Minimized Compact Automaton for Clumps over Degenerate Patterns

Eugenia Furletova, Jan Holub, Mireille Regnier

Minimized Compact Automaton for Clumps over Degenerate Patterns

E. Furletova[1] ⋆ and J. Holub[2⋆⋆] and M. Régnier[3,4⋆⋆⋆]

[1] Institute of Mathematical Problems of Biology, 142290, Institutskaya, 4, Pushchino, Russia, email: Furletova@lpm.org.ru
[2] FIT, Czech Technical University in Prague, Thákurova 2700/9, Czech Republic, email: Jan.Holub@fit.cvut.cz
[3] CNRS, Ecole polytechnique, LIX, Palaiseau 91128, France, email: Mireille.Regnier@inria.fr
[4] INRIA

Abstract. Clumps are sequences of overlapping occurrences of a given pattern that play a vital role in the study of distribution of pattern occurrences. These distributions are sused for finding functional fragments in biological sequences. In this paper we present a minimized compacted automaton (Overlap walking automaton, *OWA*) recognizing all the possible clumps for degenerate patterns and its usage for computation of probabilities of sets of clumps. We also present Aho-Corasick like automaton, *RMinPatAut*, recognizing all the sequences ending with pattern occurrences. The states of *RMinPatAut* are equivalence classes on the prefixes of the pattern words. We have proved that *RMinPatAut* is Nerode-minimal, i.e., minimal in classical sense. We use *RMinPatAut* as an auxiliary structure for *OWA* construction. For degenerate pattern, *RMinPatAut* can be constructed in linear time on the number of its states (it is bounded by 2^m, where m the length of pattern words). *OWA* can be constructed in linear time on the sum of its size and *RMinPatAut* size.

Keywords : clump; degenerate patterns; overlap automaton; minimal automaton; pvalue; pattern occurrences.

1 Introduction

Computational biology applications motivate abundant research on motif detection techniques and statistical analysis. In the past decade, this was enhanced by the arising of so-called High Throughput and Next Generation Sequencing. Beyond classical measures of overrepresentation of motifs such as z-scores and p-values or probability weight matrices [Sto00], the knowledge of the pattern occurrences distribution allows for a more precise detection of genomic signals. It is tightly related to the combinatorics of *clumps*, sequences consisting of overlapping occurrences of a set of words, called the *pattern*. For instance, they were shown to play a fundamental role in the Chen-Stein method of compound Poisson approximations [RS98]. Basic properties of clumps are given in [SRS07,BCFN08] and formulas that compute pattern occurrences p-value from the generating functions of clumps are provided in [RFI14]. Usually this formulas give high precision, i.e., in more than 10

⋆ Evgeniia Furletova was supported by the grants 14-04-32220-mol_a and 14-01-93106-NCNILa from RFBR and Metchnikov fellowship from Campus France.
⋆⋆ This research has been partially supported by the OP VVV MEYS funded project $CZ.02.1.01/0.0/0.0/16_019/0000765$ "Research Center for Informatics".
⋆⋆⋆ Mireille Régnier was supported by Digiteo grant RNAomics and Inria-Cnrs grant CARNAGE

digits after decimal point, see [RFI14]. The main goal of our study is to find a fast efficient method for computation of generating functions coefficients.

In this paper we consider patterns defined by degenerate (indeterminate) strings. A degenerate symbol over an alphabet Σ is a non-empty subset of Σ, and a degenerate string is a sequence of such symbols. Degenerate strings arise in Molecular Biology, Musicology and Cryptography. It is one of the most widely used representation of significant fragments in biological sequences [Sto00]. Pattern matching problems related to degenerate strings are of current interest [HS03,CIK$^+$16]. In particular, in paper [IKM16] a problem of finding clumps over degenerate patterns in strings was considered.

The aim of our study is to construct an efficient algorithm for computation of probability of set of clumps $C(n, k)$ of length n that consist of k overlapping pattern occurrences. Most efficient approaches to this problem are based on specific automata and some of them use Aho-Corasick (pattern matching) automaton and its minimization, see [BCR$^+$07,BCFN08,RFYR14]. Recall that Aho-Corasick automaton [AC75] recognizes texts $\Sigma^* \cdot \mathcal{H}$, where Σ is an alphabet, $\mathcal{H}$ is a pattern. We also use automata approach. We propose Overlap walking automaton (OWA) that recognizes all possible clumps. The probabilities of the sets of clumps are computed by traversals of OWA. This idea comes from assembling, our automaton is an analogue of overlap graph [MKS10]. Traversal of overlap graph returns a sequence consisting of overlapping words. The similar idea is also used for searching of shortest superstring [BJL$^+$93]. The states of OWA are equivalence classes on the states of automaton *AutoClump* proposed in [RFI14]. To construct OWA we use auxiliary automaton *RMinPatAut*, the states of OWA are the final states of *RMinPatAut*.

In this study *RMinPatAut* plays an auxiliary role. However, it is of interest in itself. We propose an equivalence relation (R-equivalence) on prefixes of pattern words. It is shown that for degenerate patterns, R-equivalent prefixes are also Nerode-equivalent, i.e., they are equivalent in classical sense. *RMinPatAut* (R-minimal pattern matching automaton) is the minimal automaton according to this relation that recognizes texts $\Sigma^* \cdot \mathcal{H}$, i.e., its states are equivalence classes on the states of Aho-Corasick automaton. For degenerate patterns this automaton can be constructed with time and space complexities proportional to its size. Patterns describing biological objects can reach huge sizes. For example, degenerate string FLXXTXXXRXXXAXXQXXXLXXF (symbols except of X stand for specific amino acids, X stands for any amino acid) describing protein familia GAS_VESICLE_C from the database Prosite [SCea12] corresponds to a pattern containing about 10^{29} words. Straightforward enumeration of the words and construction of corresponding Aho-Corasick automaton are impossible. *RMinPatAut* for this pattern has only 1 480 states and 4 900 edges. Its construction takes several seconds using a common personal computer. The problem of Aho-Corasick automaton minimization in linear time on its size is of current interest in literature [ABN12] due to its application for different pattern matching problems [SH13]. In present, the problem is not solved for general patterns. Some papers propose pseudo-minimal automata that can be constructed in linear or sub-linear time [ABN12,KNR07]. We have proved that for degenerate patterns, *RMinPatAut* is also Nerode-minimal.

This paper is organized as follows. Section 2 is devoted to the statement of the problem. In Section 3 the general definitions are given. Section 4 is related to R-equivalence relation and $RMinPatAut$ construction. Section 5 is related to clumps recognizing automaton OWA. In Section 6 we describe an algorithm for computing probabilities of clumps sets of given cardinalities and lengths, for Bernoulli models. The algorithm computes the probabilities by a traversal of OWA. It can be easily extended for Markov and hidden Markov models taking into account the specificities of the models. In Section 7, the results of computer experiments on sizes of proposed automata are given.

2 Background

Σ^k represents all strings of length k over a given alphabet Σ. Σ^* represents all strings over Σ. A word v in Σ^* is a factor of w in Σ^* if exist two words u and u' in Σ^* such that $w = uvu'$. Moreover, v is a prefix of w if $u = \varepsilon$ and v is a suffix of w if $u' = \varepsilon$. Any prefix (resp. suffix) v of w such that $v \neq w$ is a proper prefix (resp. suffix). Observe that ε is both a proper prefix and suffix of any word. The sets of prefixes (proper prefixes) and suffixes (proper suffixes) of a word w are denoted as $Pref(w)$ ($PPref(w)$) and $Suff(w)$ ($PSuff(w)$). One notes $Suff_k(w) = Suff(w) \cap \Sigma^k$. For a set S of strings of the same length we define $||S||$ to be the cardinality of the set (number of strings) and $|S|$ to be the common length of strings in S; $|S|$ is called length of S.

Pattern $\mathcal{H}$ is a set of words in Σ of the same length m; $Pref(\mathcal{H})$ ($PPref(\mathcal{H})$) is the set of (proper) prefixes of the words from $\mathcal{H}$.

Definition 1. *Given a word $w = xu$, where $w, x, u \in \Sigma^*$, string u is called the extension of x into w (denoted as $u = x^{-1}.w$).*

Definition 2. *Given two words h and f in pattern $\mathcal{H}$,*

$$\mathcal{A}_{h,f} = \{w^{-1}.f \mid w \text{ is a suffix of } h \text{ and } w \text{ is a prefix of } f; w \neq \varepsilon\}$$

A word in $\mathcal{A}_{h,f}$ is called a (h, f)-correlation.

Definition 3. *A clump c is a word of the form $w_0.w_1 \dots w_p$, where $p \geq 0$ and $w_0 = h_0 \in \mathcal{H}$, such that there exist $h_1, \dots, h_p \in \mathcal{H}$ that satisfy $w_i \in \mathcal{A}_{h_{i-1},h_i} \setminus \{\varepsilon\}$ and $|w_i| < m$, for all $i \in \{1, \dots, p\}$. The ordered list $s = \{w_0, w_1, \dots, w_p\}$ is called a clump decomposition for c; p is called the cardinality of decomposition s of c and denoted $card(s)$.*

Observe that there can be more than one decompositions for a given clump c. Also,the cardinality of a clump decomposition represents the number of strings extending the first pattern string h_0.

Example 1. Let $\mathcal{H} = \{\texttt{acatata}, \texttt{atacaca}, \texttt{atagata}\}$. The word `atagata.caca.catata` is a clump. The ordered list $\{w_0, w_1, w_2\}$ where $w_0 =$ `atagata`, $w_1 =$ `caca` and $w_2 =$ `catata` is a decomposition with cardinality 2.

Combinatorics of clumps has application to the pattern occurrences p-value computation. p-value is the probability to find at least s occurrences of pattern $\mathcal{H}$ in a random text of length n. It was shown in [RFI14] that p-value is totally determined by generating function of clumps, see Proposition 1 below.

Definition 4. *Let c be a clump with length $|c|$ and probability $Prob(c)$. Let z denote some formal variable. The sign generating function of clumps is*

$$G(z) = \sum_c \left(Prob(c) \cdot z^{|c|} \sum_s (-1)^{card(s)} \right) \ , \tag{1}$$

where c ranges over all clumps and s ranges over all possible decompositions of c.

Proposition 1. *(Regnier, 2014) The p-value of one occurrence of $\mathcal{H}$ in a random text of length n can be approximated by*

$$p\text{-}value \ \approx 1 - \frac{1}{\rho(1 - G'(\rho))} \cdot \rho^{-n}, \tag{2}$$

where ρ is the root of $1 - z + G(z) = 0$ closest to 1.

The function $G(z)$ can be approximated by the formula

$$G(z) \approx \sum_{n \le N} z^n \sum_{k \le K} (-1)^k Prob(C(n,k)),$$

where $C(n,k)$ is the set of clumps of length n and cardinality k; N, K are integer numbers.

In this paper we propose an efficient method to compute $Prob(C(n,k))$, see Section 6. We propose equivalence relation on clumps from $C(n,k)$ that allows to compute immediately probabilities of classes avoiding enumeration of all possible clumps. The probabilities are computed by traversal of clumps recognizing automaton OWA, see Sections 5 and 6.

3 Preliminaries

3.1 Pattern matching automaton

Let $L_{\mathcal{H}}$ is the set of all strings having suffix in $\mathcal{H}$, i.e. $L_{\mathcal{H}} = \Sigma^*.\mathcal{H}$. Language $L_{\mathcal{H}}$ actually represents a pattern matching task where a pattern (a string from $\mathcal{H}$) is found in input text and reported at the position of the last symbol of the pattern. Suffix link $s\ell(x)$ of $x \in \Sigma^* \backslash \{\varepsilon\}$ is the longest proper suffix of x that is a proper prefix of a word from $\mathcal{H}$, i.e., $s\ell(x) = \max\{y \mid y \in PSuff(x) \cap PPref(\mathcal{H})\}$. Denote, $s\ell^{(0)}(x) = x$ and $s\ell^{(i)}(x) = s\ell(s\ell^{(i-1)}(x))$. We also define non-proper suffix link $ns\ell(x) = \max\{y \mid y \in Suff(x) \cap Pref(\mathcal{H})\}$ which considers also non-proper suffix and prefix. It actually cuts out the longest prefix of patterns in $\mathcal{H}$ found in text x, so it transforms a word from $\Sigma^*.Pref(\mathcal{H})$ to a word from $Pref(\mathcal{H})$. When $ns\ell(x)$ is applied to any word $x \in Pref(\mathcal{H})$ it returns x.

Definition 5. *Pattern matching automaton* $PatAut(\mathcal{H}) = (Q, \Sigma, \delta, s\ell, q_0, F)$ *is a deterministic finite automaton with suffix links which accepts language* $L_{\mathcal{H}}$*, where* Q *is the set of states;* δ *is mapping* $Q \times \Sigma \to Q$*;* $s\ell$ *is mapping* $Q \to Q$*;* q_0 *is the initial state;* F *is the set of terminal states.*

The automaton $PatAut(\mathcal{H})$ can be constructed as follows, see [CHL07]:

- $Q = Pref(\mathcal{H})$;
- For any $q \in Q$ and $a \in \Sigma$,
$$\delta(q,a) = \begin{cases} qa, & \text{if } qa \in Pref(\mathcal{H}), \\ \varepsilon, & \text{if } q = \varepsilon \text{ and } a \notin Pref(\mathcal{H}), \\ \delta(s\ell(q), a), & \text{otherwise} \end{cases}$$
- $q_0 = \varepsilon$;
- $F = Pref(\mathcal{H}) \cap \Sigma^*.\mathcal{H}$.

We extend mapping δ to work with strings instead of just symbols of alphabet as follows: $\hat{\delta}(q, \varepsilon) = q$ and $\forall a \in \Sigma, u \in \Sigma^*, \hat{\delta}(q, ua) = \delta(\hat{\delta}(q, u), a)$.

The automaton $PatAut(\mathcal{H})$ can be constructed in linear time in $Q \times \Sigma$.

Here we consider the case where all words in $\mathcal{H}$ have the same length m. In this case $F = \mathcal{H}$.

Definition 6. *[Ner58,HU79,Fle05] Given a language* $L \subseteq \Sigma^*$*, strings* $x, y \in \Sigma^*$ *are Nerode-equivalent (with respect to* L*)* ($x \equiv_L y$) *if*

$$\forall t \in \Sigma^* : xt \in L \Leftrightarrow yt \in L.$$

3.2 Degenerate strings

Definition 7 (Degenerate symbol). *Consider an extended alphabet* $\Sigma' = 2^{\Sigma} \setminus \{\varepsilon\}$*. Any symbol* $\alpha \in \Sigma', ||\alpha|| > 1$ *is called degenerate.*

Example 2. Given $\Sigma = \{\mathtt{a}, \mathtt{c}, \mathtt{g}, \mathtt{t}\}$. $\Sigma' = \{\{\mathtt{a}\}, \{\mathtt{c}\}, \{\mathtt{g}\}, \{\mathtt{t}\}, \{\mathtt{a}, \mathtt{c}\}, \{\mathtt{a}, \mathtt{g}\}, \{\mathtt{a}, \mathtt{t}\}, \{\mathtt{c}, \mathtt{g}\}, \{\mathtt{c}, \mathtt{t}\}, \{\mathtt{g}, \mathtt{t}\}, \{\mathtt{a}, \mathtt{c}, \mathtt{g}\}, \{\mathtt{a}, \mathtt{c}, \mathtt{t}\}, \{\mathtt{a}, \mathtt{g}, \mathtt{t}\}, \{\mathtt{c}, \mathtt{g}, \mathtt{t}\}, \{\mathtt{a}, \mathtt{c}, \mathtt{g}, \mathtt{t}\}\}$.

$\{\mathtt{a}, \mathtt{c}\}$ is a degenerate symbol, $\{\mathtt{a}\}$ is not a degenerate symbol.

Definition 8 (Degenerate string). *Any string containing at least one degenerate symbol is called degenerate. Given a degenerate string* $\omega = \omega[1..n] \in \Sigma'^n$*, value* $v(\omega)$ *of degenerate string* ω *is defined as* $v(\omega) = \{x \in \Sigma^n \mid x[i] \in \omega[i], \forall i, 1 \leq i \leq n\}$.

Example 3. Given $\omega = \{\mathtt{c}, \mathtt{t}\}\{\mathtt{a}\}\{\mathtt{g}, \mathtt{t}\}$. $v(\omega) = \{\mathtt{cag}, \mathtt{cat}, \mathtt{tag}, \mathtt{tat}\}$.

Below we use Greek symbols to denote degenerate symbols and degenerate strings. We use Latin symbols to denote words in Σ^*.

We also denote degenerate string describing pattern $\mathcal{H}$ by π, i.e., $\mathcal{H} = v(\pi)$. Observe that, for a degenerate string ω, the words from $v(\omega)$ have the same length that is denoted by $|v(\omega)|$ (length of $v(\omega)$).

4 Pattern matching automaton minimization

In this section we define an equivalence relation (R-equivalence) on words from $Pref(\mathcal{H})$ for degenerate patterns describing $\mathcal{H}$. We prove that R-equivalent words are also Nerode-equivalent. Based on R-equivalence relation we propose an algorithm of pattern matching automaton minimization. The time and space complexities of the algorithm are linear on the number of states of the minimal automaton (*RMinPatAut*). The number of states of *RMinPatAut* is bounded by 2^m.

4.1 *R*-equivalence relation

Let π be a degenerate pattern and $\mathcal{H} = v(\pi)$.

Definition 9 ($\sim^R$ equivalency). *Let $\pi \in \Sigma'^*$ and $\mathcal{H} = v(\pi)$. Two words $x, y \in Pref(\mathcal{H})$ are R-equivalent with respect to $L_{\mathcal{H}}$ (denoted as $\sim^R$) if $x = y = \varepsilon$ or:*

$$|x| \;=\; |y| \text{ and} \tag{3}$$
$$s\ell(x) \sim^R s\ell(y) \;. \tag{4}$$

Denote by $\Re(\mathcal{H})$ the partition of $Pref(\mathcal{H})$ into the set of R-equivalence classes.

Example 4. Degenerate string $\pi = \{\mathtt{c},\mathtt{t}\}\{\mathtt{a}\}\{\mathtt{a},\mathtt{c},\mathtt{g},\mathtt{t}\}$ defines a pattern $\mathcal{H} = v(\pi)$. Consider words $\mathtt{caa}, \mathtt{cac}$ and $\mathtt{cat}$ from $\mathcal{H}$. By definition of suffix link, $s\ell(\mathtt{caa}) = \varepsilon$, $s\ell(\mathtt{cac}) = \mathtt{c}$ and $s\ell(\mathtt{cat}) = \mathtt{t}$. Note that $s\ell(\mathtt{c}) = s\ell(\mathtt{t}) = \varepsilon$. Therefore $\mathtt{c} \sim^R \mathtt{t}$ and $\mathtt{cac} \sim^R \mathtt{cat}$. However, the word $\mathtt{caa}$ is not R-equivalent to $\mathtt{cac}$ and $\mathtt{cat}$. Finally,

$$\Re(\mathcal{H}) = \{\{\varepsilon\}, \{\mathtt{c},\mathtt{t}\}, \{\mathtt{ca},\mathtt{ta}\}, \{\mathtt{caa},\mathtt{cag},\mathtt{taa},\mathtt{tag}\}, \{\mathtt{cac},\mathtt{cat},\mathtt{tac},\mathtt{tat}\}\}.$$

Proposition 2. *The number of R-equivalence classes of length k, $k \geq 1$, is bounded by 2^{k-1}. $||\Re(\mathcal{H})|| \leq 2^m$, where $m = |\mathcal{H}|$.*

Proof. It can be proved by induction on lengths of classes. There is only one class of length 0 and one class of length 1. Let the proposition be true for classes of length $k-1$, i.e., the number of classes of length $k-1$ is bounded by 2^{k-2}. Then the total number of classes of lengths smaller than or equal to $k-1$ is bounded by 2^{k-1}. By Definition 9, suffix links of words from the same class are also R-equivalent. Hence classes of length k are in one to one correspondence with the classes of suffix links of their words. Then the number of classes of length k is bounded by the total number of classes of smaller lengths, i.e., 2^{k-1}. This leads to the indicated bounds. □

The proposed boundary is reachable. For example, for the pattern represented by degenerate string $a \cdot \{a,b\}^{m-1}$, where $a, b \in \Sigma$; $a \neq b$ and $||\Sigma|| \geq 2$, the number of R-classes is 2^m. Note, this boundary is independent of the cardinality of the alphabet.

Proposition 3. *Let $\pi \in \Sigma'^*$ and $\mathcal{H} = v(\pi)$. Let x, y be in $Pref(\mathcal{H})$ and $|x| = |y|$. Then*

$$\forall t \in \Sigma^* : xt \in Pref(\mathcal{H}) \Leftrightarrow yt \in Pref(\mathcal{H}) \;.$$

Proof. Let $\pi = \alpha_1 \dots \alpha_m$. As $x, y \in \mathit{Pref}(\mathcal{H})$ and $|x| = |y|$ we have to check only if $t \in v(\alpha_{|x|+1}\alpha_{|x|+2} \dots \alpha_{|x|+|t|})$. □

Proposition 4. *Let $x, y \in \mathit{Pref}(\mathcal{H})$ and $x \sim^R y$. Let k be an integer such that $s\ell^{(k)}(x) = \varepsilon$. Then $s\ell^{(i)}(x) \sim^R s\ell^{(i)}(y), \forall i, 0 < i \leq k$.*

Proof. It follows directly from the definition of R-equivalency. □

Theorem 1. *Let $\pi \in \Sigma'^*$ and $x, y \in \mathit{Pref}(v(\pi))$ such that $x \sim^R y$. Then $\forall a \in \Sigma$*

1. *$s\ell(xa) \sim^R s\ell(ya)$;*
2. *$xa \sim^R ya$, if $xa, ya \in \mathit{Pref}(v(\pi))$.*

Proof. The first assertion is proven by induction on $|x|$ ($|x| = |y|$).

Base of induction. Let $x = y = \varepsilon$. Then $\forall a \in \Sigma, xa = ya = a$ and $s\ell(xa) = s\ell(ya) = \varepsilon$.

Induction hypothesis. Let the theorem be true for all $x, y \in \mathit{Pref}(v(\pi))$ such that $x \sim^R y$ and $|x|, |y| < k$.

Step of induction. We will prove the theorem for x and y such that $|x| = |y| = k$.

Consider the first assertion. It holds that $s\ell^{(i)}(x) \sim^R s\ell^{(i)}(y), \forall i, 0 < i \leq \ell$, such that $s\ell^{(\ell)}(x) = \varepsilon$ (see Proposition 4). Therefore $|s\ell^{(i)}(x)| = |s\ell^{(i)}(y)|$ and symbol a in both $s\ell^{(i)}(x)a$ and $s\ell^{(i)}(y)a$ matches or mismatches the same degenerate symbol $\pi[|s\ell^{(i)}(x)a|]$. Note, $s\ell(xa)$ equals either (a) to $s\ell^{(i)}(x)a$ if $s\ell^{(i)}(x)a \in \mathit{Pref}(v(\pi))$ for some $0 < i \leq \ell$, or (b) to ε if no such i exists.

Consider case (a): The above implies

$$s\ell^{(i)}(x)a \in \mathit{Pref}(v(\pi)) \Leftrightarrow s\ell^{(i)}(y)a \in \mathit{Pref}(v(\pi)) \ . \tag{5}$$

To prove that $s\ell(xa) \sim^R s\ell(ya)$ we have to show that $|s\ell^{(i)}(x)a| = |s\ell^{(i)}(y)a|$ (condition (3)) and $s\ell(s\ell^{(i)}(x)a) \sim^R s\ell(s\ell^{(i)}(y)a)$ (condition (4)). $|s\ell^{(i)}(x)a| = |s\ell^{(i)}(y)a|$ holds as $|s\ell^{(i)}(x)| = |s\ell^{(i)}(y)|$ and $a \in \Sigma$.

$x \sim^R y$ implies that $s\ell^{(i)}(x) \sim^R s\ell^{(i)}(y)$. Then by the induction hypothesis, (both $|s\ell^{(i)}(x)|$ and $|s\ell^{(i)}(y)|$ are shorter than k) $s\ell^{(i)}(x) \sim^R s\ell^{(i)}(y)$ implies $s\ell(s\ell^{(i)}(x)a) \sim^R s\ell(s\ell^{(i)}(y)a)$. Because of (5) it also implies $s\ell(s\ell(xa)) \sim^R s\ell(s\ell(ya))$.

Consider case (b): As $s\ell(xa) = \varepsilon$ also $s\ell(ya) = \varepsilon$ and the first assertion (i.e., $\varepsilon \sim^R \varepsilon$) obviously holds.

The second assertion directly follows from the first one and from the definition of R-equivalency, because $|xa| = |ya|$. □

Corollary 1. *Let $x, y \in \mathit{Pref}(\mathcal{H})$ and $x \sim^R y$. Then*

$$\forall t \in \Sigma^*, \hat{\delta}(x, t) \sim^R \hat{\delta}(y, t) \text{ in } \mathit{PatAut}(\mathcal{H}).$$

Proof. Proof follows from Definition 5, definition of $\hat{\delta}$, and Theorem 1. □

Theorem 1 leads to an efficient method of R-equivalence classes creation.

4.2 R'-equivalency

R-equivalency is defined for prefixes of pattern words only. Here we extend the equivalence relation for any strings from Σ^*.

Proposition 5. *Let $\pi \in \Sigma'^*$ and $\mathcal{H} = v(\pi)$. $\forall x \in \Sigma^*, x \equiv_{L_\mathcal{H}} ns\ell(x)$.*

Proof. By definition of $\equiv_{L_\mathcal{H}}$, the assertion $x \equiv_{L_\mathcal{H}} ns\ell(x)$ rewrites

$$\forall t \in \Sigma^* : xt \in L_\mathcal{H} \Leftrightarrow ns\ell(x)t \in L_\mathcal{H}. \tag{6}$$

a) If $|t| \geq |\pi|$, then Formula (6) holds as $L_\mathcal{H} = \Sigma^*.\mathcal{H}$ and both strings xt and $ns\ell(x)t$ share the same suffix of length greater or equal to the length of strings in $\mathcal{H}$.
b) If $|t| < |\pi|$, then we denote by ut the suffix of xt of lenght $|\pi|$ (u is a suffix of x). If $ut \in \mathcal{H}$ then $u \in$ *Pref*$(\mathcal{H})$. $ns\ell(x)$ returns the longest suffix of x that is in *Pref*$(\mathcal{H})$. Therefore $u = ns\ell(x)$ or $u \in$ *Suff*$(ns\ell(x))$. □

Corollary 2. *Let $\pi \in \Sigma'^*$ and $\mathcal{H} = v(\pi)$. $\forall x, y \in \Sigma^*, x \equiv_{L_\mathcal{H}} y$ iff $ns\ell(x) \equiv_{L_\mathcal{H}} ns\ell(y)$.*

As a consequence of Proposition 5 we can extend definition of R-equivalency to all strings over a given alphabet.

Definition 10 ($\sim^{R'}$ equivalency). *Let $\pi \in \Sigma'^*$ and $\mathcal{H} = v(\pi)$. Two words $x, y \in \Sigma^*$ are R'-equivalent with respect to $L_\mathcal{H}$ (denoted as $\sim^{R'}$) if :*

$$ns\ell(x) \sim^R ns\ell(y). \tag{7}$$

Denote by $\Re'(\mathcal{H})$ the partition of Pref$(\mathcal{H})$ into the set of R'-equivalence classes.

Proposition 6. *Let $\pi \in \Sigma'^*$ and $\mathcal{H} = v(\pi)$. $\Re(\mathcal{H}) = \Re'(\mathcal{H})$.*

Proof. The proof directly follows from Proposition 5. □

4.3 R-equivalency and Nerode-equivalency

We will prove that for values of degenerate pattern, two prefixes of $\mathcal{H}$ are R-equivalent iff they are Nerode-equivalent.

Lemma 1. *Let $x, y \in$ Pref$(\mathcal{H})$ such that $x \equiv_{L_\mathcal{H}} y$. Then $|x| = |y|$.*

Proof. Assume w.l.o.g. that $|x| > |y|$. Then $\forall t \in \Sigma^*, xt \in \mathcal{H} \subset L_\mathcal{H}$, $|yt| < |xt| = m$ and $yt \notin L_\mathcal{H}$ as the shortest string of $L_\mathcal{H}$ has length equal to m. This contradicts the condition. □

Lemma 2. *Let $x, y \in$ Pref$(\mathcal{H})$ such that $x \equiv_{L_\mathcal{H}} y$. Then $s\ell(x) \equiv_{L_\mathcal{H}} s\ell(y)$.*

Proof. Assume that $s\ell(x)$ is not equivalent to $s\ell(y)$. Then there exists $t \in \Sigma^*$, $|t| < m$, such that $s\ell(x)t$ belongs to $L_{\mathcal{H}}$ (i.e., $s\ell(x)t$ has a suffix in $\mathcal{H}$) and $s\ell(y)t$ does not belong to $L_{\mathcal{H}}$. But both xt ($xt = x's\ell(x)t$, $x' \neq \varepsilon$) and yt ($yt = y's\ell(y)t$, $y' \neq \varepsilon$) are in $L_{\mathcal{H}}$. Then yt has suffix $h \in \mathcal{H}$ and $s\ell(y)t$ has no suffix from $\mathcal{H}$. It means that $s\ell(y)t$ is a proper suffix of h. Therefore $yt = h$, otherwise suffix h in yt overlaps y with a word w, where $w = s\ell^{(i)}(y)$, $i \geq 1$, and $|w| > |s\ell(y)|$, that contradicts to the definition of suffix link. Note, $|xt| > m$ because $s\ell(x)t$ ends with a word from $\mathcal{H}$ of length m. By Lemma 1, $|xt| = |yt| > m$. Therefore yt can not be equal to any word from $\mathcal{H}$. Thus yt does not belong to $L_{\mathcal{H}}$, and y is not equivalent to x. This contradicts the condition of the proposition. □

Corollary 3. *Let $x, y \in Pref(\mathcal{H})$ such that $x \equiv_{L_{\mathcal{H}}} y$. Then $s\ell^{(i)}(x) \equiv_{L_{\mathcal{H}}} s\ell^{(i)}(y), \forall i, 0 < i \leq \ell$, such that $s\ell^{(\ell)}(x) = \varepsilon$.*

Theorem 2. *Let $x, y \in Pref(\mathcal{H})$. Then $x \equiv_{L_{\mathcal{H}}} y \Leftrightarrow x \sim^R y$.*

Proof. I. Let $x \equiv_{L_{\mathcal{H}}} y$. It follows from Lemmas 1 and 2 that $x \sim^R y$.

II. Let $x \sim^R y$. We will prove that $x \equiv_{L_{\mathcal{H}}} y$. Let $xt \in L_{\mathcal{H}}, t \in \Sigma^*$. By construction of $PatAut(\mathcal{H})$, $xt \in L_{\mathcal{H}}$ iff $\hat{\delta}(x,t) \in \mathcal{H}$. By Corollary 1, $\hat{\delta}(x,t) \sim^R \hat{\delta}(y,t)$. Then $\hat{\delta}(y,t) \in \mathcal{H}$ and $yt \in L_{\mathcal{H}}$. □

Corollary 4. *Let $x, y \in \Sigma^*$. Then $x \equiv_{L_{\mathcal{H}}} y \Leftrightarrow x \sim^{R'} y$.*

Proof. By Corollary 2, $x \equiv_{L_{\mathcal{H}}} y \Leftrightarrow ns\ell(x) \equiv_{L_{\mathcal{H}}} ns\ell(y)$. As $ns\ell(x), ns\ell(y) \in Pref(\mathcal{H})$ then by Theorem 2, $ns\ell(x) \equiv_{L_{\mathcal{H}}} ns\ell(y) \Leftrightarrow ns\ell(x) \sim^R ns\ell(y)$. And by definition of R'-equivalence, $ns\ell(x) \sim^R ns\ell(y) \Leftrightarrow x \sim^{R'} y$. □

4.4 *R*-minimal pattern matching automaton

In this section we propose a construction of a minimal pattern matching automaton with respect to R-equivalence relation (R-minimal automaton). According to Theorem 2, R-minimal automaton is also Nerode-minimal. For degenerate patterns, R-equivalence allows to build minimal automaton in time linear to the number of its states.

Definition 11. *$RMinPatAut(\mathcal{H}) = (Q_M, \Sigma', \delta_M, s\ell_M, q_0^M, F_M)$ is a deterministic finite automaton with suffix links which accepts language $\Sigma^* \cdot \mathcal{H}$, where $Q_M = \Re(\mathcal{H})$; δ_M is a mapping $Q_M \times \Sigma'$ to Q_M; $s\ell_M$ is a mapping Q_M to Q_M; $q_0^M = \{\varepsilon\}$; $F_M = Q_M \cap 2^{\mathcal{H}}$.*

Let $q, q' \in Q_M$. Mapping δ_M is defined as follows: $\delta_M(q, \alpha) = q'$, $\forall \alpha \in \Sigma'$, such that $\forall a \in \alpha, qa \subseteq q'$. For each $q \in Q_M$, suffix link $s\ell_M(q)$ is equal to q' if for any word $x \in q$, $s\ell(x) \in q'$. Denote, $s\ell_M^{(0)}(q) = q$, $s\ell_M^{(i)}(q) = s\ell_M(s\ell_M^{(i-1)}(q))$.

Note, if $\delta_M(q, \alpha) = q'$ then $\forall \alpha' \subseteq \alpha, \delta_M(q, \alpha') = q'$. Denote by $label(q, q')$ the maximal degenerate label α for which $\delta_M(q, \alpha) = q'$, i. e.

$$label(q, q') = \{a \in \Sigma \mid \delta_M(q, a) = q'\}.$$

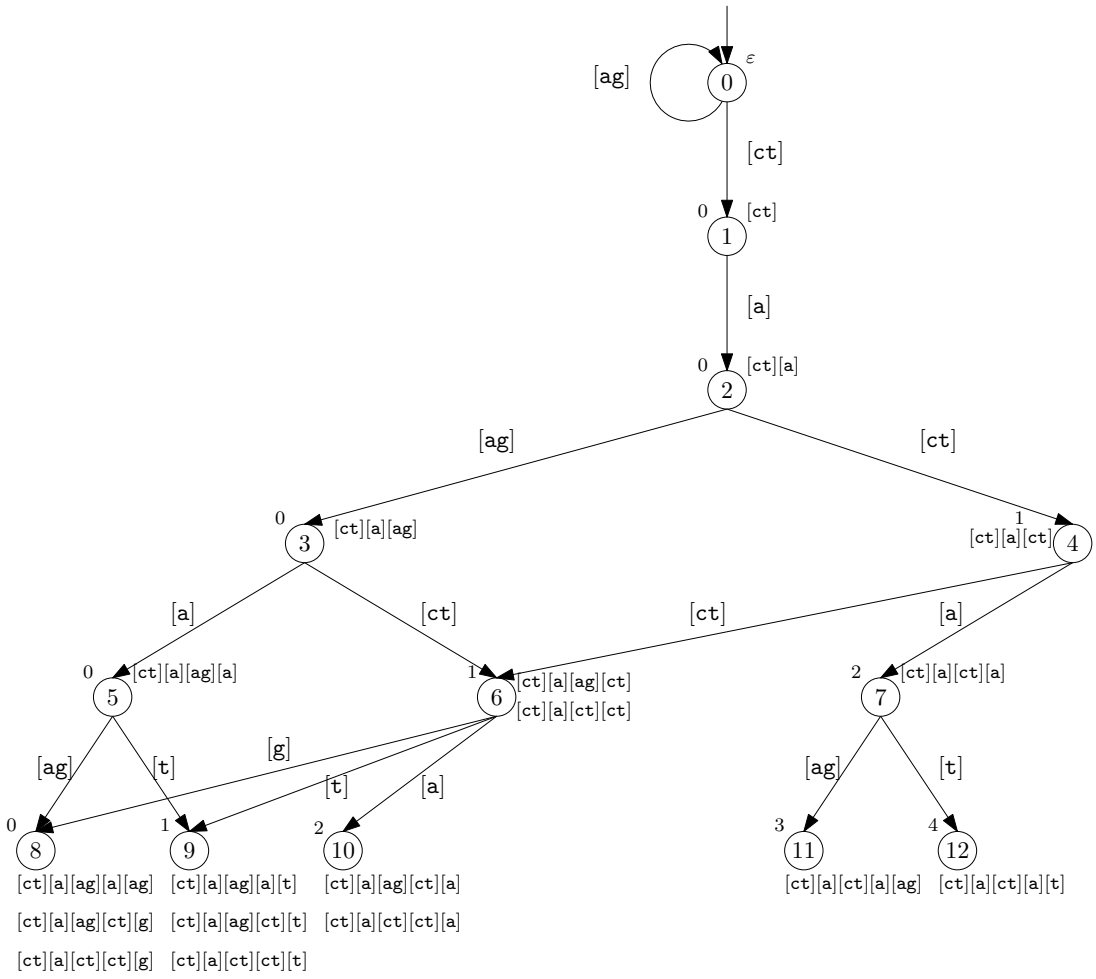

Fig. 1. *RMinPatAut*($\mathcal{H}$) for pattern $\mathcal{H} = v(\pi)$, $\pi = \{\mathtt{c},\mathtt{t}\}\{\mathtt{a}\}\{\mathtt{a},\mathtt{c},\mathtt{g},\mathtt{t}\}\{\mathtt{a},\mathtt{c},\mathtt{t}\}\{\mathtt{a},\mathtt{g},\mathtt{t}\}$. The digits near the nodes are the numbers of the nodes corresponding to their suffix links.

In a graphical representation of the automaton the edge between two states q and q' with degenerate label α (i.e., $q' = \delta_M(q, \alpha)$) corresponds to the union of multiple edges between these states labeled by symbols of Σ that belong to α. For convenience, we draw a single edge $\langle q, q' \rangle$ with degenerate label $label(q, q')$. The edge $\langle q, q' \rangle$ is called forward. Note, a path leading from one state to another state spells a degenerate string.

An example of *RMinPatAut* is given in Figure 1[5]. The self-loop in the initial state provides reading any string preceding an occurrence of word of *Pref*($\mathcal{H}$). Note that for a state there can be several incoming and outgoing forward edges but only one suffix link. Each state of *RMinPatAut*($\mathcal{H}$) corresponds to an R-equivalence class of *Pref*($\mathcal{H}$). The final states (leaves) correspond to R-equivalence classes on $\mathcal{H}$. The number of forward edges is bounded by $||\Sigma|| \times ||Q_M||$.

[5] In the figures, notation [ct] is used instead of {c, t} due to space reasons.

Theorem 3. *RMinPatAut($\mathcal{H}$) is minimal automaton according to both R-equivalence and Nerode-equivalence accepting the language $\Sigma^* \cdot \mathcal{H}$.*

Proof. Proof follows from the construction of *RMinPatAut*. □

4.5 Algorithm of *R*-minimal pattern matching automaton creating

The algorithm constructing *RMinPatAut* is based on Theorem 1 and Proposition 7 below. According to Theorem 1, if we have determined that states x, y from *Pref*($\mathcal{H}$) are equivalent then we know that transitions $\hat{\delta}(x,t)$ and $\hat{\delta}(y,t)$ in *PatAut*, $t \in \Sigma^*$, are also equivalent.

Proposition 7 gives an efficient method to create suffix links of $RMinPatAut$ nodes. It is a generalization of the observation that is used for the construction of suffix links of pattern matching automaton (the failure function in Aho-Corasick algorithm [AC75]). The observation is as follows. Let $x, xa \in$ *Pref*($\mathcal{H}$), $a \in \Sigma$. Then $s\ell(xa) = s\ell^{(i)}(x)a$, where i is the smallest possible integer for which $s\ell^{(i)}(x)a \in$ *Pref*($\mathcal{H}$), or $s\ell(xa) = \varepsilon$. Due to this observation the suffix links for all prefixes of words from $\mathcal{H}$ may be built in time linear to their number.

Denote,

$$I(q,\alpha) = \{i; \delta_M(s\ell_M^{(i)}(q), \alpha) \text{ is defined}\}.$$

Proposition 7. *Let $q, q' \in Q_M$, where $q' = \delta_M(q,\alpha)$, $\alpha \in \Sigma'$ ($\alpha \subseteq$ label(q,q')). Two cases occur:*

1. *If $I = \emptyset$ then $s\ell_M(q') = q_0^M$; otherwise*
2. *$s\ell_M(q') = \delta_M(s\ell_M^{(i)}(q), \alpha)$, where $i = min(I)$.*

Proof. The proof follows from the definition of suffix link. □

Corollary 5. *Let $q \in Q_M$, $\alpha \subseteq \pi[|q|+1]$. Also let there exist $e \in Q_M$ and integer $i > 0$ s. t. $\alpha \subseteq$ label$(s\ell_M^{(i)}(q), e)$, but for all $j < i$ and forward successors p of $s\ell_M^{(j)}(q)$, $p \in Q_M$, label$(s\ell_M^{(j)}(q), p) \cap \alpha = \emptyset$ [6]. Then there exists $q' \in Q_M$ s.t. $\delta_M(q,\alpha) = q'$, i.e. the words from $q\alpha$ are R-equivalent and $q\alpha \subseteq q'$. Moreover $s\ell_M(q') = e$.*

Proof. The proof follows from the definition of R-equivalency and Proposition 7. □

Example 5. The scheme of suffix links detection is demonstrated in Fig 2. Consider state q'. To find $sl_M(q')$ one has to consider predecessor q of q' ($q' = \delta_M(q,\alpha)$) and label $\alpha = \{a,b\}$ of the edge leading from q to q'. There exists a forward edge leading from $sl_M(q)$ to e' with label $\{a,b\}$. Then $sl_M(q') = e'$. Note, there also exists forward edge leading from $g = sl_M^{(2)}(q)$ to e'' with label $\{a,b,c\}$ containing α. But $sl_M(q') \neq e''$ because of $|sl_M(q)| > |sl_M^{(2)}(q)|$.

Remark 1. If a state has several forward predecessors, R-equivalence allows to use any of them for suffix-link detection. The result will be the same. For example, see state 6 in Fig 1.

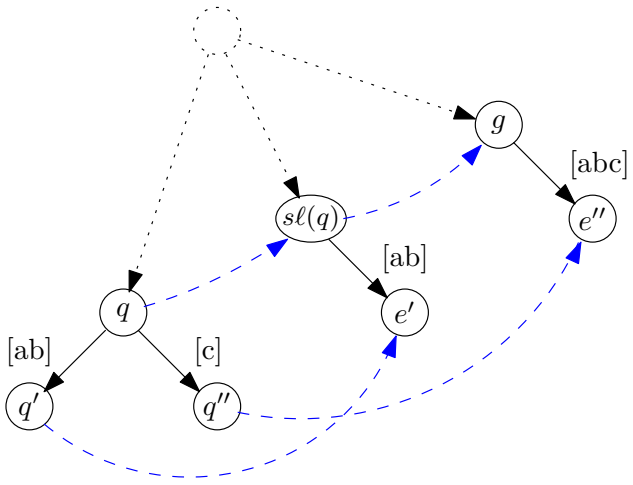

Fig. 2. Suffix links detection. Straight lines correspond to forward edges, blue dashed arcs correspond to suffix links, dotted node and lines represent higher levels of the graph.

Now we describe the algorithm constructing *RMinPatAut* for degenerate string $\pi = \pi[1 \dots m]$, see Algorithm 1. First, the algorithm creates the states of levels 0 and 1 (lengths of elements of states), see steps 1–4 in Algorithm 1. There exists only one state of level 1 because all words in $Pref(\mathcal{H})$ of length 1 have unique proper suffix ε and, therefore, are R-equivalent. Then, by induction on $k = 2, \dots, m$, it creates the states of level k, see steps 5–35 in Algorithm 1. The algorithm looks over the states of length $k-1$ and creates their forward successors (respectively mapping δ_M) of length k. Let q be a state of length $k-1$ and $\alpha = \pi[k]$.

The algorithm divides $q\alpha$ into R-equivalence classes (words of the same class have the same suffix link) and finds their suffix links, this step is based on Corollary 5. To find forward successors of q the algorithm traverses its suffix link predecessors $sl_M^{(i)}(q)$ in the order of increasing of i, see steps 11–24 in Algorithm 1. Let $r = s\ell^{(i)}(q)$ be a considered predecessor. If there exists $e \in Q_M$ s. t. $\alpha' = label(r, e) \cap \alpha \neq \emptyset$ then the words from $q\alpha'$ are R-equivalent, i. e. $q\alpha' \subseteq q'$, $q' \in Q_M$. Moreover $s\ell(q') = e$. If state p of length k with suffix link e is already built; then the algorithm sets: $\delta_M(q, \alpha') = p$ and $label(q, p) = \alpha'$. Otherwise the algorithm creates new state q' and sets: $s\ell(q') = e$; $\delta_M(q, \alpha') = q'$ and $label(q, q') = \alpha'$. Then the algorithm sets $\alpha = \alpha \setminus \alpha'$ and continues the procedure till $s\ell^{(i)}(q) = \{\varepsilon\}$ or $\alpha = \emptyset$.

The work of the algorithm is demonstrated in the example below.

Example 6. Consider the pattern $\mathcal{H} = v(\pi)$, where $\pi = \{\mathtt{c}, \mathtt{t}\}\{\mathtt{a}\}\{\mathtt{a}, \mathtt{c}, \mathtt{g}, \mathtt{t}\}\{\mathtt{a}, \mathtt{c}, \mathtt{t}\}\{\mathtt{a}, \mathtt{g}, \mathtt{t}\}$ and Figure 1. Let all the states of level ≤ 3 be created; we denote i-th state of *RMinPatAut* as q_i. Build the states of level 4, that are forward successors of q_3 and q_4. Consider q_3 and $\alpha = \pi[4]$. $s\ell_M(q_3) = q_0^M$. There exists a forward edge leading from q_0^M to q_1 with label

[6] If $\alpha \subseteq label(s\ell_M^{(i)}(q), e)$ and there exist $j < i$, $p \in Q_M$ s. t. $\alpha' = label(s\ell_M^{(j)}(q), p) \cap \alpha \neq \emptyset$ then the words from $q\alpha'$ are not R-equivalent to words from $q\beta$, $\beta = \alpha \setminus \alpha'$

Algorithm 1: Constructing of *RMinPatAut*

Data: $\pi = \pi[1 \dots m] \in (\Sigma')^*, m \geq 1$ and $||v(\pi)|| \geq 1$
Result: *RMinPatAut*($\mathcal{H}$)

$RMinPatAut(\mathcal{H}) \leftarrow (Q_M, \Sigma', \delta_M, q_0^M, s\ell_M, F_M)$;
$q_0^M \leftarrow newState$; $q_1 \leftarrow newState$; $Q_M \leftarrow \{q_0^M, q_1\}$;
$\delta_M(q_0^M, \pi[1]) \leftarrow q_1$; $s\ell_M(q_1) \leftarrow q_0^M$; $label(q_0^M, q_1) \leftarrow \pi[1]$;
$Q_1 \leftarrow \{q_1\}$;
for $k \leftarrow 2, \dots, m$ **do**
    $Q_k \leftarrow \emptyset$;
    $R \leftarrow \emptyset$;
    for *each* $q \in Q_{k-1}$ **do**
        $r \leftarrow q$;
        $\alpha \leftarrow \pi[k]$; `/* α - symbols to be processed */`
        while $(r \neq q_0^M) \wedge (\alpha \neq \emptyset)$ **do**
            $r \leftarrow s\ell_M(r)$;
            while *exists forward edge* $\langle r, e \rangle$ *s. t.* $label(r, e) \cap \alpha \neq \emptyset$ **do**
                $\alpha' \leftarrow label(r, e) \cap \alpha$;
                if $(e, p) \in R$ *for some* $p \in Q_k$ **then**
                    $\delta_M(q, \alpha') \leftarrow p$; $label(q, p) = \alpha'$;
                else
                    $q' \leftarrow newState$; $Q_M \leftarrow Q_M \cup \{q'\}$; $Q_k \leftarrow Q_k \cup \{q'\}$;
                    $\delta_M(q, \alpha') \leftarrow q'$; $s\ell_M(q') \leftarrow e$; $label(q, q') = \alpha'$;
                    $R \leftarrow R \cup \{(e, q')\}$;
                end
                $\alpha \leftarrow \alpha \setminus \alpha'$;
            end
        end
        if $\alpha \neq \emptyset$ **then**
            if $(q_0^M, p) \in R$ *for some* $p \in Q_k$ **then**
                $\delta_M(q, \alpha) \leftarrow p$; $label(q, p) = \alpha$;
            else
                $q' \leftarrow newState$; $Q_M \leftarrow Q_M \cup \{q'\}$; $Q_k \leftarrow Q_k \cup \{q'\}$;
                $\delta_M(q, \alpha) \leftarrow q'$; $s\ell_M(q') \leftarrow q_0^M$; $label(q, q') = \alpha$;
                $R \leftarrow R \cup \{(q_0^M, q')\}$;
            end
        end
    end
end
$F_M \leftarrow Q_m$;

$\beta = \{c, t\}$. One has $\beta \cap \alpha = \{c, t\}$. Then the algorithm builds new state for successor q' ($q' \supseteq q_3 \cdot \{c, t\}$) of q_3, where $s\ell_M(q') = q_1$, and forward edge $\langle q_3, q' \rangle$ with label $\{c, t\}$. There is no forward edge leading from q_0^M with label having non-empty intersection with $\alpha \setminus \beta = \{a\}$. Thus the algorithm builds the second forward successor q'' of q_3 with the suffix link q_0^M and forward edge $\langle q_3, q'' \rangle$ with label $\{a\}$. According to Figure 1, $q'' = q_5$ and $q' = q_6$. Analogously, for state q_4. State q_4 has successor with suffix link to q_1. State q_6 of length 4 with suffix link q_1 was already built while processing state q_3. Thus instead of creating a new state the algorithm creates new edge $\langle q_4, q_6 \rangle$.

The time and space complexities of Algorithm 1 are $\mathcal{O}(||\Sigma|| \times m \times ||Q_M||)$ and $\mathcal{O}(||Q_M|| + ||E_M||)$ respectively, where $||E_M||$ is the number of edges of *RMinPatAut*, i.e., the sum of the numbers of forward edges and suffix links. The factor m in the time complexity is the bound of the number of suffix link predecessors of a state.

5 Clumps recognizing automaton

In this section we define a R-minimal compact automaton (overlap walking automaton, *OWA*) recognizing all possible clumps. Our automaton is based on following ideas. Consider a compact automaton with the initial state corresponding to the empty word and terminal states corresponding to the words from a pattern, see Figure 3 and Example 7. It has two types of edges: 1) the edges leading from the initial to all terminal states, each edge is labeled by one word from the pattern; 2) the edges between the terminal states. There exists an edge from state v to another state w if a suffix u of v is also a prefix of w (in this case v can be extended into a clump $v.u^{-1}w$ of cardinality one ending with w). The label of the edge is extension $u^{-1}w$. Any path leading from the initial state to terminal state w in this automaton corresponds to a clump ending with w. All possible clumps over $\mathcal{H}$ can be generated by its traversals. This construction is a modified overlap graph that is widely used for genome assembly [KM95] and for solving some of pattern matching problems, e.g., finding a shortest linear or cyclic superstring [BJL+93]. Analogous idea is used in the paper [RFI14] for clumps probabilities computation. Here we present a similar automaton, but the terminal states of our automaton are R-equivalence classes on the pattern words. Note, described structure can be transformed from the pattern matching automaton *PatAut*. Analogously, we use *RMinPatAut* to construct *OWA*.

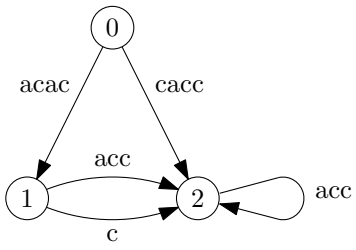

Fig. 3. Clumps recognizing automaton for $\mathcal{H} = \{\texttt{acac}, \texttt{cacc}\}$.

Example 7. Let $\mathcal{H} = \{\texttt{acac}, \texttt{cacc}\}$. In Figure 3 a compact automaton recognizing all clumps over $\mathcal{H}$ is shown. The initial state corresponds to the empty word, nodes 1 and 2 correspond to words $h_1 = \texttt{acac}$ and $h_2 = \texttt{cacc}$ from $\mathcal{H}$, respectively. There are two overlaps, `c` and `cac`, between h_1 and h_2. Then there are two edges leading from node 1 to node 2 with labels `acc` and `c` (`acc` and `c` are extensions of `c` and `cac` into h_2). Also there is a self-loop in state 2 labeled by `acc`. All possible clumps for $\mathcal{H}$ can be generated

by traversing the automaton: k-clumps correspond to paths of length $k+1$ leading from the initial to other states. The clumps of cardinality zero are the words from the pattern spelling the paths $0 \xrightarrow{\texttt{acac}} 1$ and $0 \xrightarrow{\texttt{cacc}} 2$. There are only three different clumps of cardinality one: `acac.acc`, `acac.c` and `cacc.acc`. These clumps correspond to the paths $0 \xrightarrow{\texttt{acac}} 1 \xrightarrow{\texttt{acc}} 2$, $0 \xrightarrow{\texttt{acac}} 1 \xrightarrow{\texttt{c}} 2$ and $0 \xrightarrow{\texttt{cacc}} 2 \xrightarrow{\texttt{acc}} 2$.

Below we assume that $\mathit{RMinPatAut}(\mathcal{H}) = (Q_M, \Sigma', \delta_M, s\ell_M, q_0^M, F_M)$ is constructed.

Let $q \in Q_M, f \in F_M$. In $\mathit{RMinPatAut}(\mathcal{H})$ each path (i.e., sequence of forward edges) from q to f spells a degenerate string. Denote by $e(q,f)$ a union of all such degenerate strings for all existing paths from q to f in $\mathit{RMinPatAut}(\mathcal{H})$. In other words, value of $e(q,f)$ consists of all correlations of the words from q into the words from f, i.e.,

$$v(e(q,f)) = \bigcup_{w \in q, h \in f} \{w^{-1}.h\}.$$

Note that the degenerate strings from set $e(q,f)$ have the same length $m - |q|$.

Example 8. Consider $\mathit{RMinPatAut}(\mathcal{H})$ in Figure 1. $F_M = \{f_8, f_9, f_{10}, f_{11}, f_{12}\}$, f_i stands for the terminal node with number i in Figure 1. For node q_3 (internal node), $e(q_3, f_8) = \{\{\texttt{a}\}\{\texttt{ag}\}, \{\texttt{ct}\}\{\texttt{g}\}\}$, $e(q_3, f_9) = \{\{\texttt{a}\}\{\texttt{t}\}, \{\texttt{ct}\}\{\texttt{t}\}\}$, $e(q_3, f_{10}) = \{\{\texttt{ct}\}\{\texttt{a}\}\}$, $e(q_3, f_{11}) = e(q_3, f_{12}) = \emptyset$.

Definition 12. *Overlap walking automaton* $\mathit{OWA}(\mathcal{H}) = (\tilde{Q}, \Sigma', \tilde{\delta}, \tilde{q_0}, \tilde{F})$ *is the R-minimal*[7] *deterministic compacted finite automaton recognizing all possible clumps of* $\mathcal{H}$*, where* $\tilde{q_0}$ *is the initial state;* $\tilde{Q}$ *is a finite set of states;* $\tilde{\delta}$ *is a mapping* $\tilde{Q} \times (\Sigma'^* \setminus \{\varepsilon\})$ *to* $\tilde{Q}$*;* $\tilde{F} \subset \tilde{Q}$ *is a set of final states.*

$\mathit{OWA}(\mathcal{H})$ is constructed from $\mathit{RMinPatAut}(\mathcal{H})$ as follows: $\tilde{q_0} = \{\varepsilon\}$; $\tilde{Q} = F_M \cup \{\tilde{q_0}\}$, $\tilde{F} = F_M$. For each $f \in \tilde{F}$, $\tilde{\delta}(\tilde{q_0}, e(\tilde{q_0}, f)) = f$; and for each $g \in \tilde{F}$ and i, where $e(s\ell_M^{(i)}(g), f) \neq \emptyset$ and $s\ell_M^{(i)}(g) \neq \{\varepsilon\}$, $\tilde{\delta}(g, e(s\ell_M^{(i)}(g), f)) = f$. We assume, if $\tilde{\delta}(g, \omega) = f$ then $\tilde{\delta}(g, \omega') = f$, where $\omega, \omega' \in \Sigma'$, $\omega' \subseteq \omega$.

Transition $\tilde{\delta}(g, e(q,f))$, where $g = q = \tilde{q_0}$ or $q = s\ell_M^{(i)}(g)$ (if $g \in \tilde{F}$), corresponds to an edge $\langle g, e(q,f), f\rangle$ in the graph representation of OWA, and $e(q,f)$ is the label of the edge. Note, there are at most m edges leading from g to f for $g, f \in \tilde{F}$; and exactly $||\tilde{F}||$ edges leading from $\tilde{q_0}$ to the other states. The total number of OWA edges is bounded by $O(m \times ||\tilde{F}||^2)$. An example of $\mathit{OWA}(\mathcal{H})$ is given in Figure 4[8].

RMinPatAut can be transformed to *OWA* by eliminating all non-terminal states except the initial state. The label on each edge leading from the initial state $\tilde{q_0}$ to a terminal state f of *OWA* is union of labels of all paths of forward edges leading from q_0^M to the state corresponding to f in *RMinPatAut*. There exists an edge $\langle g, e(s\ell_M^{(i)}(g), f), f\rangle$, where $g, f \in \tilde{F}$, iff there exists a path of forward edges leading from $s\ell^{(i)}(g)$ to f, i.e., each word from $s\ell^{(i)}(g)$ can be extended to a word from f. The label of the edge is union of labels of

[7] With respect to *R-equivalence*.

[8] Numbers of nodes of *OWA* in the example correspond to the numbers of nodes of *RMinPatAut*.

all such paths. There can be several edges leading from g to f that correspond to different suffix link predecessors of g.

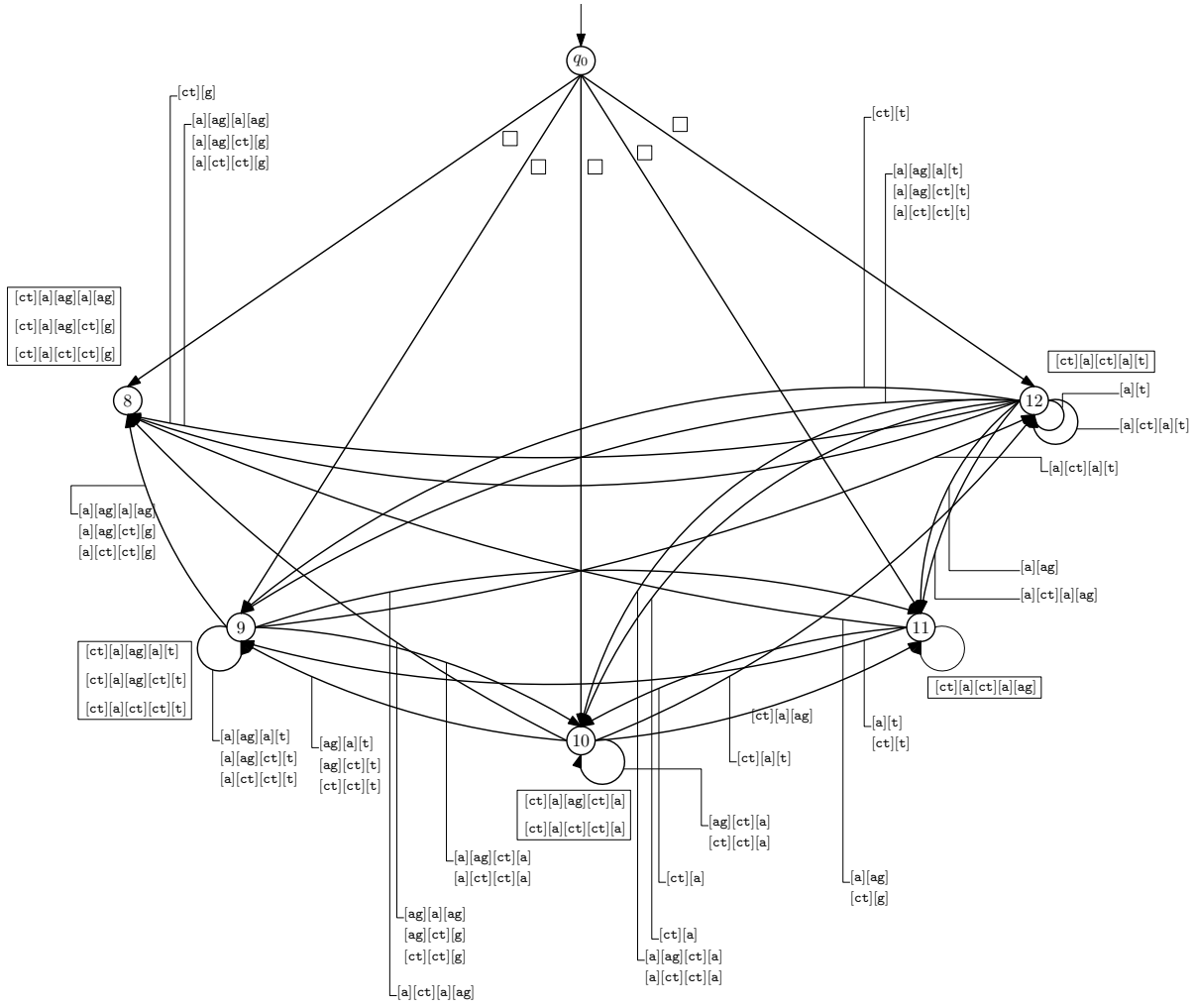

Fig. 4. $OWA(\mathcal{H})$ for $\mathcal{H} = \{\mathtt{c},\mathtt{t}\}\{\mathtt{a}\}\{\mathtt{a},\mathtt{c},\mathtt{g},\mathtt{t}\}\{\mathtt{a},\mathtt{c},\mathtt{t}\}\{\mathtt{a},\mathtt{g},\mathtt{t}\}$. Each symbol $\Box$ in transition label represents a set of degenerate strings associated to the destination state. The set is in the frame near the destination state.

In Section 6 we use OWA to compute probabilities of clumps. For this purpose, labels $e(q,f)$ of edges $\langle g, e(q,f), f\rangle$ can be replaced by probabilities of their values. For instance, for a Bernoulli model, the set $e(q,f)$ can be replaced by the pair (p,l), where $p = Prob(v(e(q,f)))$, $l = |e(q,f)| = m - |q|$. The edges of this automaton are 4-tuples $\langle g, p, l, f\rangle$. In a Bernoulli case,

$$Prob(v(e(q,f))) = \sum_{w \in v(e(q,f))} \prod Prob(w[i]).$$

Below we consider Bernoulli models. The procedures described below can be easily extended for Markov or hidden Markov models by taking into account properties of the models.

Note, $F_M = \tilde{F}$, and $q_0^M = \tilde{q_0} = \{\varepsilon\}$. Below, for simplicity, we will use in some cases notations F to refer to both F_M and $\tilde{F}$, and q_0 to refer to both q_0^M and $\tilde{q_0}$.

We describe Algorithm 2 constructing $OWA(\mathcal{H})$ for clumps probabilities computation. The algorithm proceeds in three steps. First, the algorithm creates sets $\tilde{Q}$, $\tilde{F}$ and OV (overlaps set), where

$$OV = \{s\ell_M^{(i)}(f) \mid f \in F, i \geq 1\}.$$

It can be done by traversing $RMinPatAut(\mathcal{H})$ from the leaves to the root (bottom-up traversal) following the suffix links. Second, for all $q \in OV \cup F$, $f \in \Delta(q)$, where

$$\Delta(q) = \{f \in F \mid e(q, f) \neq \emptyset\},$$

the algorithm computes the probabilities $Prob(v(e(q, f)))$. It also can be done by depth-first or bottom-up traversal of $RMinPatAut(\mathcal{H})$, see Algorithm 3 (bottom-up traversal). Then Algorithm 2 creates edges between states of $OWA(\mathcal{H})$. Edges between terminal states can be created by enumerating all 3-tuples $\langle g, q, f\rangle$, where $g \in F$; $q = s\ell_M^{(i)}(g), 1 \leq i \leq m$; and $f \in \Delta(q)$. After construction of $OWA(\mathcal{H})$, the automaton $RMinPatAut(\mathcal{H})$, set OV and sets $\Delta(q)$ can be removed.

We analyze the time and space complexities of Algorithm 2. The complexities of the algorithm is determined by step 2 (i.e., by the complexities of Algorithm 3) and step 3. Running of the step 3 needs $\mathcal{O}(m \times ||F||^2)$ operations and space. Algorithm 3 needs $\mathcal{O}(||\Sigma|| \times ||Q_M|| \times ||F||)$ operations and $\mathcal{O}(||Q_M|| \times ||F||)$ space respectively. Thus the time and space complexities (excluding storing of $RMinPatAut$) of OWA creating are $\mathcal{O}(m \times ||F||^2 + ||\Sigma|| \times ||Q_M|| \times ||F||)$ and $\mathcal{O}(m \times ||F||^2 + ||Q_M|| \times ||F||)$.

6 Computation of clumps sets probabilities

In this section we describe an algorithm to compute probabilities of clumps sets.

Denote by $C(n, k)$ the set of clumps of length n and cardinality k.

Definition 13. *Two clumps of the same lengths and cardinalities are R-equivalent if they end with R-equivalent words from $\mathcal{H}$. Denote by $C(n, k, f)$ the equivalence class of clumps of length n and cardinality k ending with words from f, $f \in F$.*

One has

$$C(n, k) = \bigcup_{f \in F} C(n, k, f).$$

Transition from q_0 to a terminal state f of OWA corresponds to the set of clumps $C(m, 0, f)$ of cardinality 0, where $C(m, 0, f) = f$. Each path of length $k + 1$ leading from q_0 to f

Algorithm 2: Constructing of $OWA(\mathcal{H})$

Input: $RMinPatAut = (Q_M, \Sigma', \delta_M, s\ell_M, q_0^M, F_M)$
Output: $OWA(\mathcal{H}) = (\tilde{Q}, \Sigma', \tilde{\delta}, \tilde{q_0}, \tilde{F})$

$\tilde{q_0} \leftarrow q_0^M$, $\tilde{F} \leftarrow F_M$, $\tilde{Q} \leftarrow \tilde{F} \cup \{\tilde{q_0}\}$;
1. Creating of set OV
$OV \leftarrow \{q_0\}$; // $q_0 = q_0^M = \tilde{q_0}$
foreach $f \in F$ **do**
  $r \leftarrow s\ell_M(f)$; // $F = F_M = \tilde{F}$
  while $r \neq q_0$ **do**
    if *r is not in OV* **then**
      $OV \leftarrow OV \cup \{r\}$;
      $r \leftarrow s\ell_M(r)$;
    else
      break;
    end
  end
end
2. For each $q \in OV$, creating of set $\Delta(q)$ and computing of $Prob(v(e(q, f)))$, $f \in \Delta(q)$
Run Algorithm 3;
3. Creating of edges $< g, p, l, f >$
foreach $g \in F$ **do**
  $r \leftarrow s\ell_M(g)$;
  while $r \neq q_0$ **do**
    foreach $f \in \Delta(r)$ **do**
      $p \leftarrow Prob(v(e(r, f)))$;
      $l \leftarrow m - |r|$;
      Add edge $< g, p, l, f >$;
      $r \leftarrow s\ell_M(r)$;
    end
  end
end

corresponds to a set of clumps of length n and cardinality k that is a subset of $C(n, k, f)$. This idea is formalized in Theorem 4.

Let $f \in F$. Denote by $Edges(f)$ the subset of OWA edges with tail f, i.e.,

$$Edges(f) = \{\langle g, p, l, f\rangle\},$$

where $g \in F$, $p = Prob(v(e(sl_M^{(i)}(g), f)))$, $sl_M^{(i)}(g) \neq q_0$ and $l = |e(sl_M^{(i)}(g), f)|$.

Theorem 4. *Let $f \in F$, $n \geq m$ and $k \geq 1$. Then*

$$Prob(C(m, 0, f)) = Prob(f); \tag{8}$$

$$Prob(C(n, k, f)) = \sum_{\langle g,p,l,f\rangle \in Edges(f)} Prob(C(n - l, k - 1, g)) \cdot p, \tag{9}$$

Proof. The proof follows from Definition 13. □

Algorithm 3: Computation of $Prob(v(e(q,f)))$

Input: $RMinPatAut = (Q_M, \Sigma', \delta_M, s\ell_M, q_0^M, F_M)$
Output: $\Delta(q)$ and $Prob(v(e(q,f)))$, $q \in OV$, $f \in \Delta(q)$

```
1  foreach f ∈ F do
2  |   Δ(f) ← {f};
3  |   Prob(v(e(f, f))) ← 1;
4  end
   Bottom-up traversal of RMinPatAut
5  foreach k = m − 1, . . . , 0 do
6  |   foreach q ∈ Q_M s.t. |q| = k do
7  |   |   foreach forward edge ⟨q, r⟩ do
8  |   |   |   α = label(q, r);
9  |   |   |   Δ(q) ← Δ(q) ∪ Δ(r);
10 |   |   |   foreach f ∈ Δ(r) do
11 |   |   |   |   Prob(v(e(q, f)))+ = Prob(α) · Prob(v(e(r, f)));
12 |   |   |   end
13 |   |   end
14 |   end
15 |   foreach q ∈ Q_M \ OV s.t. |q| = k + 1 do
16 |   |   Delete Δ(q) and probabilities Prob(v(e(q, f)));
17 |   end
18 end
```

Note that for $n < m$, $C(n,k,f) = \emptyset$.

We describe Algorithm 4 computing $Prob(C(n,k))$, for $k = 0, \ldots, K$ and all achievable n, $n \geq m$. For the computation, Algorithm 4 traverses K times $OWA(\mathcal{H})$. Initially, it computes $Prob(C(m,0))$, see lines 4–7 of Algorithm 4. Then the algorithm computes $Prob(C(n,k))$ by induction on clumps cardinality $k = 1, \ldots, K$, see lines 8–23; for a given k, probabilities of clumps of achievable lengths n are computed. For a given k and each $f \in F$, it additionally computes $Prob(C(n,k,f))$ by traversal of $OWA(\mathcal{H})$; $Prob(C(n,k))$ is the sum of these probabilities. During the procedure, two arrays of lists *NewProbsClumps* and *PrevProbsClumps* of the same size $||F||$ are used; for a state $f \in F$, *NewProbsClumps*$[f]$ and *PrevProbsClumps*$[f]$ contain the lists $\{\langle Prob(C(n,k,f)), n\rangle\}_{n \geq m}$ and $\{\langle Prob(C(n,k-1,f)), n\rangle\}_{n \geq m}$ respectively. Also during of processing of a state $f \in F$ the algorithm uses auxiliary array *NextGen* of size $N = m \times (K+1) - K$, where N is the boundary of clumps lengths; initially, *NextGen*$[n] = 0$. For f, the algorithm traverses all edges $\langle g, p, l, f\rangle$ from *Edges*(f) and computes $Prob(C(n,k-1,g)) \cdot p$, where $Prob(C(n,k-1,g)) \in$ *PrevProbsClumps*$[g]$. The values are added to *NextGen*$[n+l]$. After processing of f, *NextGen*$[n]$ contains $Prob(C(n,k,f))$, $m \leq n \leq N$.

Algorithm 4 has time and space complexities $\mathcal{O}(K \times N \times (||F|| + ||E_{OWA}||))$ and $\mathcal{O}(N \times ||F|| + ||E_{OWA}||)$, where $N = m \times (K+1) - K$ is the boundary of clumps length, and $||E_{OWA}||$ is the number of *OWA* edges, $||E_{OWA}|| \leq m \times ||F||^2$. The boundary of the space complexity includes *OWA* size.

Algorithm 4: Computation of clumps sets probabilities

Input: $OWA(\mathcal{H})$, K - maximal cardinality of clumps
Output: $Prob(C(n,k))$, $k = 0, \ldots, K$ and $m \leq n \leq N$

1. Initialization
$N = m \times (K+1) - K$; // boundary of clumps length
array of lists $NewProbsClumps[||F||]$;
array of lists $PrevProbsClumps[||F||]$;
foreach $f \in F$ **do**
  $Prob(C(m,0)) += Prob(f)$;
  $PrevProbsClumps[f] \leftarrow \{\langle Prob(f), m\rangle\}$;
end

2. Inductive computation
foreach $k = 1, \ldots, K$ **do**
  foreach $f \in F$ **do**
    array $NextGen$[N];
    foreach *edge* $\langle g, p, l, f\rangle \in Edges(f)$ **do**
      // $p = Prob(v(e(sl^{(i)}(g), f)))$ and $l = |e(sl^{(i)}(g), f)|$
      foreach $\langle Prob(C(n, k-1, g)), n\rangle \in PrevProbsClumps[g]$ **do**
        $NextGen[n+l] += Prob(C(n,k-1,g)) \cdot p$;
      end
    end
    foreach n *such that* $NextGen[n] \neq 0$ **do**
      $Prob(C(n,k)) += NextGen[n]$; // $NextGen[n] = Prob(C(n,k,f))$
      $NewProbsClumps[f] \leftarrow NewProbsClumps[f] \cup \{\langle NextGen[n], n\rangle\}$;
    end
  end
  $PrevProbsClumps \leftarrow NewProbsClumps$;
  Clear $NewProbsClumps$;
end

7 Computer experiments

We have performed computer experiments to estimate real sizes of automata $RMinPatAut(\mathcal{H})$ and $OWA(\mathcal{H})$. We have considered three types of degenerate patterns:

- amino acids patterns from database PROSITE [SCea12] (release from September 21, 2016). PROSITE contains patterns (regular expressions) and profiles (position weight matrices) that describe more than a thousand protein families, modular protein domains and functionally or structurally important residues. We have considered only those patterns that can be represented as degenerate string, see Definition 8. Totally 969 patterns of lengths 3–50 were considered; they contain from 1 to 10^{61} words.
- nuclear acids patterns from database YEASTRACT [TMea14]. YEASTRACT contains patterns describing transcription factor binding sites for *Saccharomyces cerevisiae*. Also only those patterns that can be represented as degenerate string in IUPAC alphabet were considered (some patterns correspond to several transcription factors). 266 different patterns of lengths 5–55 were considered; they contain from 1 to $1.2 \cdot 10^{24}$ words.

About 10 degenerate patterns from the databases PROSITE and YEASTRACT were not considered due to our algorithms can not process these patterns in reasonable time. For example, the pattern `TTTGCN{97}GCAAA` from YEASTRACT is overly huge.

– randomly generated nuclear acids patterns of lengths 12 and 17. We have generated degenerate strings in IUPAC alphabet assuming uniform distribution of IUPAC letters; 1000 patterns of lengths 12 and 17 containing from 1 to 23 887 872 words were generated.

For each of this patterns we have created *RMinPatAut*($\mathcal{H}$) and *OWA*($\mathcal{H}$) and computed the numbers of their states and edges. The automata sizes are most critical parameters for complexities of presented algorithms. The summary of these experiments is presented in Table 1. More detailed results are given in `http://www.lix.polytechnique.fr/~regnier/OWA/Appendix_A4.xls`. Also some examples of data are given in Table 2. The experiments show that for most of cases the numbers of states and edges of our automata are significantly smaller than the patterns sizes, especially for PROSITE. Note, the number of states of initial pattern matching automaton (AHO-CORASICK automaton) is $\lambda \cdot ||\mathcal{H}||$, $1 \leq \lambda \leq |\mathcal{H}|$. For more than 90% of considered patterns, the constructed automata have small sizes (< 1000 nodes or edges) despite the huge patterns sizes. For example, the pattern GAS_VESICLE_C from PROSITE contains about $10^{20.4}$ words, but *OWA* for this patterns has only 12 states and 206 edges, see Table 2. Thus generally our algorithms work very fast and use small memory space.

Note, that the number of edges of *OWA* is quadratic on the number of its states. In some cases this number can be large. Also note that for several patterns the numbers of states and edges of *OWA* are 2 and 1 correspondingly. The reason is that the words in the patterns do not intersect each other.

8 Conclusion

In this paper we present the following results. First, we introduce R-equivalence relation on prefixes of pattern words and prove that R-equivalent words are also Nerode-equivalent, i.e., they are equivalent in classical sense.

Second, we propose construction of minimal pattern matching automaton *RMinPatAut* recognizing language $\Sigma^* \cdot \mathcal{H}$. The states of this automaton are R-equivalence classes on prefixes of pattern words. For degenerate patterns, we present an algorithm that builds *RMinPatAut* with a time complexity proportional to its size. Algorithm is also based on the fact that prefixes of the same lengths of words from a degenerate pattern have the same extensions to pattern words. In this paper *RMinPatAut* is an auxiliary structure for clumps recognizing automaton constructing but it is of interest itself, it can be applied for solving a range pattern matching problems.

Third, we propose automaton *OWA* that recognizes clumps for patterns defined by degenerate strings. One can compute clumps sets probabilities by traversals of *OWA*. The states of *OWA* are R-equivalence classes on pattern words. This automaton induces partition on clumps. Each path leading from the initial state to a final state defines a class of clumps. Such approach allows to efficiently compute probabilities of classes instead of enumerating of individual clumps. To construct *OWA* we use *RMinPatAut*.

Data set		*RMinPatAut*		*OWA*	
		States $\|\|Q_M\|\|$	Edges $\|\|E_M\|\|$	States $\|\|\tilde{Q}\|\|$	Edges $\|\|E_{OWA}\|\|$
PROSITE	min	4	7	2	1
	50 %	32	79	3	4
	90 %	277	849	13	232
	max	70 303	174 065	5 442	147 611 094
YEASTRACT	min	6	11	2	1
	50 %	11	21	2	1
	90 %	30	73	3	9
	max	325 917	916 931	10 830	133 995 890
random 12	min	14	28	2	1
	50 %	39	99	5	26
	90 %	100	278	14	358
	max	932	2 752	292	171 073
random 17	min	22	46	2	1
	50 %	64	166	5	35
	90 %	149	418	14	392
	max	1 602	4 590	111	38 628

Table 1. Summary of computer experiments. We have considered four sets $Q_M, E_M, \tilde{Q}$ and E_{OWA} for each type of patterns. A value lying on intersection of a row with item "min" or "max" and column assigned to a set X is the minimal or maximal element of X. A value lying on intersection of a row with an item of the form "α%" and a column assigned to a set X is α-percentile of elements of X.

Computer experiments show that, for patterns defined by degenerate strings, sizes of proposed automata are significantly smaller than the patterns sizes. Efficiency of our approach increases for larger alphabets. In the future, we plan to construct analogous automata for another types of patterns.

References

ABN12. O. AitMous, F. Bassino, and C.l Nicaud. An efficient linear pseudo-minimization algorithm for Aho-Corasick automata. In *23rd Annual Symposium, CPM 2012, Helsinki, Finland, Proceedings*, pages 110–123, 2012.

AC75. A. V. Aho and M. J. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

BCFN08. F. Bassino, J. Clément, Julien Fayolle, and P. Nicodeme. Constructions for Clumps Statistics. In Michael Drmota; Mark Ward, editor, *5th International Colloquium on Mathematics and Computer Science, Sep 2008, Blaubeuren, Germany, Discrete Mathematics and Theoretical Computer Science Proceedings*, pages 183–198, 2008.

BCR+07. V. Boeva, J. Clément, M. Régnier, M. Roytberg, and V. Makeev. Exact p-value calculation for heterotypic clusters of regulatory motifs and its application in computational annotation of cis-regulatory modules. *Algorithms for molecular biology*, 2(13):25 pages, 2007.

BJL+93. A. Blum, T. Jiang, M. Li, J. Tromp, and M. Yannakakis. Linear approximation of shortest superstrings. *Journal of the ACM*, 41(4), 1993.

CHL07. M. Crochemore, C. Hancart, and T. Lecroq. *Algorithms on Strings. Pattern matching automata.* Cambridge University Press, New York, 2007.

Data set	Pattern information			*RMinPatAut*		*OWA*	
	Pattern (or PROSITE signature)	Length $\|\mathcal{H}\|$	Size $\|\|\mathcal{H}\|\|$	States $\|\|Q_M\|\|$	Edges $\|\|E_M\|\|$	States $\|\|\tilde{Q}\|\|$	Edges $\|\|E_{OWA}\|\|$
PROSITE	GAS_VESICLE_C	23	$10^{20.4}$	230	674	12	206
	ERGTX	35	10^{38}	63 510	174 065	5 030	147 611 094
	INTRADIOL_DIOXYGENAS	29	$10^{28.9}$	1 480	4 900	89	17 676
	RNA_POL_D_30KD	41	$10^{34.7}$	1 068	3 363	2	1
	SRCR_1	38	10^{41}	28 356	92 377	2 073	18 947 048
YEASTRACT	WYTTCAYRTGS	11	32	18	39	2	1
	TTAANNCAAANNCNGNYT	18	8 192	30	68	3	8
	MGCN(9)MGS	15	$10^{6.3}$	644	1 605	145	32 486
	TGTTTCCN(18)TGTTTCT	31	$10^{10.8}$	754	2 157	29	946
	ATGAACAN(40)ATGAAACA	55	10^{24}	325 917	916 931	10 830	133 995 890
random 12	GVSDDBAKYYYC	12	2 592	72	200	2	1
	NNMBHBKRSMGH	12	41 472	54	141	5	64
	GVHWNDDNMRRD	12	62 208	161	393	57	4 197
	BVHHNBBYHDGK	12	104 976	480	1 472	21	928
	BSNBNBNHKNBR	12	497 664	932	2 752	189	100 416
random 17	YTWVRKKDBSDTSYGWR	17	82 944	79	227	2	1
	CHNVKSMDVKMRTKKWD	17	497 664	105	294	3	3
	CYDNNCYKRTHWNHNWV	17	10^{6}	79	198	12	186
	NVVSYHBTHDBNNDNDA	17	$10^{7.3}$	341	1 049	32	2 451
	VWDNRWBWHNSDMDNNM	17	$10^{7.4}$	561	1 759	111	38 628

Table 2. Examples of automata sizes for patterns from considered datasets.

CIK+16. M. Crochemore, C. S. Iliopoulos, R. Kundu, M. Mohamed, and F. Vayani. Linear algorithm for conservative degenerate pattern matching. *Engineering applications of artificial intelligence*, 51:109–114, 2016.

Fle05. A. C. Fleck. A simplified view of Nerode equivalence. *Computing Letters*, 1(3):93–96, 2005.

HS03. J. Holub and W. F. Smyth. Algorithms on indeterminate strings. In M. Miller and K. Park, editors, *Proceedings of the 14th Australasian Workshop on Combinatorial Algorithms (AWOCA'03)*, pages 36–45. Seoul National University, Seoul, Korea, 2003.

HU79. J. E. Hopcroft and J. D. Ullman. *Introduction to automata, languages and computations.* Addison-Wesley, Reading, MA, 1979.

IKM16. C. S. Iliopoulos, R. Kundu, and M. Mohamed. Efficient computation of clustered-clumps in degenerate strings. In *12th IFIP WG 12.5 International Conference and Workshops, AIAI 2016, Thessaloniki, Greece*, pages 510–519, September 16–18, 2016.

KM95. J.D. Kececioglu and E. F. Myers. Combinatorial algorithms for DNA sequence assembly. *Algorithmica*, 13(1–2):7–51, 1995.

KNR07. G. Kucherov, L. No, and M. Roytberg. Subset seed automaton. In *12th International Conference, CIAA 2007, Praque, Czech Republic*, pages 180–191, July 16–18, 2007.

MKS10. J. R. Miller, S. Koren, and G. Sutton. Assembly algorithms for next-generation sequencing data. *Genomics*, 95(6):315–327, 2010.

Ner58. A. Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958.

RFI14. M. Regnier, B. Fang, and D. Iakovishina. Clump Combinatorics, Automata, and Word Asymptotics. In Michael Drmota; Mark Ward, editor, *ANALCO' 2014, Portland, United States.*, pages 62–73, Jan 2014.

RFYR14. M. Régnier, E. Furletova, V. Yakovlev, and M. Roytberg. Analysis of pattern overlaps and exact computation of P-values of pattern occurrences numbers: case of Hidden Markov Models. *Algorithms for molecular biology*, 9(25), 2014.

RS98. G. Reinert and S. Schbath. Compound Poisson approximation for occurrences of multiple words in Markov chains. *Journal of Computational Biology*, 5(2):223–253, 1998.

SCea12. C. J. A. Sigrist, E. Castro, and et al. New and continuing developments at PROSITE. *Nucleic Acids Res*, 2012.

SH13. A. Saxena S. Hasib, M. Motwani. Importance of Aho-Corasick string matching algorithm in real world applications. *International journal of computer science and information technologies*, 4(3):467–469, 2013.

SRS07. V.T. Stefanova, S. Robin, and S. Schbath. Waiting times for clumps of patterns and for structured motifs in random sequences. *Discrete Applied Mathematics*, 155:868–880, 2007.

Sto00. G. D. Stormo. DNA binding sites: representation and discovery. *Bioinformatics*, 16:16–23, 2000.

TMea14. M. C. Teixeira, P. T. Monteiro, and et al. The YEASTRACT database: an upgraded information system for the analysis of gene and genomic transcription regulation in Saccharomyces cerevisiae. *Nucleic Acids Res*, 42:D161–D166, 2014.