The Size of Subsequence Automaton

Zdeněk Troníček[1] * and Ayumi Shinohara[2,3]

[1] Department of Computer Science & Engineering, Faculty of Electrical Engineering
Czech Technical University, Prague
[2] Department of Informatics, Kyushu University 33, Fukuoka 812-8581, Japan
[3] PRESTO, Japan Science and Technology Corporation (JST)
tronicek@fel.cvut.cz, ayumi@i.kyushu-u.ac.jp

Abstract. Given a set of strings, the subsequence automaton accepts all subsequences of these strings. We will derive a lower bound for the maximum number of states of this automaton. We will prove that the size of the subsequence automaton for a set of k strings of length n is $\Omega(n^k)$ for any $k \geq 1$. It solves an open problem posed by Crochemore and Troníček [2] in 1999, in which only the case $k \leq 2$ was shown.

1 Introduction

A *subsequence* of a string T is any string obtainable by deleting zero or more symbols from T. Given a set P of strings, a *common subsequence* of P is a string that is a subsequence of every string in P. Motivation for study of subsequences comes from many domains, *e.g.* from molecular biology, signal processing, coding theory, and artificial intelligence.

An example of the problem with great practical impact is the longest common subsequence (LCS) problem. The problem is defined as follows: given a set P of strings, we are to find a common subsequence of P that has maximal length among all common subsequences of P. The decision version can be, for example, to decide whether a given string is a common subsequence of P. Another problem, which comes from artificial intelligence, is the problem of separating two sets of strings: given two sets P (positive) and N (negative) of strings, we are to find a string that best separates them. A string S separates sets P and N if S is a subsequence of P and simultaneously is not a subsequence of any string in N. The decision version is defined as follows: given two sets P and N of strings and a string S, we are to decide whether S separates P and N. If the problem is supposed to be answered for several strings S then it is sensible to preprocess the sets P and N. We can build the automaton accepting all common subsequences of P and the automaton that accepts any string that is a subsequence of at least one string in N. With these automata we can decide the problem in time linear in the length of S. Both automata were studied and described. The first one is called the Common Subsequence Automaton (CSA) and two algorithms for its building, off-line [2] and on-line [6], were designed. The second automaton is

* supported by GAČR grant 201/01/1433

known as the Directed Acyclic Subsequence Graph (DASG) and three building algorithms are available: right-to-left [1], left-to-right [2], and on-line [4].

In this paper, we investigate the number of states of the CSA. As the language accepted by the CSA is a subset of the language accepted by the DASG for the same strings, the automata are very similar. If we use the off-line or on-line algorithm, the set of states of the CSA is a subset of states of the DASG. The only previous results are, according to our knowledge, the lower bound for the maximum number of states of the DASG for two strings proved in [3] and of the CSA for two strings derived in [6]. We will prove the same lower bound for any (fixed) number of strings. The paper is organized as follows. In Section 2 we will recall the definition of the CSA from [2] and in Section 3 we will examine the asymptotic behaviour of the number of states of the CSA in the worst case.

Let Σ be a finite alphabet of size σ. A finite automaton is, in this paper, a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is an input alphabet, $\delta : Q \times \Sigma \to Q$ is a transition function, q_0 is the initial state, and $F \subseteq Q$ is the set of final states. Notation $\langle i, j \rangle$ means the interval of integers from i to j, including both i and j. All strings in this paper are considered on alphabet Σ, if not stated otherwise.

2 Definition of CSA

Let P denote the set of strings $T_1, T_2, \ldots, T_k$. Let n_i be the length of T_i and $T_i[j]$ be j-th symbol of T_i for all $j \in \langle 1, n_i \rangle$ and all $i \in \langle 1, k \rangle$. Given $T = t_1 t_2 \ldots t_n$ and $i, j \in \langle 1, n \rangle, i \leq j$, notation $T[i \ldots j]$ means the string $t_i t_{i+1} \ldots t_j$.

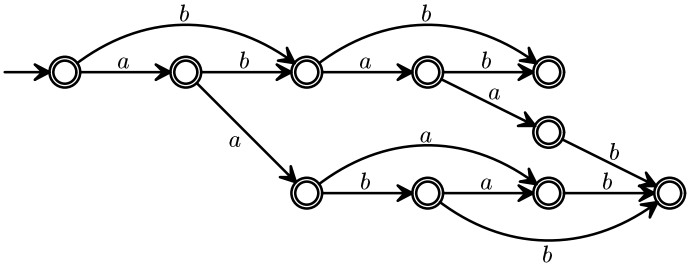

Fig. 1. The CSA for strings *ababab* and *aabaab*.

Definition 1. *We define a position point of the set P as an ordered k-tuple $[p_1, p_2, \ldots, p_k]$, where $p_i \in \langle 0, n_i \rangle$ is a position in string T_i. If $p_i \in \langle 0, n_i - 1 \rangle$ then it denotes the position in front of $(p_i + 1)$-th symbol of T_i, and if $p_i = n_i$ then it denotes the position behind the last symbol of T_i for all $i \in \langle 1, k \rangle$.*

A position point $[p_1, p_2, \ldots, p_k]$ is called *initial position point* if $p_i = 0$ for all $i \in \langle 1, k \rangle$. We denote by $ipp(P)$ the initial position point of P and by $Pos(P)$ the set of all position points of P.

Definition 2. *For a position point $[p_1, p_2, \ldots, p_k] \in Pos(P)$ we define the common subsequence position alphabet as the set of all symbols which are contained simultaneously in $T_1[p_1{+}1 \ldots n_1], \ldots, T_k[p_k{+}1 \ldots n_k]$, i.e. $\Sigma_{cp}([p_1, p_2, \ldots, p_k]) = \{a \in \Sigma : \forall i \in \langle 1, k\rangle \exists j \in \langle p_i + 1, n_i\rangle : T_i[j] = a\}$.*

Definition 3. *For $a \in \Sigma$ and a position point $[p_1, p_2, \ldots, p_k] \in Pos(P)$ we define the common subsequence transition function:*
$csf([p_1, p_2, \ldots, p_k], a) = [r_1, r_2, \ldots, r_k]$, where $r_i = min\{j : j > p_i$ and
$T_i[j] = a\}$ for all $i \in \langle 1, k\rangle$ if $a \in \Sigma_{cp}([p_1, p_2, \ldots, p_k])$, and
$csf([p_1, p_2, \ldots, p_k], a) = \emptyset$ otherwise.
Let csf^* be reflexive-transitive closure of csf.

Lemma 1. *The automaton $(Pos(P), \Sigma, csf, ipp(P), Pos(P))$ accepts a string S iff S is a subsequence of P.*

Proof. See [2]. □

The automaton from Lemma 1 is called the *Common Subsequence Automaton* (CSA) for strings $T_1, T_2, \ldots, T_k$. An example of the CSA is in Fig. 1.

Up to now, two algorithms for building the CSA have been described. The first one is off-line and uses the position points. The second one is on-line and in each step loads one input string into the automaton.

We will briefly describe the off-line algorithm. The algorithm generates step by step all reachable position points (states). At each step we process one position point. First, we will find the common subsequence position alphabet for this point and then determine the common subsequence transition function for each symbol of that alphabet. When the position point has been processed, we continue with a next point until transitions of all reachable position points are determined. The complexity of the algorithm depends on the number of states of the resulting automaton. Providing that the total number of states is $O(t)$, the algorithm requires $O(k\sigma t)$ time.

3 Number of states of CSA

We will investigate the number of states (reachable position points) of the CSA for a set of strings on binary alphabet. First, we will introduce an auxiliary structure. The *generating tree* is defined as a general tree where nodes and edges are labeled with an integer value. If the node is labeled with value v, we say that it is of order v. A node of order v has output edges labeled with $1, 2, \ldots, v$. Every edge is going to the node labeled with the same value as this edge. If the root of the tree is of order k, we say that the tree is of order k. An example of the tree of order 3 is in Fig. 2. We will use the tree to describe the set of strings. A path from the root corresponds to a string on alphabet $\{a, b\}$. All nodes but the root will contribute by b. An edge labeled with ℓ will add a^ℓ. For example, path $root \xrightarrow{4} node(4) \xrightarrow{2} node(2) \xrightarrow{1} node(1)$ corresponds to string $aaaabaabab$. A node has at most one output edge for a given value, therefore no two strings generated by the tree are identical.

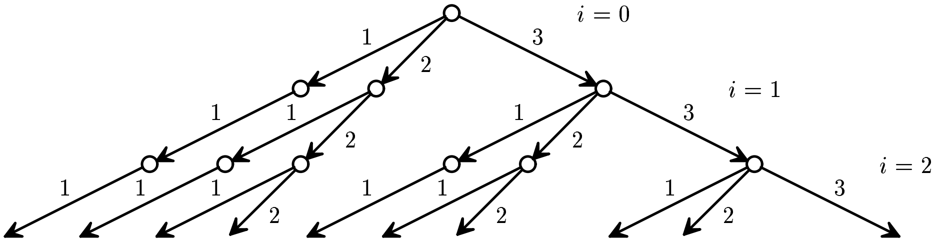

Fig. 2. The generating tree of order 3.

In the subsequent, we will consider the tree of order k and denote, for $i \in Z, i \geq 0$, by $p(k,i)$ the number of nodes on ith level of this tree. Furthermore, we will denote by $p_j(k,i), 1 \leq j \leq k$ the number of nodes on ith level which are labeled with value of j. There is just one node of order k on each level, that is $p_k(k,i) = 1$. On the 0th level, there is only one node (root). A node of order j on ith level has descendants of orders $1, 2, \ldots, j$ on $(i+1)$th level. In other words, the node of order j on ith level is a descendant either of the node of order j or of higher order on $(i-1)$th level. The number of these nodes of higher order is the same as the number of nodes of order $j+1$ on ith level. That is, $p_j(k,i) = p_j(k,i-1) + p_{j+1}(k,i)$, where $i > 0$ and $1 \leq j < k$. This formula is known from combinatorics and holds for combinatorial numbers. For further discussion we will use Pascal's triangle which is a common means for expressing the relations between combinatorial numbers.

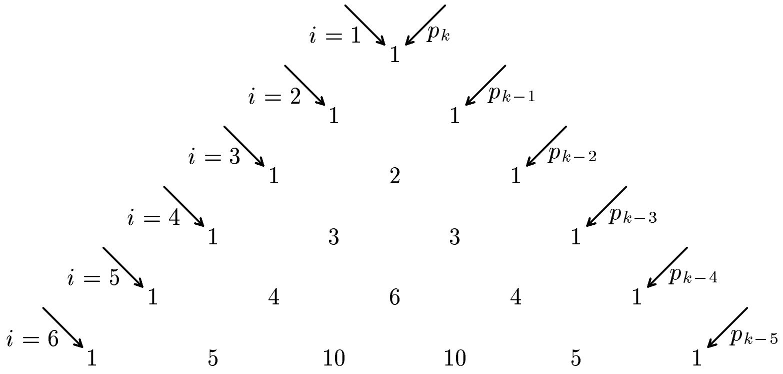

Fig. 3. The top of Pascal's triangle.

From Pascal's triangle we get:

$$p_k(k,i) = \binom{i-1}{0},\ p_{k-1}(k,i) = \binom{i}{1}, \ldots, p_1(k,i) = \binom{i+k-1}{k-1}.$$

The number of nodes on ith level is hence

$$p(k,i) = \sum_{j=1}^{k} p_j(k,i) = \binom{i+k-1}{k-1}.$$

And the total number of nodes up to nth level is

$$p(k) = \sum_{i=1}^{n} \binom{i+k-1}{k-1} = \binom{n+k}{k}.$$

The formula for $p(k)$ determines how many strings ending with b will be generated by the tree of order k if we consider the first n levels of this tree.

Lemma 2. *Let $k \in Z, k \geq 2, n_i \in Z$ for all $i \in \langle 1,k \rangle$, and $n_j \geq 2n_{j+1}$ for all $j \in \langle 1, k-1 \rangle$. Let $T_1 = (ab)^{n_1}, T_2 = (aab)^{n_2}, \dots, T_k = (a^k b)^{n_k}, L_k = \{T_1, T_2, \dots, T_k\}$, and δ denote the transition function of the CSA for L_k. Let M_k be the set of strings generated by the generating tree of order k. Then for all $u, v \in M_k, u \neq v$, is $\delta^*(ipp(P), u) \neq \delta^*(ipp(P), v)$.*

Proof. (by induction in k): Let M_k^ℓ denote the set of strings from the ℓth level of the generating tree of order k.
1. $k = 2 : M_2^\ell = \{(ab)^\ell, aab(ab)^{\ell-1}, (aab)^2(ab)^{\ell-2}, \dots, (aab)^\ell\}$. If two strings contain different number of b's they cannot result in shift to the same position in T_2. Therefore we can consider only the strings with the same number of b's. For given $\ell \in Z, \ell \geq 1$, all strings in M_2^ℓ have the same number of b's. But simultaneously, no such two strings have the same number of a's. Thus, the transition function will finish at the same position in T_2 and always at different position in T_1.
2. Let $n_{k+1} \in Z, n_k \geq 2n_{k+1}$. We add $T_{k+1} = (a^{k+1}b)^{n_{k+1}}$ into set L_k, that is $L_{k+1} = L_k \cup \{T_{k+1}\}$. According to the induction hypothesis the lemma holds for k. We will show that it holds for $k+1$ too by using induction in height h of the tree:
(a) $h = 1 : M_{k+1}^1 = \{ab, aab, \dots, a^{k+1}b\}$. No two strings from this set have the same number of a's, thus the lemma holds.
(b) The hypothesis says that the lemma holds for $M_{k+1}^1, M_{k+1}^2, \dots, M_{k+1}^h$. We will prove that it holds also for M_{k+1}^{h+1}. From the generating tree we get: $M_{k+1}^{h+1} = M_k^{h+1} \cup \{a^{k+1}bs, s \in M_{k+1}^h\}$. According to the induction hypotheses the lemma holds for M_k^{h+1} and M_{k+1}^h. Any string from M_k^{h+1} will result in shift to the $(h+1)$th b in T_k. Furthermore, $a^{k+1}b$ will make shift to the second b in T_k and because any string in M_{k+1}^h contains h symbols b, the lemma holds also for M_{k+1}^{h+1}. □

Lemma 3. *Let $k \in Z, k \geq 2, n_i \in Z$ for all $i \in \langle 1,k \rangle$, and $n_j \geq 2n_{j+1}$ for all $j \in \langle 1, k-1 \rangle$. Let $T_1 = (ab)^{n_1}, T_2 = (aab)^{n_2}, \dots, T_k = (a^k b)^{n_k}, L_k = \{T_1, T_2, \dots, T_k\}$, and δ denote the transition function of the CSA for L_k. Let M_k be the set of strings generated by the generating tree of order k. Then for all $u, v \in M_k$ and all $x, y \in \{a, aa, \dots, a^k\}, ux \neq vy$ is $\delta^*(ipp(P), ux) \neq \delta^*(ipp(P), vy)$.*

Proof. We put $\delta^*(ipp(P), u) = p$ and $\delta^*(ipp(P), v) = q$. If $p = q$ then according to Lemma 2 is $u = v$ and hence $x \neq y$. The lemma is obviously true in this case. If $p \neq q$ then either $p_k = q_k$ or $p_k \neq q_k$. If $p_k = q_k$ then either $x = y$ or $x \neq y$. For $x \neq y$ is the lemma obvious. We will consider $x = y$. From $p \neq q$ we get that there exists $\ell \in Z, 1 \leq \ell \leq k$ such that $p_\ell \neq q_\ell$. If we skip the same number of a's in T_ℓ, the positions will still differ. Hence the lemma holds in this case. For $p_k \neq q_k$ the lemma also holds because by reading any string from $\{a, aa, \ldots, a^k\}$ we will never cross b in T_k. □

The Lemma 3 shows that we can consider not only strings ending with b, *i.e.* ending in nodes of the generating tree, but we can add any string from $\{a, aa, \ldots, a^k\}$ as a suffix.

Lemma 4. *Let $k \in Z, k \geq 2, n_i \in Z$ for all $i \in \langle 1, k\rangle$ and $n_j \geq 2n_{j+1}$ for all $j \in \langle 1, k-1\rangle$. Let $T_1 = (ab)^{n_1}, T_2 = (aab)^{n_2}, \ldots, T_k = (a^k b)^{n_k}, L_k = \{T_1, T_2, \ldots, T_k\}$ and $n = n_k$ Then the number of states of the CSA for L_k is $\Omega(n^k)$.*

Proof. The lemma directly follows from Lemma 2 and the formula for the number of strings generated by the generating tree. □

We note again that the result of Lemma 2 is applicable also for the DASG.

4 Conclusion

We checked that the maximum number of states of the subsequence automaton for k strings of length $O(n)$ is $\Omega(n^k)$. We also dealt with the problem of tight upper bound for the number of states. By exhaustive searching we found the worst cases for several lengths of input strings and verified in [5] that the sequence of the maximum numbers of the states does not form any well-known integer sequence.

References

1. R. A. Baeza-Yates. Searching subsequences. *Theor. Comput. Sci.*, 78(2):363–376, 1991.
2. M. Crochemore and Z. Troníček. Directed acyclic subsequence graph for multiple texts. Rapport I.G.M. 99-13, Université de Marne-la-Vallée, 1999.
3. M. Crochemore and Z. Troníček. On the size of DASG for multiple texts. In *Proceedings of the Symposium on String Processing and Information Retrieval 2002*, pages 58–64, Lisbon, 2002.
4. H. Hoshino, A. Shinohara, M. Takeda, and S. Arikawa. Online construction of subsequence automata for multiple texts. In *Proceedings of the Symposium on String Processing and Information Retrieval 2000*, La Coruña, Spain, 2000. IEEE Computer Society Press.
5. N. J. A. Sloane. The on-line encyclopedia of integer sequences. `http://www.research.att.com/~njas/sequences/`.
6. Z. Troníček. Common subsequence automaton. In *International Conference on Implementation and Application of Automata 2002*, Tours, France, 2002.