Inclusion Problems in Trace Monoids *

Karine Shahbazyan
Institute for Informatics and Automation Problems
Yerevan, Armenia
e-mail: shahb@ipia.sci.am

Yuri Shoukourian
Institute for Informatics and Automation Problems
Yerevan, Armenia
e-mail: shouk@sci.am

ABSTRACT

We suggest a number efficient pattern matching algorithms for dependence graphs of traces. A graph $G = (V, E, \lambda)$ of partial order labeled by letters from alphabet Σ belongs to this class iff there exists a reflexive and symmetric relation $D \subset \Sigma \times \Sigma$ such that if $v <_G v'$ and $(\lambda(v).\lambda(v')) \in D \rightarrow (v, v') \in E$. We consider pattern matching problems related to problems of recognizing frequent patterns in structured data.

Keywords

Pattern matching, Trace Monoid, dependence graph.

1. INTRODUCTION

The efficient algorithms for strings are widely known [1-11]. In trace theory a string is regarded as a representative of a class of strings that are equivalent in the following sense. Given the independence relation on the alphabet two strings are equivalent if one can be received from the other by transposition of adjacent independent letters. In this paper a number string matching algorithms are considered for arbitrary dependence relation on the alphabet, that generalize usual string matching problems without independent letters. Problem of inclusion a string into another string in this context acquires the form of trace matching: given the strings x and y – the representatives of equivalence classes (traces) pattern $p = [x]$ and target $t = [y]$ correspondingly to decide whether exist such strings $x' \in p$ and $y' \in t$ that x' is subsequence of y'. The task is to find efficient algorithm solving the problem.

We study here two algorithms for inclusion problem. The first algorithm preprocesses on-line the fixed pattern $x \in p$ and then on-line scans the string $y \in t$ reading each symbol of y only once. The algorithm runs in time $O(|t| + |p|)$. The second algorithm does not call for preprocessing of the target or pattern. It scans concurrently the pattern and the target, reading each symbol of x only once, but returning to the beginning of the target string $y \in t$. This algorithm runs in time $O(|t||p|)$. Further we consider two problems related to problems of recognizing frequent patterns in structured data: 1) given the integer w count the number of w-trace-windows of string-representative of target t which include the trace-pattern; 2) count the number of minimal factors of target t where pattern p is included.

*This research has been partially supported by the grant ISTC A-1451.

2. PRELIMINARIES

In this section we provide necessary background information. The details can be found in [12].

Let Σ is a finite alphabet, $D \subseteq \Sigma \times \Sigma$ is a reflexive and symmetric **dependence** relation, $I = (\Sigma \times \Sigma) \setminus D$ is **independence** or **commutation** relation. The relation I induces an equivalence relation $\underset{I}{\sim}$ over Σ^*. Two strings $x, y \in \Sigma^*$ are equivalent under $\underset{I}{\sim}$ if there exists a sequence $z_1, \ldots, z_k$ of strings such that $x = z_1$, $y = z_k$, and for all i $(1 \leq i < k)$ there exist strings z_i', z_i'' and letters a_i, b_i satisfying

$$z_i = z_i' a_i b_i z_i''$$
$$z_{i+1} = z_i' b_i a_i z_i'' \qquad \text{and} \quad (a_i, b_i) \in I,$$

i.e., two strings are equivalent by $\underset{I}{\sim}$ if one can be obtained from the other by successive transpositions of neighboring independent letters. As is known the defined equivalence is a congruence and operation is defined as follows: $[s][t] = [st]$. Then the set $M(\Sigma, D)$ of equivalence classes of strings from Σ^* under $\underset{I}{\sim}$ is a **trace monoid,** and its elements are called **traces**. A trace t is denoted by $[x]$ for any representative string $x \in t$. The **length** $|t|$ of a trace t is the length of any of its representatives $x \in t$. For any string $x \in \Sigma^*$ we denote by $|x|$ the length of x, by $|x|_a$ the number of occurrences of the letter a in the string x.

In the sequel we need two representations of traces: representation by a tuple of strings and representation by dependence graph.

Every trace $t \in M(\Sigma, D)$ has unique representation as a labeled directed acyclic graph G_t defining a labeled partial order. Graph G_t is defined recursively: G_e is empty graph. $G_{t[a]}$ arises from the graph G_t by adding to it a node labeled with symbol a and new edges leading to it from all nodes of G_t labeled with symbols that a causally depends on. The graph G_t is called **dependence graph**.

The graph G_t can be related to the set of strings induced by all causal orderings of G_t. This set of strings composes the trace t.

The second representation of a trace - by a tuple of strings.

Let $\{\Sigma_1, \ldots, \Sigma_m\}$ be a clique cover of the dependence alphabet (Σ, D), that is a family of subsets of Σ such that

$$\bigcup_{i=1}^{m} \Sigma_i = \Sigma, \quad \Sigma_i \times \Sigma_i \subset D \quad (i = 1, 2, \ldots, m),$$
$$(a, b) \in D \;\Leftrightarrow\; \exists\, i : \; a, b \in \Sigma_i.$$

Then trace $t \in M(\Sigma, D)$ can be represented by a m-tuple of strings

$$\pi(t) = \{\pi_1(t), \ldots, \pi_m(t)\},$$

where $\pi_i(t) \in \Sigma_i^*$, and $\pi_i(t)$ is the projection of any string $y \in t$ to Σ_i [11].

Given two traces $p, t \in M(\Sigma, D)$ we say that p is a **prefix** of t if $t = pq$ where $q \in M(\Sigma, D)$. We denote by $Pref(t)$ the set of prefixes of trace t. If $t = pqr$ where $p, q, r \in M(\Sigma, D)$ then q is called a **factor** of t. Note that if $p \in Pref(t)$ then exist $x, y \in \Sigma^*$ such that $t = [y]$, $p = [x]$ and x is a string-prefix of y.

Let a trace t be given by its representation $\pi(t)$. Then we can represent any prefix s of t (with respect to $\pi(t)$) by the integer array denoted by $\underline{s} = (k_1, \ldots, k_m) \in N^m$ keeping in mind that $k_i = |\pi_i(s)|$, $(i = 1, \ldots, m)$ and $\pi_i(s)$ is string-prefix of $\pi_i(t)$ of length k_i. Thus $(0, 0, \ldots, 0) = \underline{e}$ represents the empty prefix e , while the whole trace t is represented by $\underline{t} = (n_1, \ldots, n_m)$ where $n_i = |\pi_i(t)|$ [5].

The **graph of prefixes** of the trace t is a directed acyclic graph $G^{pref}(t) = (V, U, \mu)$ with labeled edges, such that $V = \{\underline{s} \mid s \in Pref(t)\}$, U is the set of ordered pairs $(\underline{s}, \underline{r})$, μ is the edge labeling function and the edge $(\underline{s}, \underline{r})$ has label $a \in \Sigma$, if

$$s, r \in Pref(t),$$
$$s = r[a].$$

where $[a]$ is the trace that consists of single one-letter string a.

Below we need the definitions of automata, processing traces [11].

For the monoid $M = M(\Sigma, D)$ an M-**automaton** $A = (M, Q, \delta, q_0, F)$ consists of a finite set Q of states, an initial state $q_0 \in Q$, a subset $F \subseteq Q$ of final states, and a transition function δ from $Q \times M$ to Q satisfying the following conditions:

$\forall q \in Q \ : \ \delta(q, e) = q, \ \ (e \text{ is empty trace}),$
$\forall q \in Q, \ \forall t_1, t_2 \in M(\Sigma, D) \ : \ \delta(q, t_1 t_2) = \delta(\delta(q, t_1), t_2).$

The set of traces $T \subseteq M = M(\Sigma, D)$ recognized by an automaton A is defined by $T = \{t \in M \mid \delta(q_0, t) \in F\}$.

Zielonka's Automaton (asynchronous automaton over M) is a M-automaton $A = (M, Q, \delta, q_0, F)$ which satisfies the following additional conditions:

1. The state set Q is a cartesian product $Q = \prod_{i=1}^{n} Q_i$.
2. With each $a \in \Sigma$ there is an associated index set $K(a) \subseteq \{i \mid i \in \{1, 2, \ldots, n\}\}$ such that $K(a) \cap K(b) = \emptyset$ iff $(a, b) \in I$.
3. Transition mapping δ is given by a collection of partially defined mappings

$$\{\delta_a : \prod_{i \in K(a)} Q_i \to \prod_{i \in K(a)} Q_i\}_{a \in \Sigma},$$

¿From now on we fix the monoid $M = M(\Sigma, D)$, the clique cover $\{\Sigma_1, \ldots, \Sigma_m\}$ of (Σ, D) and the integer m- the number of cliques in the clique cover.

3. INCLUSION OF TRACES

We start with the definition of the term *inclusion of traces* and then study the inclusion problem.

Definition.

Trace p is **included** in trace t if for some $t_i, p_i \in M(\Sigma, D)$, $i = 0, \ldots, n$ holds

$$t = t_0 p_1 t_1 \ldots t_{n-1} p_n t_n, \tag{1}$$

$$p = p_1 p_2 \ldots p_n. \tag{2}$$

In this case we say also that t **contains** p and write $p \subset t$. The representation (1) we call **inclusion**.

Inclusion problem. Given two traces $p, t \in M(\Sigma, D)$, to decide whether p is included in t.

Let some r be a prefix of trace t : $r \in Pref(t)$. We denote by $R_t(r)$ the set of labels of edges outgoing from the node $\underline{r}$ in the graph of prefixes $G^{pref}(t)$. Hence $a \in R_t(r) \Leftrightarrow s = r[a] \in Pref(t)$.

To any $a \in \Sigma$ we associate an index set $J(a) = \{i \in \{1, 2, \ldots m\} | a \in \Sigma_i\}$ such as $(a, b) \in I \Leftrightarrow J(a) \cap J(b) = \emptyset$.

Statement 1. *Let a trace t be given by its representation $\pi(t) = \{\pi_1(t), \ldots, \pi_m(t)\}$ where $\pi_i(t) = \pi_{i,1}(t) \ldots \pi_{i,n_i}(t) \in \Sigma_i^*$.*

1.1 *If $r \in Pref(t)$, $z \in \Sigma^*$ then $s = r[z] \in Pref(t)$ iff there is a directed path in $G^{pref}(t)$ from $\underline{r}$ to $\underline{s}$ that carries the string z.*

A path in $G^{pref}(t)$ is a maximal path iff it carries some string $z \in t$.

1.2 *For any $r \in Pref(t)$ if $\underline{r} = (k_1, \ldots, k_m)$ then the set $R_t(r)$ may be computed as*

$$R_t(r) = \{a \mid i \in J(a) \Rightarrow \ \pi_{i,k_i+1} = a\}.$$

1.3 *For any $r \in Pref(t)$ and $a \in R_t(r)$ if $\underline{r} = (k_1, \ldots, k_m)$ then representation $\underline{s}=\underline{r[a]}= (k'_1, \ldots, k'_m)$ can be computed as*

$$k'_i = k_i \qquad \text{if} \quad i \notin J(a),$$

$$k'_i = k_i + 1 \qquad \text{if} \quad i \in J(a),$$

$$i = 1, 2, \ldots, m.$$

1.4 *For any $r \in Pref(t)$ and $(a, b) \in I$, $a \in R_t(r)$ and $s = r[a]$ holds:*

$$b \in R_t(s) \Leftrightarrow b \in R_t(r),$$

$$r[a][b] = r[b][a].$$

Let $p, t \in M(\Sigma, D)$. It is a question of construction two M-automata solving inclusion problem. The first automaton $A(p)$ is determined by the *pattern* p and recognizes the target $t \in M(\Sigma, D)$ iff p is **included** in t.

The second automaton $B(t)$ is determined by the *target* t and recognizes the pattern $p \in M(\Sigma, D)$ iff p is **included** in t.

We suppose $|t| > |p| > 1$.

Statement 2. *For any trace $p \in M(\Sigma, D)$ there exists a deterministic asynchronous Zielonka's automaton $A(p)$ that recognizes $t \in M(\Sigma, D)$ if and only if $p \subset t$.*

Statement 3. *There exists an on-line algorithm to solve inclusion problem $p \subset t$ with time complexity $O(m|t|)$ for any $p, t \in M(\Sigma, D)$ represented by strings $x \in p$ and $y \in t$. The space complexity is $O(m|p|)$.*

This algorithm serves for a base of the following counting algorithms in sections 3, 4.

Statement 4. *Let $p, t \in M(\Sigma, D)$.*

4.1. *$p \subset t$ iff the dependence graph G_p is a subgraph of dependence graph G_t.*

4.2. *$p \subset t$ iff for every string $y \in t$ there is a string $x \in p$ such that x is a subsequence of the string y.*

Proof. It is clear that 4.1 implies 4.2. Proof is straightforward from definitions.

Remark. The validity of $p \subset t$ does not imply that for every string $x \in p$ there is a string $y \in t$ such that x is a subsequence of y. Here is an example.

Example. Consider the monoid $M(\Sigma, D)$ where $D = \{(a, b), (b, c)\}$ $\{a, b, c\}$. Let $t = [abc]$. The trace t is composed of single string $y = abc$. If $p = [ac]$ then the string $ca \in p$, however the string ca is not subsequence of y.

Let $s, t \in M(\Sigma, D)$. By statement 4.1 and definition of inclusion, $s \subset t$ if and only if for any strings $x = a_1 \dots a_{|s|} \in s$, $y = b_1 \dots b_{|t|} \in t$ there exists an embedding of index sets $\varphi_{xy} : X_s \to X_t$ (here $X_s = \{1, \dots, |s|\}, X_t = \{1, \dots, |t|\}$) such that

1) $\varphi_{x,y}$ preserves the label : $a_i = b_{\varphi_{x,y}(i)}$,

2) $\varphi_{x,y}$ preserves the order of G_p:

$$\text{if } i <_{G_p} j \text{ and } (a_i, a_j) \in D \text{ then } \varphi_{x,y}(i) < \varphi_{x,y}(j). \quad (3)$$

We call such embedding **correct**.

Let an arbitrary $X \subseteq X_t$. Denote by $\overline{X} = \bigcup_{k=0}^{K} Y_k$, where $Y_0 = X$, $Y_k = \{i \in X_t \mid \exists j \in Y_{k-1} : \ i < j, \ (y_i, y_j) \in D\}$.

Statement 5. Let $s, t \in M(\Sigma, D)$, $x = a_1 \dots a_{|s|} \in s$, $y = b_1 \dots b_{|t|} \in t$.

If $s \subset t$, $a \in \Sigma$, then $s[a] \subset t$ if and only if

$$\exists\ i_0 \le |t| : \ b_{i_0} = a, \quad i_0 \notin \overline{\varphi_{xy}(X_s)}. \quad (4)$$

Algorithm 2 for inclusion problem.

In accordance to Statement 5, algorithm scans the string $x = a_1 \dots a_{|p|}$ reading each letter once and transforming the string y into a sequence of strings $y_0 = y, y_1, \dots, y_{|p|}$. On $i-$st step ($i = 1, \dots, |p|$) when reading a_i, the string y_{i-1} is transformed as follows: if $y_{i-1} = b_1 \dots b_n$ then the next string $y_i = b'_1 \dots b'_{n'}$ is obtained from y_{i-1} by deleting the letters of the set $\overline{\{b_{i_0}\}}$ where b_{i_0} is the first occurence of the letter a_i in the string y_{i-1}, i.e., $a_i = b_{i_0}$.

The time complexity of the algorithm is $O(|p||t|)$, the space complexity is $O(|t| + |p|)$.

Statement 6. *For any trace $t \in M(\Sigma, D)$ there exists a deterministic asynchronous Zielonka's automaton $B(t)$ that recognizes $p \in M(\Sigma, D)$ if and only if $p \subset t$.*

4. COUNTING w-TRACE-WINDOWS CONTAINING PATTERN

The problem. Given two traces p , $t \in M(\Sigma, D)$ represented by their representative-strings $x \in p$ and $y \in t$, count the number of w-trace-windows of $y \in t$ containing pattern trace p.

Definitions. Let the string $y = b_1 \dots b_n \in \Sigma^*$. A w- **string-window** of size w on string y is a substring $b_{i+1} \dots b_{i+w}$, $(i = 0, \dots, n - w)$ of length w. The trace $[b_{i+1} \dots b_{i+w}]$ is a w-**trace-window** of size w of the string y.

Remark. Evidently the number of w-windows containing p can be different for strings $y, y' \in t$ if $y \neq y'$ even if $y \underset{I}{\sim} y'$.

A **stamped prefix** of $p \in M(\Sigma, D)$ is an ordered pair $(\underline{r}, n)$ where $r \in Pref(p)$, $n \in N$. A **configuration** is a sequence of stamped prefixes of $p \in M(\Sigma, D)$.

Statement 7. *There exists an algorithm counting the number of w-trace-windows of string $y \in t$, containing trace p represented by a string $x \in p$. Time complexity of this algorithm is $O(mw|t|)$, the space-complexity is $O(|w||p|)$.*

Idea of algorithm. Scanning the string $y = b_1 \dots b_n \in \Sigma^*$ the algorithm constructs successive configurations C_i for every letter b_i of y . A stamped prefix $(\underline{r}, u)$ ($r \in Pref(p)$) enters the configuration C_i iff the trace-window $[b_{i-u+1} \dots b_i]$ of length $u \le w$ contains the prefix r. Clearly, given C_i and b_{i+1} it is possible to count C_{i+1}. If C_i contains a stamped prefix $(\underline{p}, u)$ where $u \le w$ then the w-trace-window $[b_{i-w+1} \dots b_i]$ contains p.

5. COUNTING THE NUMBER OF MINIMAL FACTORS CONTAINING PATTERN

Let (1) and (2) be valid, i.e., $p \subset t$. The factor $f = p_1 t_1 \dots t_{n-1} p_n$ of trace t containing p is **minimal** if each proper factor of the trace f does not contain p.

The problem. Given two traces $p, t \in M(\Sigma, D)$ represented by strings $x \in p$ and $y \in t$, count the number of different minimal factors of the trace t containing the trace p.

Statement 8. *There exists an algorithm counting the number of minimal factors of the trace t containing the trace p. The time complexity of the algorithm is $O(m|t|^2)$, the space-complexity is $O(|t|)$.*

Idea of algorithm. Let $t = qrs$ and $q, r, s, p \in M(\Sigma, D)$ and r be a minimal factor of t containing p. Then for each string $y = b_1 \dots b_{|t|} \in t$ there exists a trace-window (without restriction on the length) $[b_k \dots b_l]$ such that $p \subset r \subset [b_k \dots b_l]$.

Evidently, the number of minimal factors of the trace t if $t = [y]$ is equal to the number of such substrings of y whose proper trace-substrings do not contain p. Thus we need to find and count such minimal substrings. Scanning the string y, algorithm associates to each letter b_i of the string y a configuration

$$C_i = (\underline{r_1}, \dots, \underline{r_l}), \quad r_k \in Pref(p), \quad k = 1, \dots, l,$$

i.e., sequence of prefixes of $p \in M(\Sigma, D)$ (not the stamped prefixes as in the previous section) such that $r_k \subset [b_{v_k} \dots b_i]$ and $v_1 < \dots < v_l < i$. Suppose that for some $j \in \{1, \dots, l\}$ holds:

$$p \subset r_1, \dots, p \subset r_j, \ p \not\subset r_{j+1}, 1 \le j \le l. \tag{5}$$

Then all $[b_{v_k} \dots b_i]$ for $k \le j$ contain the same minimal trace-factor of the t , a trace-factor of y containing minimal trace-factor of t is found out and the counter must be increased by 1. It is possible to consider only trace-windows $[b_k \dots b_l]$ with $b_k \in R_p(e)$.

The configuration C_{i+1} for j, satisfying (5), is constructed as follows :

If $C_i = (\underline{r_1}, \dots, \underline{r_l})$ and $b_{i+1} = a$, put $s_k = r_{j+k}[a]$ if $r_{j+k}[a] \in Pref(p)$ and

$s_k = r_{j+k}$ if $r_{j+k}[a] \notin Pref(p)$ for $k = 1, \dots, l-j$. Then

$$C_{i+1} = (\underline{s_1}, \dots, \underline{s_{l-j}}) \text{ if } a \notin R_p(e), \ j < l,$$

$$C_{i+1} = (\underline{s_1}, \dots, \underline{s_{l-j}}, \underline{[a]}) \text{ if } a \in R_p(e), \ j < l$$

$$C_{i+1} = \{\underline{e}\} \text{ if } j = l, \ a \notin R_p(e),$$

$$C_{i+1} = \{\underline{[a]}\} \text{ if } j = l, \ a \in R_p(e),$$

where e is empty string.

So scanning the string y we can count the number of minimal factors.

Remark. The algorithm only counts the *number* of minimal factors, not minimal factors themselves. To obtain the minimal factors of t we need to accomplish the algorithm by a procedure that deletes from the minimal substrings redundant letters. This will increase the complexity to $O(m|t|^3)$.

REFERENCES

[1] Y.Chi, H.Wang, P.S.Yu, R.R.Muntz. Catch the moment: maintaining closed frequent itemsets over a date stream sliding window.// Knowledge and Information Systems,– 2006. – v 10, N 3.

[2] H-F.Li. M-K.Shan, S-Y.Lee. DSM-FI: an efficient algorithm for mining frequent itemsets in data streams.// Knowledge and Information Systems – 2008. – 17, N1 .

[3] C.Ba, M.H.Ferrari, M.A.Musicante. Composing Web Services with PEWS : a trace-theoretical approach.// 4-th European Conference on WEB Services. – 2006.

[4] H.Mannila, H.Toivonen, A Verkamo. Discovering Frequent Episodes in Sequences.// Proc. KDD Conf.– 1955.

[5] A.Avellone, M.Goldwurm. Analysis of algorithms for the recgnition of rational and context-free trace languages// Theoretical Informatics and Applications – 1998.– 32.

[6] Y.Chi, R.Muntz, S.Nijssen J.Kok. Frequent Subtree Mining. - An Overview//. Fundamenta Informatcae – 2001. –XXI .

[7] L.Boasson, P. Cegielski, I. Guessarian, Y.Matiyasevich. Window-Accumulated Subsequence Problem is linear.// Annals of Pure and Applied Logic. – 2001.– 113.

[8] P. Cegielski, I. Guessarian, Y.Matiyasevich. Tree Inclusion Problems.// RAIRO - Theoretical Informatics and Applications, 42, 5-20 (2008).

[9] I. Guessarian, P. Cegielski. Tree Inclusions in Windows and Slices.// CSIT Conference – 2000.

[10] J.Messner Pattern Matching in Trace Monoids.// Symposium on Theoretical Aspects of Computer Science. – 1997.

[11] V.Dikert, G.Rozenberg. The Book of Traces.// Handbook of formal languages, v. 3, Beyond Words, Springer-Verlag N.Y., 1997.