The Set-Set Closest Common Subsequence Problem[1]

Gabriela Andrejková

Department of Computer Science, Faculty of Science, P. J. Šafárik University, Jesenná 5, 041 54 Košice, Slovakia

e-mail: `andrejk@kosice.upjs.sk`

Abstract. Efficient algorithm is presented that solves a general case of the Common Subsequence Problem, in which both input strings contain sets of symbols with *membership values* in the sets. The problem arises from a searching of the sets of most similar strings.

Key words: Subsequence, common subsequence, measure of the string, dynamic programming, design and analysis of algorithms.

1 Introduction

The motivation to the Closest Common Subsequence (CCS Problems) can be found in the typing of a text on the keyboard. The following mistakes can be made in typing some string: (1) Typing a different character, usually from the neighbour area of the given character. (2) Inserting a single character into the source string. (3) Omitting (skipping) any single source character. (4) Transposition of two elements. It means, we have some words with mistakes. The problem is how to find the strings they are very similar very closed to the exact strings. Very important role has here the common subsequence of similar strings.

The common subsequence problem of two strings is to determine one of the subsequences that can be obtained by deleting zero or more symbols from each of the given strings. It is possible to demand some additional properties for the common subsequence. Usually, it is the greatest length of the common subsequence, but we can consider some different measures for the common subsequence.

The longest common subsequence problem (*LCS Problem*) of two strings is to determine the common subsequence with the maximal length. Algorithms for this problem can be used in chemical and genetic applications and in many problems concerning data and text processing [4, 8, 10]. Further applications include the string-to-string correction problem [8] and determining the measure of differences between text files [4]. The length of the longest common subsequence (*LLCS Problem*) can determine

[1] This research was partially supported by Slovak Grant Agency for Science VEGA, project No. 1/4375/97

the measure of differences (or similarities) of text files. The simulation method for the approximate strings and sequence matching using the Levenstein metric can be found in J. Holub [7] and the algorithm for the searching of the subsequences is in Z. Troníček and B. Melichar [11].

D. S. Hirschberg and L. L. Larmore [6] have discussed a generalization of LCS Problem, which is called Set-Set LCS Problem (*SSLCS Problem*). In this case both strings are strings of subsets over an alphabet Ω. In the paper [6] is presented the $O(m \cdot n)$-time algorithm for the general SSLCS Problem.

In this paper we present algorithms for more general case of the Common Subsequence Problem, it means Closest Common Subsequence Problem *SSCCS Problem* for two strings of symbol sets with membership values of elements in the sets.

2 Basic Definitions

In this section, some basic definitions and results concerning to CCS Problem, SCCS and SSCCS Problem are presented.

Let Ω be a finite alphabet, $|\Omega| = s, P(\Omega)$ the set of all subsets of $\Omega, |P(\Omega)| = 2^s$.

Let $A = a_1 a_2 \dots a_m, a_i \in \Omega, 1 \leq i \leq m$ be a string over an alphabet Ω, where $|A| = m$ is the length of the string A.

The string $C \in \Omega^*, C = c_1 \dots c_p$ is *a subsequence* of the string $A = a_1 \dots a_m$, if a monotonous increasing sequence of natural numbers $i_1 < \cdots < i_p$ exists such that $c_j = a_{i_j}, 1 \leq j \leq p$. The string C is *a common subsequence* of two strings A, B if C is a subsequence of A and C is a subsequence of B. $|C|$ is the length of the common subsequence. The classical problem to find the longest common subsequence is defined and solved in Hirschberg [5]. In the classical problem, each element in the string is in his position as full member, but sometimes we are not sure about it in texts. The element should be in his position with 70%, it means, the element is in his position with 0.7 membership value. Sometimes, we can suppose that in some position should be one element of some set of elements with membership values.

Let $\mu_A(a_i) \in (0, 1\rangle, 1 \leq i \leq m$, be some membership values of elements in the string A. The pair (A, μ_A) is *the string A with the membership function μ_A, m-string μA* for short. $Val(\mu A)$ is a measure of μA defined by the (1).

$$Val(\mu A) = \Sigma_{i=1}^{m} \mu_A(a_i) \tag{1}$$

The string $\mu C = (C, \mu_C)$ is *a subsequence with the membership function μ_C, shortly m-subsequence* of the m-string μA if C is a subsequence of the string A and $0 < \mu_C(c_t) \leq \mu_A(a_{i_t})$, for $1 \leq t \leq p$. The m-subsequence μC is *a closest m-subsequence* if $Val(\mu C) = \Sigma_{j=1}^{p} \mu_C(c_j) = \Sigma_{j=1}^{p} \mu_A(a_{i_j})$.

The string μC is *a common m-subsequence* of two m-strings μA and μB if μC is a m-subsequence of μA and μC is a m-subsequence of μB.

The string μC is a *closest common m-subsequence* of the m-strings μA and μB if μC is a common m-subsequence with the maximal value $Val(\mu C)$. It means, if μD is a common m-subsequence of the strings μA and μB then $Val(\mu D) \leq Val(\mu C)$.

If μC is a closest common m-subsequence of the m-strings, μA and μB then $\mu_C(c_t) = min\{\mu_A(a_{k_t}), \mu_B(b_{l_t})\}$, for $1 \leq t \leq p$.

The CCS Problem: Let μA and μB be m-strings. To find a closest common subsequence of the m-strings μA and μB, $CCS(\mu A, \mu B)$ for short.

The MCCS Problem is to find the measure of CCS m-string, $MCCS$ for short. It means, $MCCS(\mu A, \mu B) = Val(CCS(\mu A, \mu B))$. $\diamond$

Algorithms for CCS and MCCS Problem Andrejková [2].

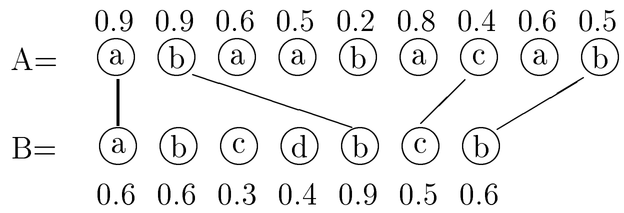

Figure 1. The closest common subsequence of two m-strings A and B.

Example 1. $\Omega = \{a, b, c\}, A = abaabacab, m = 9$, $\mu_A = (.9, .9, .6, .5, .2, .8, .4, .6, .5)$, $B = abcdbcb, n = 7, \mu_B = (.6, .6, .3, .4, .9, .5, .6)$. The string $C = abcb$ is a subsequence, $C' = abbcb$ is the longest common subsequence of the strings A and B, and $\mu C"$, $C" = abcb, \mu_{C"} = (.6, .9, .4, .5)$ is the closest common subsequence of the m-strings μA and $\mu B, Val(\mu C") = MCCS(\mu A, \mu B) = 2.4$ as it is shown in Figure 1.

A string of sets $\mathcal{B}$ over an alphabet Ω, *set-string* for short, is any finite sequence of sets from $P(\Omega)$. Formally, $\mathcal{B} = B_1 B_2 \dots B_n, B_i \in P(\Omega), 1 \leq i \leq n$, n is the number of sets in $\mathcal{B}$. The length of the symbol string described by $\mathcal{B}$ is $N = \Sigma_{i=1}^{n} |B_i|$. A string of symbols $C = c_1 c_2 \dots c_p, c_i \in \Omega, 1 \leq i \leq p$, is a *subsequence of symbols* (subsequence, for short) of the set-string $\mathcal{B}$ if there is a nondecreasing mapping $F : \{1, 2, \dots, p\} \rightarrow \{1, 2, \dots, n\}$, such that

1. if $F(i) = k$ then $c_i \in B_k$, for $i = 1, 2, \dots, p$
2. if $F(i) = k$ and $F(j) = k$ and $i \neq j$ then $c_i \neq c_j$.

Let $\mathcal{A} = A_1 \dots A_m, \mathcal{B} = B_1 \dots B_n, 1 \leq m \leq n$, be two set-strings of sets over an alphabet Ω. The string of symbols C is *a common subsequence of symbols* of $\mathcal{A}$ and $\mathcal{B}$ is C a subsequence of symbols of $\mathcal{A}$ and C is a subsequence of symbols of the set-string $\mathcal{B}$.

As similar as for strings, let define *m-set* as a set with membership function defined on its elements.

Let $\mu_{B_i}, i = 1, 2, \dots, n$ be the membership functions of the sets $B_i, i = 1, 2, \dots, n$ in the string $\mathcal{B}$. It means, $\mu\mathcal{B} = \mu B_1 \mu B_2 \dots \mu B_n$. $\mu\mathcal{B}$ is the m-set-string $\mathcal{B}$ of m-sets $B_i, i = 1, 2, \dots, n$ with the membership functions μ_{B_i}, m-set-string $\mu\mathcal{B}$ for short. The *weight of the m-set* $B \in \mathcal{P}(\Omega)$ with membership function μ_B is defined by

$$W(B) = \sum_{x \in B} \mu_B(x) \qquad (2)$$

A string μC is a *m-subsequence* of the m-set-string $\mu\mathcal{B}$ if (1) μC is the subsequence of symbols of the set-string $\mathcal{B}$ and (2) if $c = c_i, c_i \in B_k$ then $\mu_C(c_i) \leq \mu_{B_k}(c)$.

The m-string μC is *a common m-subsequence* of the m-set-strings $\mu\mathcal{A}$ and $\mu\mathcal{B}$ if μC is m-subsequence of $\mu\mathcal{A}$ and μC is m-subsequence of $\mu\mathcal{B}$.

The string μC is *a closest common m-subsequence* of the m-set-strings $\mu\mathcal{A}$ and $\mu\mathcal{B}$ if μC is a common subsequence with maximal value $Val(\mu C)$. Note that μC is not in general unique.

The SSCCS Problem: Let $\mu\mathcal{A}, \mu\mathcal{B}$ be two m-set-strings. The Set-Set Closest Common Subsequence problem of the m-set-strings $\mu\mathcal{A}$ and $\mu\mathcal{B}$, ($SSCCS(\mu\mathcal{A}, \mu\mathcal{B})$ for short, consists of finding a closest common m-subsequence μC with the maximal value $Val(\mu C)$.

The MSSCCS Problem consists of finding the measure of $SSCCS$ m-set-string, $MSSCCS(\mu\mathcal{A}, \mu\mathcal{B})$ for short.

It means, $MSSCCS(\mu\mathcal{A}, \mu\mathcal{B}) = Val(SSCCS(\mu\mathcal{A}, \mu\mathcal{B}))$, $\diamond$

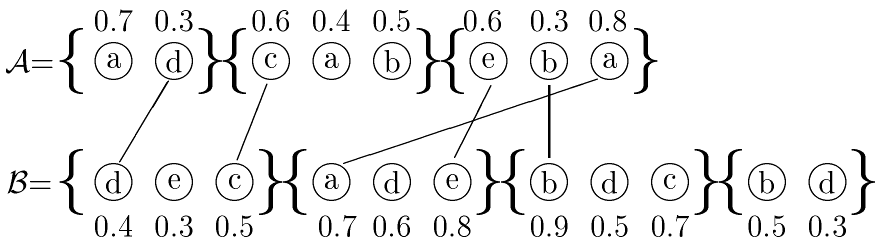

Figure 2. The closest common m-subsequence of two m-set-strings $\mathcal{A}$ and $\mathcal{B}$.

Example 2. Let $\mathcal{A} = \{a, d\}\{c, a, d\}\{e, b, a\}, m = 3$, $\mu_{A_1} = (.7, .3), \mu_{A_2} = (.6, .4, .5), \mu_{A_3} = (.6, .3, .8), \mathcal{B} = \{d, e, c\}\{a, d, e\}\{b, d, c\}\{b, d\}, n = 4$. $\mu_{B_1} = (.4, .3, .5), \mu_{B_2} = (.7, .6, .8), \mu_{B_3} = (.9, .5, .7), \mu_{B_4} = (.5, .3)$. The membership values are described according to the named order in the set. For example, $\mu_{A_1}(a) = 0.7, \mu_{A_1}(d) = 0.3$. Then $MSSCCS(\mu\mathcal{A}, \mu\mathcal{B}) = 2.4$ as it is shown in the Figure 2.

3 ALGORITHM FOR MSSCCS Problem

The basic idea of the algorithm starts from the definition of $MSSCCS$ Problem.

$$MSSCCS(\mu\mathcal{A}, \mu\mathcal{B}) = max_{\mu C}\{Val(\mu C) : \mu C \text{ is the common m-subsequence of } \mu\mathcal{A} \textit{ and } \mu\mathcal{B}\} \tag{3}$$

In the following part of the paper we will use the m-set only and for a simpler description we will omit the symbol μ in the names of sets.

A *flattening* of a sequence of sets is defined as a concatenation, in order of the sequence, of strings formed by some permutation of individual elements of the sets in the sequence. For example, a flattening of the m-set-string $\mathcal{A}$ in Example 2 is

$\mathcal{A}_{fl1} = dabaceba, \mu_{\mathcal{A}_{fl1}} = (.3, .7, .5, .4, .6, .6, .3, .8)$ and so is $\mathcal{A}_{fl2} = daacbbae, \mu_{\mathcal{A}_{fl2}} = (.3, .7, .4, .6, .5, .3, .8, .6)$.

If we have some flattening of both set-strings then it is possible to apply the $MCCS$ algorithm, Andrejková [2]. It is necessary to compute $MCCS$ values of all pairs of all flattenings both set-strings but it is too much time consuming.

If we have the flattening of one set-string and the second is a set-string then it is possible to use the $MSCCS$ algorithms. But it is necessary to compute $MSCCS$ value for all flattenig of one string. It is to much time consuming too. Both algorithms have an exponential time complexity.

It is possible to use the following algorithm of polynomial time complexity. The algorithm works in two steps:

1. to create the string of symbols for each of set-string; each set can be encoded as the string of all permutations of its elements (the length of such string is $k^2 - 2 \cdot k + 4$, k is the number of elements in set [9]); for example, the shortest m-string of elements in the m-set-string $\mathcal{A}$ in example 2 is *dadcabcabcbeabeab* and so is *adacabcabcebaebae*.

2. to apply the MCCRS algorithm, Andrejková [1] for the two in the previous step constructed m-strings (each element of the m-set can be used once at most);

The algorithm works in polynomial time: $O(M^2 \cdot N^2 \cdot K)$, where $M = \Sigma_{i=1}^{m} |A^i|$, $N = \Sigma_{j=1}^{n} |B^j|$, and K is the number of elements in the closest common restricted subsequence.

We formulate the following algorithm with the better time complexity according to Hirshberg's idea for SSLCS Problem [6]. The algorithm works with *intersection*, *union* and *equivalence*, *difference* of m-sets. It is possible to use many definitions of them but the following [3, 12] are more obvious:

Let A, B be the m-sets with membership functions μ_A, μ_B, and $x \varepsilon A$ explains a membership of the element x to the m-set A, $\mu_A(x) > 0$, then

1. *intersection* "$\cap_m$" of two m-sets is defined:
$$A \cap_m B =_{def} \{x | x\varepsilon A \wedge x \varepsilon B\}, \mu_{A \cap_m B}(x) = min\{\mu_A(x), \mu_B(x)\}$$

2. *union* "$\cup_m$" of two m-sets is defined:
$$A \cup_m B =_{def} \{x | x\varepsilon A \vee x \varepsilon B\}, \mu_{A \cup_m B}(x) = max\{\mu_A(x), \mu_B(x)\}$$

3. *equivalence* "$=_m$" of two m-sets is defined:
$$A =_m B \Leftrightarrow x\varepsilon A \wedge x\varepsilon B \wedge x\varepsilon A \cap_m B \wedge x \varepsilon A \cup_m B$$

4. *difference* "$-_m$" of two m-sets is defined:
$$A -_m B =_{def} \{x | x\varepsilon A \wedge x \not\varepsilon B\}, \mu_{A-B}(x) = \mu_A(x)$$

5. A is *m-subset* of B, $A \subseteq_m B$, if and only if $\forall x \varepsilon A$ is fulfilled $x \varepsilon B$ and $0 < \mu_A(x) \leq \mu_B(x)$.

3.1 Description of the simple algorithm A.

For convenience, we define $A_0 = B_0 = \emptyset$.

We define $Ent(i, j)$ to be the set of quintuples (k, F_f, F_u, G_f, G_u) such that:

(1) k is the measure of γ, a common m-subsequence of some flattening of $A_1 \dots A_i$ and some flattening of $B_1 \dots B_j$, defined by (1).

(2) *free m-set* $F_f \subseteq_m A_i$ is the m-set of elements of A_i which are not used by γ,

(3) *free m-set* $G_f \subseteq_m B_j$, is the m-set of elements of B_j not used by γ,

(4) *m-set of used elements* $F_u \subseteq_m A_i$ is the m-set of elements of A_i used by γ, and

(5) *m-set of used elements* $G_u \subseteq_m B_j$ is the m-set of elements of B_j used by γ.

Example 3. $(1.2, \{(c, 0.5), (a, 0.4)\}, \{(b, 0.5)\}, \{(a, 0.4)\}, \{(d, 0.6), (e, 0.8)\})$ is in $Ent(2, 2)$ for m-set-strings in Example 2.

We refer to such quintuple as an *entry*. The measure of the CCS of the some flattening of $A_1 \dots A_m$ and some flattening of $B_1 \dots B_n$ is then, by definition, the largest k such that $(k, F_f, F_u, G_f, G_u) \in Ent(m, n)$ for some m-sets F_f, F_u, G_f and G_u. $Ent(0, 0)$ contains just one entry, namely $(0, \emptyset, \emptyset, \emptyset, \emptyset)$, while $Ent(i, j)$ can be computed dynamically from $Ent(i-1, j)$ and $Ent(i, j-1)$. The problem is that the cardinality of $Ent(i, j)$ could become very large, making such an algorithm exponential in the worst case.

Let $e = (k, F_f, F_u, G_f, G_u) \in Ent(i-1, j)$ and $F^s, F_u \subseteq_m F^s$ be the m-set with the following property: $x \varepsilon F^s \Leftrightarrow x \varepsilon F_u$ and $\mu_{F_u}(x) \leq \mu_{F^s}(x) = \mu_{A_i}(x)$. It means, the m-set F^s is the maximal m-set that has the same elements as the m-set F_u, but membership values of elements in F^s are the same as in the m-set A_i.

Let S be any subset of $A_i \cap_m G_f$. We say that e **vertically generates** $e' \in Ent(i, j)$ iff

1. $e' = (k + W(S) - W(A_i \cap_m G_u) + W(S'), A_i -_m S, F_u, G_f -_m S, G_u)$ for any subset S' of $A_i \cap_m G^s, W(S') > W(A_i \cap_m G_u)$, or

2. $e' = (k + W(S), A_i -_m S, F_u, G_f -_m S, G_u)$ and for each subset S' of $A_i \cap_m G^s$ is $W(S') \leq W(A_i \cap_m G_u)$.

The element $e' \in Ent(i, j)$ and it is shown by the following: If α is common m-subsequence with a measure $k = Val(\alpha)$ of the flattening of $A_1 \dots A_{i-1}$ and some flattening of $B_1 \dots B_j$, where $F_f \subseteq_m A_{i-1}$ and $G_f \subseteq_m B_j$ are free m-sets, and β is a m-sequence consisting of the elements of $S \subseteq_m A_i \cap G_f$ written in any order, then $\alpha\beta$ (having measure $k + Val(S)$) is common subsequence of a flattening of $A_1 \dots A_i$ and a flattening of $B_1 \dots B_j$, with free m-sets $A_i - S$ and $G_f - S$. The used elements from m-set G_u can be used with some better membership values and it is evaluated by the comparison of the weights of the sets $A_i \cap_m G^s$ and $A_i \cap_m G_u$. If $W(A_i \cap_m G_u) < W(A_i \cap_m G^s)$ then there exists some better using of elements in A_i.

Similarly, if $(k, F_f, F_u, G_f, G_u) \in Ent(i, j-1)$ and $S \subseteq_m F_f \cap_m B_j$ and S' is any subset of $B_j \cap_m F^s), W(S') > W(B_j \cap_m F_u)$, we say that (k, F_f, F_u, G_f, G_u) **horizontally generates** $(k + W(S) - W(B_j \cap_m F_u) + W(S'), F_f -_m S, F_u, B_j -_m S, G_u) \in Ent(i, j)$ or $(k+W(S), F_f -_m S, F_u, B_j -_m S, G_u) \in Ent(i, j)$ according to the relation $W(B_j \cap_m F_u) < / \geq W(S')$.

Lemma 1 *If $e \in Ent(i, j)$ for $i+j \geq 0$ then e is generated by some element of either $Ent(i-1, j)$ or $Ent(i, j-1)$.*

Proof. $e = (k, F_f, F_u, G_f, G_u) \in Ent(i, j)$, it means $e = \alpha\beta$, β is the part of elements in $A_i \cap_m B_j$. According to above construction, the part β is the prolongation of some element $e' \in Ent(i-1, j)$ *or* $Ent(i, j-1)$. In the part α should be elements with higher membership values. $\square$

The element $e \in Ent(i, j)$ is generated from elements in $E(i-1, j)$ or $Ent(i, j-1)$ using of two sets: the free subset and the used subset of B_j, respectively A_i. The following algorithm is a dynamic programming algorithm in which the boundary conditions are set and then the interval entries are determined:

Algorithm A.

for all i **do** $Ent(i, 0) := \{(0, A_i, \emptyset, \emptyset, \emptyset)\}$

for all j **do** $Ent(0, j) := \{(0, \emptyset, \emptyset, B_j, \emptyset)\}$

for $i := 1$ **to** m **do**

for $j := 1$ **to** n **do**

$Ent(i, j) :=$ {**all entries vertically generated from** $Ent(i-1, j)$}

$\bigcup$ {**all entries horizontally generated from** $Ent(i, j-1)$}

max_k := the largest k **such that** $(k, F_f, F_u, G_f, G_u) \in Ent(m, n)$ **for some** F_f, F_u, G_f, G_u**.**

3.2 Description of a better algorithm

The above algorithm may be very time-consuming because of too many quintuples is necessary to analyze. We will speed the algorithm by eliminating consideration of many quintuples.

If $(k, F_f, F_u, G_f, G_u), (k', F_f^{'}, F_u^{'}, G_f^{'}, G_u^{'}) \in Ent(i, j)$, we say that (k, F_f, F_u, G_f, G_u) *dominates* $(k', F_f^{'}, F_u^{'}, G_f^{'}, G_u^{'})$ $((k', F_f^{'}, F_u^{'}, G_f^{'}, G_u^{'}) \preceq (k, F_f, F_u, G_f, G_u))$ if the following conditions hold:

1. $d = k - k^{'} \geq 0$,
2. $\big(W(F_f^{'} - F_f) \leq d$ and $F_u^{'} \subseteq_m F_u\big)$ or $\big(W(F_u^{'} - F_u) \leq d$ and $F_f^{'} \subseteq_m F_f\big)$,
3. $\big(W(G_f^{'} - G_f) \leq d$ and $G_u^{'} \subseteq_m G_u\big)$ or $\big(W(G_u^{'} - G_u) \leq d$ and $G_f^{'} \subseteq_m G_f\big)$.

The relation "$\preceq$" is a transitive, antisymmetric and reflexive relation. The elements of $Ent(i,j)$ can be ordered according to relation "$\preceq$", it means they are ordered in some *chains*. The last element of the chain has maximal measure and that is very important.

Lemma 2 *Any element of $Ent(i,j)$ which is not maximal with respect to the relation "$\preceq$" can be discarded during execution of the algorithm without affecting the final value of max_k.*

Proof. It will be proved by downward induction on both indices i and j. The value of max_k is obtained from $Ent(m,n)$ in the last step and all other elements may be discarded with no effect.

Suppose $i+j < m+n$ and $e' \in Ent(i,j), e'$ is not maximal. Let $e \in Ent(i,j)$ is maximal. It means, $e' \preceq e$. It is necessary to prove that maximal element of $Ent(i+1,j)$ or $Ent(i,j+1)$ which is generated by e' can be generated by e too. And the element e' can be discarded.

Let $e = (k, F_f, F_u, G_f, G_u), e' = (k', F'_f, F'_u, G'_f, G'_u)$ and e' vertically generates f'. f' should have two forms for some m-set $P \subseteq A_{i+1} \cap G'_f$

(a) $f' = (k' + W(P) - W(A_{i+1} \cap_m G'_u) + W(A_{i+1} \cap_m G'^s), A_{i+1} -_m P, P, G'_f -_m P, G'_u)$, if $W(A_{i+1} \cap_m G'_u) < W(A_{i+1} \cap_m G'^s)$, or

(b) $f' = (k' + W(P), A_{i+1} -_m P, P, G'_f -_m P, G'_u)$, if $W(A_{i+1} \cap_m G'_u) \geq W(A_{i+1} \cap_m G'^s)$.

Let $S = P \cap G_f$, and f is vertically generated by e. f should have two forms: (1) $f = (k + W(S) - W(A_{i+1} \cap_m G_u) + W(A_{i+1} \cap_m G^s), A_{i+1} -_m S, S, G_f -_m S, G_u)$ or (2)$f = (k + W(S), A_{i+1} -_m S, S, G_f -_m S, G_u)$. It is necessary to analyze four cases to prove the Lemma (a)-(1), (a)-(2), (b)-(1), (b)-(2). We start with the first one, it means (a)-(1), and $W(A_{i+1} \cap_m G'_u) < W(A_{i+1} \cap_m G'^s)$ and $W(A_{i+1} \cap_m G_u) < W(A_{i+1} \cap_m G^s)$.

Since $e' \preceq e, d = k - k'$,
$d \geq 0$, $\big((W(F'_f - F_f) \leq d$ and $F'_u \subseteq_m F_u)$ or $(W(F'_u - F_u) \leq d$ and $F'_f \subseteq_m F_f)\big)$, and $\big((W(G'_f - G_f) \leq d$ and $G'_u \subseteq_m G_u)$ or $(W(G'_u - G_u) \leq d$ and $G'_f \subseteq_m G_f)\big)$. Then $W(P -_m S) = W(P -_m P \cap_m G_f) = W(P -_m G_f) \leq W(F'_f -_m F_f) \leq d$ and $W(P -_m S) \leq W(P) - W(S)$. Let $d' = (W(P) - W(S)) - (W(A_{i+1} \cap_m G^s) - W(A_{i+1} \cap_m G'^s)) - (W(A_{i+1} \cap_m G_u) - W(A_{i+1} \cap_m G'_u) \leq d$ We prove that $f' \preceq f$, it means f' is not maximal or $f = f'$. According to definition of "$\preceq$" it is necessary to check three conditions 1.-3.

1. $z = k + W(S) - W(A_{i+1} \cap_m G_u) + W(A_{i+1} \cap_m G^s) - (k' + W(P) - W(A_{i+1} \cap_m G'_u) + W(A_{i+1} \cap_m G'^s)) = k - k' - (W(P) - W(S)) + W(A_{i+1} \cap_m G^s) - W(A_{i+1} \cap_m G'^s) + W(A_{i+1} \cap_m G_u) - W(A_{i+1} \cap_m G'_u \geq d - d' \geq 0$

2. $W(P -_m S) \leq d$ and $A_{i+1} -_m P \subseteq_m A_{i+1} -_m S$

3. $W(G'_f -_m P -_m (F_f -_m S)) = W(G'_f -_m G_f) \leq d$ and $G'_u \subseteq_m G_u$.

The rest three cases can be proved by a very similar method. And the vertical case is similar. □

If $e = (k, F_f, F_u, G_f, G_u) \in Ent(i,j)$, we define the *horizontal child* of e to be $hor(e) = k + W(F_f \cap B_{j+1}) - W(A_i \cap_m G_u) + W(A_i \cap_m G^s), F_f - B_{j+1}, F_u, B_{j+1} - F_f, G_u)$ or $hor(e) = k + W(F_f \cap B_{j+1}), F_f - B_{j+1}, F_u, B_{j+1} - F_f, G_u)$ and define the *vertical child* of e to be $ver(e) = k + W(A_{i+1} \cap G_f) - W(B_{j+1} \cap G_u) + W(B_{j+1} \cap G^s), B_{j+1} - G_f, F_u, G_f - B_{j+1}, G_u)$or $ver(e) = k + W(A_{i+1} \cap G_f), B_{j+1} - G_f, F_u, G_f - B_{j+1}, G_u)$. We define $MaxEnt(i,j)$ to be the set of maximal elements of $Ent(i,j)$ under the dominance relation "$\preceq$".

Lemma 3 *Any entry horizontally generates at most one maximal entry and vertically generates at most one maximal entry.*

Proof. Let $e = (k, F_f, F_u, G_f, G_u) \in Ent(i,j)$. The only elements vertically generated by e which can be maximal are in the $ver(e)$, since they dominates any others vertically generated by e. Similarly, $hor(e)$ dominates any entries horizontally generated by e. □

We say that (k, F_f, F_u, G_f, G_u) *strongly dominates* (k', F_f, F_u, G_f, G_u) if $k > k'$. If $S \subseteq Ent(i,j)$, defines $Dom(S) \subseteq S$ to be the set obtained by deleting every element of S which is strongly dominated by another element of S. We now inductively define sets $Chain(i,j) \subseteq Ent(i,j)$ by:

1. $Chain(i,0) = \{(0, A_i, \emptyset, \emptyset, \emptyset)\}$,
2. $Chain(0,j) = \{(0, \emptyset, \emptyset, B_j, \emptyset)\}$,
3. $Chain(i,j) = Dom(\{ver(e) | e \in Chain(i-1,j)\} \cup \{hor(e) | e \in Chain(i,j-1)\})$.

We refer to entries $Chain(i,j)$ as **weakly maximal**. We observe the following lemma.

Theorem 1 $MaxEnt(i,j) \subseteq Chain(i,j)$.

Proof. By induction. For $i = 0$ or $j = 0$ the two sets $MaxEnt(i,j)$ and $Chain(i,j)$ are identical. For $i, j > 0$, and $e \in MaxEnt(i,j)$ must be vertical or horizontal child of some maximal element, which is weakly maximal by induction. It means, e must be weakly maximal, since it is maximal and thus cannot be deleted by operator Dom. □

Using the results of the Lemmas 2 and 3 and Theorem 1 we have the following algorithm:

Algorithm B.

{Using weakly maximal entries.}

for all i do $Chain(i,0) := \{(0, A_i, \emptyset, \emptyset, \emptyset)\}$;

for all j do $Chain(0,j) := \{(0, \emptyset, \emptyset, B_j, \emptyset)\}$;

for i:=1 to m do

for j:=1 to n do

begin

$Chain(i,j) := \emptyset$**;**

for all $(k, F_f, F_u, G_f, G_u) \in Chain(i, j-1)$ **do begin**

$help := W(B_j \cap_m F^s) - W(B_j \cap_m F_u)$;

if $help \leq 0$ **then**

insert $(k + W(F \cap B_j), F_f - B_j, F_u, B_j - F_f, G_u)$ **into** $Chain(i,j)$

else insert $(k + W(F_f \cap B_j) + help, F_f - B_j, F_u, B_j - F_f, G_u)$ **into** $Chain(i,j)$

end;

for all $(k, F_f, F_u, G_f, G_u) \in Chain(i-1, j)$ **do begin**

$help := W(A_i \cap_m G^s) - W(A_i \cap_m G_u)$;

if $help \leq 0$ **then**

insert $(k + W(A_i \cap G), A_i - G_f, F_u, G_f - A_i, G_u)$ **into** $Chain(i,j)$

else insert $(k + W(G_f \cap A_i) + help, A_i - G_f, F_u, G_f - A_i, G_u)$ **into** $Chain(i,j)$

end;

delete all nonweakly maximal elements from $Chain(i,j)$

end

$max\_k :=$ **the maximum value of** k **such that** $(k, F_f, F_u, G_f, G_u) \in Chain(m,n)$ **for some** F_f, F_u, G_f **and** G_u**.**

The algorithm works in $O(m \cdot n \cdot K \cdot t)$-time, where K is the maximal number of elements in $Chain(i,j)$ and t is the maximal time spent for computing the intersection of two sets. The algorithm works in $O(m \cdot n \cdot k)$-space, where k is the maximal number of elements in the m-sets A_i, B_j. In the next section we show the idea of some efficient implementation of the algorithm.

3.3 Efficient implementation of algorithm B.

We show the structure of $Chain(i,j)$ that will help obtain an efficient implementation of algorithm B. We begin by defining a transitive reflexive relation $\triangleleft$ on $Ent(i,j)$. We say that $(k, F_f, F_u, G_f, G_u) \triangleleft (k', F'_f, F'_u, G'_f, G'_u)$ if $F_f \subseteq_m F'_f, G'_f \subseteq_m G_f, F_u =_m F'_u$ and $G_u =_m G'_u$.

Lemma 4 *(a) If* $e, e' \in Ent(i-1, j)$*, and if* $e \triangleleft e'$*, then* $ver(e) \triangleleft ver(e')$*.*

(b) If $e, e' \in Ent(i, j-1)$*, and if* $e \triangleleft e'$*, then* $hor(e) \triangleleft hor(e')$*.*

(c) If $e \in Ent(i, j-1)$ *and* $e' \in Ent(i-1, j)$*, then* $hor(e) \triangleleft ver(e')$*.*

Proof. (a) $ver(e) = (k, F_f \cap_m B_j, F_u, B_j -_m F_f, G_u)$ and $ver(e') = (k', F' -_m B_j, F_u^{'}, B_j -_m F_f^{'}, G_u^{'})$. It follows from $F_f \subseteq_m F_f^{'}$ that $F_f - B_j \subseteq_m F_f^{'} - B_j$ and $B_j - F_f^{'} \subseteq_m B_j - F_f$, i.e. $ver(e) \lhd ver(e')$.
(b) Similar to the proof of (a).
(c) Let $e = (k, F_f, F_u, G_f, G_u)$ and $e' = (k', F_f^{'}, F_u^{'}, G_f^{'}, G_u^{'})$. Then $hor(e) = (k, F_f -_m B_j, F_u, B_j -_m F_f, G_u)$ and $ver(e') = (k', F_f^{'} -_m B_j, F_u^{'}, B_j -_m F_f^{'}, G_u^{'})$. It can be seen that $F_f \subseteq_m A_i$ since $e \in Ent(i, j-1)$, and that $G_f^{'} \subseteq_m B_j$ since $e' \in Ent(i-1, j)$. And we have $F_f - B_j \subseteq_m A_i - G_f^{'}$ and $G_f^{'} - A_i \subseteq_m B_j - F_f$, i.e. $hor(e) \lhd ver(e')$.
□

Lemma 5 *The relation $\lhd$ imposes a total ordering on $Chain(i, j)$.*

Proof. We need to prove that for any distinct $f, f' \in Chain(i, j)$, either $f \lhd f'$ or $f' \lhd f$ but not both. If $f' \lhd f$ and $f \lhd f'$, then f and f' would have the same free sets, which implies they must be identical, else the one with the smaller value of k would not be weakly maximal. Thus, we need only show that f and f' are comparable. We do this by induction on i and j.

$Chain(0, j)$ contains just one entry, namely $(0, \emptyset, \emptyset, B_j, \emptyset)$, and hence is ordered. Similarly, $Chain(i, 0)$ contains only the entry $(0, A_i, \emptyset, \emptyset, \emptyset)$.

Suppose, $i, j > 0$, and $f, f' \in Chain(i, j)$. Both f and f' must be generated by maximal entries e and e', respectively. We consider three cases. If f and f' are the vertical children of e and e', respectively, then by induction, e and e' are comparable, hence f and f' are comparable by above Lemma. If f and f' are the horizontal children of e and e' the proof is similar. If f is the horizontal child of e and f' is vertical child of e', then f and f' are comparable by above Lemma too. □

Lemma 6 *$Chain(i, j)$ has the number of elements at most $1 + |A_i| + |B_j|$, where $|A_i|, |B_j|$ are the numbers of elements in the m-sets A_i, B_j, for $i = 1, \dots, m, j = 1, \dots, n$.*

Proof. The main idea of the proof is in the following: Each element from m-set should be used once at most but with some different membership value.

If $e \in (k, F_f, F_u, G_f, G_u) \in Ent(i, j)$, define *signature* of e to be $|F_f| - |G_f|$, which must lie in the interval $[-|B_j|..|A_i|]$. Since $Chain(i, j)$ is ordered under the relation $\lhd$, each entry must have different signature. □

It means, the algorithm works in $O(m \cdot n \cdot L \cdot t)$-time, where L is the maximal number of the numbers in $\{1 + |A_i| + |B_j|, i = 1, \dots m, j = 1, \dots n\}$ and $t = max\{|A_i|, |B_j|, i = 1, \dots m, j = 1, \dots n\}$ is the maximal time spent for computing of the intersection of two sets. The algorithm works in $O(n \cdot L \cdot t)$-space.

4 CONCLUDING REMARKS

Polynomial algorithms for the solutions of the SSCCS and MSSCCS Problem with membership functions have been presented. The algorithms work in $O(m \cdot n \cdot L \cdot t)$-time and $O(n \cdot L \cdot t)$-space, where $L = max\{1 + |A_i| + |B_j|, i = 1, \dots m, j = 1, \dots n\}$ and $t = max\{|A_i|, |B_j|, i = 1, \dots m, j = 1, \dots n\}$ is the maximal time spent for computing of the intersection of two sets.

References

[1] Andrejková, G.: *The longest restricted common subsequence problem.* Proceedings of the Prague Stringology Club Workshop'98, Prague, 1998, p. 14-25.

[2] Andrejková, G.: The Set Closest Common Subsequence Problem, *Proceedings of 4-th International Conference on Applied Informatics '99*, Eger-Noszvaj, (1999), p. 8.

[3] Gottwald, S.: Fuzzy Sets and Fuzzy Logic, *Vieweg, Wiesbaden*, (1993), p. 216.

[4] Heckel, P.: *A technique for isolating differences between files.* Comm. ACM 21, 4 (Apr. 1978), p. 264–268.

[5] Hirschberg, D. S.: *Algorithms for longest common subsequence problem.* Journal ACM 24, 4 (Oct 1977), p. 664–675.

[6] Hirschberg, D. S., Larmore, L. L.: *Set-Set LCS Problem.* Algorithmica 4 (1989), p. 503–510.

[7] Holub, J.: *Dynamic Programming for Reduced NFAs for Approximate String and Sequence Matching.* Proceedings of the Prague Stringology Club Workshop'98, Prague, 1998, p. 73-82.

[8] Lowrance, R., Wagner, R. A.: *An extension of the string-to-string correction problems.* Journal ACM 22, 2 (Apr. 1975), p. 177–183.

[9] Mohanty, S. P.: *Shortest string containing all permutations.* Discrete Mathematics 31, 1980, p. 91–95.

[10] Needleman, S. B., Wunsch, Ch. D.: *A general method applicable to the search for similarities in the amino acid sequence of two proteins.* Journal Mol. Biol. 48, 1970, p. 443–453.

[11] Troníček, Z., Melichar, B.: *Directed Acyclic Subsequence Graph.* Proceedings of the Prague Stringology Club Workshop'98, Prague, 1998, p. 107-118.

[12] Zadeh, L.: Fuzzy sets as a basis for a theory of possibility. Fuzzy Sets Syst. 1, 1978, p. 3-28.