ **if** exists $r \in \mathsf{d}(q_u)$ where $\mathsf{level}(r) = 0$ **then**
25: $\quad\quad\quad\quad \boldsymbol{y} \leftarrow \mathsf{lfactor}(q_t).a$
26: $\quad\quad\quad\quad \mathsf{lfactor}(q_u) \leftarrow y$
27: $\quad\quad\quad\quad$ **if** IsSeed$(\boldsymbol{y}, \mathsf{d}(q_u), k)$ (Algorithm 5.4) **then**
28: $\quad\quad\quad\quad\quad l_m \leftarrow$ SmallestDistanceSeed$(\mathsf{d}(q_u), \boldsymbol{y})$ (Algorithm 5.3)
29: $\quad\quad\quad\quad\quad$ **if** $|\boldsymbol{y}| > k$ or $l_m < |\boldsymbol{y}|$ **then**
30: $\quad\quad\quad\quad\quad\quad \mathsf{L}^{\mathrm{s}}_{\mathsf{H}^k}(\boldsymbol{w}) \leftarrow \mathsf{L}^{\mathrm{s}}_{\mathsf{H}^k}(\boldsymbol{w}) \cup \{(\boldsymbol{y}, l_m)\}$
31: $\quad\quad\quad\quad\quad$ **end if**
32: $\quad\quad\quad\quad$ **end if**
33: $\quad\quad\quad$ **end if**
34: $\quad\quad$ **end for**
35: $\quad$ **end if**
36: $\quad$ discard all elements of $\mathsf{d}(q_t)$ but the last one
37: **end while**

Algorithm 5.2 Compute state of a partially constructed k-approximate deterministic suffix automaton $\tilde{\mathcal{M}}^{\mathsf{H},\boldsymbol{w}}_{\mathrm{S^D}}(k) = (Q_\mathrm{D}, A_{\boldsymbol{w}}, \delta_\mathrm{D}, q_{0\mathrm{D}}, F_\mathrm{D})$.

Input: $\mathcal{M}^{\mathsf{H},k}_{\mathrm{S^N}}(\boldsymbol{w}) = (Q_\mathrm{N}, A_{\boldsymbol{w}}, \delta_\mathrm{N}, q_{0\mathrm{N}}, F_\mathrm{N})$, partially constructed $\tilde{\mathcal{M}}^{\mathsf{H},\boldsymbol{w}}_{\mathrm{S^D}}(k)$, state $q_t \in Q_\mathrm{D}$, symbol $a \in A_{\boldsymbol{w}}$, queue L of states.
Output: Modified partially constructed $\tilde{\mathcal{M}}^{\mathsf{H},\boldsymbol{w}}_{\mathrm{S^D}}(k)$ with possibly added successor q_u of state q_t for symbol a, modified queue L.

create new state q_u
$\mathsf{depth}(q_u) \leftarrow \mathsf{depth}(q_t) + 1$
$\delta_\mathrm{D}(q_t, a) \leftarrow q_u$
for all $r^i \in \mathsf{d}(q_t)$ (in order by $\mathsf{d}(q_t)$) **do**
    append all $r^j \in \delta_\mathrm{N}(r^i, a)$ to $\mathsf{d}(q_u)$
end for
if $|\mathsf{d}(q_u)| > 0$ **then**
    if exists $r \in \mathsf{d}(q_u)$ where $\mathsf{level}(r) = 0$ **then**
        $Q_\mathrm{D} \leftarrow Q_\mathrm{D} \cup \{q_u\}$
        $\mathsf{enqueue}(L, q_u)$
        if $r^u_{|\mathsf{d}(q_u)|} \in F_\mathrm{N}, \mathsf{d}(q_u) = \{r^u_1, \ldots r^u_{|\mathsf{d}(q_u)|}\}$ **then**
            $F_\mathrm{D} \leftarrow F_\mathrm{D} \cup \{q_u\}$
        end if
    end if
end if

Algorithm 5.3 The smallest Hamming distance of a k-approximate seed of $\boldsymbol{w}$.

Input: d-subset $\mathsf{d}(q) = \{r_1, r_2, \ldots, r_{|\mathsf{d}(q)|}\}$ of partially constructed deterministic k-approximate suffix automaton for string $\boldsymbol{w}$ provided that $\boldsymbol{u} = \mathsf{lfactor}(q)$ is a k-approximate seed of $\boldsymbol{w}$.
Output: The smallest distance l_m of $\boldsymbol{u}$.

procedure SmallestDistanceSeed($\mathsf{d}(q), \boldsymbol{u}$)
    $C \leftarrow \mathsf{d}(q)$ (consider C to be end set)
    $l_{\max} \leftarrow \max_{r \in C}\{\mathsf{level}(r)\}$
    $l \leftarrow l_{\max}$
    repeat
        for all $r \in C : \mathsf{level}(r) = l$ **do**
            remove r from C
        end for
        $l \leftarrow l - 1$
    until IsSeed($\boldsymbol{u}, C, l$) (Algorithm 5.4)
    $l_m \leftarrow l + 1$
    return l_m
end procedure

Algorithm 5.4 Determine whether string $\boldsymbol{u}$ is an l-approximate seed of $\boldsymbol{w}$ with Hamming distance.

Input: Partially constructed deterministic l-approximate suffix automata $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w}) = (Q, A_{\boldsymbol{w}}, \delta, q_0, F)$ and $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w}^{\mathrm{R}}) = (Q^{\mathrm{R}}, A_{\boldsymbol{w}}, \delta^{\mathrm{R}}, q_0^{\mathrm{R}}, F^{\mathrm{R}})$, l-approximate end set $C = \{e_1, e_2, \ldots, e_{|C|}\}$ of $\boldsymbol{u}$, maximum Hamming distance l.

Output: Resolution whether $\boldsymbol{u}$ is an l-approximate seed of $\boldsymbol{w}$ with respect to C.

1: **procedure** IsSeed($\boldsymbol{u}, C, l$)
2: **if** for all $i = 2, 3, \ldots, |C| : e_i - e_{i-1} \leq |\boldsymbol{u}|$
3: **and** $\exists \boldsymbol{p} \in \mathsf{pref}(\boldsymbol{u}), |\boldsymbol{p}| \geq |\boldsymbol{w}| - e_{|C|} : \delta^*(q_0, \boldsymbol{p}) = q_1 : r^{q_1}_{|\mathsf{d}(q_1)|} \in \mathsf{d}(q_1) \wedge q_1 \in F \wedge \mathsf{level}(r^{q_1}_{|\mathsf{d}(q_1)|}) \leq l$
4: **and** $\exists \boldsymbol{s} \in \mathsf{suff}(\boldsymbol{u}), |\boldsymbol{s}| \geq e_1 - |\boldsymbol{u}| : \delta^{\mathrm{R}*}(q_0^{\mathrm{R}}, \boldsymbol{s}) = q_2 : r^{q_2}_{|\mathsf{d}(q_2)|} \in \mathsf{d}(q_2) \wedge q_2 \in F^{\mathrm{R}} \wedge \mathsf{level}(r^{q_2}_{|\mathsf{d}(q_2)|}) \leq l$ **then**
5: **return** true
6: **else**
7: **return** false
8: **end if**
9: **end procedure**

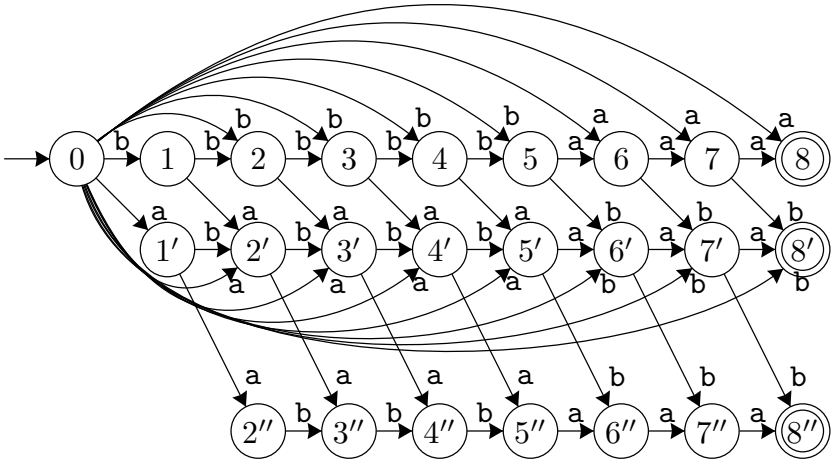

Figure 5.4: Transition diagram of the nondeterministic 2-approximate suffix automaton $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SN}}(\boldsymbol{w})$ for string $\boldsymbol{w} = \mathtt{bbbbbaaa}$ and Hamming distance from Example 5.2.4

left language. This factor is a 2-approximate cover of $\boldsymbol{w}$, and therefore it is seed of $\boldsymbol{w}$ (see Figure 5.2). When all positions with the maximum distance (i.e., 8) are not considered, bbba is still a seed of $\boldsymbol{w}$, as proper prefix b of bbba is a 1-approximate cover of uncovered suffix a of $\boldsymbol{w}$ (see Figure 5.3). See resulting table of all seeds and their distances in Table 5.1.

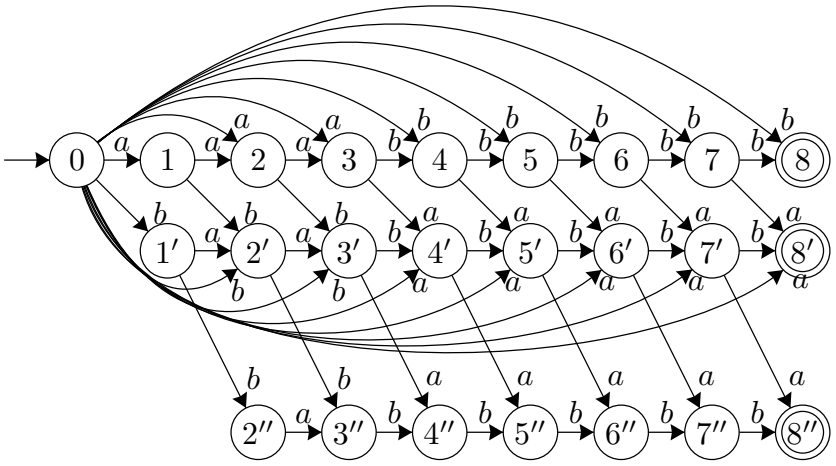

Figure 5.5: Transition diagram of the nondeterministic 2-approximate suffix automaton $\mathcal{M}^{\mathsf{H},k}_{\mathrm{S^N}}(\boldsymbol{w}^{\mathrm{R}})$ for reversion of string $\boldsymbol{w}$, i.e., $\boldsymbol{w}^{\mathrm{R}}$ = aaabbbbb, and Hamming distance from Example 5.2.4

Table 5.1: All 2-approximate seeds of string $\boldsymbol{w}$ = bbbbbaaa with Hamming distance and their smallest distances l_m; $\boldsymbol{p}$ is used prefix of a seed, $\boldsymbol{s}$ is used suffix (both computed by Algorithm 5.4); see Example 5.2.4

seed	d-subset	l_m	occurrences	$\boldsymbol{p}$	$\boldsymbol{s}$
ba	$2'3'4'5'67'8'$	1	2,3,4,5,6,7,8	ε	ε
baa	$3''4''5''6'78'$	2	3,4,5,6,7,8	ε	ε
bba	$3'4'5'67'8''$	1	3,4,5,6,7	b	ε
bbb	$3456'7''$	2	3,4,5,6,7	b	ε
baaa	$6''7'8$	2	6,7,8	ε	aa
bbaa	$4''5''6'78'$	2	4,5,6,7,8	ε	aa
bbba	$4'5'67'8''$	1	4,5,6,7	b	ε
bbbb	$456'7''$	2	4,5,6,7	b	ε
bbaaa	$6''7'8$	2	6,7,8	ε	a
bbbaa	$5''6'78'$	1	6,7,8	ε	a
bbbba	$5'67'8''$	1	5,6,7	b	ε
bbbbb	$56'7''$	2	5,6,7	b	ε
bbbaaa	$6''7'8$	1	7,8	ε	a
bbbbaa	$6'78'$	1	6,7,8	ε	ε
bbbbba	$67'8''$	1	6,7	b	ε
bbbbaaa	$7'8$	1	7,8	ε	ε
bbbbbaa	$78'$	1	7,8	ε	ε
bbbbbaaa	8	0	8	ε	ε

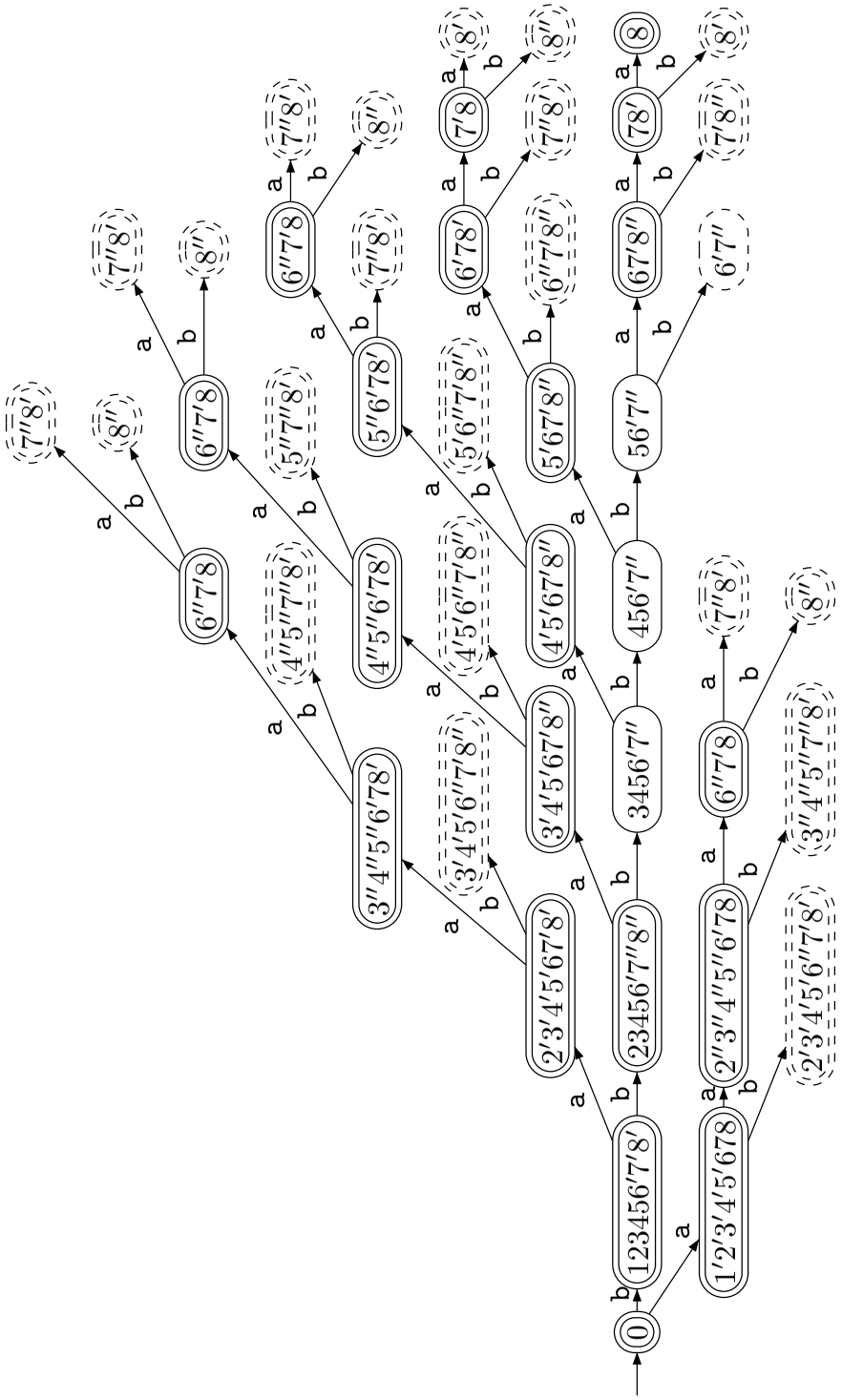

Figure 5.6: Transition diagram of the constructed part of the 2-approximate suffix automaton $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w})$ for string $\boldsymbol{w}$ = bbbbbaaa and Hamming distance from Example 5.2.4; dashed states are removed as their left language do not contain exact factor of $\boldsymbol{w}$ and thus they are not states of $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w})$

5.2.1.1 Correctness

Lemma 5.2.5. Automata $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w})$, $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w}^{\mathrm{R}})$ constructed by Algorithm 5.1 are deterministic k-approximate suffix automata for Hamming distance and string $\boldsymbol{w}$, $\boldsymbol{w}^{\mathrm{R}}$, respectively.

Proof. Let us prove the correctness just for $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w})$, as $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w}^{\mathrm{R}})$ is constructed the same way. Algorithm 5.1 constructs new state q_u for every state q_t and every alphabet symbol a using Algorithm 5.2. Therefore, every state q_u corresponds to some string $\mathsf{lfactor}(q_u)$ (line 25). Moreover, transition from state q_t to state q_u for symbol a is defined. As for each state and each symbol this is done just once, the resulting automaton is deterministic.

D-subsets are constructed by Algorithm 5.2 as follows: for every element of $\mathsf{d}(q_t)$, elements in $\mathsf{d}(q_u)$ are appended based on transition function of nondeterministic suffix automaton $\mathcal{M}^{\mathsf{H},k}_{\mathrm{S^N}}(\boldsymbol{w}) = (Q_{\mathrm{N}}, A_{\boldsymbol{w}}, \delta_{\mathrm{N}}, q_{0\mathrm{N}}, F_{\mathrm{N}})$. Therefore, elements of $\mathsf{d}(q_u)$ correspond to k-approximate occurrences of $\mathsf{lfactor}(q_u)$. Let us use induction.

- If just initial state is dequeued, lines 4–6 of Algorithm 5.2 define d-subset of newly created state q_u as direct successors of $q_{0\mathrm{N}}$ for symbol a. Therefore, q_u corresponds to k-approximate factor a of $\boldsymbol{w}$ (there exists transition from $q_{0\mathrm{N}}$ for it) and $\mathsf{d}(q_u)$ is a k-approximate end set of $\mathsf{lfactor}(q_u)$ in $\boldsymbol{w}$ and q_u is a correct state of deterministic k-approximate suffix automaton.
- If correct state q_t of $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w})$ is dequeued, lines 4–6 of Algorithm 5.2 define each element of d-subset of newly created state q_u as a direct successor of each element (state) of $\mathcal{M}^{\mathsf{H},k}_{\mathrm{S^N}}(\boldsymbol{w})$. As each element of $\mathsf{d}(q_t)$ represents k-approximate position of $\mathsf{lfactor}(q_t)$ in $\boldsymbol{w}$, elements of $\mathsf{d}(q_u)$ represent k-approximate positions of $\mathsf{lfactor}(q_t).a = \mathsf{lfactor}(q_u)$ (based on $\delta_{\mathrm{N}}(q_t, a)$).

Due to line 7, states with $\mathsf{lfactor}(q_u)$ having no k-approximate occurrence are ignored (as they do not correspond to a k-approximate factor of $\boldsymbol{w}$). Every state is enqueued and therefore later processed (its successors are constructed). When an element of $\mathsf{d}(q_u)$ is final state of corresponding nondeterministic k-approximate suffix automaton, q_u becomes final. Just the last element is checked, because following the definition of d-subset, it corresponds to state with maximum depth in nondeterministic k-approximate suffix automaton (state with less depth cannot represent k-approximate suffix – it is a property of that automaton).

If $\mathsf{d}(q_u)$ contains no zero-level element, $\mathsf{lfactor}(q_u)$ has no (exact) occurrence in $\boldsymbol{w}$ and therefore cannot represent restricted k-approximate seed. □

Lemma 5.2.6. Algorithm 5.4 correctly determines whether for given maximum Hamming distance l given string $\boldsymbol{u}$ is an l-approximate seed of given string $\boldsymbol{w}$ considering just occurrences of $\boldsymbol{u}$ in $\boldsymbol{w}$ given by end set C.

Proof. In order to $\boldsymbol{u}$ be an l-approximate seed of $\boldsymbol{w}$, $\boldsymbol{u}$ must be an l-approximate cover of some extension of $\boldsymbol{w}$. Assuming d-subset C is equal to l-approximate end set of $\boldsymbol{u}$ in $\boldsymbol{w}$, in line 2 there is a check whether $\boldsymbol{u}$ is an l-approximate cover of $\boldsymbol{w}[r_1 - |\boldsymbol{u}| + 1..r_C]$ (by Lemma 4.1.8).

In conjunction with the previous check, in line 3, there is a check whether $\boldsymbol{u}$ is an l-approximate cover of a right extension of $\boldsymbol{w}[r_1 - |\boldsymbol{u}| + 1..|\boldsymbol{w}|]$: it is true when $\boldsymbol{p}$ is a prefix of $\boldsymbol{u}$ such that:

- $\boldsymbol{p}$ is at least as long as the (uncovered) suffix of $\boldsymbol{w}$ starting right after the rightmost l-approximate position of $\boldsymbol{u}$ in $\boldsymbol{w}$, and
- $\boldsymbol{p}$ is l-approximate suffix of $\boldsymbol{w}$:
  - $\boldsymbol{p}$ is checked to be accepted by k-approximate suffix automaton for $\boldsymbol{w}$ when $k \geq l$ (necessary condition),
  - for the last element $r^{q_1}_{|\mathsf{d}(q_1)|}$ of d-subset of q_1 where $\mathsf{lfactor}(q_1) = \boldsymbol{p}$, it is checked that the following holds: $\mathsf{level}(r^{q_1}_{|\mathsf{d}(q_1)|}) \leq l$, therefore $\boldsymbol{p}$ has l-approximate position $r^{q_1}_{|\mathsf{d}(q_1)|}$ in $\boldsymbol{w}$; $r^{q_1}_{|\mathsf{d}(q_1)|}$ is equal to $|\boldsymbol{w}|$, because in order q_1 to be final, it must contain such $r^{q_1}_{|\mathsf{d}(q_1)|}$ as the last element of its d-subset (it is a property of nondeterministic k-approximate suffix automaton and consequence of line 12 of Algorithm 5.2). Therefore, $\boldsymbol{p}$ is l-approximate suffix of $\boldsymbol{w}$.

In condition with the previous checks, in the next line, there is a check whether $\boldsymbol{u}$ is an l-approximate cover of an extension of $\boldsymbol{w}$. □

Lemma 5.2.7. Algorithm 5.3 correctly computes the smallest Hamming distance of given k-approximate seed $\boldsymbol{u}$, given string $\boldsymbol{w}$ and given k-approximate end set of $\boldsymbol{u}$ in $\boldsymbol{w}$.

Proof. The Algorithm starts with level l equal to maximum level such that there is some l-approximate occurrence of $\boldsymbol{u}$ in $\boldsymbol{w}$. During each iteration, elements having level equal to l of end set C are removed, i.e., l-approximate occurrences of $\boldsymbol{u}$ in $\boldsymbol{w}$ are no longer considered. In the end of each iteration, l is decremented by 1. The iteration is being done until $\boldsymbol{u}$ is a (approximate) seed of $\boldsymbol{w}$ considering just occurrences in C. This check is correct due to Lemma 5.2.6. When this check returns false, the positions removed during the last iteration are needed, therefore 1 is added to previously decremented l as a result. □

Lemma 5.2.8. Algorithm 5.1 correctly computes set of all restricted k-approximate seeds of given string $\boldsymbol{w}$ with given maximum Hamming distance k.

Proof. By proof of Lemma 5.2.5, automata $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w})$, $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w}^{\mathrm{R}})$ are correct deterministic k-approximate suffix automata for $\boldsymbol{w}$, $\boldsymbol{w}^{\mathrm{R}}$, respectively. As every state of $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w})$ is processed, every k-approximate factor $\boldsymbol{y}$ of $\boldsymbol{w}$ is examined (line 25). If d-subset of a state does not contain a zero-level element (line 24), it is ignored and its left language not processed, as it cannot represent restricted k-approximate seed. Then the check whether $\boldsymbol{y}$ is a k-approximate seed is done and its smallest distance l_m is computed. Then, $\boldsymbol{y}$ is considered to be a k-approximate seed if its smallest distance is less than its length and if it is longer than given maximum distance k. If this is not true, we would say that $\boldsymbol{y}$ has a k-approximate occurrence in every position in $\boldsymbol{w}$. □

5.2.1.2 Time and space complexities

Note 5.2.9. As parts of automata $\mathcal{M}_{\mathrm{S^D}}^{\mathsf{H},k}(\boldsymbol{w})$ and $\mathcal{M}_{\mathrm{S^D}}^{\mathsf{H},k}(\boldsymbol{w}^{\mathrm{R}})$ are constructed the same way in Algorithm 5.1, the time and space complexities of their construction are the same.

Lemma 5.2.10. Left languages of states of deterministic suffix automaton $\tilde{\mathcal{M}}_{\mathrm{S^D}}^{\mathsf{H},k}(\boldsymbol{w}) = (Q_{\mathrm{D}}, A_{\boldsymbol{w}}, \delta_{\mathrm{D}}, q_{0\mathrm{D}}, F_{\mathrm{D}})$ are distinct, i.e.,

$$q_1, q_2 \in Q_{\mathrm{D}}; q_1 \neq q_2 \Rightarrow \mathsf{lfactor}(q_1) \neq \mathsf{lfactor}(q_2)$$

By contradiction. Let us have the following consideration: if there exist two states $q_1, q_2 \in Q_{\mathrm{D}}, q_1 \neq q_2$ and $\mathsf{lfactor}(q_1) = \mathsf{lfactor}(q_2) = \boldsymbol{u}$, it means existence of two distinct sequences of transitions: $\delta_{\mathrm{D}}^*(q_{0\mathrm{D}}, \boldsymbol{u}) = q_1$ and $\delta_{\mathrm{D}}^*(q_{0\mathrm{D}}, \boldsymbol{u}) = q_2$. As Algorithm 5.1 creates new state for every $a \in A_{\boldsymbol{w}}$ (line 23) and for every such state, the string of its left language is defined as unique (line 25), the resulting automaton $\tilde{\mathcal{M}}_{\mathrm{S^D}}^{\mathsf{H},k}(\boldsymbol{w})$ is deterministic, so such distinct sequences of transitions for the same string are not possible, thus either $q_1 = q_2$ or $\mathsf{lfactor}(q_1) \neq \mathsf{lfactor}(q_2)$. □

Lemma 5.2.11. Number of states of deterministic suffix automaton $\tilde{\mathcal{M}}_{\mathrm{S^D}}^{\mathsf{H},k}(\boldsymbol{w})$ for string $\boldsymbol{w}$ and maximum Hamming distance k for searching all restricted k-approximate seeds is

$$\frac{1}{2} \cdot \left(|\boldsymbol{w}|^2 + |\boldsymbol{w}|\right) - R_{\mathsf{H}}^k(\boldsymbol{w}) + 1 \tag{5.1}$$

Proof. Idea is stated in proof of Lemma 4.2.37. □

Note 5.2.12. As restricted k-approximate seeds of string $\boldsymbol{w}$ are exact factors of $\boldsymbol{w}$, it is meaningful to consider effective alphabet $A_{\boldsymbol{w}}$ only and $|A_{\boldsymbol{w}}| \leq |\boldsymbol{w}|$ always holds (recall that effective alphabet $A_{\boldsymbol{w}}$ consists only of symbols that occur in $\boldsymbol{w}$). It is also meaningless to consider high k, because every factor of $\boldsymbol{w}$ having length less or equal to k is always k-approximate seed of $\boldsymbol{w}$. Thus $k \leq |\boldsymbol{w}|$ always holds. Usually k and $|A_{\boldsymbol{w}}|$ are independent of $|\boldsymbol{w}|$.

Lemma 5.2.13. Number of states being created of deterministic k-approximate suffix automaton $\mathcal{M}_{\mathrm{S^D}}^{\mathsf{H},k}(\boldsymbol{w})$ for string $\boldsymbol{w}$ and maximum Hamming distance k using Algorithm 5.1 for searching all restricted k-approximate seeds is at most

$$|A_{\boldsymbol{w}}| \cdot \left(\frac{1}{2} \cdot \left(|\boldsymbol{w}|^2 + |\boldsymbol{w}|\right) - R_{\mathsf{H}}^k(\boldsymbol{w})\right) + 1$$

Proof. By Lemma 5.2.11, number of states of deterministic suffix automaton $\tilde{\mathcal{M}}_{\mathrm{S^D}}^{\mathsf{H},k}(\boldsymbol{w}) = (\tilde{Q}_{\mathrm{D}}, A_{\boldsymbol{w}}, \tilde{\delta}_{\mathrm{D}}, \tilde{q}_0^D, \tilde{F}_{\mathrm{D}})$ is equal to (5.1). But using Algorithm 5.1 there are also constructed (but not stored) more states that have strings not being factors of string $\boldsymbol{w}$ in their left languages. For every state $q_t \in \tilde{Q}_{\mathrm{D}}$ there is constructed a successor q_u for each symbol $a \in A_{\boldsymbol{w}}$ (line 23), but not every state q_u is in $\tilde{Q}_{\mathrm{D}}$ (line 24). Number of such successors varies from 0 to $|A_{\boldsymbol{w}}|$ for each state of $\tilde{\mathcal{M}}_{\mathrm{S^D}}^{\mathsf{H},k}(\boldsymbol{w})$ (but the initial one, which has successors in $\tilde{Q}_{\mathrm{D}}$ for all $a \in A_{\boldsymbol{w}}$), thus there could be at most $|A_{\boldsymbol{w}}| \cdot \left(\frac{1}{2} \cdot (|\boldsymbol{w}|^2 + |\boldsymbol{w}|) - R_{\mathsf{H}}^k(\boldsymbol{w})\right) + 1$ states of automaton $\mathcal{M}_{\mathrm{S^D}}^{\mathsf{H},k}(\boldsymbol{w})$ constructed. □

Lemma 5.2.14. For every d-subset of deterministic suffix automaton $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w})$ constructed for string $\boldsymbol{w}$ and maximum Hamming distance k by Algorithm 5.1 holds that there are no two elements having the same depth.

Proof. Idea stated in proof of Lemma 4.2.41. □

Lemma 5.2.15. Number of elements of all d-subsets of deterministic k-approximate suffix automaton $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w})$ constructed for string $\boldsymbol{w}$ and maximum Hamming distance k is not greater than

$$\frac{1}{2} \cdot \left(|\boldsymbol{w}|^3 + |\boldsymbol{w}|^2\right) - |\boldsymbol{w}| \cdot R^k_{\mathsf{H}}(\boldsymbol{w}) + 1$$

Proof. Idea stated in proof of Lemma 4.2.42. □

Lemma 5.2.16. Number of elements of all d-subsets of deterministic suffix automaton $\mathcal{M}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w})$ constructed for string $\boldsymbol{w}$ over effective alphabet $A_{\boldsymbol{w}}$ and maximum Hamming distance k using Algorithm 5.1 is not greater than

$$|A_{\boldsymbol{w}}| \cdot \left(\frac{1}{2} \cdot \left(|\boldsymbol{w}|^3 + |\boldsymbol{w}|^2\right) - |\boldsymbol{w}| \cdot R^k_{\mathsf{H}}(\boldsymbol{w})\right) + 1$$

Proof. Clearly holds by Lemma 5.2.13 and 5.2.15. □

Lemma 5.2.17. Time complexity of the check whether d-subset $\mathsf{d}(q)$ of a state q of deterministic suffix automaton $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w})$ represents a seed $\boldsymbol{u} = \mathsf{lfactor}(q)$ of string $\boldsymbol{w}$ (Algorithm 5.4) is at most

$$|\mathsf{d}(q)| + 2 \cdot |\boldsymbol{u}| - 3$$

that is

$$\mathcal{O}(|\boldsymbol{w}|)$$

Proof. The check whether seed $\boldsymbol{u}$ covers central part of string $\boldsymbol{w}$ (comparison of each two subsequent elements' depth, line 2) takes $|t|-1$ time, where $|t| \leq |\mathsf{d}(q)|$, and $|\mathsf{d}(q)| = \mathcal{O}(|\boldsymbol{w}|)$ by Lemma 5.2.14, therefore it takes $\mathcal{O}(|\boldsymbol{w}|)$ time.

The check of existence of a prefix of $\boldsymbol{u}$ to cover uncovered suffix of $\boldsymbol{w}$ (line 3) takes at most $|\boldsymbol{u}| - 1$ time, that is $\mathcal{O}(|\boldsymbol{w}|)$, as it is found during processing $\boldsymbol{u}$ as input for already constructed part of automaton $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{S^D}}(\boldsymbol{w})$: using transition function δ symbol by symbol takes at most $|\boldsymbol{u}|-1$ time, selection of the last element of d-subset of state q_1 takes constant time, checking whether it is final and its level needs constant time. Therefore, line 3 means at most $|\boldsymbol{u}| - 1$ steps for each string $\boldsymbol{u}$.

The check of existence of a suffix of $\boldsymbol{u}$ to cover uncovered prefix of $\boldsymbol{w}$ takes also at most $|\boldsymbol{u}| - 1$ time (similarly to the previous case). □

Lemma 5.2.18. Having string $\boldsymbol{w}$, its seed $\boldsymbol{u}$ and given maximum Hamming distance k, time complexity of the computation of the smallest distance of seed $\boldsymbol{u} = \mathsf{lfactor}(q)$ of $\boldsymbol{w}$ (Algorithm 5.3) is at most

$$7 - 2 \cdot |\boldsymbol{u}| + k \cdot (2 \cdot |\mathsf{d}(q)| + 2 \cdot |\boldsymbol{u}| - 6)$$

that is

$$\mathcal{O}\left(k \cdot |\boldsymbol{w}|\right)$$

Proof. Retrieving the maximum level $l_{\max}$ of d-subset $\mathsf{d}(q)$ takes $|\mathsf{d}(q)|$ time. Then there are at most $l \leq k$ iterations in Algorithm 5.3. Each iteration means removal of some elements of a d-subset (needs $|\mathsf{d}(q)|$ time for the first iteration – worst case, during that at least one element is removed) and check whether $\boldsymbol{u}$ is still seed of string $\boldsymbol{w}$ (takes $|\mathsf{d}(q)| + 2 \cdot |\boldsymbol{u}| - 4$ for the first iteration – Lemma 5.2.17). □

Note 5.2.19. Number of all restricted k-approximate seeds is $\mathcal{O}(|\boldsymbol{w}|^2)$ (like number of factors). Thus, the sum of their lengths is $\mathcal{O}(|\boldsymbol{w}|^3)$, denoted by $\|\mathsf{L}^{\mathrm{s}}_{\mathsf{H}^k}(\boldsymbol{w})\|$.

Theorem 5.2.20. Time complexity of computation of all restricted k-approximate seeds with their smallest distance for string $\boldsymbol{w}$ over effective alphabet $A_{\boldsymbol{w}}$ with maximum Hamming distance k (Algorithm 5.1) is

$$\mathcal{O}(k \cdot |A_{\boldsymbol{w}}| \cdot |\boldsymbol{w}|^3)$$

Proof. Time needed for construction of nondeterministic k-approximate suffix automaton $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SN}}(\boldsymbol{w}) = (Q_{\mathrm{N}}, A_{\boldsymbol{w}}, \delta_{\mathrm{N}}, q_{0\mathrm{N}}, F_{\mathrm{N}})$ for string $\boldsymbol{w}$ and maximum Hamming distance k is $\mathcal{O}(k \cdot |A_{\boldsymbol{w}}| \cdot |\boldsymbol{w}|)$ time (Lemma 3.1.35, Algorithm 3.7). For each state of deterministic suffix automaton $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w})$ (their number is stated in Lemma 5.2.11) and for each symbol of $A_{\boldsymbol{w}}$, new d-subset is constructed (lines 4–6 of Algorithm 5.2). As each element of any d-subset may be constructed in constant time (just using already known transition function δ_{N}) and the elements are naturally ordered (no need to sort – for proof see Lemma 4.2.6), construction of all d-subsets is done in at most $|A_{\boldsymbol{w}}| \cdot (\frac{1}{2} \cdot (|\boldsymbol{w}|^3 + |\boldsymbol{w}|^2) - |\boldsymbol{w}| \cdot R^k_{\mathsf{H}}(\boldsymbol{w})) + 1$ time (given by number of all elements – Lemma 5.2.15). Each d-subset is checked whether it contains element with zero level in linear time with size of the d-subset (and thus in $\mathcal{O}(|\boldsymbol{w}|)$ time for each state). The left language extraction of state is done in linear time for each state (line 25 of Algorithm 5.1). Algorithm 5.4 runs in linear time for each state (Lemma 5.2.17), Algorithm 5.3 needs $\mathcal{O}(k|\boldsymbol{w}|)$ time for each state (Lemma 5.2.18). □

Lemma 5.2.21. During construction of deterministic suffix automaton $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w})$ for string $\boldsymbol{w}$ (Algorithm 5.1), the number of states stored in queue L (or L^{R}, respectively) is at most equal to the number of states having depths i and $i-1$ for $1 \leq i \leq |\boldsymbol{w}|$.

Proof. For each state q_t of $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w})$ and possibly for each alphabet symbol, new state q_u and its d-subset are stored in a memory and put into queue L. Because L is a queue, state with depth i is processed after all states having depth $i-1$ have been processed (line 20). After storing all the successors of q_t in L, d-subset of q_t is discarded and only one element is preserved (line 36). □

Lemma 5.2.22. During construction of deterministic suffix automaton $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w})$ for string $\boldsymbol{w}$ (Algorithm 5.1), there are at most $\mathcal{O}(|\boldsymbol{w}|^2)$ elements of d-subsets stored in memory at a time.

Proof. Due to Lemma 5.2.21 and because all new constructed states q_u for some q_t correspond to factors of equal length i (line 25) and just d-subsets of states corresponding to factors of lengths i and $i-1$ are stored in memory at a time. Number of factors of string $\boldsymbol{w}$ of equal length i is at most $\min(|\boldsymbol{w}|-i+1, |A_{\boldsymbol{w}}|^i)$. Number of k-approximate positions of such factor is also at most $|\boldsymbol{w}|-i+1$. In such case, there are $\mathcal{O}(|\boldsymbol{w}|^2)$ elements in queue L, so number of elements in L stored at a time is $\mathcal{O}(|\boldsymbol{w}|^2) + \mathcal{O}(|\boldsymbol{w}|^2) = \mathcal{O}(|\boldsymbol{w}|^2)$. □

Theorem 5.2.23. Space complexity of computation of all restricted k-approximate seeds of string $\boldsymbol{w}$ with maximum Hamming distance k is

$$\mathcal{O}\left(|\boldsymbol{w}|^2\right)$$

Proof. Space complexity of construction of nondeterministic k-approximate suffix automaton $\mathcal{M}_{\mathrm{S^N}}^{\mathsf{H},k}(\boldsymbol{w})$ is $\mathcal{O}(k \cdot |A| \cdot |\boldsymbol{w}|)$ (Lemma 3.1.35). By Lemma 5.2.22, number of elements of d-subsets stored in memory at a time is $\mathcal{O}(|\boldsymbol{w}|^2)$, as no more elements of d-subsets than those in queue L plus $\mathcal{O}(|\boldsymbol{w}|)$ new are in memory at a time. By Lemma 5.2.11, number of states of deterministic suffix automaton $\tilde{\mathcal{M}}_{\mathrm{S^D}}^{\mathsf{H},k}(\boldsymbol{w})$ is $\mathcal{O}(|\boldsymbol{w}|^2)$ (they all are stored in memory with one element each). As the automaton $\tilde{\mathcal{M}}_{\mathrm{S^D}}^{\mathsf{H},k}(\boldsymbol{w})$ is constructed as trie, number of transitions is also $\mathcal{O}(|\boldsymbol{w}|^2)$. □

5.2.2 Approximate seeds

Every k-approximate seed of string $\boldsymbol{w}$ is not an exact factor of $\boldsymbol{w}$ in general. As for computation of all restricted k-approximate seeds, a trie-like deterministic k-approximate suffix automaton is used for indexing $\boldsymbol{w}$, however, states having no zero-level element in their d-subsets and their successors must be constructed for searching all k-approximate seeds.

As the problem of searching all k-approximate seeds is a generalization of searching all restricted k-approximate seeds, some results and algorithms stated for the latter problem hold here as well, others need to be modified.

Lemma 5.2.24. Algorithm 5.3 is correct without any modification for computation of the smallest distance of any k-approximate seed of a string with Hamming distance.

Proof. There is no assumption about seed $\boldsymbol{u}$ of $\boldsymbol{w}$ being exact factor of $\boldsymbol{w}$. □

Lemma 5.2.25. Algorithm 5.4 is correct without any modification for decision whether a string is an l-approximate seed of a string with Hamming distance for arbitrary l.

Proof. Assuming t represents a partial d-subset of state q where $\boldsymbol{u} = \mathsf{lfactor}(q)$, there is no assumption that $\boldsymbol{u}$ is an exact factor of $\boldsymbol{w}$ (this is not assumption of Algorithm 5.4 that does not depend on it). Line 2 does not depend on $\boldsymbol{u}$ being an exact factor of $\boldsymbol{w}$. Line 3 checks existence of a prefix of $\boldsymbol{u}$, its length related to a position and whether it belongs to left language of a state whose d-subset contains element of l-limited level. There is no assumption about $\boldsymbol{u}$ being an exact factor of $\boldsymbol{w}$. Similarly for the next line. Therefore, Algorithm 5.4 does not depend on whether $\boldsymbol{u}$ is a restricted l-approximate seed of $\boldsymbol{w}$. □

Proposition 5.2.26. Assume the following modification of Algorithm 5.2: remove lines 8 and 14. Then after removing the assumption of working with effective alphabet $A_{\boldsymbol{w}}$ and constructing state of $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w})$, using the algorithm it is possible to construct state of a k-approximate deterministic suffix automaton that is capable of indexing k-approximate seeds.

Proof. Originally, in Algorithm 5.2 is assumed construction of state of a deterministic k-approximate suffix automaton $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w})$ that contains only states having at least one zero-level element in their d-subsets. Line 8 checks whether there is such an element in d-subset of newly created state.

After removal of lines 8 and 14, there is no assumption about left language of newly created state q_u in terms of being an exact factor. The state is created as a successor of given state q_t for given symbol a without any other assumption, lines 4–6 define its d-subset based on transition function of nondeterministic k-approximate suffix automaton without any limit regarding number of exact positions. □

Proposition 5.2.27. Assume the following modification of Algorithm 5.1: remove lines 24 and 33. Then remove the assumption of working with effective alphabet $A_{\boldsymbol{w}}$ and with automata $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w})$ and $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w}^{\mathrm{R}})$, i.e., with automata having only states with at least one zero-level element in their d-subsets. Instead, work with $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w})$ and $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w}^{\mathrm{R}})$. Then after application of Proposition 5.2.26, the algorithm computes correctly the set of all k-approximate seeds of a string with Hamming distance.

Proof. After application of Proposition 5.2.26, states created using Algorithm 5.2 are no longer limited to those having only states with at least one zero-level element in their d-subsets. The previous lemmas state that Algorithms 5.4 and 5.3 are not limited to restricted k-approximate seeds. Assuming not working with $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w})$ and $\tilde{\mathcal{M}}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w}^{\mathrm{R}})$, after removal of lines 24 and 33 where the check whether newly created state contains a zero-level element in its d-subset, the Algorithm still computes correctly all the restricted k-approximate seeds of given string with Hamming distance, but does not limit them to exact factor of the string. □

5.2.2.1 Time and space complexities

Lemma 5.2.28. Lemma 5.2.10 holds for algorithm stated by Proposition 5.2.27.

Proof. There is no assumption about exact positions or factor in proof of Lemma 5.2.10. □

Lemma 5.2.29. In the deterministic suffix automata $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w})$ and $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w}^{\mathrm{R}})$ constructed using algorithm stated by Proposition 5.2.27 for string $\boldsymbol{w}$, alphabet A and maximum Hamming distance k, the number of states of each automaton is no more than

$$|\boldsymbol{w}| \cdot k \cdot \binom{|\boldsymbol{w}|}{k} \cdot \frac{(|A|-1)^k - 1}{|A|-1} \tag{5.2}$$

if $k \leq \frac{|\boldsymbol{w}|}{2}$

Proof. By Lemma 5.2.28, there exist no two states of $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w}) = (Q_{\mathrm{D}}, A_{\boldsymbol{w}}, \delta_{\mathrm{D}}, q_{0\mathrm{D}}, F_{\mathrm{D}})$ having the same k-approximate factor of $\boldsymbol{w}$ in their left languages (as $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w})$ is a trie-like automaton, each factor corresponds to exactly one state and vice versa). As d-subset of each state successor is based on d-subset elements and transition function of nondeterministic k-approximate suffix automaton (lines 4–6 of Algorithm 5.2) and states with empty d-subset are not considered (line 7), each state corresponds just to string being a k-approximate factor of $\boldsymbol{w}$, therefore there cannot be more states of $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w})$ than number of all k-approximate factors of $\boldsymbol{w}$ with Hamming distace.

The number of k-approximate factors of $\boldsymbol{w}$ of length n is by [24, proof of Proposition 2] equal to $\sum_{i=0}^{k} \binom{n}{i}(|A|-1)^i$. It is true that

$$\sum_{i=0}^{k} \binom{n}{i} \cdot (|A|-1)^i < k \cdot \binom{n}{k} \sum_{i=0}^{k} (|A|-1)^i = k \cdot \binom{n}{k} \cdot \frac{(|A|-1)^k - 1}{|A|-1} \tag{5.3}$$

assuming $k \leq \frac{n}{2}$. The number of all k-approximate factors of $\boldsymbol{w}$ is

$$\sum_{j=0}^{|\boldsymbol{w}|} \sum_{i=0}^{k} \binom{j}{i} \cdot (|A|-1)^i$$

To find the higher bound, the previous formula needs to be simplified. Using $\binom{|\boldsymbol{w}|}{k}$ constantly for every factor, the number of k-approximate factors of $\boldsymbol{w}$ cannot be greater than what stated in the Lemma.

□

Lemma 5.2.30. The total number of states constructed by algorithm stated by Proposition 5.2.27 for alphabet A for automata $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w})$ and $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w}^{\mathrm{R}})$ is no more than $|A|$-times more than number of states of automata $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w})$ and $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w}^{\mathrm{R}})$.

Proof. For every symbol $a \in A$ and every state q of each of $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w})$ and $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w}^{\mathrm{R}})$, new successor state q' is created (line 23 of Algorithm 5.1). If $\mathsf{d}(q') = \emptyset$, state q' is not considered and no successor of q' is constructed. Therefore, no more than $|A|$ states are created for each existing state of $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w})$ and $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w}^{\mathrm{R}})$. □

Lemma 5.2.31. Lemma 5.2.14 holds for automata $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w})$ and $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w}^{\mathrm{R}})$ constructed by algorithm stated by Proposition 5.2.27.

Proof. There is no assumption about exact positions or exact factor in proof of Lemma 5.2.14. □

Lemma 5.2.32. Number of elements of all d-subsets of automata $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w})$ and $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w}^{\mathrm{R}})$ constructed by algorithm stated by Proposition 5.2.27 for string $\boldsymbol{w}$ is no more than $|\boldsymbol{w}|$-times more than number of states of automata $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w})$ and $\mathcal{M}^{\mathsf{H},k}_{\mathrm{SD}}(\boldsymbol{w}^{\mathrm{R}})$.

Proof. Idea stated in proof of Lemma 4.2.42. □

Lemma 5.2.33. The total number of elements of all d-subsets constructed by algorithm stated by Proposition 5.2.27 for alphabet A for automata $\mathcal{M}_{\mathrm{SD}}^{\mathsf{H},k}(\boldsymbol{w})$ and $\mathcal{M}_{\mathrm{SD}}^{\mathsf{H},k}(\boldsymbol{w}^{\mathrm{R}})$ is the same as number of elements of automata $\mathcal{M}_{\mathrm{SD}}^{\mathsf{H},k}(\boldsymbol{w})$ and $\mathcal{M}_{\mathrm{SD}}^{\mathsf{H},k}(\boldsymbol{w}^{\mathrm{R}})$ (there are no extra elements constructed and immediately discarded).

Proof. State q constructed by Algorithm 5.2 is not state of $\mathcal{M}_{\mathrm{SD}}^{\mathsf{H},k}(\boldsymbol{w})$ or $\mathcal{M}_{\mathrm{SD}}^{\mathsf{H},k}(\boldsymbol{w}^{\mathrm{R}})$ if and only if its d-subset is empty (line 7). Therefore, such state adds no extra elements of d-subset. □

Lemma 5.2.34. Lemmas 5.2.17 and 5.2.18 hold for computation of all k-approximate seeds of a string with Hamming distance.

Proof. By Lemmas 5.2.24 and 5.2.25, there is no need to change Algorithm 5.3 and 5.4 to compute all k-approximate seeds. □

Theorem 5.2.35. Time complexity of computation of all k-approximate seeds over alphabet A of string $\boldsymbol{w}$ with their smallest Hamming distance (algorithm stated by Proposition 5.2.27) is

$$\mathcal{O}\left(|\boldsymbol{w}|^2 \cdot k \cdot \binom{|\boldsymbol{w}|}{k} \cdot |A|^k\right)$$

Proof. Most parts of proof of Theorem 5.2.20 hold here as well (see the previous lemmas). The difference is that the proposed algorithm processes more states and thus more strings.

As number of states of automata $\mathcal{M}_{\mathrm{SD}}^{\mathsf{H},k}(\boldsymbol{w})$ and $\mathcal{M}_{\mathrm{SD}}^{\mathsf{H},k}(\boldsymbol{w}^{\mathrm{R}})$ is at most $|\boldsymbol{w}| \cdot k \cdot \binom{|\boldsymbol{w}|}{k} \cdot \frac{(|A|-1)^k-1}{|A|-1}$, construction of all d-subsets is done in at most $|\boldsymbol{w}|^2 \cdot k \cdot \binom{|\boldsymbol{w}|}{k} \cdot \frac{(|A|-1)^k-1}{|A|-1}$ time. □

Theorem 5.2.36. Space complexity of computation of all k-approximate seeds over alphabet A of string $\boldsymbol{w}$ with their smallest Hamming distance (algorithm stated by Proposition 5.2.27) is

$$\mathcal{O}\left(|\boldsymbol{w}| \cdot k \cdot \binom{|\boldsymbol{w}|}{k} \cdot |A|^k\right)$$

Proof. Most parts of proof of Theorem 5.2.23 hold here as well. The difference is the number of states of automata $\mathcal{M}_{\mathrm{SD}}^{\mathsf{H},k}(\boldsymbol{w})$ and $\mathcal{M}_{\mathrm{SD}}^{\mathsf{H},k}(\boldsymbol{w}^{\mathrm{R}})$ and the number of elements of d-subsets held in memory in a time.

The number of states of the deterministic automata held in memory is (5.2) (all the states are held in memory). The number of d-subsets stored in a memory is due to Lemma 5.2.21 equal to number of states having depths n and $n-1$ which is equal to number of k-approximate factors of $\boldsymbol{w}$ of length n and $n-1$ given by (5.3). Size of each d-subset is at most $|\boldsymbol{w}|$. As each of automata $\mathcal{M}_{\mathrm{SD}}^{\mathsf{H},k}(\boldsymbol{w})$ and $\mathcal{M}_{\mathrm{SD}}^{\mathsf{H},k}(\boldsymbol{w}^{\mathrm{R}})$ is constructed as a trie, the number of transitions is by one less than the number of states. □

5.3 Experimental results

The algorithms for computation of all k-approximate seeds with Hamming distance and restricted k-approximate seeds with Hamming distance were implemented, run and measured under the same conditions as experiments with implementation of searching covers (Section 4.3). Instead of peak number of states stored in memory at a time, maximum (peak) number of states stored in the queue L at a time was recorded.

5.3.1 Restricted approximate seeds

Some properties of the input are shown at Figures 5.7 and 5.8 – number of found restricted k-approximate seeds with Hamming distance in a prefix of input sequence. As expected, the number of found restricted k-approximate seeds does not necessarily grow with length of input; for the same input it grows with k as more factors are k-approximate seeds with higher k. For particular input data, computation with high value of k (e.g., $k = 9$) does not produce results considered meaningful, as number of restricted k-approximate seeds is high compared to length of input.

Time needed to run the computation depending on input length and for a few different values of maximum Hamming distance is shown at Figure 5.9. As expected, the elapsed time grows similarly to the number of states of constructed deterministic automaton, i.e., similarly to number of exact factors of the input that need to be processed and checked. In contrast to the number of states, the time depends also on k. The dependence on the number of states on k is depicted at Figure 5.11 Another view is shown at Figure 5.10 where the elapsed time together with number of found restricted k-approximate seeds is depicted depending on maximum Hamming distance k for particular input length n. As expected by Theorem 5.2.20, elapsed time grows with both n and k.

Memory consumption needed for computation depending on input length and for particular value of maximum Hamming distance k is shown at Figure 5.12; the same figure shows how sum of sizes of all d-subsets stored in memory at a time depends on input length compared to real memory consumption. Similar view for number of states stored in memory at a time is depicted at Figure 5.13. The memory consumption and the sum of sizes of all stored d-subsets grow similarly depending on input length. Similar holds for memory consumption and the number of states. Another view is shown at Figure 5.14 where the memory needed compared to the sum of sizes of all d-subsets stored in memory at a time is depicted depending maximum Hamming distance k for particular input length n. Note that as stated in Theorem 5.2.23, the memory consumption and the sum of sizes of all stored d-subsets do not grow with k.

The complete data set from experimental run is shown in Table B.3.

5.3.2 Approximate seeds

Some properties of the input are shown at Figures 5.7 and 5.16 – number of found k-approximate seeds with Hamming distance in a prefix of input sequence. For particular

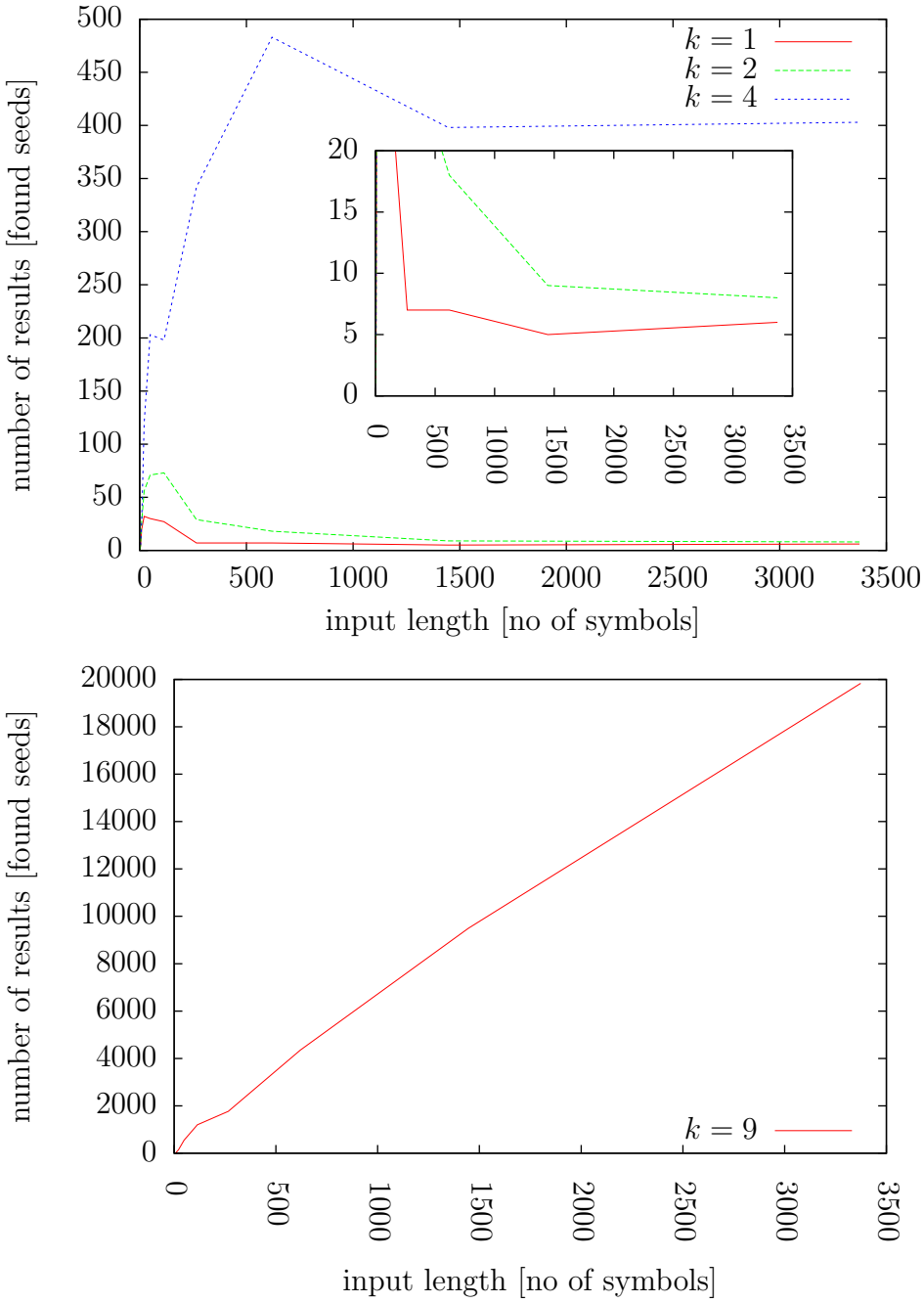

Figure 5.7: Number of all restricted k-approximate seeds for particular maximum Hamming distance

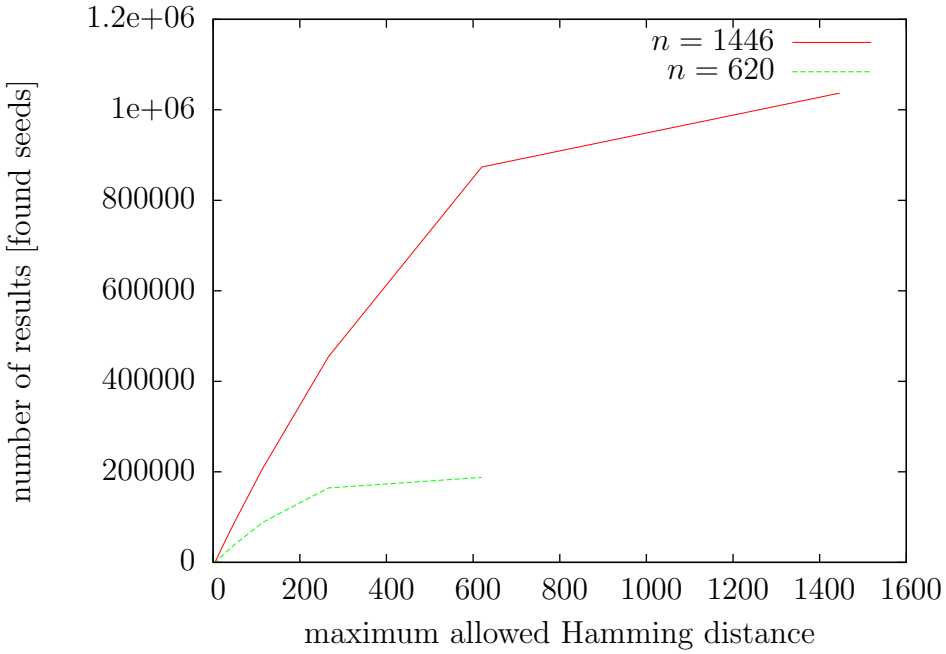

Figure 5.8: Number of all restricted k-approximate seeds for particular input length

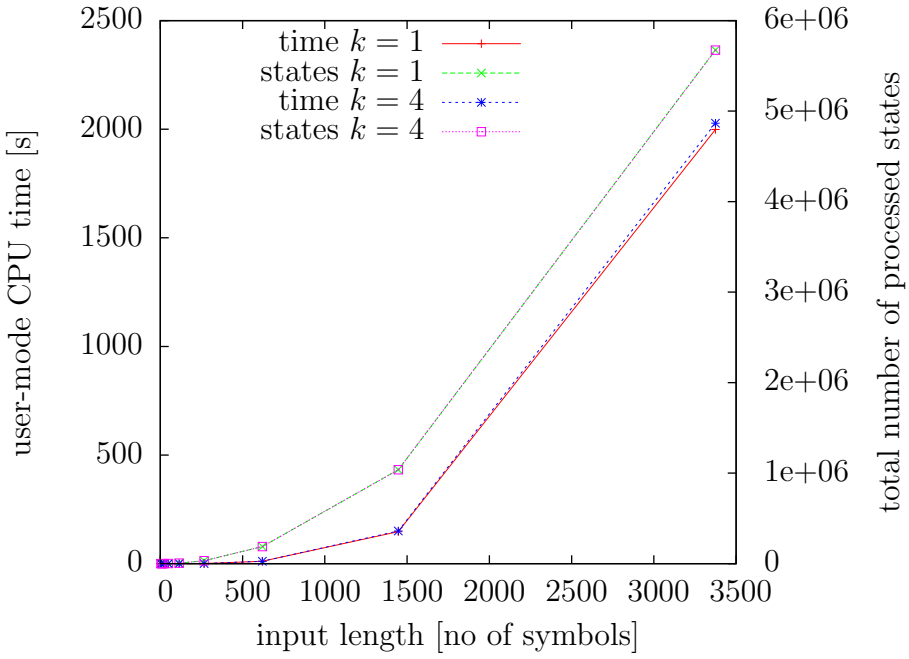

Figure 5.9: Elapsed time for computation of all restricted k-approximate seeds for particular maximum Hamming distance k compared to number of processed states

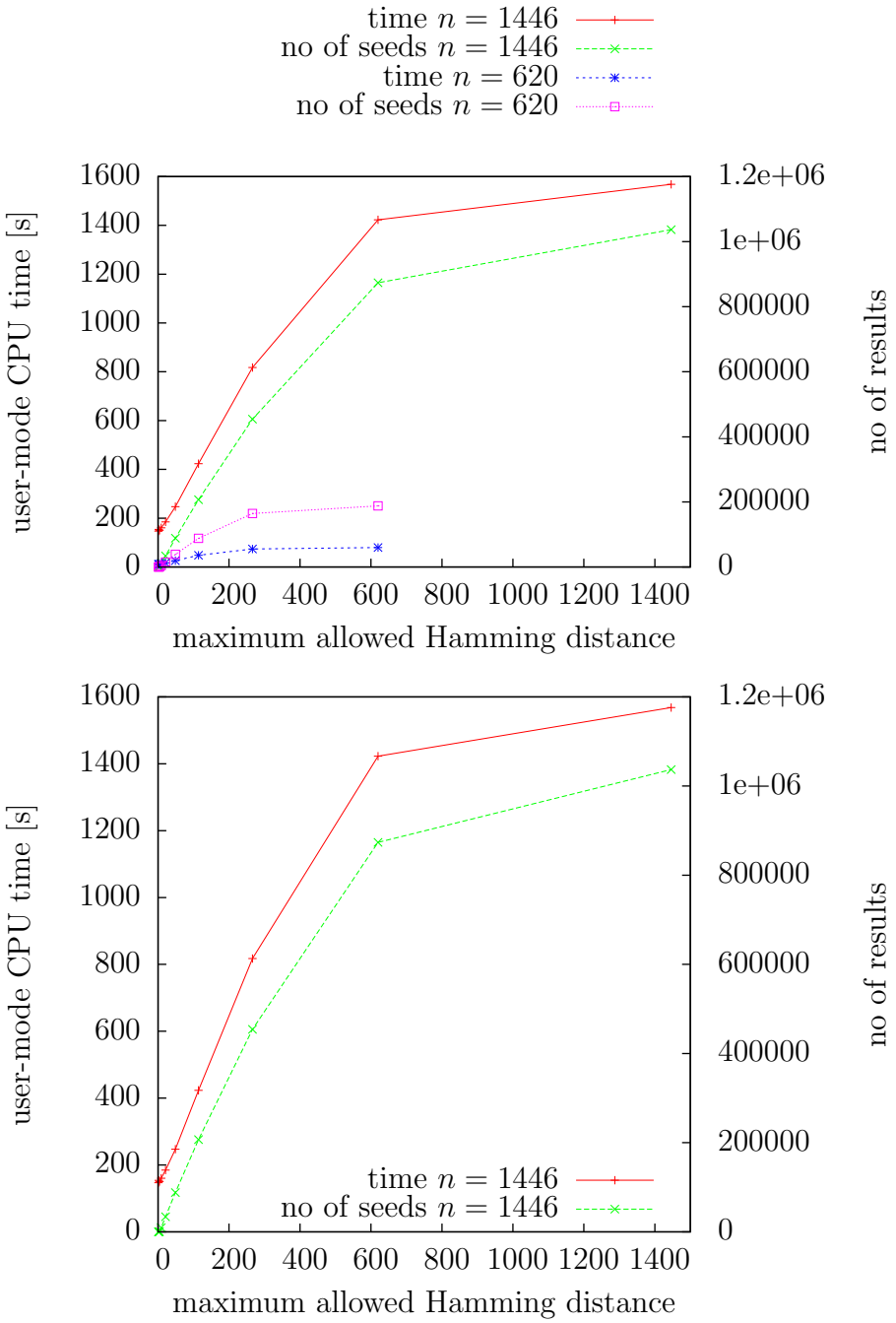

Figure 5.10: Elapsed time for computation of all restricted k-approximate seeds for particular input length n compared with number of found results

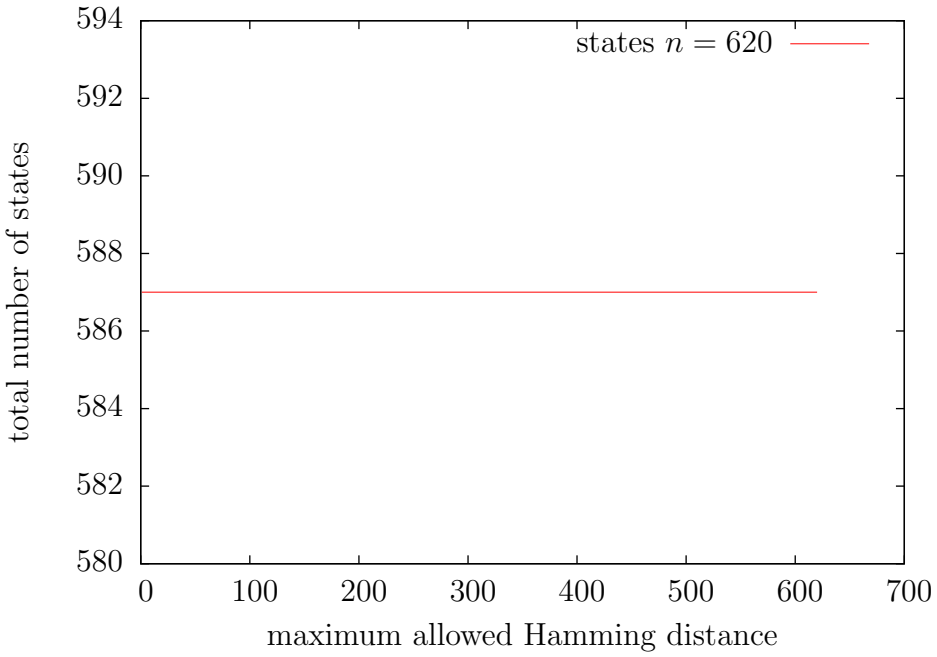

Figure 5.11: Number of states during computation of restricted k-approximate seeds for particular input length n

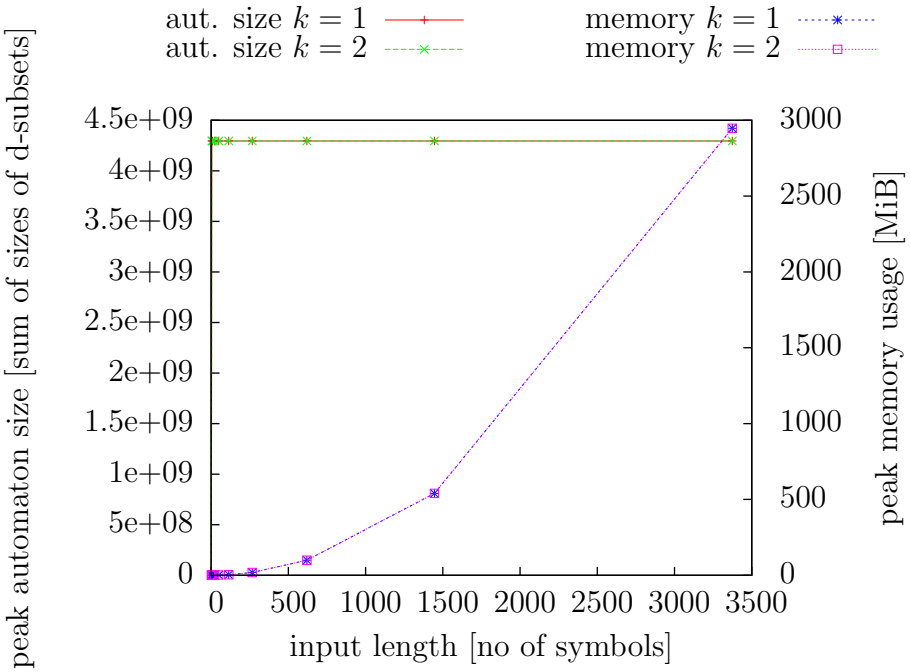

Figure 5.12: Memory needed for computation of all restricted k-approximate seeds for particular maximum Hamming distance compared with sum of sizes of all d-subsets

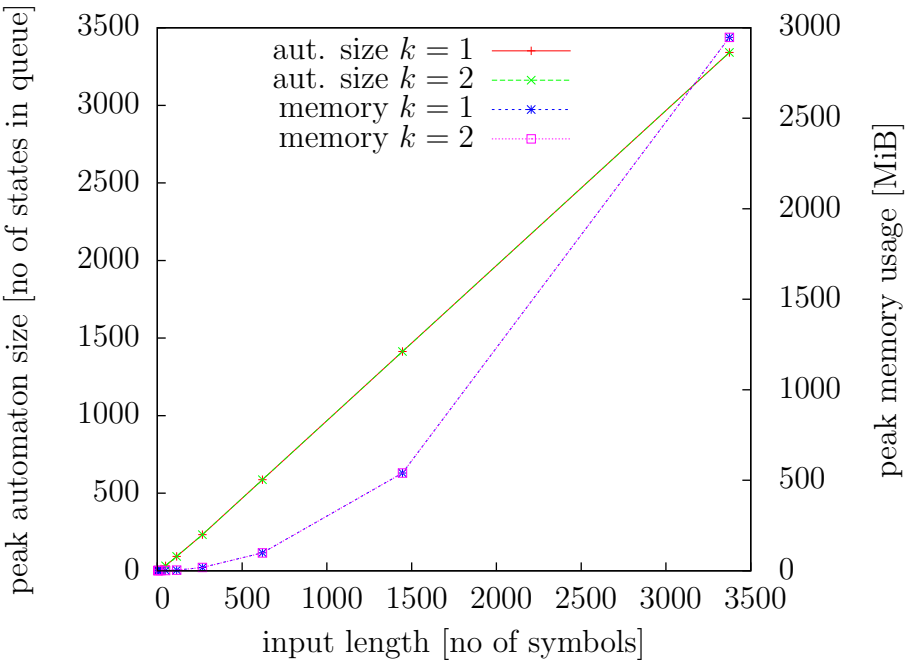

Figure 5.13: Memory needed for computation of all restricted k-approximate seeds for particular maximum Hamming distance compared with peak number of states stored in queue L at a time

input, the number of found k-approximate seeds grow fast for even $k = 2$, too much to consider the result meaningful.

Time needed to run the computation depending on input length and for a few different values of maximum Hamming distance is shown at Figure 5.17. As expected, the elapsed time grows similarly to the number of states of constructed deterministic automaton, i.e., similarly to number of k-approximate factors of the input that need to be processed and checked. The dependence on the number of states on k is depicted at Figure 5.19. Another view is shown at Figure 5.18 where the elapsed time together with number of found k-approximate seeds is depicted depending on maximum Hamming distance k for particular input length n. As expected by Theorem 5.2.35, elapsed time grows with both n and k.

Memory consumption needed for computation depending on input length and for particular value of maximum Hamming distance k is shown at Figure 5.20; the same figure shows how sum of sizes of all d-subsets stored in memory at a time depends on input length compared to real memory consumption. Similar view for number of states stored in memory at a time is depicted at Figure 5.21. The memory consumption and the sum of sizes of all stored d-subsets grow similarly depending on input length. Another view is shown at Figure 5.22 where the memory needed compared to the sum of sizes of all d-subsets stored in memory at a time is depicted depending maximum Hamming distance k for particular input length n.

The complete data set from experimental run is shown in Table B.4.

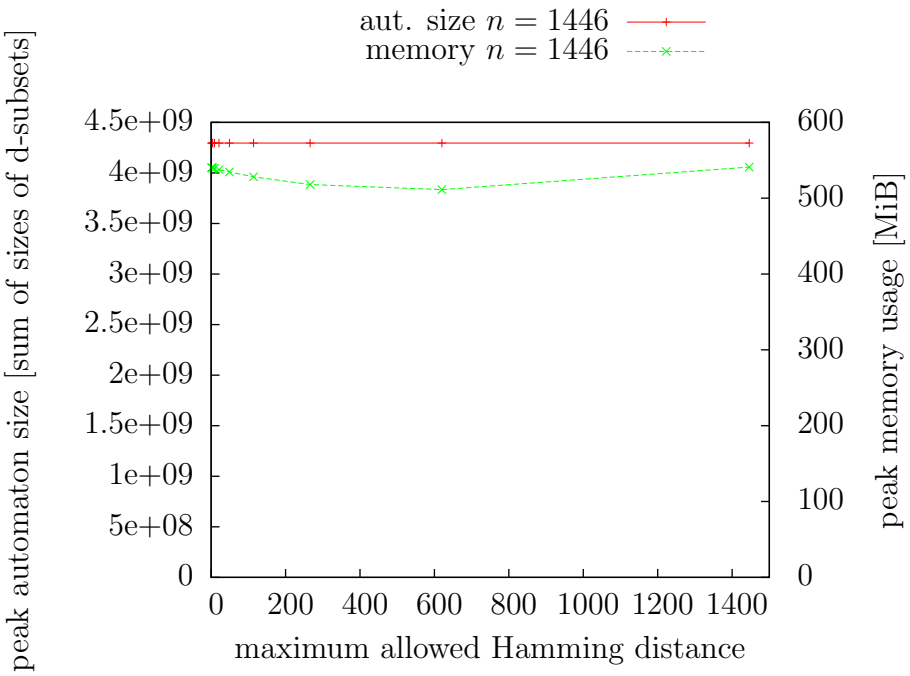

Figure 5.14: Memory needed for computation of all restricted k-approximate seeds for particular input length n

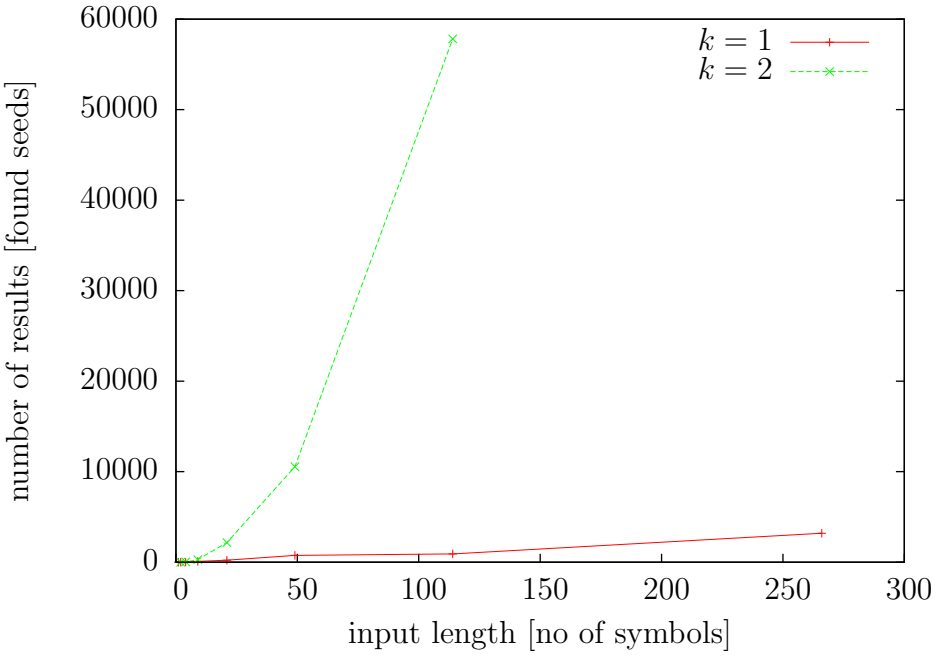

Figure 5.15: Number of all k-approximate seeds for particular maximum Hamming distance

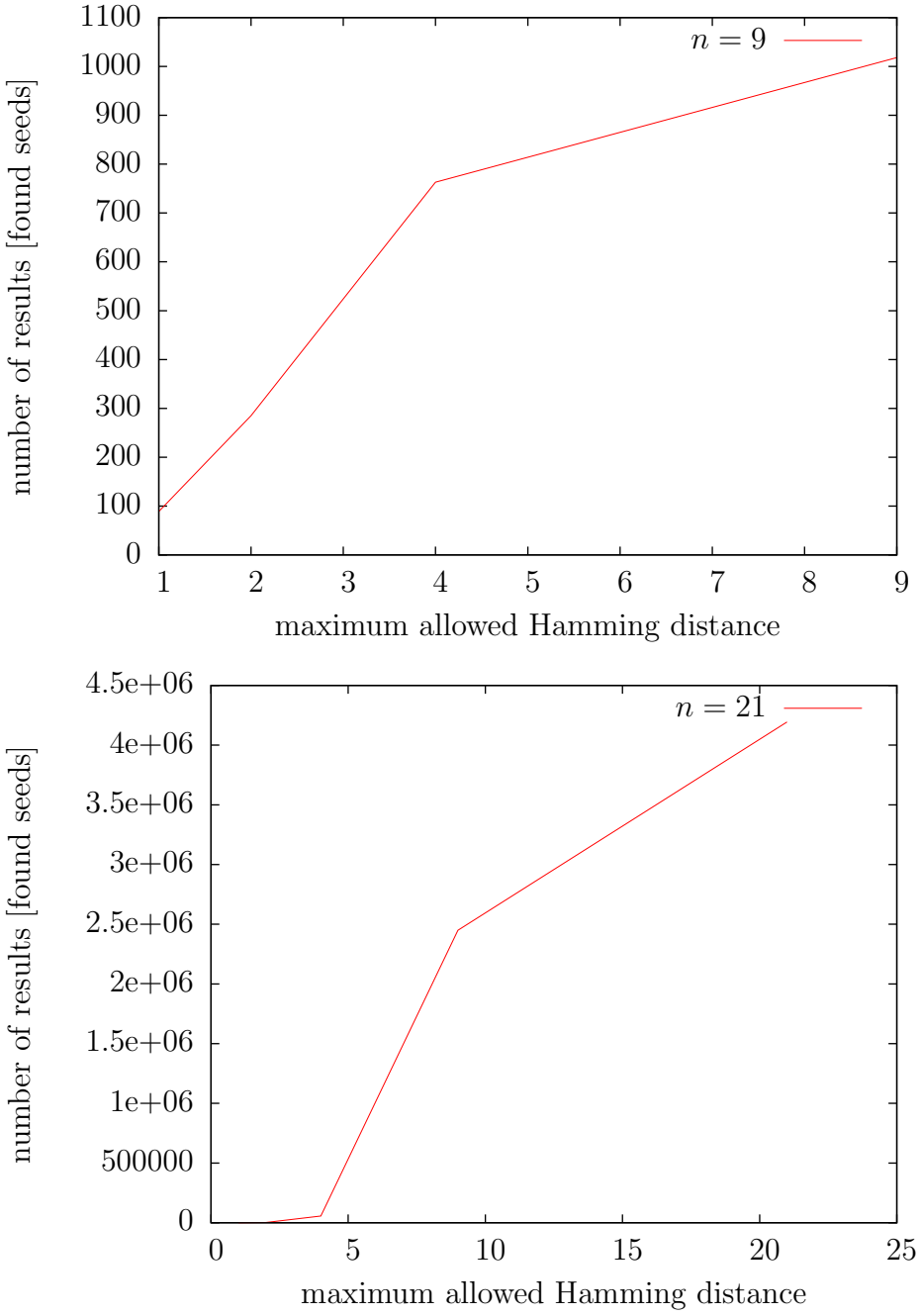

Figure 5.16: Number of all k-approximate seeds for particular input length

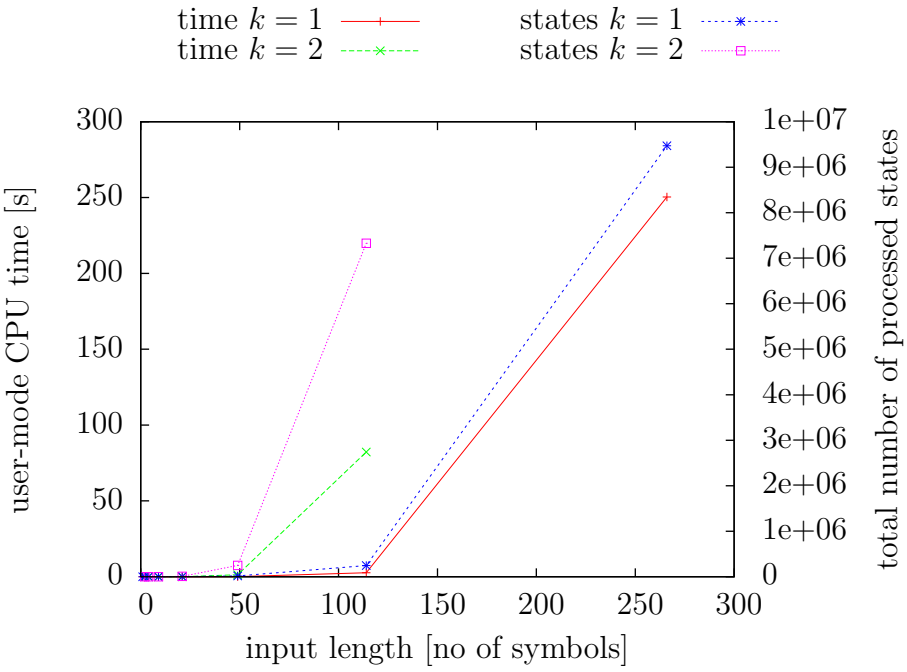

Figure 5.17: Elapsed time for computation of all k-approximate seeds for particular maximum Hamming distance k compared to number of processed states

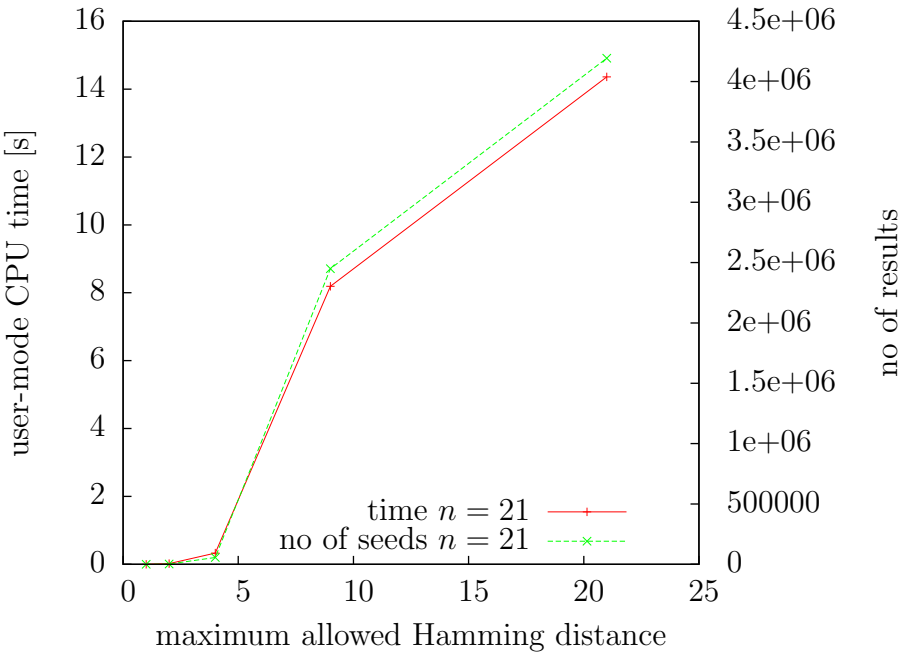

Figure 5.18: Elapsed time for computation of all k-approximate seeds for particular input length n compared with number of found results

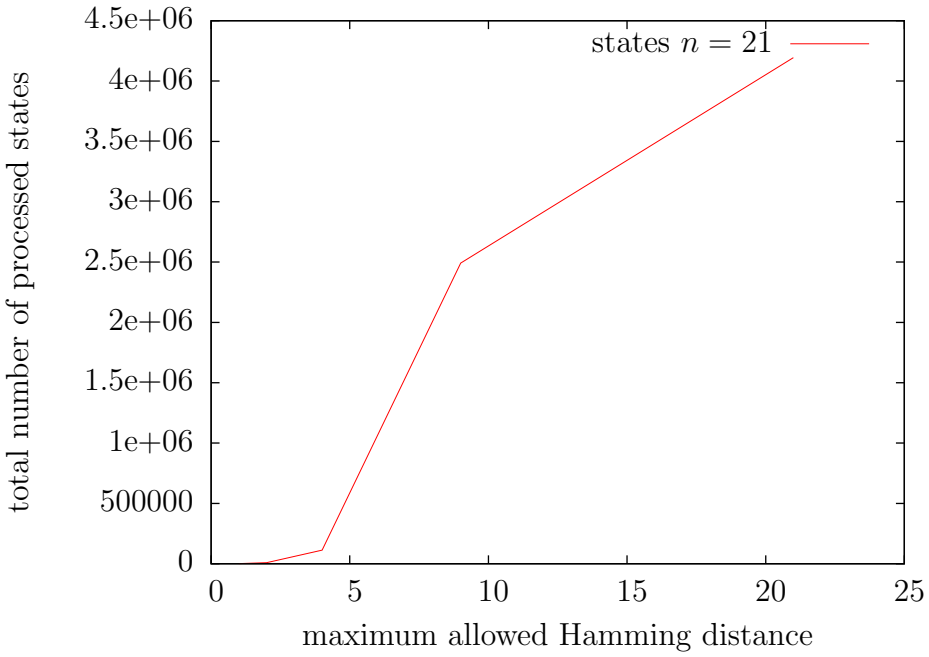

Figure 5.19: Number of states processed during computation of k-approximate seeds for particular input length n

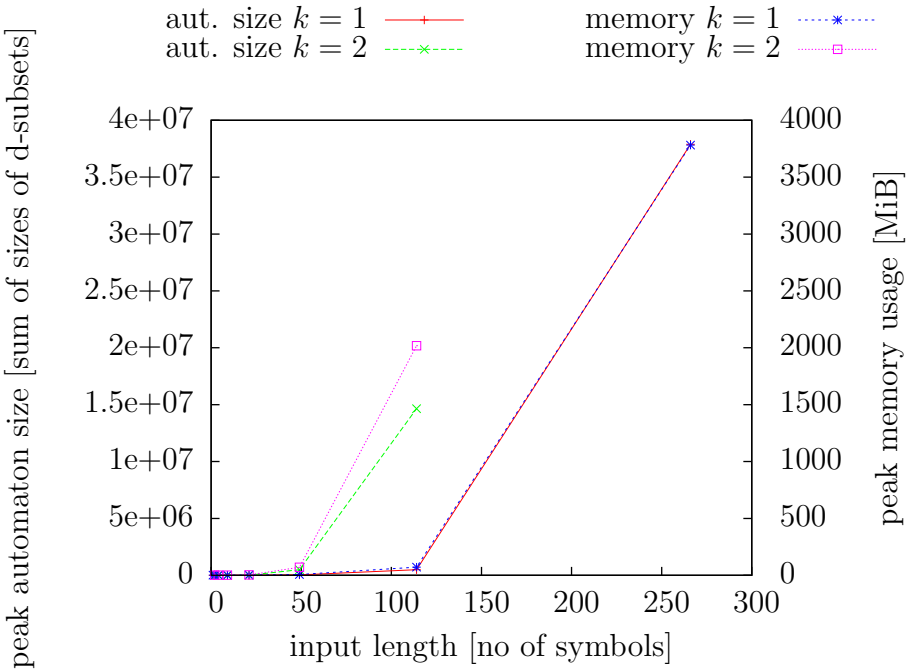

Figure 5.20: Memory needed for computation of all k-approximate seeds for particular maximum Hamming distance compared with sum of sizes of all d-subsets

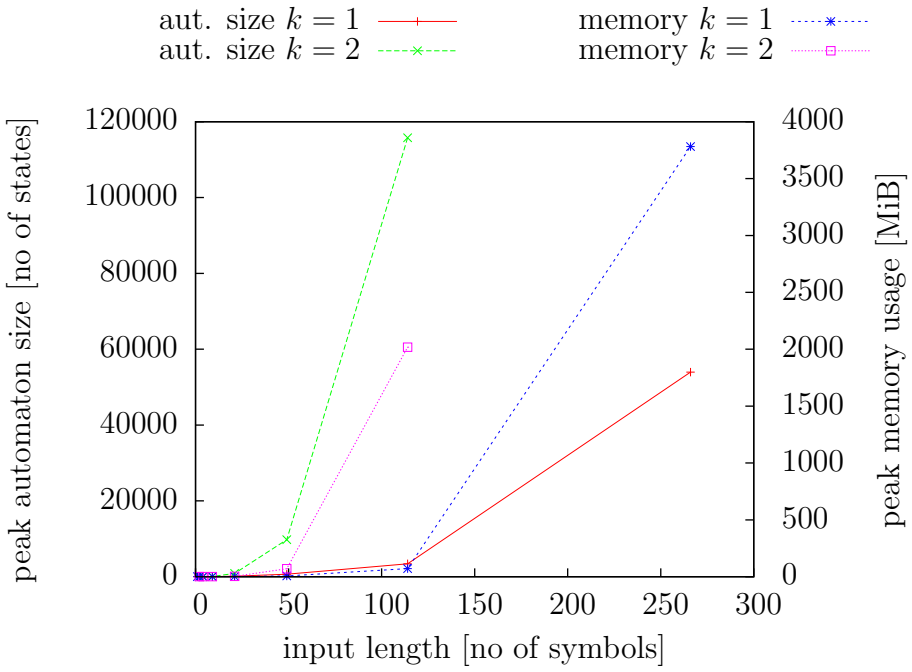

Figure 5.21: Memory needed for computation of all k-approximate seeds for particular maximum Hamming distance compared with number of states

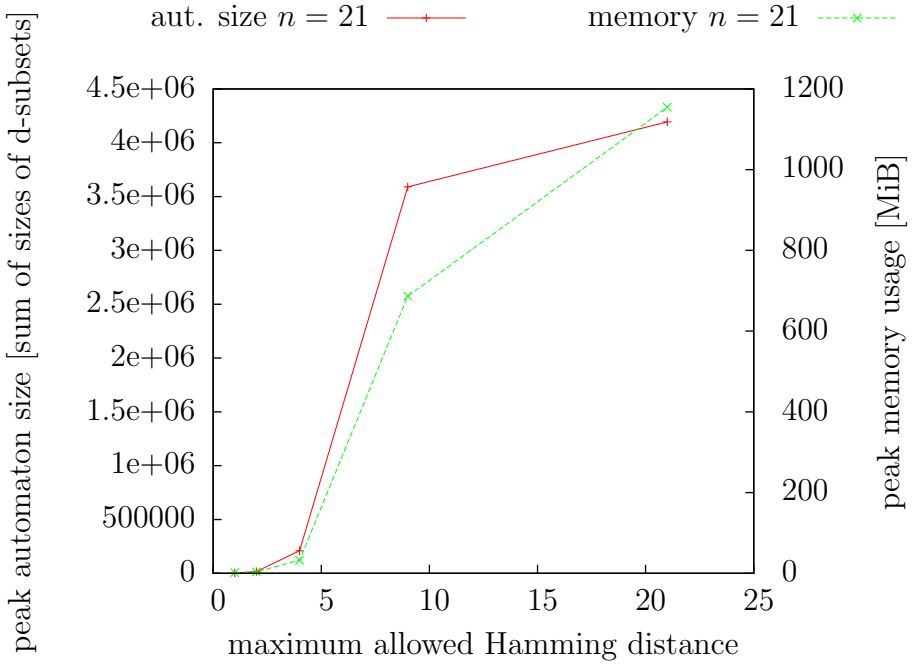

Figure 5.22: Memory needed for computation of all k-approximate seeds for particular input length n

Chapter 6

Conclusions

In this chapter we summarize this thesis and its contributions. We also present a list of suggested future work to extend research presented in this thesis.

6.1 Summary

In this dissertation thesis, some results and algorithms related to searching covers and seeds are presented. It is divided in several chapters.

In the introductory Chapter 1, the problems are stated along with motivation. Brief summary of related work is also given there.

In the next two Chapters 2 and 3, theoretical background and preliminaries are given, i.e. basic definitions, precise problem statements, and results from related works that are important to know about covers and seeds and ways how to compute them. Known properties of covers and seeds are summarized, along with exact and approximate indexing strings using finite automata.

The following chapters present novel algorithms for computing covers and seeds.

In this thesis, it is shown that computing string regularities, in particular, covers and seeds, is another area where it is possible to use finite automata as a basic formalism.

6.2 Contributions of the dissertation thesis

Contributions of this dissertation thesis are the following ones:

Chapter 4 This chapter presents solutions for the problems of searching all covers of a string, searching all restricted k-approximate covers and all k-approximate covers of a string with Hamming distance. All the solutions are based on finite automata. For all the algorithms, their correctness is proved along with their time and space complexities. Experimental results show real performance of their implementation. Time complexity

of the solution of the problem of searching all restricted k-approximate covers of string $\boldsymbol{w} \in A^*$ is $\mathcal{O}(|\boldsymbol{w}|^3 \cdot (|A| + k))$.

Chapter 5 This chapter presents solutions for the problems of searching all restricted k-approximate seeds of a string and searching all k-approximate seeds of a string. The algorithm for the latter problem presented here is the only known solution. All the algorithms are based on finite automata. For all of them, their correctness and also their time and space complexities are proved. Experimental results show real performance of implementation of the algorithms. Time complexity of the solution of the problem of searching all restricted k-approximate covers of string $\boldsymbol{w} \in A^*$ is $\mathcal{O}(|\boldsymbol{w}|^3 \cdot (|A| + k))$.

6.3 Future work

Distance Algorithms for computing approximate covers and seeds given in Chapters 4 and 5 deal with Hamming distance only. They heavily use the fact that Hamming distance is defined for strings of equal length. Therefore, they cannot work with distance functions allowing insertions and deletions. Pattern-matching finite automata for such distance functions are already known and adopting them for approximate indexing and computation of covers and seeds is one area for further research. The algorithms presented in this thesis consider maximum distance as fixed number of errors allowed for the whole input. Extending the algorithms based on finite automata to another model, e.g., fixing the allowed number of errors to a window of given length is also an area to research.

External memory As shown in Section 5.3, implementation of introduced algorithms for searching approximate and restricted approximate seeds requires high amount of memory for a long input, e.g., for processing whole nucleotide. Available operating memory limits searching approximate seeds in such long input data. External memory algorithm for computation of approximate seeds (and also covers) is an open problem.

Approximate covers with Hamming distance as NP-hard It is known that searching approximate (non-restricted) covers with Levenshtein distance is NP-hard. It is to be found out whether the all smallest distance k-approximate covers problem with Hamming distance, for which the algorithm is introduced in this thesis, is also NP-hard.

Parallel computation Searching all approximate and restricted approximate covers and seeds in parallel is an open problem.

Searching regularities in trees Regularities can be searched not only in strings, but also in rooted trees. Cover of a tree can be defined as follows: tree (template) $\mathcal{U}$ is a cover of rooted tree $\mathcal{T}$ if subgraphs of $\mathcal{T}$ ($\{\mathcal{I}_1, \ldots, \mathcal{I}_m\}$) isomorphic to $\mathcal{U}$ occur in $\mathcal{T}$ in such a way that path from root to any leaf in $\mathcal{T}$ contains only nodes within any of

$\{\mathcal{I}_1, \ldots, \mathcal{I}_m\}$. Similarly as finite automata are considered basic formalism in algorithms for string regularities in this thesis, pushdown automata are considered to be formalism for algorithms for tree pattern matching and indexing (a few years ago, algorithmic discipline called Arbology was founded, it deals with algorithms on trees using pushdown automata). One possible way of further research is an algorithm for searching all covers of rooted trees based on pushdown automata and similar principles to those presented in this thesis.

Bibliography

[1] Aho, A. V.; Corasick, M. J. Efficient string matching: an aid to bibliographic search. *Commun. ACM*, volume 18, no. 6, 1975: pp. 333–340, ISSN 0001-0782, doi:10.1145/360825.360855.

[2] Antoniou, P.; Crochemore, M.; Iliopoulos, C. S.; et al. Conservative String Covering of Indeterminate Strings. In *Proceedings of the Prague Stringology Conference 2008*, edited by J. Holub; J. Žďárek, 2008, ISBN 978-80-01-04145-1, pp. 108–115.

[3] Apostolico, A.; Ehrenfeucht, A. Efficient detection of quasiperiodicities in strings. Technical report, 1990.

[4] Apostolico, A.; Ehrenfeucht, A. Efficient detection of quasiperiodicities in strings. *Theoretical Computer Science*, volume 119, no. 2, 1993: pp. 247–265, ISSN 0304-3975, doi:10.1016/0304-3975(93)90159-Q.

[5] Apostolico, A.; Farach, M.; Iliopoulos, C. S. Optimal Superprimitivity Testing for Strings. *Information Processing Letters*, volume 39, no. 1, 1991: pp. 17–20, ISSN 0020-0190, doi:10.1016/0020-0190(91)90056-N.

[6] Berkman, O.; Iliopoulos, C. S.; Park, K. S. The subtree max gap problem with application to parallel string covering. *Information and Computation*, volume 123, no. 1, 1995: pp. 127–137.

[7] Blumer, A.; Blumer, J.; Ehrenfeucht, A.; et al. Linear size finite automata for the set of all subwords of a word – an outline of results. *Bulletin of the EATCS*, volume 21, 1983: pp. 12–20.

[8] Blumer, A.; Blumer, J.; Haussler, D.; et al. The Smallest Automaton Recognizing the Subwords of a Text. *Theoretical Computer Science*, volume 40, 1985: pp. 31–55.

[9] Breslauer, D. An On-Line String Superprimitivity Test. *Information Processing Letters*, volume 44, no. 6, 1992: pp. 345–347. Available from: `citeseer.ist.psu.edu/breslauer95line.html`

[10] Breslauer, D. Testing String Superprimitivity in Parallel. *Information Processing Letters*, volume 49, no. 5, 1994: pp. 235–241. Available from: citeseer.ist.psu.edu/breslauer92testing.html

[11] Brodal, G. S.; Pedersen, C. N. S. Finding maximal quasiperiodicities in strings. In *Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching*, Springer-Verlag, 2000, pp. 397–411.

[12] Christodoulakis, M.; Iliopoulos, C. S.; Park, K. S.; et al. Approximate Seeds of Strings. In *Proceedings of the Prague Stringology Conference '03*, 2003, ISBN 80-01-02823-2, pp. 25–36.

[13] Christodoulakis, M.; Iliopoulos, C. S.; Park, K. S.; et al. Approximate Seeds of Strings. *Journal of Automata, Languages and Combinatorics*, volume 10, no. 5/6, 2005: pp. 609–626.

[14] Christodoulakis, M.; Iliopoulos, C. S.; Park, K. S.; et al. Implementing Approximate Regularities. *Mathematical and Computer Modelling*, volume 42, October 2005: pp. 855–866.

[15] Christou, M.; Crochemore, M.; Guth, O.; et al. On the right-seed array of a string. In *Computing and Combinatorics*, Springer, 2011, pp. 492–502.

[16] Christou, M.; Crochemore, M.; Guth, O.; et al. On left and right seeds of a string. *Journal of Discrete Algorithms*, volume 17, 2012: pp. 31–44.

[17] Christou, M.; Crochemore, M.; Iliopoulos, C. S.; et al. Efficient seeds computation revisited. In *Proceedings of the 22nd annual conference on Combinatorial pattern matching*, CPM'11, Berlin, Heidelberg: Springer-Verlag, 2011, ISBN 978-3-642-21457-8, pp. 350–363. Available from: http://dl.acm.org/citation.cfm?id=2018243.2018275

[18] Crochemore, M. An Optimal Algorithm for Computing the Repetitions in a Word. *Information Processing Letters*, volume 12, no. 5, 1981: pp. 244–250.

[19] Crochemore, M. Transducers and repetitions. *Theoretical Computer Science*, volume 45, 1986: pp. 63–86, ISSN 0304-3975, doi:DOI:10.1016/0304-3975(86)90041-1. Available from: http://www.sciencedirect.com/science/article/B6V1G-48CX3XH-12/2/e9a2d3a533c848545053d1337c11f4cc

[20] Crochemore, M.; Epifanio, C.; Gabriele, A.; et al. On the Suffix Automaton with Mismatches. In *Implementation and Application of Automata*, volume 4783, edited by J. Holub; J. Žďárek, Czech Technical University in Prague, Czech Republic: Springer, 2007, ISBN 978-3-540-76335-2, ISSN 1611-3349, pp. 144–156, doi:10.1007/978-3-540-76336-9_15.

[21] Crochemore, M.; Landau, G. M.; Ziv-Ukelson, M. A Subquadratic Sequence Alignment Algorithm for Unrestricted Scoring Matrices. *SIAM Journal on Computing*, volume 32, no. 6, 2003: pp. 1654–1673, doi:10.1137/S0097539702402007, `http://epubs.siam.org/doi/pdf/10.1137/S0097539702402007`. Available from: `http://epubs.siam.org/doi/abs/10.1137/S0097539702402007`

[22] Damerau, F. J. A Technique for Computer Detection and Correction of Spelling Errors. *Commun. ACM*, volume 7, no. 3, March 1964: pp. 171–176, ISSN 0001-0782, doi:10.1145/363958.363994. Available from: `10.1145/363958.363994`

[23] Ehrenfeucht, A.; Haussler, D. A new distance metric on strings computable in linear time. *Discrete Applied Mathematics*, volume 20, no. 3, 1988: pp. 191–203, ISSN 0166-218X, doi:10.1016/0166-218X(88)90076-5. Available from: `http://www.sciencedirect.com/science/article/pii/0166218X88900765`

[24] Epifanio, C.; Gabriele, A.; Mignosi, F. Languages with mismatches and an application to approximate indexing. In *Developments in Language Theory*, Springer, 2005, pp. 224–235.

[25] Flouri, T. *Indexing Degenerate Strings*. Master's thesis, Czech Technical University in Prague, 2008, available from: `http://dip.felk.cvut.cz`.

[26] sys-process/time – /usr/bin/time -f '%M' reports incorrect memory usage size with linux 2.6.34. [online]. Available from: `http://bugs.gentoo.org/show_bug.cgi?id=332253`, November 2011, [cit. 2014-02-25].

[27] Hamming, R. W. Error detecting and error correcting codes. *Bell System Tech. J.*, volume 29, 1950: pp. 147–160, ISSN 0005-8580.

[28] Hopcroft, J. E.; Ullman, J. D. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 1979.

[29] Iliopoulos, C. S.; Mohamed, M.; Mouchard, L.; et al. String regularities with don't cares. *Nordic journal of Computing*, volume 10, no. 1, 2003: pp. 40–51, ISSN 1236-6064.

[30] Iliopoulos, C. S.; Moore, D.; Park, K. S. Covering a String. In *CPM '93: Proceedings of the 4th Annual Symposium on Combinatorial Pattern Matching*, London, UK: Springer-Verlag, 1993, ISBN 3-540-56764-X, pp. 54–62.

[31] Iliopoulos, C. S.; Mouchard, L. Fast Local Covers. Technical report TR-98-03, Department of Computer Science, King's College London, March 1998.

[32] Iliopoulos, C. S.; Park, K. S. A work-time optimal algorithm for computing all string covers. *Theoretical Computer Science*, volume 164, no. 1-2,

1996: pp. 299–310, ISSN 0304-3975, doi:DOI:10.1016/0304-3975(96)00047-3. Available from: http://www.sciencedirect.com/science/article/B6V1G-3WP2DWG-K/2/ea0700bce08fbab3785dea44f8c17160

[33] Kececioglu, J.; Sankoff, D. Exact and Approximation Algorithms for Sorting by Reversals, with Application to Genome Rearrangement. *Algorithmica*, volume 13, no. 1-2, 1995: pp. 180–210, ISSN 0178-4617, doi:10.1007/BF01188586.

[34] Kociumaka, T.; Kubica, M.; Radoszewski, J.; et al. A linear time algorithm for seeds computation. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, SIAM, 2012, pp. 1095–1112. Available from: http://dl.acm.org/citation.cfm?id=2095116.2095202

[35] Levenstein, V. Binary codes capable of correcting spurious insertions and deletions of ones. *Problems of Information Transmission*, volume 1, no. 1, 1965: pp. 8–17.

[36] Li, Y.; Smyth, W. F. Computing the Cover Array in Linear Time. *Algorithmica*, volume 32, no. 1, 2002: pp. 95–106.

[37] Melichar, B. A simple method of complete indexing. In *Proceedings of DATASEM'97*, CS-COMPEX, 1997, pp. 183–188.

[38] Melichar, B.; Holub, J.; Polcar, T. Text Searching Algorithms, Volume I. November 2005, available from: http://stringology.org/athens.

[39] Moore, D.; Smyth, W. F. Computing the covers of a string in linear time. In *SODA '94: Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms*, Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1994, ISBN 0-89871-329-3, pp. 511–515.

[40] Moore, D.; Smyth, W. F. An optimal algorithm to compute all the covers of a string. *Information Processing Letters*, volume 50, 1994: pp. 101–103.

[41] Moore, D.; Smyth, W. F. A correction to "An optimal algorithm to compute all the covers of a string". *Information Processing Letters*, volume 54, no. 2, 1995: pp. 101–103, ISSN 0020-0190, doi:10.1016/0020-0190(94)00235-Q.

[42] Rabin, M. O.; Scott, D. Finite automata and their decision problems. *IBM journal of research and development*, volume 3, no. 2, 1959: pp. 114–125.

[43] Ristad, E. S.; Yianilos, P. N. Learning String-Edit Distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 20, no. 5, 1998: pp. 522–532, ISSN 0162-8828, doi:10.1109/34.682181.

[44] Sim, J. S.; Park, K. S.; Kim, S. R.; et al. Finding Approximate Covers of Strings. *Journal of Korea Information Science Society*, volume 29, 2002: pp. 16–21.

[45] Smyth, B. *Computing patterns in strings*. Pearson Education, 2003.

[46] Tichy, W. F. The String-to-String Correction Problem with Block Moves. *ACM Trans. Comput. Syst.*, volume 2, no. 4, November 1984: pp. 309–321, ISSN 0734-2071, doi: 10.1145/357401.357404.

[47] Voráček, M.; Melichar, B. Searching for Regularities in Strings using Finite Automata. In *Proceedings of Workshop 2005*, volume A, Czech Technical University in Prague, 2005, ISBN 80-01-03201-9, pp. 264–265.

[48] Voráček, M. Algorithms on Generalized Strings. Dissertation thesis proposal, Czech Technical University in Prague, 2005.

[49] Voráček, M. Computing Covers in Generalized Strings. In *POSTER 2005*, Czech Technical University in Prague, Faculty of Electrical Engineering, 2005.

[50] Voráček, M.; Melichar, B. Searching for Regularities in Generalized Strings using Finite Automata. In *Proceedings of the International Conference on Numerical Analysis and Applied Mathematics*, WILEY – VCH Verlag, 2005, ISBN 3-527-40652-2, pp. 809–812.

[51] Voráček, M.; Melichar, B. Computing Seeds in Generalized Strings. In *Proceedings of Workshop 2006*, Czech Technical University in Prague, 2006, pp. 138–139.

Reviewed publications of the author relevant to the thesis

[A.1] Guth, O.; Melichar, B.; Balík, M. Searching all approximate covers and their distance using finite automata. *Information technologies – applications and theory*, 2008, 21–26. ISBN 978-80-969184-9-2.

[A.2] Guth, O.; Melichar, B. Finite Automata Approach to Computing All Seeds of Strings with the Smallest Hamming Distance. *IAENG International Journal of Computer Science*, 2009, 36, 2. ISSN 1819-656X.

- Journal version of *Searching All Seeds of Strings with Hamming Distance using Finite Automata*.

[A.3] Guth, O.; Melichar, B. Using Finite Automata Approach for Searching Approximate Seeds of Strings. In: *Intelligent Automation and Computer Engineering*. Springer Netherlands, 2010. p. 347–360. ISSN 1876-1100. ISBN 978-90-481-3516-5.

- Book-chapter version of *Searching All Seeds of Strings with Hamming Distance using Finite Automata*.

Remaining publications of the author relevant to the thesis

[A.4] Guth, O. Searching Approximate Covers of Strings Using Finite Automata [lecture]. London Stringology Days, 2009.

Appendix A

Acronyms

Acronyms

DFA deterministic finite automaton.

FA finite automaton.

NFA nondeterministic finite automaton.

Appendix B

Experimental results

Complete set of data retrieved during experimental run of implementations of algorithms given in Chapters 4 and 5 is shown in this appendix. Details about environment are given in Section 4.3. Note that values of memory consumption are 4 times higher than real memory requirements. The values shown are the following:

n input length,

k maximum Hamming distance,

time number of seconds needed to run,

mem4 memory consumption in KiB times 4,

 # res number of results (regularities) found,

d-sub sum of sizes of all d-subsets stored in memory at a time,

states proc. total number of states being processed,

p. # states maximum number of states stored in memory at a time.

Table B.1: Experimental run of computing restricted k-approximate covers of a string

n	k	time	mem4	# res	d-sub	states proc.	p. # states
1	1	0.00	5344	1	1	0	0
1	5	0.00	5344	1	1	0	0
1	6	0.00	5360	1	1	0	0
2	1	0.00	5376	1	3	2	1
2	2	0.00	5392	1	3	2	1
2	5	0.00	5392	1	3	2	1
2	6	0.00	5360	1	3	2	1

n	k	time	mem4	# res	d-sub	states proc.	p. # states
4	1	0.00	5376	2	7	3	2
4	2	0.00	5360	2	8	4	2
4	4	0.00	5376	2	10	6	3
4	5	0.00	5392	2	10	6	3
4	6	0.00	5376	2	10	6	3
9	1	0.00	5376	6	24	13	4
9	2	0.00	5392	16	36	25	7
9	4	0.00	5376	19	41	25	7
9	5	0.00	5376	20	42	25	7
9	6	0.00	5392	21	43	26	7
9	9	0.00	5376	23	45	28	8
21	1	0.00	5376	13	78	47	10
21	2	0.00	5392	26	106	65	10
21	4	0.00	5408	73	168	120	16
21	5	0.00	5392	86	180	123	16
21	6	0.00	5424	100	199	134	18
21	9	0.00	5440	128	219	139	19
21	21	0.00	5456	138	231	142	20
49	1	0.00	5440	18	226	261	22
49	2	0.00	5456	46	321	418	24
49	4	0.00	5504	149	471	542	29
49	5	0.00	5520	194	563	573	30
49	6	0.00	5520	256	631	628	32
49	9	0.00	5584	392	811	711	37
49	21	0.00	5696	773	1172	827	47
49	49	0.00	5808	826	1225	830	48
114	1	0.00	5520	11	540	547	23
114	2	0.00	5552	25	773	956	26
114	4	0.00	5648	118	1164	1399	31
114	5	0.00	5696	244	1406	1620	33
114	6	0.00	5744	379	1585	1791	37
114	9	0.02	5888	837	2202	2354	43
114	21	0.05	6352	2894	4213	4230	73
114	49	0.07	7056	5121	6427	5256	112
114	114	0.10	7568	5253	6555	5259	113
266	1	0.00	5760	1	1063	1363	35
266	2	0.01	5840	8	1527	2240	36
266	4	0.03	6064	231	2426	3528	41
266	5	0.05	6192	434	3043	4195	43
266	6	0.07	6272	650	3441	4819	46
266	9	0.14	6608	1468	4976	6734	58
266	21	0.66	7776	5811	10319	12603	85

n	k	time	mem4	# res	d-sub	states proc.	p. # states
266	49	1.31	10176	15016	20610	22678	154
266	114	2.12	13920	30420	33691	32579	249
266	266	2.49	16912	32704	35511	32818	265
620	1	0.00	6176	1	1848	2650	35
620	2	0.02	6384	2	2825	4425	36
620	4	0.17	6832	312	4847	7160	41
620	5	0.27	7088	746	6014	8557	43
620	6	0.41	7312	1281	6955	9904	46
620	9	0.90	8000	3386	10146	14000	58
620	21	3.73	10672	13381	22093	27494	85
620	49	8.70	16544	36519	47822	54478	154
620	114	28.17	28176	84508	97350	104473	258
620	266	46.97	49104	161590	171851	174198	483
620	620	45.81	66592	187501	192510	188074	619
1446	1	0.01	7104	1	3193	5274	35
1446	2	0.07	7552	1	5339	8789	36
1446	4	0.95	8544	288	9601	14789	41
1446	5	1.35	9088	1169	11823	17826	43
1446	6	2.24	9552	2552	13936	20712	46
1446	9	5.38	11088	7711	20443	29421	58
1446	21	33.53	16992	30377	45886	59784	85
1446	49	67.88	30416	84030	103211	123653	154
1446	114	178.75	59856	200831	227186	255745	258
1446	266	601.54	122736	445092	481417	513428	483
1446	1446	1075.53	334784	1035679	1046181	1037619	1445
3374	1	0.04	9312	1	6389	11460	35
3374	2	0.20	10288	1	11244	19185	36
3374	4	2.54	12544	314	20878	32888	41
3374	5	7.10	13776	1746	25665	39598	43
3374	6	13.33	14800	4635	30449	46111	46
3374	9	54.58	18288	16406	44698	65379	58
3374	21	180.45	31664	70972	101096	136330	85
3374	49	430.49	62464	197058	232310	288047	154
3374	114	1680.45	133120	489241	526205	613961	258
3374	266	4442.34	291088	1139096	1176005	1305806	483
3374	1446	16967.32	1232160	4676285	4723904	4848067	2152
7872	1	0.09	14544	1	15344	26179	35
7872	2	0.64	16912	1	26710	43873	36
7872	4	15.38	22192	269	49345	76008	41
7872	5	53.07	25072	2075	60574	91472	43
7872	6	103.50	27472	7506	71932	106556	46
7872	9	384.34	35680	34939	106004	150521	58

n	k	time	mem4	# res	d-sub	states proc.	p. # states
7872	21	1518.96	66608	162175	235310	313236	85
7872	49	3531.85	138496	464483	540650	665214	154
18368	1	0.49	26720	1	35757	61184	35
18368	2	4.10	32208	1	62304	102776	36
18368	4	62.76	44656	221	115723	178896	41
18368	5	275.59	51408	2156	142384	215141	43
18368	6	657.84	57040	10458	168821	250262	46
18368	9	2544.93	76096	68492	247916	352255	58
42858	1	1.47	55232	1	84309	144232	36
42858	2	13.14	68240	1	146883	243816	37
42858	4	384.00	97360	197	272509	424239	41
42858	5	1382.90	113184	2019	335065	510363	43
100002	1	4.49	121232	1	198072	341750	36
100002	2	60.04	152000	1	345259	577582	37
100002	4	1258.31	219584	191	640067	1005713	41
100002	5	5882.42	257456	1838	787024	1209294	43
233338	1	16.30	276096	1	455097	1055336	344
233338	2	164.93	346864	1	795401	1700203	356
233338	4	5064.43	505248	219	1475701	2920335	500
233338	5	23044.04	590912	1810	1816018	3537303	505
544455	1	51.00	641632	1	1069220	15894619	4271
544455	2	443.33	808128	1	1867858	32432644	6258
544455	4	19776.65	1178032	191	3462263	37736086	6461
1270395	1	146.87	1487168	1	2485591	33034573	4271
1270395	2	1684.25	1872144	1	4348154	56179904	6258

Table B.2: Experimental run of computing k-approximate covers of a string

n	k	time	mem4	# res	d-sub	states proc.	p. # states
1	1	0.00	5376	1	1	0	0
2	1	0.00	5360	1	3	2	1
2	2	0.00	5360	1	3	2	1
4	1	0.00	5376	4	8	5	2
4	2	0.00	5376	10	10	12	3
4	4	0.00	5376	10	10	14	3
9	1	0.00	5376	20	29	41	7
9	2	0.00	5376	69	40	100	7
9	4	0.00	5376	282	45	310	8
9	9	0.00	5376	497	45	510	8
21	1	0.00	5392	34	87	191	10
21	2	0.00	5376	238	126	1022	16

n	k	time	mem4	# res	d-sub	states proc.	p. # states
21	4	0.00	5408	5822	191	17224	19
21	9	0.44	5424	652033	231	842324	20
21	21	1.12	5472	2097113	231	2097150	20
49	1	0.00	5424	40	230	1430	24
49	2	0.00	5456	486	346	11271	29
49	4	0.16	5504	42081	554	443796	36
49	9	434.85	5616	199579965	944	1224051456	47
114	1	0.00	5520	12	572	4139	26
114	2	0.01	5568	96	843	37731	31
114	4	0.75	5680	6576	1398	2013024	41
114	9	6729.00	5936	233095011	2619	2850072540	66
266	1	0.01	5760	2	1104	30146	36
266	2	0.43	5872	10	1692	860695	41
266	4	153.80	6112	6811	2901	320648687	54
620	1	0.02	6192	1	1876	38260	36
620	2	0.54	6416	4	3038	1007793	41
620	4	174.64	6928	2700	5419	352723997	54
1446	1	0.04	7120	1	3206	54493	36
1446	2	0.79	7584	4	5421	1310776	41
1446	4	229.92	8592	853	9937	421813337	54
3374	1	0.08	9312	1	6470	95233	36
3374	2	1.50	10320	1	11375	2118277	41
3374	4	368.92	12592	685	21079	620733183	54
7872	1	0.19	14544	1	15349	197778	36
7872	2	3.50	16928	1	26730	4286741	41
7872	4	824.83	22240	276	49598	1208369035	54
18368	1	0.51	26720	1	35761	455327	36
18368	2	9.69	32224	1	62323	10021221	41
18368	4	2298.03	44688	221	115975	2908354181	54
42858	1	1.60	55248	1	84316	1102128	37
42858	2	29.98	68256	1	146903	25236612	41
42858	4	7113.99	97408	197	272961	3541742876	54
100002	1	4.92	121392	1	198078	2694970	37
100002	2	92.48	151856	1	345276	64286286	41
100002	4	21881.98	219440	191	640384	4189713029	54
233338	1	28.16	276096	1	455118	40240736	356
233338	2	2946.28	346848	1	795429	2444805448	500
544455	1	10137.50	642688	1	1072362	1588774481	6258

Table B.3: Experimental run of computing restricted k-approximate seeds of a string

n	k	time	mem4	# res	d-sub	states proc.	p. # states
1	1	0.00	5376	1	1	1	1
2	1	0.00	5392	1	4	3	2
2	2	0.00	5392	1	4	3	2
4	1	0.00	5392	5	4294967294	7	2
4	2	0.00	5376	5	4294967294	7	2
4	4	0.00	5376	5	4294967294	7	2
9	1	0.00	5424	20	4294967295	29	5
9	2	0.00	5408	27	4294967294	29	5
9	4	0.00	5424	27	4294967291	29	5
9	9	0.00	5456	27	4294967295	29	5
21	1	0.00	5616	32	4294967285	143	11
21	2	0.00	5616	56	4294967295	143	11
21	4	0.00	5648	120	4294967295	143	11
21	9	0.00	5680	141	4294967288	143	11
21	21	0.00	5712	141	4294967288	143	11
49	1	0.00	6832	30	4294967287	831	32
49	2	0.00	6816	71	4294967258	831	32
49	4	0.00	6848	203	4294967289	831	32
49	9	0.00	6848	553	4294967260	831	32
49	21	0.01	6960	829	4294967260	831	32
49	49	0.01	7136	829	4294967260	831	32
114	1	0.03	14112	27	4294967289	5260	93
114	2	0.04	14096	73	4294967183	5260	93
114	4	0.04	14064	198	4294967294	5260	93
114	9	0.06	13936	1200	4294967195	5260	93
114	21	0.11	13680	3315	4294967195	5260	93
114	49	0.16	14080	5258	4294967195	5260	93
114	114	0.16	15088	5258	4294967195	5260	93
266	1	0.54	75600	7	4294967257	32819	233
266	2	0.56	75552	29	4294967221	32819	233
266	4	0.60	75440	342	4294966533	32819	233
266	9	0.79	74992	1768	4294966533	32819	233
266	21	1.40	73920	6562	4294966533	32819	233
266	49	2.68	72800	15914	4294966533	32819	233
266	114	4.13	73328	31864	4294966533	32819	233
266	266	4.31	78656	32765	4294966533	32819	233
620	1	10.85	406048	7	4294967274	188075	587
620	2	10.93	405808	18	4294967142	188075	587
620	4	11.27	405440	483	4294966860	188075	587

n	k	time	mem4	# res	d-sub	states proc.	p. # states
620	9	12.68	404272	4354	4294966810	188075	587
620	21	16.72	401488	15078	4294966810	188075	587
620	49	25.93	396368	38959	4294966810	188075	587
620	114	47.89	388944	88024	4294966810	188075	587
620	266	73.61	387792	164381	4294966810	188075	587
620	620	79.56	413408	187894	4294966810	188075	587
1446	1	147.13	2213120	5	4294967287	1037620	1413
1446	2	154.25	2212464	9	4294967242	1037620	1413
1446	4	150.36	2211392	398	4294966720	1037620	1413
1446	9	160.28	2208944	9497	4294963462	1037620	1413
1446	21	184.62	2202720	33760	4294963462	1037620	1413
1446	49	247.33	2189248	88275	4294963462	1037620	1413
1446	114	423.42	2163632	206858	4294963462	1037620	1413
1446	266	817.31	2121856	454283	4294963462	1037620	1413
1446	620	1422.73	2094576	873822	4294963462	1037620	1413
1446	1446	1568.28	2216080	1036951	4294963462	1037620	1413
3374	1	1998.85	12072096	6	4294967274	5674323	3341
3374	2	2001.72	12070368	8	4294967101	5674323	3341
3374	4	2027.52	12067840	403	4294966224	5674323	3341
3374	9	2164.07	12068432	19834	4294955118	5674323	3341
3374	21	2281.86	12046784	76638	4294955118	5674323	3341
3374	49	2738.35	12017056	203218	4294955118	5674323	3341
3374	114	3995.05	11944176	491773	4294962103	5674323	3341
3374	266	7487.34	11793984	1137943	4294964842	5674323	3341
3374	620	15862.71	11548688	2472090	4294966968	5674323	3341
3374	1446	29825.05	11326432	4694896	4294966968	5674323	3341
3374	3374	34478.91	11903536	5672715	4294966968	5674323	3341

Table B.4: Experimental run of computing k-approximate seeds of a string

n	k	time	mem4	# res	d-sub	states proc.	p. # states
1	1	0.00	5392	1	1	1	1
2	1	0.00	5376	3	7	5	3
2	2	0.00	5392	3	7	6	4
4	1	0.00	5392	17	28	19	8
4	2	0.00	5408	23	31	25	11
4	4	0.00	5424	27	31	30	16
9	1	0.00	5568	89	226	123	26
9	2	0.00	5776	285	532	312	77
9	4	0.00	6336	763	1023	766	291
9	9	0.00	6640	1018	1023	1022	512

n	k	time	mem4	# res	d-sub	states proc.	p. # states
21	1	0.00	7328	215	2910	1477	135
21	2	0.02	15360	2165	16584	8524	871
21	4	0.33	132592	56521	210036	112565	17089
21	9	8.19	2811360	2449052	3590309	2491013	820205
21	21	14.36	4730048	4194298	4194303	4194302	2097152
49	1	0.08	27616	758	37114	18607	657
49	2	1.30	288656	10550	497448	249950	9794
114	1	2.66	291504	917	494263	247247	3401
114	2	82.19	8266608	57821	14644579	7328846	115788
266	1	250.40	15488224	3191	37884389	9471896	53980