The Minimum DAWG for All Suffixes of a String and Its Applications

Shunsuke Inenaga[1], Masayuki Takeda[1,2], Ayumi Shinohara[1,2], Hiromasa Hoshino[1], and Setsuo Arikawa[1]

[1] Department of Informatics, Kyushu University
33 Fukuoka 812-8581, Japan
{s-ine,takeda,ayumi,hoshino,arikawa}@i.kyushu-u.ac.jp
[2] PRESTO, Japan Science and Technology Corporation (JST)

Abstract. For a string w over an alphabet Σ, we consider a composite data structure called the *all-suffixes directed acyclic word graph* (*ASDAWG*). $ASDAWG(w)$ has $|w| + 1$ initial nodes, and the dag induced by all reachable nodes from the k-th initial node conforms with $DAWG(w[k:])$, where $w[k:]$ denotes the k-th suffix of w. We prove that the size of the *minimum* $ASDAWG(w)$ ($MASDAWG(w)$) is $\Theta(|w|)$ for $|\Sigma| = 1$, and is $\Theta(|w|^2)$ for $|\Sigma| \geq 2$. Moreover, we introduce an *on-line* algorithm which directly constructs $MASDAWG(w)$ for given w, whose running time is linear with respect to its size. We also demonstrate some application problems, *beginning-sensitive pattern matching*, *region-sensitive pattern matching*, and *VLDC-pattern matching*, for which ASDAWGs are useful.

1 Introduction

In the field of information retrieval, pattern matching on strings is one of the most fundamental and important problems. A variety of patterns have been considered so far, according to various kinds of purposes and aims. The most basic one is a *substring* pattern. Let Σ be a finite alphabet. We call an element in Σ a *character*, and one in Σ^* a *string*. We say a pattern string p is a substring of a text string w if $w = upv$ for some strings $u, v \in \Sigma^*$. When a text w is fixed and a pattern p is flexible, once constructing a suitable data structure for w, we can solve the substring matching problem in $O(|p|)$ time, where $|p|$ denotes the length of p. In order to solve the problem efficiently, much attention has extensively been paid to inventing efficient data structures, such as suffix trees [21,16,20], directed acyclic word graphs (DAWGs) [3,5], compact directed acyclic word graphs (CDAWGs) [4,7,11], suffix arrays [14], compact suffix arrays [13], suffix cacti [12], compressed suffix arrays [18,8], and so on.

Meanwhile, the problem finding a *subsequence* pattern has also been widely studied. We say a pattern p is a subsequence of a text w if p can be obtained by removing zero or more characters from w. By means of the directed acyclic subsequence graph (DASG) for w, we can examine whether or not p is a subsequence of w in $O(|p|)$ time [2,6]. An *episode* pattern is a "length-bounded" version of a subsequence pattern [15]. An episode pattern is given in the form of a pair of a string p and an integer k, as $\langle p, k \rangle$. If p is a subsequence of x such that x is a substring of w with $|x| \leq k$, we say that the episode pattern $\langle p, k \rangle$ matches w. The episode directed acyclic subsequence graphs (EDASGs) were introduced in [19], for a practical solution to the problem.

Now we propose a new kind of pattern matching problem: *Given a text string* $w = w_1 w_2 \cdots w_n$ *(*$w_i \in \Sigma$*), a string* p *and an integer* k*, examine whether or not* p *is a substring of* $w[k:]$ *where* $w[k:] = w_k \ldots w_n$*.* (NOTE: if $k > |w|$, the answer is always NO.) We name the pattern $\langle p, k \rangle$ a *beginning-sensitive pattern*, a *BS-pattern* for short. For any string $w \in \Sigma^*$ $DAWG(w)$ denotes the DAWG of w. Using the DAWGs for all suffixes of w, this problem is solvable in $O(|p|)$ time. This simple collection of the DAWGs is called the *naive all-suffixes directed acyclic word graph* for w, written as the naive $ASDAWG(w)$. Since the size of $DAWG(w)$ is $O(|w|)$, that of the naive $ASDAWG(w)$ is $O(|w|^2)$.

In this paper we introduce a new composite data structure, named the *minimum* $ASDAWG(w)$ and denoted by $MASDAWG(w)$. $MASDAWG(w)$ is the minimization of the naive $ASDAWG(w)$. We show that the size of $MASDAWG(w)$ is $\Theta(|w|)$ if $|\Sigma| = 1$, and $\Theta(|w|^2)$ if $|\Sigma| \geq 2$. Also, we produce an *on-line* algorithm that *directly* constructs $MASDAWG(w)$ in time linear in the size of $MASDAWG(w)$.

We show further two applications of $MASDAWG(w)$, one of which is as follows. Let $\Pi = (\Sigma \cup \{\star\})^*$, where $\star$ is a *wildcard* that matches any string. A pattern $q \in \Pi$ such as $q = a{\star}ba{\star}c$ is called a *variable-length-don't-care's pattern* (*VLDC-pattern*), where $a, b \in \Sigma$. The *language* $L(q)$ of a pattern $q \in \Pi$ is the set of strings obtained by replacing $\star$'s in q with strings. For example, $L(a{\star}ba{\star}c) = \{aubavc \mid u, v \in \Sigma^*\}$. This language corresponds to a class of the *pattern languages* proposed by Angluin [1]. We declare that the smallest automaton to recognize all possible VLDC-patterns matching a text w is a variant of $MASDAWG(w)$.

Finding a good rule to separate given two sets of strings, often referred to as *positive examples* and *negative examples*, is a critical task in knowledge discovery and data mining. In [9], an efficient method, with which a subsequence pattern is considered as a rule for the separation, was given, and in [10] one using an episode pattern was proposed. $MASDAWG(w)$ is believed certainly to be a good "weapon" to develop a practical algorithm to find the best VLDC-patterns to distinguish given two sets of strings efficiently. In fact, our experimental result has shown that the average size of the MASDAWGs for random texts of length 1 to 500 over a binary alphabet is proportional to $|w|^{1.24}$, in spite of the theoretical space complexity, $\Theta(|w|^2)$.

2 All-Suffixes Directed Acyclic Word Graphs

Strings x, y, and z are said to be a *prefix*, *substring*, and *suffix* of string $w = xyz$, respectively. The sets of prefixes, substrings, and suffixes of a string w are denoted by $Prefix(w)$, $Sub(w)$, and $Suffix(w)$, respectively. The empty string is denoted by ε, that is, $|\varepsilon| = 0$. Let $\Sigma^+ = \Sigma^* - \{\varepsilon\}$. The substring of a string w that begins at position i and ends at position j is denoted by $w[i:j]$ for $1 \leq i \leq j \leq |w|$. For convenience, let $w[i:j] = \varepsilon$ for $j < i$. Let $w[i:] = w[i:|w|]$ for $1 \leq i \leq |w|+1$. Assume S is a subset of Σ^*. For any string $u \in \Sigma^*$, $u^{-1}S = \{x \mid ux \in S\}$.

Let $w \in \Sigma^*$. We define an equivalence relation $\equiv_w$ on Σ^* by

$$x \equiv_w y \Leftrightarrow x^{-1}\mathit{Suffix}(w) = y^{-1}\mathit{Suffix}(w).$$

Let $[x]_w$ denote the equivalence class of a string $x \in \Sigma^*$ under $\equiv_w$. The longest element in the equivalence class $[x]_w$ for $x \in \mathit{Sub}(w)$ is called its *representative*.

Definition 1 (Directed Acyclic Word Graph (DAWG)). *$\mathit{DAWG}(w)$ is the dag (V, E) such that*

$$V = \{[x]_w \mid x \in \mathit{Sub}(w)\},$$
$$E = \{([x]_w, a, [xa]_w) \mid x, xa \in \mathit{Sub}(w) \text{ and } a \in \Sigma\}.$$

If we designate the node $[\varepsilon]_w$ of $\mathit{DAWG}(w)$ as the initial state and the nodes $[x]_w$ with $x \in \mathit{Suffix}(w)$ as the final states, then the resulting automaton is the smallest automaton that accepts the set $\mathit{Suffix}(w)$ [5].

Definition 2 (All-Suffixes DAWG (ASDAWG)). *$\mathit{ASDAWG}(w)$ is a kind of deterministic automaton with $|w|+1$ initial nodes, designated by integers $0, 1, \ldots, |w|$, in which the subgraph consisting of the nodes reachable from the k-th initial node and of their out-going edges is $\mathit{DAWG}(w[k+1:])$.*

The simple collection of $\mathit{DAWG}(w[1:])$, $\mathit{DAWG}(w[2:])$,..., $\mathit{DAWG}(w[n])$, $\mathit{DAWG}(w[n+1:])$ ($n = |w|$) is an example of $\mathit{ASDAWG}(w)$, referred to as the *naive* $\mathit{ASDAWG}(w)$. The number of nodes of the naive $\mathit{ASDAWG}(w)$ is $O(|w|^2)$. By minimizing the naive $\mathit{ASDAWG}(w)$, we can obtain the *minimum* $\mathit{ASDAWG}(w)$, which is denoted by $\mathit{MASDAWG}(w)$. The naive $\mathit{ASDAWG}(abba)$ and $\mathit{MASDAWG}(abba)$ are shown in Fig. 1. The minimization is performed based on the equivalence relation defined as follows. Let an ordered pair $\langle u, [x]_u \rangle$ denote a node $[x]_u$ of $\mathit{DAWG}(u)$. Each node of the naive $\mathit{ASDAWG}(w)$ can be represented by a pair $\langle u, [x]_u \rangle$ with $u \in \mathit{Suffix}(w)$ and $x \in \mathit{Sub}(u)$. The equivalence relation, denoted by $\sim_w$, is defined by

$$\langle u, [x]_u \rangle \sim_w \langle v, [y]_v \rangle \Leftrightarrow x^{-1}\mathit{Suffix}(u) = y^{-1}\mathit{Suffix}(v) \; .$$

A node of $\mathit{MASDAWG}(w)$ corresponds to an equivalence class under $\sim_w$. We write $\langle u, [x]_u \rangle$ simply as $\langle u, [x] \rangle$ in case no confusion occurs.

Proposition 1. *Let $u \in \mathit{Suffix}(w)$. Let x be a nonempty substring of u. We factorize u as $u = hxt$ and assume h is the shortest such string. Then, $\langle hxt, [x] \rangle$ is equivalent to $\langle sxt, [x] \rangle$ for every suffix s of h. (NOTE: The string x is not necessarily the representative of $[x]_u$.)*

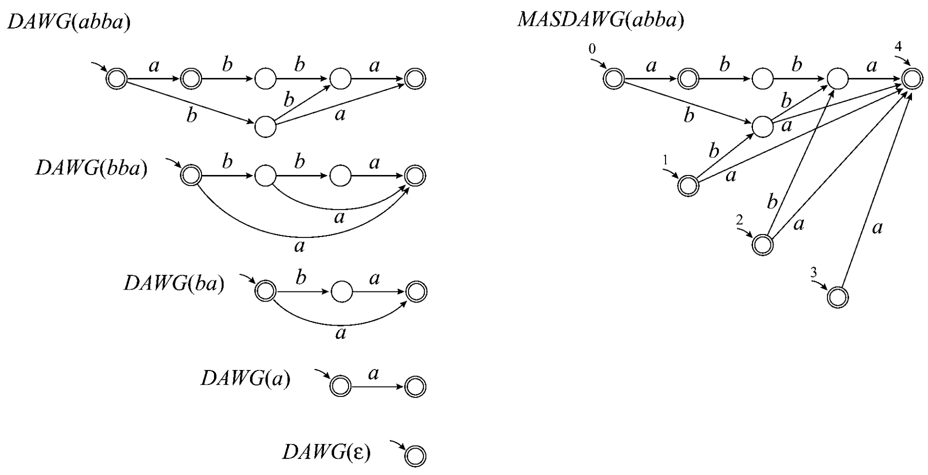

Fig. 1. $DAWG(x)$ for each string $x \in \mathit{Suffix}(w)$ is shown on the left, where $w = abba$. The collection of them is the naive $ASDAWG(w)$. On the right $MASDAWG(w)$ is displayed. While there are in total 16 nodes and 16 edges in the former, there are 9 nodes and 12 edges in the latter. For example, nodes $\langle abba, [b] \rangle$ and $\langle bba, [b] \rangle$ are equivalent due to Case 1 of Lemma 3 and merged into one. Also, $\langle abba, [abb] \rangle$, $\langle bba, [bb] \rangle$, and $\langle ba, [b] \rangle$ are merged into one node, where the first two are equivalent due to Case 2 and the last two are equivalent due to Case 3. The upper four sink nodes are equivalent due to Case 2 and the lowest one is equivalent to them (see Lemma 2), and therefore the five are merged into the same sink node

Let $h_0, h_1, \ldots, h_r$ be the suffixes of the string h arranged in the decreasing order of their length. The above proposition implies an existence of the chain of equivalent nodes

$$\langle h_0 xt, [x] \rangle, \langle h_1 xt, [x] \rangle, \ldots, \langle h_r xt, [x] \rangle.$$

In case more than one string belong to $[x]_u$, the chain length r is maximized by choosing the shortest one as x. The chain, however, does not necessarily break at the node $\langle h_r xt, [x] \rangle$. The shortest string in $[x]_u$ is not necessarily the shortest in $[x]_{h_r xt}$: Shorter one may exist. Thus we need a more precise discussion.

Lemma 1. *Let $h \in \Sigma^+$ and $u, hu \in \mathit{Suffix}(w)$. If a node of $DAWG(u)$ is equivalent to some node of $DAWG(hu)$, then it is also equivalent to some node of $DAWG(au)$ where a is the last character of the string h.*

Proof. Let $h = ta$ ($t \in \Sigma^*$). Assume $t \neq \varepsilon$. Let $x \in \mathit{Sub}(u)$ with $x \neq \varepsilon$, and $y \in \mathit{Sub}(tau)$ with $y \neq \varepsilon$. Assume $x^{-1}\mathit{Suffix}(u) = y^{-1}\mathit{Suffix}(tau)$. We have two cases to consider.

– $x \equiv_u y$. In this case, every occurrence of the string y within tau must be included within the u part. Thus, we have $x^{-1}\mathit{Suffix}(u) = y^{-1}\mathit{Suffix}(au)$.

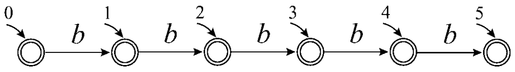

Fig. 2. *MASDAWG*(w) for $w = b^5$. For every $i = 0, 1, \ldots, 4$, the initial node $[\varepsilon]_{b^i}$ of *DAWG*(b^i) is equivalent to the node $[b]_{b^{i+1}}$ of *DAWG*(b^{i+1})

- $x \not\equiv_u y$. In this case, (1) y is written as $y = sx$ where s is a nonempty string, and (2) there is an occurrence of y within tau that covers the boundary between a and u but the x part of the occurrence of $y = sx$ is contained in the u part of the string tau. In this case, by truncating an appropriate length prefix of s we can obtain a string z as a suffix of $y = sx$ such that $x^{-1}\mathit{Suffix}(u) = z^{-1}\mathit{Suffix}(au)$.

The proof is now complete. □

The above lemma guarantees that the DAWGs sharing one node of *MASDAWG*(w) are 'consecutive'. We therefore concentrate on the relation between two consecutive DAWGs. First, we consider the equivalence of the initial node.

Lemma 2. *Suppose $b \in \Sigma$ and $u, bu \in \mathit{Suffix}(w)$. Let $y \in \mathit{Sub}(bu)$ and assume y is the representative of $[y]_{bu}$. Then, the nodes $\langle u, [\varepsilon]\rangle$ and $\langle bu, [y]\rangle$ are equivalent under $\sim_w$ if and only if $y = b$ and u is of the form b^ℓ with $\ell \geq 0$.*

See, for example, *MASDAWG*$(bbbbb)$ shown in Fig. 2.

As an extreme case of Lemma 2 where $\ell = 0$, the node $[\varepsilon]_\varepsilon$ of *DAWG*(ε) is always equivalent to the sink node $[b]_b$ of the previous *DAWG*(b).

Next, we consider the equivalence of nodes other than the initial node.

Lemma 3. *Suppose $b \in \Sigma$ and $u, bu \in \mathit{Suffix}(w)$. Let $x \in \mathit{Sub}(u)$ with $x \neq \varepsilon$. Let $y \in \mathit{Sub}(bu)$ with $y \neq \varepsilon$. Assume x and y are the representatives of $[x]_u$ and $[y]_{bu}$, respectively. The equivalence $\langle u, [x]\rangle \sim_w \langle bu, [y]\rangle$ implies that if $y \in \mathit{Prefix}(bu)$ then $y = bx$ and $x \in \mathit{Prefix}(u)$, and otherwise $y = x$. Moreover, $\langle u, [x]\rangle \sim_w \langle bu, [y]\rangle$ holds if and only if either*

(Case 1) *$x \notin \mathit{Prefix}(bu)$ and $y = x$;*
(Case 2) *$x \in \mathit{Prefix}(u)$, $x \equiv_{bu} y$, and $y = bx$; or*
(Case 3) *$x = b^i$, $y = b^{i+1}$, and u is of the form $b^\ell s$ such that $i \leq \ell$, and $s \in \Sigma^*$ does not begin with b nor contain an occurrence of b^i.*

Proof. Suppose $x^{-1}\mathit{Suffix}(u) = y^{-1}\mathit{Suffix}(bu)$. Let $u[i+1:]$ $(0 < i \leq |u|)$ be the longest member of this set.

1. When $y \in \mathit{Prefix}(bu)$. Then, $i = |y| - 1$ and $y = by'$ with $y' = u[1:i]$. Since $u[i+1:] \in x^{-1}\mathit{Suffix}(u)$, we have $u = hxu[i+1:]$ for some $h \in \Sigma^*$. Namely, x is a suffix of $y' = u[1:i]$.

(a) When $y' \notin \mathit{Prefix}(bu)$. We have $y \equiv_{bu} y'$ and

$$(y')^{-1}\mathit{Suffix}(u) \subseteq (y')^{-1}\mathit{Suffix}(bu) = y^{-1}\mathit{Suffix}(bu) \subseteq x^{-1}\mathit{Suffix}(u).$$

It derives from the assumption that $y^{-1}\mathit{Suffix}(bu) = x^{-1}\mathit{Suffix}(u)$. Thus, $(y')^{-1}\mathit{Suffix}(u) = x^{-1}\mathit{Suffix}(u)$, i.e., $x \equiv_w y'$. Since $y' \in \mathit{Prefix}(u)$, y' must be the representative of $[y']_u = [x]_u$. Consequently, we have $x = y'$.

(b) When $y' \in \mathit{Prefix}(bu)$. String y' is a prefix of $y = by'$, and therefore has a period of 1. Hence we have $y' = b^i$ and $y = b^{i+1}$. Since x is a suffix of $y' = b^i$, $x = b^j$ for some j with $0 < j \leq i$. If $j < i$, then $u[j+1:] \in x^{-1}\mathit{Suffix}(u)$, a contradiction. Thus we have $j = i$, i.e., $x = b^i$. On the other hand, $u[1:i] = y' = b^i$ and thus u is of the form $b^\ell s$ such that $\ell \geq i$ and $s \in \Sigma^*$ does not begin with b. We can show that the string s cannot contain an occurrence of $x = b^i$.

Note that we have $x \in \mathit{Prefix}(u)$ in both cases.

2. When $y \notin \mathit{Prefix}(bu)$. We have $y^{-1}\mathit{Suffix}(u) = y^{-1}\mathit{Suffix}(bu) = x^{-1}$ $\mathit{Suffix}(u)$, which implies $x \equiv_u y$. From the choice of x, y must be a suffix of x and $x = \delta y$ with $\delta \in \Sigma^*$. Assume, for a contradiction, that $x^{-1}\mathit{Suffix}(bu) \neq x^{-1}\mathit{Suffix}(u)$. Then there must be a suffix $u[j+1:]$ of u such that $j < i$ and $bu = hxu[j+1:]$ with $h \in \Sigma^*$. Since $x = \delta y$, we have $bu = h\delta yu[j+1:]$, which implies $u[j+1:] \in y^{-1}\mathit{Suffix}(bu)$, a contradiction. Hence we have $x \equiv_{bu} y$. From the choice of y, x must be a suffix of y. Thus we have $x = y$.

□

It should be noted that Case 1 and Case 2 of Lemma 3 fit to Proposition 1, whereas Case 3 is irregular in the sense that the two equivalence classes $[x]_u$ and $[y]_{bu}$ have no common member despite $\langle u, [x]\rangle \sim_w \langle bu, [y]\rangle$. See Fig. 1, which includes instances of Case 1, Case 2, and Case 3.

The *owner* of a node of $\mathit{MASDAWG}(w)$ is defined by $\mathit{DAWG}(w[k:])$ such that k is the smallest integer for which $\mathit{DAWG}(w[k:])$ shares the node. We are now ready to estimate the lower bound of the number of nodes of $\mathit{MASDAWG}(w)$.

Theorem 1. *When $|\Sigma| \geq 2$, the number of nodes of $\mathit{MASDAWG}(w)$ for a string w is $\Theta(|w|^2)$. It is $\Theta(|w|)$ for a unary alphabet.*

Proof. The proof for the case of a unary alphabet $\Sigma = \{a\}$ is not difficult. We can use Lemma 2. We now prove the lower bound in case $|\Sigma| \geq 2$. Let us consider string $w = (ab)^m(ba)^m$, where a, b are distinct characters from Σ. For each $i = 2, \ldots, m-1$, let $u_i = (ab)^i(ba)^m$. Let $x = (ba)^j$ with $0 < j < i$. It is not difficult to show that $x \not\equiv_{u_i} ax$ and $x \not\equiv_{u_i} b^{-1}x$, and therefore $[x]_{u_i} = \{x\}$. Thus x is the representative of $[x]_{u_i}$, and we can use the above lemma. Since $x \in \mathit{Prefix}(bu_i)$, $x \notin \mathit{Prefix}(u_i)$, and the first character of u_i is not b, none of the three conditions is satisfied, and therefore $\mathit{DAWG}(u_i)$ is the owner of the node corresponding to $[x]_{u_i}$. Thus, the nodes of $\mathit{MASDAWG}(w)$ corresponding to

$$[(ba)^1]_{u_i}, [(ba)^2]_{u_i}, \ldots, [(ba)^{i-1}]_{u_i}$$

are distinct and are owned by $DAWG(u_i)$. For each i with $1 < i < m$, $DAWG(u_i)$ has at least $i-1$ own nodes. Thus, $MASDAWG(w)$ has $\Omega(m^2) = \Omega(|w|^2)$ nodes. □

3 Construction

Since the construction of the naive $ASDAWG(w)$ takes $O(|w|^2)$ and the minimization can be performed in time linear in the number of edges of the naive $ASDAWG(w)$ (see [17]), we can build $MASDAWG(w)$ in $O(|w|^2)$ time. On the other hand, we have shown that the number of nodes in $MASDAWG(w)$ is $\Theta(|w|^2)$. We are therefore interested in on-line and direct construction of $MASDAWG(w)$. We obtained the following result.

Theorem 2. *$MASDAWG(w)$ can be constructed directly and on-line in linear time with respect to its size.*

The algorithm for the on-line construction of $MASDAWG(w)$ basically simulates the on-line constructions of the DAWGs for all suffixes of a string w. Fig. 3 illustrates the on-line construction of $MASDAWG(abbab)$.

We present a basic idea of the algorithm together with showing several lemmas which support it.

3.1 Suffix Link

In the construction, the *suffix links* play a key role. One main difference compared with constructing a single DAWG is that a node may have more than one suffix link. This is because $MASDAWG(w)$ may contain two distinct, equivalent nodes $\langle u, [x]\rangle$ and $\langle v, [y]\rangle$ such that the node to which the suffix link of $\langle u, [x]\rangle$ points is not equivalent to the one to which the suffix link of $\langle v, [y]\rangle$ points. We update $MASDAWG(w)$ into $MASDAWG(wa)$ as if the underlying DAWGs for $w[1:], w[2:], \ldots$ were updated simultaneously, as follows. Conceptually, we reserve all suffix links of these DAWGs, by associating each suffix link with the corresponding DAWG. Whenever two or more suffix links are duplicated, the corresponding DAWGs are consecutive due to Lemma 1. Therefore we can handle them at once. This is critical for linear-time performance of our algorithm. We traverse the dag induced by the suffix links rooted from the sink node, in the order of the corresponding DAWGs, and process each encountered node appropriately (creating a new edge to the new sink node, separating the node, or redirecting an edge to the separated node).

3.2 Compact Representation of Node Length Information

In the on-line construction of the DAWG for a single string, there occurs an event so-called *node separation* [3]. Formally, this event is described as follows. We store in each node $[x]_w$ of $DAWG(w)$ its *length*, namely, the length of the

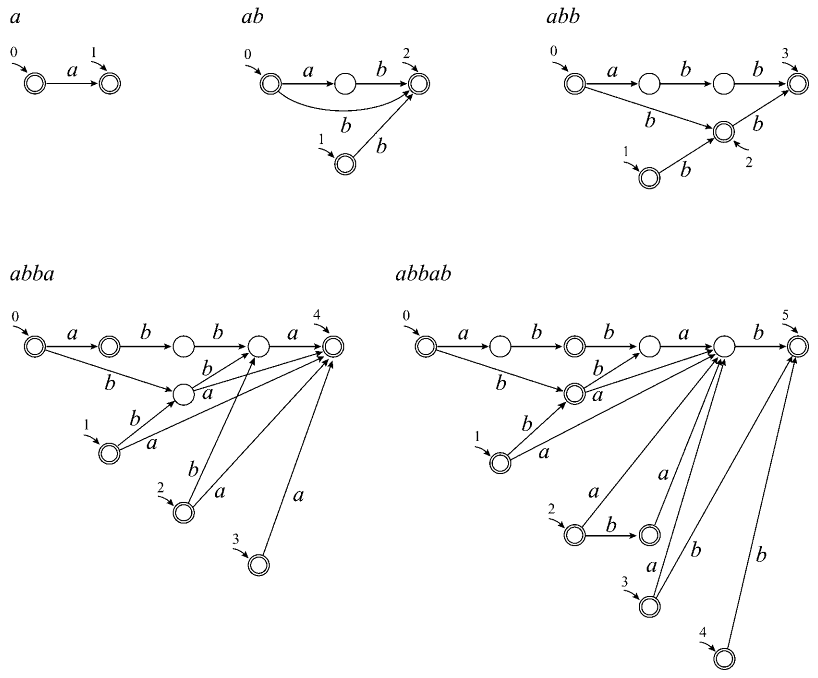

Fig. 3. On-line construction of *MASDAWG*(w) for $w = abbab$. Each initial node becomes independent whenever a newly appended character violates the condition of Lemma 2. Node separation of other type occurs only twice. One happens during the update of *MASDAWG*(ab) to *MASDAWG*(abb). The node consisting of $\langle ab, [ab]\rangle$ and $\langle b, [b]\rangle$ is separated into two nodes. This is regarded as a node separation in *DAWG*(abb). The other occurs during the update of *MASDAWG*$(abba)$ to *MASDAWG*$(abbab)$. The node consisting of $\langle abba, [abb]\rangle$, $\langle bba, [bb]\rangle$, and $\langle ba, [b]\rangle$ is separated into two. This is a special case in the sense that no node separation occurs inside any of *DAWG*$(abba)$, *DAWG*(bba), and *DAWG*(ba) (See the first case of Lemma 7.) (Note: Though each accepting node is double-circled in any step in this figure, we do not maintain it on-line. After the construction of *MASDAWG*(w) is completed, we mark every node reachable by the suffix-links-traversal from the sink node.)

representative of $[x]_w$. Consider updating *DAWG*(w) to *DAWG*(wa) where a is a character. Let z be the longest suffix of wa that also occurs within w. We call it the *longest repeated suffix* of wa. A node separation happens iff z is not the representative of $[z]_w$. The node $[z]_w$ can be detected by traversing the suffix link chain from the sink node of *DAWG*(w) in order to find its parent node $[z']_w$, which is the first encountered node on the chain that has an out-going edge labeled by a. Whenever the length of $[z]_w$ is greater than that of its parent $[z']_w$

plus one, the node $[z]_w$ of $DAWG(w)$ is separated into two nodes $[x]_{wa}$ and $[z]_{wa}$ in $DAWG(wa)$, where x is the representative of $[z]_w$.

Recall that a node of $MASDAWG(w)$ corresponds to an equivalence class under the equivalence relation $\sim_w$, and therefore two or more DAWGs may share a node of $MASDAWG(w)$. We need to know the length of the corresponding node of an arbitrary one of them. Naive solution would be to store into a node of $MASDAWG(w)$ a $(|w|+1)$-tuple of integers, the i-th value of which indicates the length of the corresponding node of the i-th DAWG, where $i = 0, 1, \ldots, |w|$. The overall space requirement is, however, proportional to $|w|^3$. Below we give an idea of compact representation of the tuple.

Lemma 4. *Let $\langle w[i+1:], [x_1]\rangle, \ldots, \langle w[i+\ell:], [x_\ell]\rangle$ be the nodes of the naive $ASDAWG(w)$ which are merged into a single node in $MASDAWG(w)$, where $0 \le i$ and $i + \ell \le |w| + 1$. We assume each of the strings $x_1, \ldots, x_\ell$ is the representatives of the equivalence class of it. Then, there exists an integer k with $1 \le k \le \ell$ such that*

$$x_j = \begin{cases} x_k, & \text{if } 1 \le j \le k; \\ x_k[j-k+1:], & \text{if } k < j \le \ell. \end{cases}$$

(See Fig. 4.)

Proof. By Lemma 3. □

For example, $MASDAWG(abb)$ in Fig. 3 has a node consisting of $\langle abb, [b]\rangle$ and $\langle bb, [b]\rangle$. Also, $MASDAWG(abba)$ has a node consisting of $\langle abba, [abb]\rangle$, $\langle bba, [bb]\rangle$, and $\langle ba, [b]\rangle$.

It follows from the above lemma that the function, which takes an integer s as an input and returns $|x_s|$ if $1 \le s \le \ell$, can be represented as a quartet $(i, \ell, k, |x_k|)$, which requires only a constant space (or $O(\log |w|)$ space). The update procedure of the quartet for each node is basically apparent, except for the nodes to be separated.

3.3 Node Separation

Recall that two or more DAWGs can share one node in $MASDAWG(w)$, and each of them has a possibility of being separated into two nodes. This seems to complicate the update of $MASDAWG(w)$. However, we can readily show the following lemma.

Lemma 5. *Suppose $b \in \Sigma$ and $u, bu \in Suffix(w)$. Let $x \in Sub(u)$ with $x \ne \varepsilon$. Let $y \in Sub(bu)$ with $y \ne \varepsilon$. Assume x and y are the representatives of $[x]_u$ and $[y]_{bu}$, respectively. Suppose $\langle u, [x]\rangle \sim_w \langle bu, [y]\rangle$. Let $a \in \Sigma$, and let z be the longest repeated suffix of bua. Suppose $z \in [y]_{bu}$. If $|z| < |y|$, then z is also the longest repeated suffix of ua, and $z \in [x]_u$. If $|z| = |y|$, then x is a repeated suffix of ua (not necessarily to be the longest).*

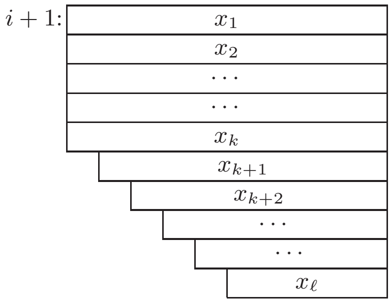

Fig. 4. The representatives x_j of $[x_j]_{w[i+j:]}$ such that the nodes $\langle w[i+j:], [x_j]\rangle$ of the naive $ASDAWG(w)$ are merged into a single node of $MASDAWG(w)$

The next lemma characterizes the node separations that occur during the update of $MASDAWG(w)$ to $MASDAWG(wa)$.

Lemma 6. *Consider the node of $MASDAWG(w)$ stated in Lemma 4 (see Fig. 4). Let z be the longest repeated suffix of $w[i+j:]a$. Suppose $z \in [x_j]_{w[i+j:]}$.*

1. *When $|z| = |x_k|$: Node separation occurs in none of the DAWGs for the strings $w[i+j:], \ldots, w[i+\ell:]$.*
2. *When $|z| < |x_k|$: Let t be the maximum integer such that z is a proper suffix of x_t. Node separation occurs in each of the DAWGs for the strings $w[i+j:], \ldots, w[i+t:]$. That is, for each $j = 1, \ldots, t$, the node $[x_j]_{w[i+j:]}$ of $DAWG(w[i+j:])$ is separated into $[x_j]_{w[i+j:]a}$ and $[z]_{w[i+j:]a}$ inside $DAWG(w[i+j:]a)$. The nodes $\langle w[i+j:]a, [x_1]\rangle, \ldots, \langle w[i+\ell:]a, [x_\ell]\rangle$ are equivalent under $\sim_{wa}$, and the new nodes $\langle w[i+j:]a, [z]\rangle, \ldots, \langle w[i+t:], [z]\rangle$ are also equivalent under $\sim_{wa}$.*

The node separations of DAWGs characterized in the above lemma lead to a node separation in the update of $MASDAWG(w)$ to $MASDAWG(wa)$. It simultaneously performs the node separations within each DAWG caused by the common z. (For the same z, we can take j as small as possible.)

The remaining problem to be overcome is that there is another kind of node separation in the update of $MASDAWG(w)$.

Lemma 7. *In the update of $MASDAWG(w)$ to $MASDAWG(wa)$, node separation of the following types may occur, where $w \in \Sigma^*$ and $a \in \Sigma$.*

1. *When $w[i+1:]$ is of the form $b^{\ell+1}s$ such that $w[i] \neq b$ or $i = 0$, $\ell \geq 1$, and $s \in \Sigma^*$ does not begin with b nor contain an occurrence of b^ℓ:*
   Assume that d is the largest integer such that s contains an occurrence of b^d. $MASDAWG(w)$ has a node consisting of

$$\langle w[i+j+1:], [b^{d+k}]\rangle, \langle w[i+j+2:], [b^{d+k-1}]\rangle, \ldots, \langle w[i+j+k], [b^{d+1}]\rangle,$$

where $k = \ell - (d + j) + 1$, for each $j = 0, 1, \ldots, d$. If $|s| > 0$, s ends with b^d, and $a = b$, then the node is separated into two nodes, one of which consists of

$$\langle w[i+j+1:]a, [b^{d+k}]\rangle, \langle w[i+j+2:]a, [b^{d+k-1}]\rangle, \ldots$$
$$, \langle w[i+j+k-1]a, [b^{d+2}]\rangle,$$

and the other consists only of $\langle w[i+j+k:]a, [b^{d+1}]\rangle$.

2. *When $w[i+1:]$ is of the form b^ℓ with $\ell \geq 1$ such that $w[i] \neq b$ or $i = 0$: MASDAWG(w) has a node consisting of*

$$\langle b^\ell, [b^j]\rangle, \langle b^{\ell-1}, [b^{j-1}]\rangle, \ldots, \langle b^{\ell-j}, [\varepsilon]\rangle,$$

for each $j = 1, \ldots, \ell$. Whenever $b \neq a$, the node is separated into two nodes, one of which consists of

$$\langle b^\ell a, [b^j]\rangle, \langle b^{\ell-1}a, [b^{j-1}]\rangle, \ldots, \langle b^{\ell-j+1}a, [b]\rangle,$$

and the other consists only of $\langle b^{\ell-j}a, [\varepsilon]\rangle$.

For an example of the first case of the above lemma, consider the update of *MASDAWG*(w) to *MASDAWG*(wb) for $w = bbbbbab$. The naive *ASDAWG*(w) and the naive *ASDAWG*(wb) are shown in Fig. 5, whereas *MASDAWG*(w) and *MASDAWG*(wb) are displayed in Fig. 6.

It should be emphasized that in the node separation mentioned in the above lemma no node separation occurs *inside a DAWG*. This kind of node separation can also be performed during the suffix link traversal started at the sink node.

4 Applications

In this section we show some applications to which the data structure ASDAWG and its variants effectively contribute.

4.1 Finding Beginning-Sensitive Patterns

Definition 3 (Beginning-Sensitive Pattern). *A* beginning-sensitive pattern *(a* BS-pattern *for short) is a pair $\langle p, i\rangle$ where p is a string in Σ^* and i is a positive integer.*

Definition 4 (BS-Pattern Matching Problem).
Instance*: a text w and a BS-pattern $\langle p, i\rangle$.*
Determine*: whether p is a substring of $w[i:]$.*

This is a natural extension of the substring pattern matching problem with $i = 1$. The BS-pattern matching problem is solvable in $O(|p|)$ time for an arbitrary pair $\langle p, i\rangle$, by using *ASDAWG*(w). For a given text w, we construct *MASDAWG*(w) with the on-line algorithm proposed in Section 3. For a BS-pattern $\langle p, i\rangle$, if $i > |w|$, the BS-pattern never matches w. Otherwise, we start with the i-th initial node of *MASDAWG*(w) and examine whether or not the string p is recognized.

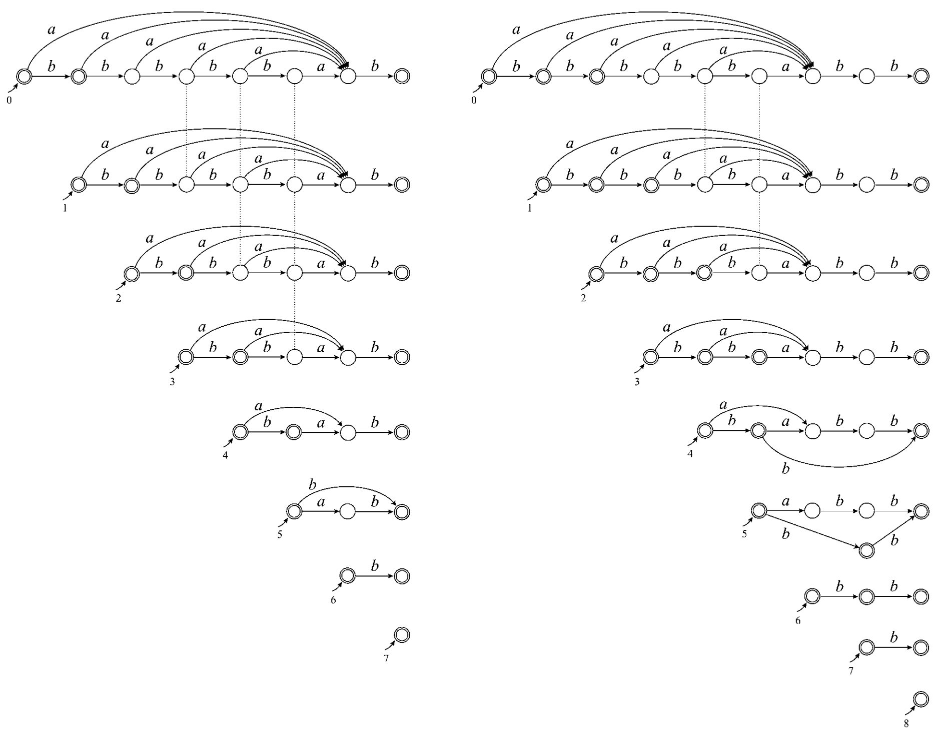

Fig. 5. The naive *ASDAWG*(*bbbbbab*) on the left, and the naive *ASDAWG*(w) on the right. The nodes connected by the dotted lines are equivalent due to Case 3. Recall the value of d mentioned in Lemma 7. In string *bbbbbab* the value of d is 1, whereas in string *bbbbbabb* $d = 2$ since the new b is added afterward

4.2 Pattern Matching within a Specific Region

Definition 5 (Region-Sensitive Pattern). *A* region-sensitive pattern *(an* RS-pattern *for short) is a triple* $\langle p, (i, j)\rangle$ *where* p *is a string in* Σ^* *and* i, j *are positive integers.*

Definition 6 (RS-Pattern Matching Problem).
Instance*: a text* w *and an RS-pattern* $\langle p, (i, j)\rangle$*.*
Determine*: whether* p *occurs within the region* $w[i : j]$ *in the text* w*.*

This is a natural extension of the BS-pattern matching problem in which $j = |w|$. For a given text w, we construct *MASDAWG*(w). We assign each node the integer for the position of the *rightmost occurrence* of the string corresponding to the node. For an RS-pattern $\langle p, (i, j)\rangle$, if $i > |w|$, the RS-pattern never matches w. Otherwise, we start with the i-th initial node of *MASDAWG*(w) and examine whether or not the string p is recognized. If it is recognized, we compare j with

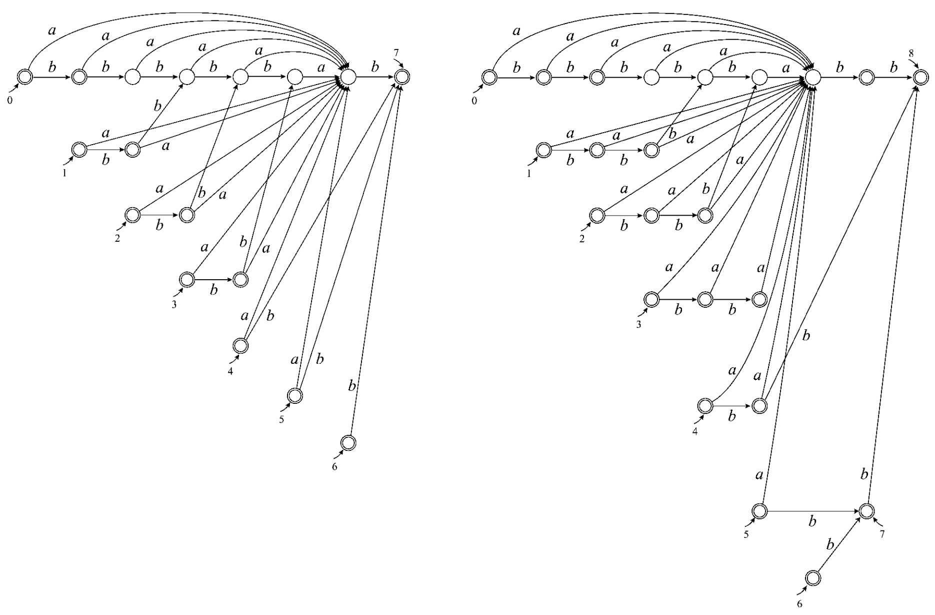

Fig. 6. $MASDAWG(bbbbbab)$ is on the left, and $MASDAWG(bbbbbabb)$ is on the right. Compare the update of $MASDAWG(bbbbbab)$ to $MASDAWG(bbbbbabb)$ with that of the naive $ASDAWG(bbbbbab)$ to the naive $ASDAWG(bbbbbabb)$ shown in Fig. 5

the integer k stored in the node at which p finally arrived. Then: If $j \leq k$, YES; Otherwise, NO. Obviously, the problem can be solved in $O(|p|)$ time.

4.3 Finding Variable-Length-Don't-Care's Patterns

Definition 7 (Variable-Length-Don't-Care's Pattern). *Let* $\Pi = (\Sigma \cup \{\star\})^*$*, where* $\star$ *is a* wildcard *that matches any string. An element* $q \in \Pi$ *is called a* variable-length-don't-care's pattern *(a* VLDC-pattern *for short).*

For instance, $\star a \star ab \star ba \star$ is a VLDC-pattern for $a, b \in \Sigma$. We say that a VLDC-pattern q matches a text string $w \in \Sigma^*$ if w can be obtained by replacing $\star$'s in q with some strings. In the running example, the VLDC-pattern $\star a \star ab \star ba \star$ matches text $ababab bbaa$ by replacing the $\star$'s with ab, b, b and a, respectively.

Definition 8 (VLDC-Pattern Matching Problem).
Instance*: a text* w *and a VLDC-pattern* q*.*
Determine*: whether* q *matches* w*.*

The smallest automaton to recognize all possible VLDC-patterns that match a text $w \in \Sigma^*$ is a variant of $MASDAWG(w)$. We call the automaton the *wildcard*

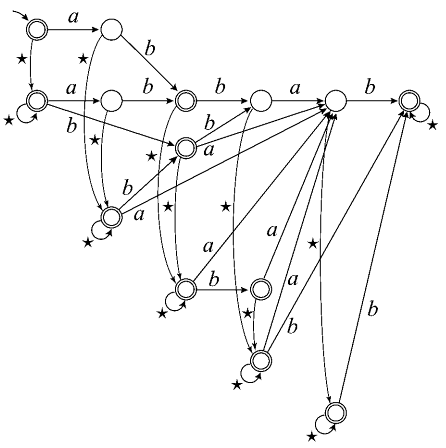

Fig. 7. *WDAWG*(w) where $w = abbab$

DAWG for w, and write it as *WDAWG*(w). *WDAWG*$(abbab)$ is displayed in Fig. 7. In *WDAWG*(w), a $\star$-transition is added between each node and the initial node of the "same layer" in *MASDAWG*(w) (see also *MASDAWG*$(abbab)$ in Fig. 3). Note that there exist two additional nodes, one of which is a unique initial node of *WDAWG*$(abbab)$. They are added in order that VLDC-patterns beginning with a can be recognized. For any $q \in \Pi$, the VLDC-pattern matching problem can be solved in $O(|q|)$ time, by using *WDAWG*(w).

References

1. D. Angluin. Finding patterns common to a set of strings. *J. Comput. Sys. Sci.*, 21:46–62, 1980. 154
2. R. A. Baeza-Yates. Searching subsequences (note). *Theoretical Computer Science*, 78(2):363–376, Jan. 1991. 153
3. A. Blumer, J. Blumer, D. Haussler, A. Ehrenfeucht, M. T. Chen, and J. Seiferas. The smallest automaton recognizing the subwords of a text. *Theoretical Computer Science*, 40:31–55, 1985. 153, 159
4. A. Blumer, J. Blumer, D. Haussler, R. McConnell, and A. Ehrenfeucht. Complete inverted files for efficient text retrieval and analysis. *J. ACM*, 34(3):578–595, 1987. 153
5. M. Crochemore. Transducers and repetitions. *Theoretical Computer Science*, 45:63–86, 1986. 153, 155
6. M. Crochemore and Z. Troníček. Directed acyclic subsequence graph for multiple texts. Technical Report IGM-99-13, Institut Gaspard-Monge, June 1999. 153
7. M. Crochemore and R. Vérin. On compact directed acyclic word graphs. In J. Mycielski, G. Rozenberg, and A. Salomaa, editors, *Structures in Logic and Computer Science*, volume 1261 of *LNCS*, pages 192–211. Springer-Verlag, 1997. 153

8. R. Grossi and J. S. Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. In *Proc. of 32nd ACM Symposium on Theory of Computing (STOC'00)*, pages 397–406, 2000. 153
9. M. Hirao, H. Hoshino, A. Shinohara, M. Takeda, and S. Arikawa. A practical algorithm to find the best subsequence patterns. In S. Arikawa and S. Morishita, editors, *Proc. The Third International Conference on Discovery Science*, volume 1967 of *LNAI*, pages 141–154. Springer-Verlag, 2000. 154
10. M. Hirao, S. Inenaga, A. Shinohara, M. Takeda, and S. Arikawa. A practical algorithm to find the best episode patterns. In K. P. Jantke and A. Shinohara, editors, *Proc. The Fourth International Conference on Discovery Science*, volume 2226 of *LNAI*, pages 435–440. Springer-Verlag, 2001. 154
11. S. Inenaga, H. Hoshino, A. Shinohara, M. Takeda, S. Arikawa, G. Mauri, and G. Pavesi. On-line construction of compact directed acyclic word graphs. In A. Amir and G. M. Landau, editors, *Proc. 12th Annual Symposium on Combinatorial Pattern Matching (CPM'01)*, volume 2089 of *LNCS*, pages 169–180. Springer-Verlag, 2001. 153
12. J. Kärkkäinen. Suffix cactus: A cross between suffix tree and suffix array. In Z. Galil and E. Ukkonen, editors, *Proc. 6th Annual Symposium on Combinatorial Pattern Matching (CPM'95)*, volume 973 of *LNCS*, pages 191–204. Springer-Verlag, 1995. 153
13. V. Mäkinen. Compact suffix array. In R. Giancarlo and D. Sankoff, editors, *Proc. 11th Annual Symposium on Combinatorial Pattern Matching (CPM'00)*, volume 1848 of *LNCS*, pages 305–319. Springer-Verlag, 2000. 153
14. U. Manber and G. Myers. Suffix arrays: A new method for on-line string searches. *SIAM J. Compt.*, 22(5):935–948, 1993. 153
15. H. Mannila, H. Toivonen, and A. I. Verkamo. Discovering frequent episode in sequences. In U. M. Fayyad and R. Uthurusamy, editors, *Proc. 1st International Conference on Knowledge Discovery and Data Mining*, pages 210–215. AAAI Press, Aug. 1995. 154
16. E. M. McCreight. A space-economical suffix tree construction algorithm. *J. ACM*, 23(2):262–272, Apr. 1976. 153
17. D. Revuz. Minimization of acyclic deterministic automata in linear time. *Theoretical Computer Science*, 92(1):181–189, Jan. 1992. 159
18. K. Sadakane. Compressed text databases with efficient query algorithms based on the compressed suffix array. In *Proc. of 11th International Symposium on Algorithms and Computation (ISAAC'00)*, volume 1969 of *LNCS*, pages 410–421. Springer-Verlag, 2000. 153
19. Z. Troníček. Episode matching. In A. Amir and G. M. Landau, editors, *Proc. 12th Annual Symposium on Combinatorial Pattern Matching (CPM'01)*, volume 2089 of *LNCS*, pages 143–146. Springer-Verlag, 2001. 154
20. E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995. 153
21. P. Weiner. Linear pattern matching algorithms. In *Proc. 14th Annual Symposium on Switching and Automata Theory*, pages 1–11, Oct. 1973. 153