Journal of the Applied Mathematics, Statistics and Informatics (JAMSI), **4** (2008), No. 1

A Heuristic Algorithm for the *N*-LCS Problem[1]

MOURAD ELLOUMI AND AHMED MOKADDEM

Abstract

Let $f=\{w_1, w_2, \dots, w_N\}$ be a set of N strings, a string s is an *N-Common Subsequence* (*N*-CS) of f, if and only if, s is a subsequence of each of the N strings of f. Hence, the *N-Longest Common Subsequence* (*N*-LCS) problem is defined as follows: Given a set of N strings $f=\{w_1, w_2, \dots, w_N\}$ find an *N*-CS of f of maximum length. When $N>2$, the *N*-LCS problem becomes NP-hard [Maier 78] and even hard to be approximated [Jiang and Li 95].
In this paper, we present A heuristic algorithm, adopting a *regions* approach, for the *N*-LCS problem, where $N\geq2$. Our algorithm is $O(N*L*log(L))$ in computing time, where L is the maximum length of a string. During each recursive call to our algorithm :
We look for the longest common substrings that appear in the different strings. If we have more than one longest common substring, we do a *filtering*.

Additional Key Words and Phrases: strings, common subsequences, divide-and-conquer strategy, algorithms, complexities.
Mathematics Subjekt Classification 2000: 68T20, 68Q25

INTRODUCTION

Let $s = s^1s^2 \dots s^m$ and $w = w^1w^2 \dots w^n$ be two strings, we say that a string s is a *subsequence* of a string w if and only if there is a mapping $F : \{1, 2, \dots, m\} \rightarrow \{1, 2, \dots, n\}$ such that $F(i) = k$ if and only if $s_i= w_k$ and F is a *monotone strictly increasing* function, i.e., $F(i) = u$, $F(j) = v$ and $i<j$ imply that $u<v$ [Hirschberg 75].

Let $f=\{w_1, w_2, \dots, w_N\}$ be a set of N strings, a string s is an *N-Common Subsequence* (*N*-CS) of f, if and only if, s is a subsequence of each of the N strings of f.

The *N-Longest Common Subsequence* (*N*-LCS) problem is defined as follows : Given a set of N strings $f=\{w_1, w_2, \dots, w_N\}$ find an *N*-CS of f of maximum length.

The *N*-LCS problem is one of the classical problems studied in Computer Science and has variants in other areas like Molecular Biology. Indeed, one variant of the *N*-LCS problem is the *Multiple Sequence Alignment* (MSA) one. In this problem, we deal with the *alignment* of a number of strings coding biological macromolecules to find similarities between these macromolecules [Pearson and Lipman 88, Altschul *et al.* 90, Evans 99, Jiang *et al.* 00, Lin *et al.* 02, Blin *et al* 05, Bereg and Zhu 05, Elloumi and Mokaddem 08]. The MSA problem is NP-Complete [Wang and Jiang 94].

[1]Permission to make digital/hard copy of part of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication, and its date of appear.

Most of the existing algorithms deal with the 2-LCS problem and its variants [Andrejkova 98, Arslan and Egecioglu 04, Arslan and Egecioglu 05], because these problems are polynomial. When N>2, the N-LCS problem becomes NP-hard [Maier 78] and even hard to be approximated [Jiang and Li 95]. In this case, it is interesting to describe polynomial algorithms to construct CS of lengths close to the one of the LCS.

We distinguish two types of algorithms of construction of a CS to a set of strings :

(*i*) Either algorithms adopting a *Dynamic Programming* (DP) approach [Bellman 57, Bellman and Dreyfus 62]. By using these algorithms, we proceed in two steps :

During the first step, we compute a distance between all the strings to be compared : the computation of the distances between longer substrings is made by using the results of the computations of the distances between shorter substrings. We reiterate this process until the distance between all the strings is computed.

During the second step, we construct recursively the CS associated with the value of the computed distance : the construction of the CS associated the distances between longer substrings is made by using CS associated with the distances between shorter substrings. We reiterate this process until the CS associated with the distance between all the strings is constructed.

These DP algorithms of construction of a CS to a set of strings are, actually, a generalization of DP algorithms of construction of a CS to two strings [Needleman et Wunsch 70, Wagner et Fischer 74, Sellers 80]. Among these algorithms, we mention the one of Sankoff [Sankoff 75], the one of Waterman *et al.* [Waterman *et al.* 76] and the one of Altschul and Lipman [Altschul and Lipman 89]. Sankoff's algorithm has complexities of $O(2^N*L^N)$ in computing time and $O(L^N)$ in memory space, where N is the number of the strings and L is the maximum length of a string, and those of Waterman *et al.* and Altschul and Lipman have comparable complexities to those of Sankoff's one.

(*ii*) Or algorithms adopting a *regions* approach : these algorithms are based on the *alignment* of common substrings. The substrings kept are those that minimize a certain *function*. The list of these substrings, sorted by their order of appearance in the strings to be compared, constitutes a CS reflecting structural similarities that exist between these strings. Among these algorithms, we mention the one of Hakata and Imai [Hakata and Imai 92], the ones of Irving and Fraser [Irving and Fraser 92], the one of Vitte [Vitte 96], the ones of Bonizzoni *et al.* [Bonizzoni *et al.* 01]. The algorithm of Hakata and Imai is of complexity $O(N*L*|A|+D*N*|A|*(log^{N-3}(L)+log^{N-2}(|A|)))$ in computing time, those of

Irving and Fraser are respectively of complexities $O(N*L*(L-l)^{N-1})$ and $O(N*l(L-l)^{N-1}+N*L*|A|)$, the one of Vitte is of complexity $O(|A|*(L/|A|)^N)$ and the ones Bonizzoni *et al.* are respectively of complexities $O(N*L^3*log(L))$ and $O(N*L^4*log(L))$, where N is the number of the strings, L is the length of a string, D is the number of dominant matches, A is the alphabet and l is the length of an LCS.

In this paper, we present A Heuristic algorithm, adopting a regions approach, for the N-LCS problem, where $N\geq 2$. Our algorithm is $O(N*L^2*log(L))$ in computing time, where L is the maximum length of a string. During each recursive call to our algorithm :

We look for the longest common substrings that appear in the different strings. If we have more than one longest common substring, we do a *filtering*. The filtering is achieved in two steps :

(*i*) During the first step, we a do a *horizontal filtering* : We filter the different occurrences, in a string, of a same longest common substring.

(*ii*) During the second step, we a do a *vertical filtering* : We filter the alignments associated with the different longest common substrings.

In the first section of this paper, we give some definitions and notations.

In the second section, we describe our algorithm of construction of a CS to a set of strings, then, we evaluate its time complexity.

In the third section, we measure the performances of our algorithm.

Finally, in the last section, we present our conclusion.

1. DEFINITIONS AND NOTATIONS

Let A be a finite alphabet, a *string* is an element of A^*, it is a concatenation of elements of A. The *length* of a string w, denoted by $|w|$, is the number of the characters that constitute this string. By convention, the null length string will be denoted by ε and $A^+=A^*\backslash\{\varepsilon\}$. A portion of w beginning at the position i and ending at the position j, $0<i\leq j\leq|w|$, is called *substring* of w. When i=1 and $1\leq j\leq n$ the substring $w_{1,j}$ is called *prefix* of w and when $1\leq i\leq n$ and $j=n$ then the substring $w_{i,n}$ is called *suffix* of w.

Let f be a set of strings, a *Common Subsequence* (CS) to the strings of f is a list of substrings that appear in the same order, and without overlappings, in all the strings of f. The *length* of a CS s, denoted by $|s|$, is the sum of the lengths of the substrings that make up *s*. The null length CS will be denoted by λ. A CS *s*, $s=[sw_1, sw_2, \dots , sw_n]$, will be denoted by $sw_1 \rightarrow sw_2 \rightarrow \dots \rightarrow sw_n$. A portion of *s* beginning at sw_i and ending at sw_j, $0<i\leq j \leq n$, is called *sub-CS* of *s* and will be denoted by $sw_i \rightarrow sw_{i+1} \rightarrow \dots \rightarrow sw_j$.

A list $p_x=[p_1, p_2, \dots , p_N]_x$ made up by the positions of the first characters of *x*, in the strings w_1, w_2, …, w_N, is called *alignment* of *x*. The *width* of an alignment p_x, denoted by $\delta(p_x)$, is defined by :

$$\delta(p_x)= max_{i\in [1 .. N]}\{p_i\}$$
$$min_{i\in [1 .. N]}\{p_i\} \qquad (1)$$

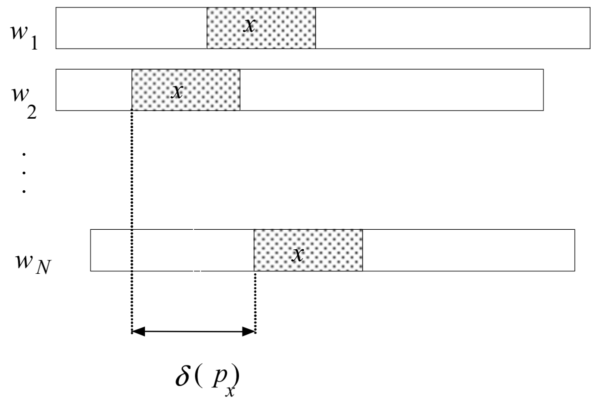

Fig. 1. $\delta(p_x)$ is the width of the alignment p_x.

2. CONSTRUCTION OF A COMMON SUBSEQUENCE

Let $f=\{w_1, w_2, \dots , w_N\}$ be a set of strings and *s* be an *N*-CS to *f*, the more *s* is made up by longer substrings the more *s* is longer. It is within this scope that we deal with the *N*-LCS problem, *N*>2.

2.1. Description of the Algorithm

Our algorithm of construction of an *N*-CS to a set of strings $f=\{w_1, w_2, \dots , w_N\}$, operates by a *divide-and-conquer* strategy [Aho *et al.* 74] : first, we locate a *longest* *common substring*, let us call it *sw*, appearing, approximately, in the same position in all the strings of *f*. This substring partitions each string w_i into three smaller substrings : w_{il}, *sw* and w_{ir} such that $w_i = w_{il} sw w_{ir}$. This partition gives rise to two new sets : $f_1=\{w_{1l}, w_{2l}, \dots , w_{Nl}\}$ and $f_2=\{w_{1r}, w_{2r}, \dots , w_{Nr}\}$. Then, we process, recursively, f_1 and f_2 in the same way as *f*. Let us call, respectively, s_1 and s_2 the *N*-CS to f_1 and f_2. Then, the *N*-CS to *f* is $s=s_1 \rightarrow sw \rightarrow s_2$.

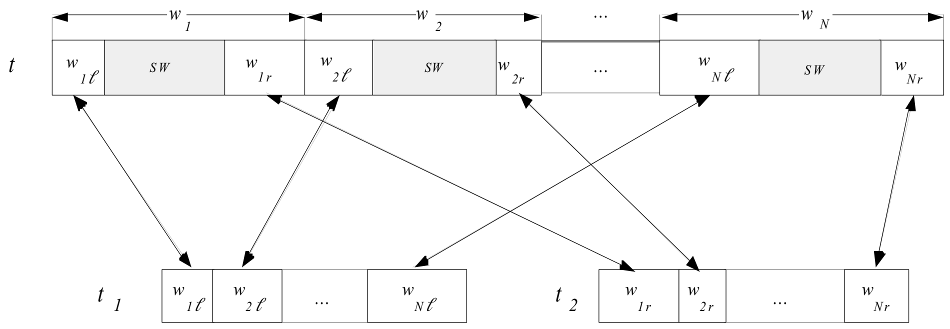

Fig. 2. t_1 (resp. t_2) is the concatenation of strings of f_1 (resp. f_2)

The construction of *the* longest common substring to the strings of *f* can be made *via* an adaptation of Ukkonen's online linear algorithm [Ukkonen 95] to construct a *Generalized Suffix Tree* (GST) that represent the different substrings that appear in the strings of *f*.

If we have more than one longest common substring, we make a *filtering*. The filtering is achieved in two steps :

During the first step, we make a *horizontal filtering* : It consists in filtering the different occurrences, in a string, of a same longest common substring. We operate as follows : with each longest common substring, we associate a *pivot string*. This string can be the one that contains the maximum number of occurrences of this longest common substring. At the level of this string, we align each occurrence of this longest common substring with the (*N*-1) nearest occurrences appearing in the (*N*-1) other strings. The alignment kept is the one that has the smallest width. Then, for a longest common substring, we consider, at the level of each string, only the occurrence taking part of the alignment that has the smallest width. Hence, for each longest common substring, we will have one occurrence per string.

During the second step, we make a *vertical filtering* : It consists in filtering the alignments associated with the different longest common substrings : we consider, only, the alignment with the smallest width. The longest common substring associated with this alignment is considered to be as the only longest common substring to the strings of *f*.

2.2. Time Complexity of the Algorithm

In this section, we study time complexity of our algorithm. Let *N* be the number of the strings and *L* be the maximum length of a string. We call *CS*, the algorithm that constructs CS, *LC_substrings*, the algorithm that constructs the longest common substrings, *Filtering*, the algorithm that filters the longest common substrings, and *Sub_CS*, the recursive algorithm that constructs sub-CS.

Time complexity of *Filtering* is $O(N*L^2)$. In fact, during each iteration (common substring) of *Filtering* :

(*i*) First, we make a horizontal filtering, i.e., we locate the corresponding minimum width alignment. This step is achieved by using a time of complexity $O(N*L)$.

(*ii*) Then, we make a vertical filtering : we have to check-up the corresponding width. Time complexity of this check-up is $O(1)$. Hence, when we make a vertical filtering then each iteration of *Filtering* is of complexity $O(N*L)$ in computing time.

Let us call *r* the maximum length of a common substring. The number of the common substrings with lengths equal to *r* is, at most, equal to $(L-r)$. Then we do, at most, $(L-r)$ iterations. Hence, *Filtering* is of complexity $O(N*L^2)$ in computing time.

Time complexity of *Sub_CS* is $O(N*L^2*log(L))$. Indeed, each call to *Sub_CS* generates one call to *LC_substrings* and another one to *Filtering*. The respective time complexities of these algorithms are $O(N*L)$ [Ukkonen 95] and $O(N*L^2)$. Then, time complexity of each call to *Sub_CS* is $O(N*L^2)$. On the other hand, each call to *Sub_CS* generates, at most, two other recursive calls to *Sub_CS*. Then, for strings with maximum length equal to *L*, we have, at most, $log_2(L)$ recursive levels. Hence, time complexity of *Sub_CS* is $O(N*L^2*log(L))$.

CS calls, successively, *LC_substrings*, *Filtering* and *Sub_CS*. The respective time complexities of these three algorithms are $O(N*L)$, $O(N*L^2)$ and $O(N*L^2*log(L))$. Then, time complexity of *CS* is $O(N*L^2*log(L))$.

3. MEASURES OF THE ALGORITHM PERFORMANCES

In this section, we measure the performances of our algorithm by comparing the lengths of the *N*-CS, constructed thanks to our algorithm, to those of the *N*-LCS.

Let *s* and *lcs* be, respectively, an *N*-CS and an *N*-LCS to *f*. Let us call the rate defined by :

$$\theta = \frac{|s|}{|lcs|} * 100 \qquad (2)$$

We have used this rate to compare the lengths of the *N*-CS, constructed thanks to our algorithm, to those of the *N*-LCS. We have made this comparison in the case where *N*>2, for strings constructed from :

(*i*) A 2 characters alphabet : the binary alphabet,

(*ii*) A 4 characters alphabet : the DNA and RNA alphabet,

(*iii*) And a 20 characters alphabet : the proteins alphabet.

For each of these alphabets, we have measured the performances of our algorithm on families of 5, 10, 20 and 50 strings of, approximately, equal lengths. The construction of each of these families is made within two steps [Ebabil 92] :

(*i*) During the first step, we generate randomly two strings of equal lengths, then, we construct a 2-LCS to these two strings by using the algorithm of Wagner and Fischer [Wagner and Fischer 74].

(*ii*) During the second step, we generate new strings, containing this 2-LCS, by inserting in a random way, characters between the substrings that constitute this 2-LCS. The set of the strings obtained in this way admits as an *N*-LCS the 2-LCS constructed during the first step.

Figures Fig. 3, 4 and 5 represent the results obtained for the different processed alphabets. All these results have been obtained through computing an average on 50 draws. The X-axis represents the strings length and the Y-axis represents the rate θ:

For a 2 characters alphabet:

(*i*) 41.06 %≤θ≤73.19% for families of 5 strings,

(*ii*) 39.31%≤θ≤72.03 for families of 10 strings,

(*iii*) 37.7%≤θ≤69.75% for families of 20 strings,

(*iv*) 34.62%≤θ≤66.06% for families of 50 strings.

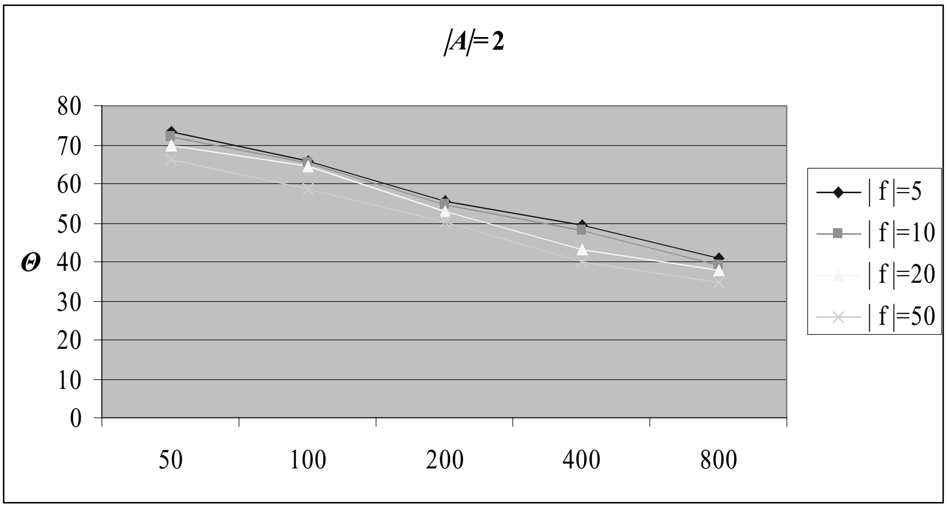

Fig. 3. Variation of the rate θ for a 2 characters alphabet.

For a 4 characters alphabet:

(*i*) 46.6%≤θ≤80.05% for families of 5 strings,

(*ii*) 46.76%≤θ≤80.35% for families of 10 strings,

(*iii*) 40.34%≤θ≤76.36% for families of 20 strings,

(*iv*) 36.62%≤θ≤71.99% for families of 50 strings.

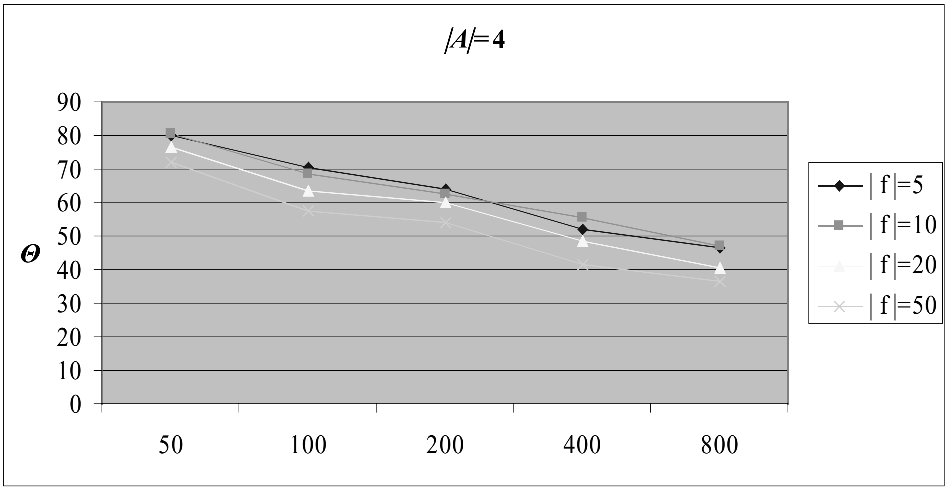

Fig. 4 Variation of the rate θ for a 4 characters alphabet.

Finally, for a 20 characters alphabet :

(*i*) 56.1%≤θ≤79.39% for families of 5 strings,
(*ii*) 58.14%≤θ≤83.48% for families of 10 strings,
(*iii*) 53.04%≤θ≤83.42% for families of 20 strings,
(*iv*) 39.78%≤θ≤82.32% for families of 50 strings.

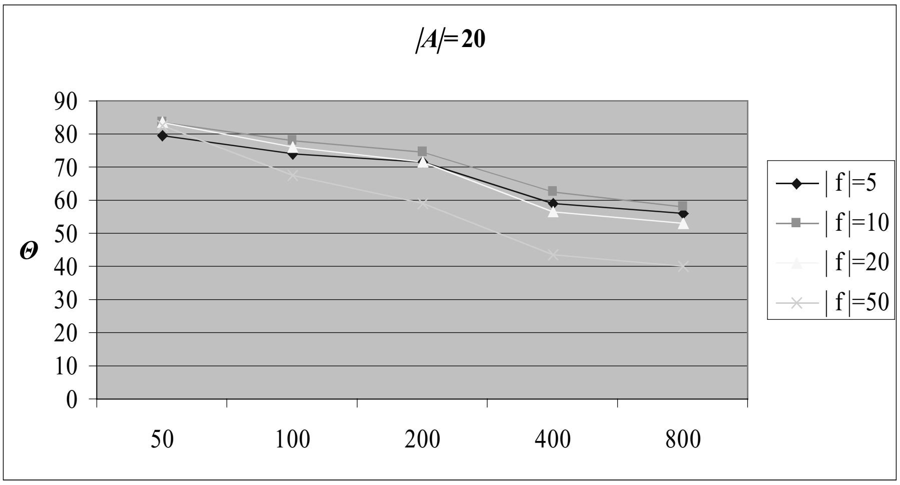

Fig. 5 Variation of the rate θ for a 20 characters alphabet.

4. CONCLUSION

In this paper, we have described A Heuristic algorithm, adopting a regions approach, for the *N-Longest Common Subsequence* (*N*-LCS) problem, *N*≥2. Our algorithm is

$O(N*L^2*log(L))$ in computing time, where N is the number of the strings and L is the maximum length of a string.

Our algorithm performed good results while having a low time complexity compared to other existing approximation algorithms [Hakata and Imai 92, Irving and Fraser 92, Vitte 96, Bonizzoni *et al.* 01]. We are now working on improving more the performances of this algorithm.

A variant of our algorithm has been developed to deal with the *Multiple Sequence Alignment* (MSA) problem [Elloumi and Mokaddem 08]. We have obtained interesting results.

REFERENCES

AHO A. V., HOPCROFT, J. E., AND ULLMAN, J. D. 1974. *The Design and analysis of computer algorithms*. Addison-Wesley Publishing Company, 60-65.

ALBERGA C. N. 1967. Strings similarity and misspellings. *Comm. of ACM 10*, *5*, 302-313.

ALTSCHUL, S. F., AND LIPMAN, D. J. 1989, Trees, stars, and multiple biological sequence alignment. *SIAM Journal of Applied Mathematics 49*, *1*, 197-209.

ALTSCHUL, S. F., GISH, W., MILLER, W., MEYERS, E. W., AND LIPMAN, D. J. 1990. Basic local alignment search tool. *J. Molecular Biology 215*, *3*, 403–410.

ANDREJKOVA, G. 1998. The longest restricted common subsequence Problem. *In Proceeding of the Prague Stringology Club Workshop*, Prague, Slovakia.

ARSLAN, A. N., AND EGECIOGLU, Ö. 2004, Algorithms for the constrained longest common subsequence problems; *In Proceeding of The Prague Stringology Conference*, Prague, Slovakia, 24-32.

ARSLAN, A. N., AND EGECIOGLU, Ö. 2005. Algorithms for the constrained longest common subsequence problems. *J. Found. Comput. Sci. 16*, *6*, 1099-1109.

BELLMAN, R. E. 1957. *Dynamic programming*, Princeton University Press, New Jersey.

Bellman, R. E., Dreyfus, S. E. 1962. *Applied dynamic programming*, Princeton University Press, New Jersey.

BEREG, S., ZHU, B. 2005. RNA multiple structural alignment with longest common subsequences. *In Proceeding of Computing and Combinatorics*, 32-41.

BEYER, W. A., STEIN, M. L., SMITH, T. F., ULLMAN, S. M. 1974. A molecular-sequence metric and evolutionary tree. *Mathematical Biosciences 19*, 9-25.

BLIN, G., FERTIN, G., RIZZI, R., AND VIALETTE, S. 2005. What makes the arc-preserving subsequence problem hard. *In Proceeding of International Conference on Computational Science*, 860-868.

BONIZZONI, P., AND VEDOVA, G. D. 2001. The complexity of multiple sequence alignment with SP-score that is a metric. *Theoretical Computer Science 259*, *1*, 63-79.

DAMERAU, F. J. 1964. A technique for computer detection and correction of spelling errors. *Comm. of ACM 7*, *3*, 171-176.

DAY, W. H. E., JOHNSON, D. S., AND SANKOFF, D. 1986. The computational complexity of inferring rooted phylogenies by parsimony. *Mathematical Biosciences 81*, 33-42.

DIXON, N. R., AND MARTIN, T. B. 1979. *Automatic speech and speaker recognition*, IEEE Press.

DOOLITTLE, R. F. 1990. Molecular evolution : computer analysis of protein and nucleic acid sequences. *Methods in Enzymology 183*.

EBABIL, T. 1992. *Méthode de subdivision pour l'alignement multiple*. Informatics and Mathematics, Master Thesis, University of Aix-Marseilles II.

ELLOUMI M., AND MOKADDEM A., 2008. An algorithm for multiple and global alignments. *In Proceedings of Workshop Algorithms in Molecular Biology,* Vienna, Austria, July 2008.

EVANS, P. 1999. *Algorithms and complexity for annotated sequence analysis*. Ph.D Thesis, University of Victoria.

FU, K. S., AND LU, S. Y. 1977. A clustering procedure for syntactic patterns. *In Proceeding of IEEE Trans. on Systems, Man and Cybernetics*, 734-742.

GOAD, W. B., AND KANEHISA, M. I. 1982. Pattern recognition in nucleic acid sequences, I. A general method for finding local homologies and symmetries. *Nucleic Acid Reseach 10*, *1*, 247-263.

GUSFIELD, D. 1990. Algorithms for inferring evolutionary trees. *Networks*.

HAKATA, K. AND IMAI, H. 1992. The longest common subsequence problem for small alphabet size between many strings. *In Proceedings of the Third International Symposium on Algorithms and Computation*, Lecture Notes in Computer Science, Springer Verlag, 469-478.

IRVING, R. W. AND FRASER, C. B., 1992. Two algorithms for the longest common subsequence of three (or more) strings. *In Proceedings of the Fourth Annual Symposium on Combinatorial Pattern Matching*, Lecture Notes in computer Science, Springer Verlag, 214-229.

JIANG, T., LIN, G., MA, B., AND ZHANG, K. 2000. The longest common subsequence problem for arc-annotated sequences. *In Proceedings of Combinatorial Pattern Matching*, 154–165.

JIANG, T., AND LI, M. 1995. On the approximation of shortest common supersequences and longest common subsequences. *SIAM Journal of Computing 24*, *5*, 1122-1139.

KANNAN, S., AND WARNOW, T. 1990. Inferring evolutionary history from DNA sequences. *In Proceedings of 31st Annual IEEE Symposium on Foundation of Computer Science*, 326-371.

LELEWER, D. A., AND HIRSCHBERG, D. S. 1987. Data compression. *ACM Computing Survey 3*, *19*, 261-287.

LIN, G. H., CHEN, Z. Z., JIANG, T., AND WEN, J. 2002. The longest common subsequence problem for sequences with nested arc annotations. *J. Comput. Syst. Sci. 65*, *3*, 465-480.

MAIER, D. 1978. The complexity of some problems on subsequences and super sequences. *J. of ACM 25*, *2*, 322-336.

MASEK, W. J., AND PATERSON, M. 1980. A faster algorithm computing string edit distances. *J. Comput. Syst. Sci. 20*, *1*, 18-31.

MORGAN, H. L. 1970. Spellings correction in systems programs. *Comm. of ACM 13*, *2*, 90-94.

PEARSON, W., AND LIPMAN, D. 1988. Improved tools for biological sequence comparison. *In Proceedings of National Academy of Science of the U.S.A. 85*, 2444-2448.

SAKOE, H., AND CHIBA, S. 1978. Dynamic programming algorithms optimization for spoken substring recognition. *IEEE Trans. Acous.*, *1*, *1*, 43-49.

SANKOFF, D. 1975. Minimal mutation trees of sequences. *SIAM Journal of Applied Mathematics 78*, 35-42.

SANKOFF, D., AND KRUSKAL, J. B. 1983. *Time warps, string edits and macromolecules: The theory and practice of sequence comparison*. Reading, MA: Addison-Wesley.

SELLERS, F. F. 1962. Bit loss and gain correction code. *IRE Trans. on Information Theory 8*, 35-38.

UKKONEN, E. 1995. On-line construction of suffix trees, *Algorithmica 14*, 249–260.

Wang, L., AND Jiang, T. 1994.On the complexity of multiple sequence alignment. *J. Computational Biology 1*, 337-348.

WAGNER, R. A., AND FISCHER, M. J. 1974. The string-to-string correction problem. *J. of ACM 21*, *1*, 168-173.

WATERMAN, M. S., SMITH, T. F., AND BEYER, W. A. 1976. Some biological sequence metrics. *Adv. Math. 20*, 367-387.

WATERMAN, M. S. 1984. General methods of sequence comparison. *Bull. Math. Biol. 46*, *4*, 473-500.

WATERMAN, M. S. 1989. *Mathematical methods for DNA sequences*, CRC Press.

WILBUR. W. J., AND LIPMAN, D. J. 1983. Rapid similarity searches of nucleic acid and protein data banks. *In Proceedings of the National Academy of Science of the U.S.A. 18*, 726-730.

Mourad Elloumi , Ahmed Mokaddem

Research Unit of Technologies of Information and Communication,
Higher School of Sciences and Technologies of Tunis. 5,
Avenue Taha Hussein,
Monfleury 1008 Tunis,
Tunisia

Received september 2007