

Exemplar Longest Common Subsequence

Paola Bonizzoni, Gianluca Della Vedova, Riccardo Dondi, Guillaume Fertin,
Raffaella Rizzi and Stéphane Vialette

Abstract

In this paper, we investigate the computational and approximation complexity of the Exemplar Longest Common Subsequence of a set of sequences (ELCS problem), a generalization of the Longest Common Subsequence problem, where the input sequences are over the union of two disjoint sets of symbols, a set of mandatory symbols and a set of optional symbols. We show that different versions of the problem are **APX**-hard even for instances with two sequences. Moreover, we show that the related problem of determining the existence of a feasible solution of the Exemplar Longest Common Subsequence of two sequences is **NP**-hard. On the positive side, we first present an efficient algorithm for the ELCS problem over instances of two sequences where each mandatory symbol can appear in total at most three times in the sequences. Furthermore, we present two fixed-parameter algorithms for the ELCS problem over instances of two sequences where the parameter is the number of mandatory symbols.

Index Terms

Longest common subsequence, comparative genomics, algorithm design and analysis, combinatorial algorithms, analysis of algorithms and problem complexity.

P. Bonizzoni is with Dipartimento di Informatica, Sistemistica e Comunicazione, Università degli Studi di Milano-Bicocca, Milano - Italy. Email: bonizzoni@disco.unimib.it.

G. Della Vedova is with Dipartimento di Statistica, Università degli Studi di Milano-Bicocca, Milano - Italy. Email: gianluca.dellavedova@unimib.it.

R. Dondi is with Dipartimento di Scienze dei Linguaggi, della Comunicazione e degli Studi Culturali, Università degli Studi di Bergamo, Bergamo - Italy. Email: riccardo.dondi@unibg.it.

G. Fertin is with LINA - FRE CNRS 2729 Université de Nantes, Nantes Cedex 3, France. Email: fertin@lina.univ-nantes.fr

R. Rizzi is with Dipartimento di Informatica, Sistemistica e Comunicazione, Università degli Studi di Milano-Bicocca, Milano - Italy. Email: rizzi@disco.unimib.it.

S. Vialette is with LRI - UMR CNRS 8623 Faculté des Sciences d'Orsay, Université Paris-Sud Bât 490, Orsay Cedex, France. Email: Stephane.Vialette@Iri.fr

I. INTRODUCTION

Algorithmic studies in comparative genomics have produced powerful tools for the analysis of genomic data which has been successfully applied in several contexts, from gene functional annotation to phylogenomics and whole genome comparison. A main goal in this research field is to explain differences in gene order in two (or more) genomes in terms of a limited number of rearrangement operations.

When there are no duplicates in the considered genomes, the computation of the similarity measure is usually polynomial-time solvable, *e.g.*, number of breakpoints, reversal distance for signed genomes, number of conserved intervals, number of common intervals, maximum adjacency disruption, summed adjacency disruption [8]–[10]. However, except for a few exceptions, several copies of the same gene or several highly homologous genes are usually scattered across the genome, and hence it is a major problem to handle those duplicates when computing the similarity between two genomes. One approach to overcome this difficulty is based on the concept of *exemplar* [11]: for each genome, an exemplar sequence is constructed by deleting all but one occurrence of each gene family. Another approach is based on *matching* [12]: in this two-step procedure, the two genomes are first made *balanced* (the number of occurrences of genes from the same family must be the same in both genomes) by removing a minimum number of genes and next a one-to-one correspondence (among genes of each family) between genes of the genomes is computed.

Unfortunately, in the presence of duplicates, most similarity measures turn out to be **NP**-hard to be computed [12]–[15] for both the exemplar and the matching models, so we generally have to rely on approximation algorithms or heuristic approaches. We discuss here one such general heuristic approach, the EXEMPLAR LCS problem, which is basically a constrained string alignment problem. The basic idea of the general framework we propose here is based on the observation that, for most similarity measures and for both the exemplar and the matching models, specific common subsequences may correspond to highly conserved sets of genes. This suggests the following greedy heuristic algorithm: find a common subsequence of significant length – but compact enough – between the two genomes, replace in the two genomes the substring that contains the common subsequence (the substring that starts at the first character of the common subsequence and ends at the last character of the common subsequence) by a new letter and

continue in a similar way. Observe that after we have identified a common subsequence of the genomes, we can establish a one-to-one correspondence between genes of the two genomes.

At each iteration of this simple heuristic algorithm, one however has to be cautious in how to choose the common subsequence, as bad choices may have a disastrous impact for the rest of the algorithm. Let us take the exemplar model as a very simple explanatory example, and suppose that we are searching for a common subsequence between two precise substrings of the two genomes. For one, if one gene family has occurrences elsewhere in the two genomes, then taking or not one occurrence of this particular gene family in the common subsequence is thus not based on necessity but on the length of the obtained solution. For another, if there do not exist any other occurrences of one gene family except one in the two considered substrings, definitively one has to take this occurrence in the common subsequence (observe that in this case the obtained common subsequence may not be the longest one). This simple example suggests to consider an LCS-like problem that deals with two types of letters (*mandatory* and *optional* symbols) to allow greater flexibility in the searching process.

In this paper we will formally define such a framework with a simple combinatorial problem that generalizes the well-known LCS problem and we will study its computational and approximation complexity. We show that some different versions of the problem are **APX**-hard even for instances with two sequences and that even determining if a feasible solution exists or not is **NP**-hard. On the positive side, the hardness of the problem can be limited in some cases; in fact we show that it is possible to determine efficiently a feasible solution, provided that each symbol appears at most three times in total in the input sequence. Finally, we present two fixed-parameter algorithms, where the parameter is the number of mandatory symbols.

II. THE PROBLEMS

The LONGEST COMMON SUBSEQUENCE problem (shortly LCS) is a well-known problem in Computational Biology. Let $s = s[1], s[2], \dots, s[m]$ and $t = t[1], t[2], \dots, t[l]$ be two sequences, s is a subsequence of t if for some $j_1 < j_2 < \dots < j_m$, $s[h] = t[j_h]$. Let S be a set of sequences, then a *longest common subsequence* of S is a longest possible sequence s that is a subsequence of each sequence in S .

A simple way to informally define a subsequence is by using the notion of *threading scheme*. First write the two sequences on two parallel lines, then a threading scheme is a set of lines, each

Problem name	Occurrences mandatory symbols	Occurrences optional symbols
ELCS(1, ≤ 1)	exactly 1	at most 1
ELCS(1)	exactly 1	unrestricted
ELCS(≥ 1, ≤ 1)	at least 1	at most 1
ELCS(≥ 1)	at least 1	unrestricted

TABLE I

VERSIONS OF EXEMPLAR LCS

one connecting two identical symbols of different sequences, so that no two lines are crossing. In this case a common subsequence consists of symbols connected by the non-crossing lines.

Given a set of sequences S , the LCS problem asks for a *longest common subsequence* of S . The complexity of the LCS problem has been deeply studied in the past. In [7] it is shown that the problem is **NP**-hard even for sequences over binary alphabet. However, when the instance of the problem consists of a fixed number of sequences, the LCS can be solved in polynomial-time via dynamic programming algorithms [4], [5], [16].

The EXEMPLAR LCS problem (ELCS) is related to the LCS problem. The input of the ELCS problem consists of a set S of sequences over alphabet $A_o \cup A_m$, $A_o \cap A_m = \emptyset$, where A_o is the set of *optional* symbols and A_m is the set of *mandatory* symbols. The output of the problem is a longest common subsequence of all sequences in S that contains all mandatory symbols. Next we state formally the ELCS problem.

Problem 1: ELCS PROBLEM

Input: a set S of sequences over alphabet $A_o \cup A_m$, where A_o is the set of *optional* symbols and A_m is the set of *mandatory* symbols. The sets A_o , A_m are disjoint.

Output: a longest common subsequence of all sequences in S that contains an occurrence of each mandatory symbol in A_m .

Given an instance S of ELCS, by *exemplar common subsequence* we mean a feasible solution of ELCS over S . It is possible to define different versions of the problem, according to the number of occurrences of each symbol in the solution, as represented in Table I. In this paper we will deal with such different versions of ELCS. First notice that ELCS(1) and ELCS(≥ 1) are

generalizations of the LCS problem. Indeed, the LCS problem can be seen as the restriction of $\text{ELCS}(1)$ and $\text{ELCS}(\geq 1)$ with an empty set of mandatory symbols. Therefore all the hardness results for LCS apply to $\text{ELCS}(1)$ and $\text{ELCS}(\geq 1)$. Moreover, we will show that the above problems are hard also on instances of only two sequences (while the LCS problem can be solved in polynomial-time for any fixed number of sequences). When dealing with the restriction of ELCS containing only a fixed number of sequences, we will denote such restriction prefixing the problem name with the number of sequences, e.g. $2\text{-ELCS}(1, \leq 1)$ is the restriction of $\text{ELCS}(1, \leq 1)$ to instances of two sequences.

III. COMPLEXITY RESULTS

In this section we investigate the complexity of the $2\text{-ELCS}(1, \leq 1)$ problem and the $2\text{-ELCS}(\geq 1, \leq 1)$ problem. More precisely we will show that both problems are **APX**-hard even when restricted to instances where each symbol appears at most twice in each input sequence.

A. Complexity of $2\text{-ELCS}(1, \leq 1)$

We prove that $2\text{-ELCS}(1, \leq 1)$ is **APX**-hard even when each symbol appears at most twice in each input sequence via an L-reduction from MAX INDEPENDENT SET problem on Cubic Graph (MISC) to $2\text{-ELCS}(1, \leq 1)$, since the MISC problem is known to be **APX**-hard [1]. The MISC problem is defined as follows:

Problem 2: MISC PROBLEM

Input: $G = (V, E)$ a cubic graph.

Output: a set $V' \subseteq V$ of maximum size, such that no two vertices $u, v \in V'$ are adjacent.

Let $G = (V, E)$ be a cubic graph. Since G is cubic, for each vertex $v_i \in V$ there are exactly three edges incident on it; denote by $e_1(v_i), e_2(v_i), e_3(v_i)$ these edges. The reduction associates with each vertex v_i a symbol v_i of A_o and a symbol x_i in A_m . Furthermore, the reduction associates with each edge $e_j \in E$ a distinct symbol $s_j \in A_m$.

Let $v_i \in V$ and let $e_1(v_i), e_2(v_i)$ and $e_3(v_i)$ be the edges incident on it. In what follows, we denote by $s(e_1(v_i)), s(e_2(v_i)), s(e_3(v_i))$ respectively the symbols of A_m associated by the reduction with edges $e_1(v_i), e_2(v_i)$ and $e_3(v_i)$. Notice that each edge $e = (v_i, v_j)$ appears in the incidence lists of both v_i and v_j , thus e will be denoted by $e_x(v_i)$ and $e_y(v_j)$, for some

$1 \leq x, y \leq 3$, in the incidence list of v_i and v_j respectively. Nonetheless observe that e is mapped to one distinct symbol of A_m , that is $s(e_x(v_i)) = s(e_y(v_j))$.

Define a *block* associated with a vertex v_i , as a string consisting of a vertex symbol v_i , the symbols associated with edges incident to v_i in G and the symbol x_i . There are two blocks associated with v_i , one contained in s_1 and defined as $b_1(v_i) = v_i s(e_1(v_i)) s(e_2(v_i)) s(e_3(v_i)) x_i$, the second contained in s_2 and defined as $b_2(v_i) = s(e_1(v_i)) s(e_2(v_i)) s(e_3(v_i)) v_i x_i$. The instance of 2-ELCS($1, \leq 1$) consists of the two sequences $s_1 = b_1(v_1) b_1(v_2) \cdots b_1(v_n)$, that is $s_1 = v_1 s(e_1(v_1)) s(e_2(v_1)) s(e_3(v_1)) x_1 v_2 \cdots x_{n-1} v_n s(e_1(v_n)) s(e_2(v_n)) s(e_3(v_n)) x_n$ and the sequence $s_2 = b_2(v_1) b_2(v_2) \cdots b_2(v_n)$ which, just as we have done for s_1 , can be expanded to $s_2 = s(e_1(v_1)) s(e_2(v_2)) s(e_3(v_3)) v_1 x_1 v_2 \cdots x_{n-1} s(e_1(v_n)) s(e_2(v_n)) s(e_3(v_n)) v_n x_n$.

Lemma 1: Each exemplar common subsequence contains the symbol x_i , and x_i is taken from blocks $b_1(v_i)$ and $b_2(v_i)$.

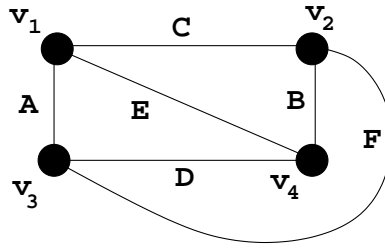
Proof: Observe that each symbol x_i is mandatory, hence it must appear in any feasible solution of 2-ELCS($1, \leq 1$). Furthermore observe that there is only one occurrence of x_i in s_1 and in s_2 . More precisely x_i occurs in block $b_1(v_i)$ in s_1 and in block $b_2(v_i)$ in s_2 . It follows that any symbol x_i in a feasible solution of 2-ELCS($1, \leq 1$) over s_1 and s_2 must be taken from blocks $b_1(v_i)$ and $b_2(v_i)$. ■

Thus, we can divide an exemplar common subsequence s in n blocks, where block i of s starts after the positions containing the symbol x_{i-1} (or with the first symbol of s if $i = 1$) and ends in the position containing x_i . Observe that, since each x_i must appear in any exemplar common subsequence, each block of s contains at least one symbol of the solution.

Lemma 2: The i -th block of an exemplar common subsequence s contains either the symbol v_i or some symbols in $s(e_1(v_i)) s(e_2(v_i)) s(e_3(v_i))$.

Proof: Observe that, by Lemma 1, block i of s can contain only symbols from blocks $b_1(v_i)$ and $b_2(v_i)$. Furthermore observe that if symbol v_i is in an exemplar common subsequence s , then s does not contain any symbol of $s(e_1(v_i)) s(e_2(v_i)) s(e_3(v_i))$ of $b_1(v_i)$ and $b_2(v_i)$, otherwise it is easy to see that this block of s will not be a subsequence of i -th block of s_1 or s_2 .

Now assume that none of the symbols of $s(e_1(v_i)) s(e_2(v_i)) s(e_3(v_i))$ belongs to the i -th block of s . Then if v_i does not belong to the i -th block of s , we can obtain a better solution adding v_i to the i -th block of s . ■



$$s_1 = v_1CAEx_1v_2CFBx_2v_3AFDx_3v_4EBDx_4$$

$$s_2 = CAEv_1x_1CFBv_2x_2AFDv_3x_3EBDv_4x_4$$

Fig. 1. The cubic graph K_4 and its associated instance of $2\text{-ELCS}(1, \leq 1)$

Hence a feasible solution s of $2\text{-ELCS}(1, \leq 1)$ over s_1, s_2 consists of $f_1x_1 \dots f_ix_i \dots f_nx_n$, where each block f_i is either v_i or a subsequence of $s(e_1(v_i))s(e_2(v_i))s(e_3(v_i))$.

Theorem 3: The $2\text{-ELCS}(1, \leq 1)$ problem is **APX**-hard even when each symbol appears at most twice in each input sequence.

Proof: Consider the symbols of a common subsequence s contained in $b_1(v_i)$ and $b_2(v_i)$. The common subsequence s contains the symbol x_i and either v_i or some symbols in $e_1(v_i)e_2(v_i)e_3(v_i)$. Observe that each edge symbol is mandatory, which means that it must appear exactly once in a common subsequence. Moreover, an edge symbol encoding edge (v_i, v_j) appears in blocks $b_1(v_i)$ and $b_1(v_j)$ of s_1 and in blocks $b_2(v_i)$ and $b_2(v_j)$ of s_2 . Thus a common subsequence takes such edge symbol either from $b_1(v_i)$ and $b_2(v_i)$ or from $b_1(v_j)$ and $b_2(v_j)$.

Let I be the set of vertices appearing in s , we will show that I is an independent set of G . Assume that symbols $v_i, v_j \in I$. Then (v_i, v_j) is not an edge of G , otherwise s in f_i and f_j contains symbols v_i and v_j respectively. An immediate consequence is that the edge symbol associated with $e = (v_i, v_j)$, that can appear only in f_i and f_j , is not contained in s . Since each edge symbol is mandatory, it must appear in any feasible solution of $2\text{-ELCS}(1, \leq 1)$, which is a contradiction. Observe that the length of a feasible solution s of $2\text{-ELCS}(1, \leq 1)$ over s_1, s_2 is $|V| + |E| + |I|$, where I is an independent set of G . Indeed, s will contain some symbols associated with an independent set I and one occurrence of each mandatory symbol. Notice that the set of mandatory symbols has size $|V| + |E|$.

On the other side, let I be an independent set of G , we can compute a feasible solution of $2\text{-ELCS}(1, \leq 1)$ over s_1, s_2 of size $|V| + |E| + |I|$, retaining in the exemplar common subsequence

only the symbols associated with vertices in I . Since I is an independent set, for each edge $e = (v_i, v_j)$ at least one of v_i, v_j is not in I , hence each symbol associated with e can be retained once in a feasible solution of 2-ELCS($1, \leq 1$) over s_1, s_2 . ■

B. Complexity of 2-ELCS($\geq 1, \leq 1$)

Next we show that 2-ELCS($\geq 1, \leq 1$) is also **APX**-hard with a reduction similar to the previous one. Let $G = (V, E)$ be a cubic graph, for each vertex $v_i \in V$, we introduce four optional symbols $v_i^a v_i^b v_i^c v_i^d$, and the blocks $b_1(v_i)$ and $b_2(v_i)$ associated with v_i in sequences s_1 and s_2 respectively are defined as follows: $b_1(v_i) = v_i^a v_i^b v_i^c v_i^d s(e_1(v_i))s(e_2(v_i))s(e_3(v_i))x_i$; $b_2(v_i) = s(e_1(v_i))s(e_2(v_i))s(e_3(v_i))v_i^a v_i^b v_i^c v_i^d x_i$. Notice that x_i and $s(e_1(v_i)), s(e_2(v_i)), s(e_3(v_i))$ are all mandatory symbols.

Since the symbols x_i are mandatory and there is only one occurrence of each x_i in s_1 and s_2 , it follows that Lemma 1 holds also for this problem. Each symbol x_i appears in blocks $b_1(v_i)$ and $b_2(v_i)$ of s_1 and s_2 respectively, and any symbol x_i in an exemplar common subsequence must be taken from the blocks of s_1, s_2 associated with v_i , that is $b_1(v_i)$ and $b_2(v_i)$. Since each mandatory edge symbol appears twice in each input sequence, it must appear once or twice in a common subsequence.

Lemma 4: The i -th block of an exemplar common subsequence s contains either sequence $v_i^a v_i^b v_i^c v_i^d$ or some symbols in $s(e_1(v_i))s(e_2(v_i))s(e_3(v_i))$.

Proof: It is easy to see that if any symbol of the sequence $v_i^a v_i^b v_i^c v_i^d$ is in a feasible solution of 2-ELCS($\geq 1, \leq 1$) over s_1, s_2 , then this solution does not contain any occurrences of symbols of sequence $s(e_1(v_1))s(e_2(v_1))s(e_3(v_1))$ in $b_1(v_i)$ and $b_2(v_i)$. This means that a feasible solution s of 2-ELCS($\geq 1, \leq 1$) over s_1, s_2 consists of $g_1 x_1 \dots g_i x_i \dots g_n x_n$, where each g_i is either a subsequence of $v_i^a v_i^b v_i^c v_i^d$ or a subsequence of $s(e_1(v_i))s(e_2(v_i))s(e_3(v_i))$.

Now assume that none of the symbols of $s(e_1(v_i))s(e_2(v_i))s(e_3(v_i))$ belongs to the i -th block of s . Then if some of the symbols $v_i^a v_i^b v_i^c v_i^d$ do not belong to the i -th block of s , we can obtain a better solution adding it to the i -th block of s . Conversely if none of the symbols of $v_i^a v_i^b v_i^c v_i^d$ belongs to the i -th block of s , then we having the sequence $v_i^a v_i^b v_i^c v_i^d$ in the i -th block of s does not shorten s . ■

Observe that each edge symbol is mandatory, which means that it must appear exactly once in an exemplar common subsequence. Thus an exemplar common subsequence takes each edge

symbol from one of the two blocks where it appears.

Theorem 5: The 2-ELCS($\geq 1, \leq 1$) problem is **APX**-hard even when each symbol appears at most twice in each input sequence.

Proof: Let I be an independent set of G , then $s = g_1x_1 \dots g_ix_i \dots g_nx_n$, where each $g_i = v_i^a v_i^b v_i^c v_i^d$ if $v_i \in I$ and $g_i = s(e_1(v_i))s(e_2(v_i))s(e_3(v_i))$ otherwise. It is immediate to note that s is a common subsequence of s_1 and s_2 of length $|V| + 3(|V| - |I|) + 4|I| = 4|V| + |I|$ and that all mandatory symbols encoding an edge are included in s . W.l.o.g. assume to the contrary that a symbol encoding the edge (v_1, v_2) is not included in s . This fact implies that $g_1 = v_1^a v_1^b v_1^c v_1^d$ and $g_2 = v_2^a v_2^b v_2^c v_2^d$, hence $v_1, v_2 \in I$, contradicting the assumption that I is an independent set of G .

Assume now that there exists a feasible solution s of 2-ELCS($\geq 1, \leq 1$) over s_1, s_2 with length $4|V| + |I|$. We can assume that, for each block, either $v_i^a v_i^b v_i^c v_i^d$ or $s(e_1(v_i))s(e_2(v_i))s(e_3(v_i))$ appears as a substring of s . Let Y be the set of blocks for which $v_i^a v_i^b v_i^c v_i^d$ is part of s . Hence the vertices corresponding to Y are an independent set of G . By a trivial counting argument, it is easy to show that for $|I|$ blocks, s includes $v_i^a v_i^b v_i^c v_i^d$. We claim that such blocks encode an independent set. W.l.o.g. assume that $v_1^a v_1^b v_1^c v_1^d$ and $v_2^a v_2^b v_2^c v_2^d$ are included in s , then there is no edge (v_1, v_2) in G , otherwise the mandatory symbol encoding such edge would not be in s . ■

IV. EXISTENCE OF A FEASIBLE SOLUTION

Given an instance of 2-ELCS, a problem related to 2-ELCS is that of determining if a feasible solution exists. In what follows we will consider a general version of the 2-ELCS problem, where the instance consists of two sequences s_1, s_2 over alphabet $A_o \cup A_m$ and we want to compute if there exists a subsequence of s_1 and s_2 containing all the mandatory symbols in A_m . Observe that computing if a feasible solution of 2-ELCS exists implies computing if a feasible solution exists for each of the problems 2-ELCS($1, \leq 1$), 2-ELCS(1), 2-ELCS($\geq 1, \leq 1$) and 2-ELCS(≥ 1). Notice that both reductions described in the previous section hold for instances that are known to admit a feasible solution, therefore they are not sufficient for dealing with the problem.

A simple observation allows to simplify the complexity of the problem: in fact only mandatory symbols are relevant, as removing all optional symbols does not change the fact that a feasible solution exists or not. Therefore in what follows we can assume that both input sequences are

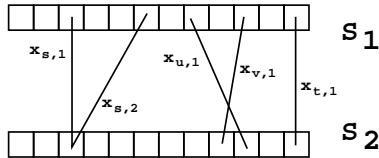


Fig. 2. Reducing 2-ELCS to 2SAT

made only of mandatory symbols. Clearly, in order to have a feasible solution, each mandatory symbol must appear in both input sequences s_1 and s_2 . It is trivial to verify in polynomial-time such property, hence in what follows we assume that all mandatory symbols appear in both input sequences.

The number of occurrences of each mandatory symbol in the instance is a fundamental parameter when studying the complexity of the 2-ELCS problem. Indeed, we will show that finding a feasible solution can be done in polynomial-time for small values of such parameter, but becomes intractable when each symbol occurs three times in each input sequence.

A. A polynomial-time algorithm

First we investigate the case where each mandatory symbol appears in total at most three times in the input sequences. We will present a polynomial-time algorithm for this case, via a reduction to 2SAT (the restriction of SATISFIABILITY to instances where each clause contains at most two literals). It is well known that 2SAT can be solved in linear time [2].

For each symbol s , let $o_1(s)$ (respectively $o_2(s)$) be the set of positions of the input sequence s_1 (resp. s_2) where s appears. Clearly both $o_1(s)$ and $o_2(s)$ are not empty and $|o_1(s)| + |o_2(s)| \leq 3$. It follows that for each symbol s , there exists one of s_1 and s_2 containing exactly one occurrence of s , while in the other sequence there are one or two occurrences of s . It follows that for each symbol s there are at most two pairs in $o_1(s) \times o_2(s)$, for otherwise $|o_1(s)| + |o_2(s)| > 3$. Let us associate with each of such pairs a variable $x_{s,i}$, where $i \in \{1, 2\}$ if there are two pairs in $o_1(s) \times o_2(s)$ and $i = 1$ if there is only one pair in $o_1(s) \times o_2(s)$. Graphically the possible variables are represented in Fig. 2 with a line connecting two identical symbols belonging to different sequences. The case $|occ_1(s)| + |occ_2(s)| = 3$ is represented by the two leftmost lines and the variables $x_{s,1}$, $x_{s,2}$, while the case $|occ_1(s)| + |occ_2(s)| = 2$ is represented by the rightmost line

and the variable $x_{t,1}$. Each truth assignment to the variables can be viewed as picking the lines corresponding to true variables.

Let C be the set of clauses of the instance of 2SAT that we are constructing. For each pair $x_{s,1}, x_{s,2}$ of variables, the clauses $\neg x_{s,1} \vee \neg x_{s,2}$ and $x_{s,1} \vee x_{s,2}$ are added to C . Moreover, for each symbol s such that there is only one pair in $o_1(s) \times o_2(s)$, add the clause $x_{s,1}$ to C (this corresponds to forcing the variable $x_{s,1}$ to be true). Two lines (or two variables) are called *crossing* if they cross in the drawing built as in Fig. 2.

If there exists a solution S of 2SAT that satisfies all the clauses in C , then S picks exactly one of the lines associated with each symbol. More formally, notice that each variable $x_{s,i}$ is associated with an occurrence of symbol s in sequence s_1 (denoted as $s_1(s, i)$) and an occurrence of symbol s in sequence s_2 (denoted as $s_2(s, i)$). A pair $x_{s,i}, x_{t,j}$ of variables is crossing if in s_1 the symbol $s_1(s, i)$ precedes $s_1(t, j)$ and in s_2 the symbol $s_2(s, i)$ does not precede $s_2(t, j)$ or, symmetrically, if in s_1 the symbol $s_1(s, i)$ does not precede $s_1(t, j)$ and in s_2 the symbol $s_2(s, i)$ precedes $s_2(t, j)$. For each pair $x_{s,i}, x_{t,j}$ of crossing variables, the clause $\neg x_{s,i} \vee \neg x_{t,j}$ is added to C .

Theorem 6: The problem of determining if a feasible solution exists for an instance of 2-ELCS where each mandatory symbol appears in total at most three times in the input sequences, can be solved in polynomial-time.

Proof: We prove that the original instance of 2-ELCS has a feasible solution if and only if the corresponding instance of 2SAT is satisfiable, that is there is a truth assignment for all variables such that all clauses in C are evaluated true. Assume that there is a feasible solution z of the instance of 2-ELCS then, for each symbol s , we pick the lines connecting the symbols retained in z . By definition of common subsequence there cannot be two crossing lines, and exactly one of the lines associated with each symbol must be picked as z in an exemplar common subsequence, thus all the symbols must belong to s . Therefore we have constructed a feasible solution of 2SAT.

Conversely given a truth assignment A for variables that satisfies all clauses in C , it follows that there are no two crossing variables in A . Indeed, for each pair of crossing variables $x_{s,i}, x_{t,j}$ a clause $\neg x_{s,i} \vee \neg x_{t,j}$ is in C and this clause can be true iff at least one of $x_{s,i}, x_{t,j}$ is false. Moreover, the two clauses $\neg x_{s,1} \vee \neg x_{s,2}$ and $x_{s,1} \vee x_{s,2}$ are true if and only if there is exactly one of the variables $x_{s,1}, x_{s,2}$ true in A and one of the variables $x_{s,1}, x_{s,2}$ false in A . Hence there

is exactly one line for each symbol, therefore it is immediate to construct a feasible solution of 2-ELCS that contains all symbols. ■

The overall complexity of the algorithm is quadratic, since we build a clause for each pair $x_{s,i}, x_{t,j}$ of crossing variables.

Notice that the above result holds for all the restrictions of 2-ELCS considered here, as no symbol appears twice in both input sequences, therefore it can appear at most once in any solution.

B. NP-hardness

In what follows we will show that slightly relaxing the constraint on the number of occurrences of each symbol makes the problem **NP**-hard.

Theorem 7: The problem of determining if a feasible solution exists for an instance of 2-ELCS where each mandatory symbol appears at most three times in each input sequence, is **NP**-hard.

Proof: We will prove the theorem reducing 3SAT to 2-ELCS, with a reduction very similar to the one shown before. Let $C = \{C_1, \dots, C_k\}$ be a set of clauses, each one consisting of at most three (possibly negated) literals. We construct an instance of 2-ELCS associating a block with each variable. The block of s_1 associated with variable x_i is defined as the symbol x_i , followed by the sequence of clauses containing x_i , followed by the sequence of clauses containing $\neg x_i$, where in each sequence the clauses are ordered according to the index in $\{C_1, \dots, C_k\}$. In s_2 the block associated with variable x_i is defined as the symbol x_i , followed by the sequence of clauses containing $\neg x_i$, followed by the sequence of clauses containing x_i (again the clauses are ordered according to the index in $\{C_1, \dots, C_k\}$). For example if x_1 appears negated in C_1 and not negated in C_2, C_3 , then the corresponding blocks are $x_1 C_2 C_3 C_1$ (in s_1) and $x_1 C_1 C_2 C_3$ (in s_2). Both sequences s_1 and s_2 consist of the sequence of all blocks associated with the variables of the original instance of 3SAT. All symbols are mandatory, also notice that each symbol appears at most three times in each sequence as each clause contains at most three literals.

Each symbol x_i appears exactly once in each sequence, hence there is no ambiguity on which occurrence is retained in any exemplar common subsequence. Consequently each symbol retained must correspond to occurrences taken from the same block. Inside the block associated with x_i , retaining the clauses where x_i appears as a positive literal is mutually exclusive with retaining the

clauses where x_i appears as a negative literal, by definition of exemplar common subsequence. The first case (that is retaining the clauses where x_i appears as a positive literal) corresponds to setting x_i to true, while the second case corresponds to setting x_i to false. In both cases the clauses retained are satisfied by the assignment of variables x_i .

Any feasible solution of 2-ELCS over sequences s_1 and s_2 must contain all symbols associated with the clauses, therefore we have computed a truth assignment of the variables that satisfies all clauses in C , completing the proof. ■

The above results have a definitive consequence on the approximability of the 2-ELCS problem where each mandatory symbol appears at most three times in both input sequences, as they rule out any polynomial-time approximation algorithm (regardless of the approximation factor).

V. INSTANCES OF MORE THAN 2 SEQUENCES

Since the problem can be extended to instances consisting of a set of sequences, it is interesting to know if the above results can be made stronger. In fact, the well-known inapproximability results in [6] for the LCS problem, immediately apply also to the $\text{ELCS}(\geq 1)$ problem, since $\text{ELCS}(\geq 1)$ is more general than LCS. A closer inspection of their proofs shows that their results also apply to all versions of ELCS, as the optimal solutions in their reductions contain at most one occurrence of each symbol, excluding any $O(n^{1-\epsilon})$ ratio polynomial-time approximation algorithm unless $\mathbf{P}=\mathbf{NP}$, even if no mandatory symbol is allowed and all symbols appear at most twice in each sequence.

VI. INSTANCES CONTAINING NO MANDATORY SYMBOL

Consider the restrictions of problems $2\text{-ELCS}(1, \leq 1)$ and $2\text{-ELCS}(\geq 1, \leq 1)$, where $A_m = \emptyset$. Observe that the two problems are equivalent, since each feasible solution of the two problems consists only of optional symbols and each optional symbol can occur at most once. Denote by $2\text{-ELCS}(*, \leq 1)$ the restriction above. Next we will show that the $2\text{-ELCS}(*, \leq 1)$ is \mathbf{NP} -hard modifying the reduction in Section III-A, replacing all the mandatory symbols by optional symbols.

First, each mandatory symbol x_i can be replaced by a sufficiently long sequence w_j of new optional symbols. Let $|w_j| = 10n$, where n represents the number of vertices of the cubic graph G , that is $n = |V|$. It follows that, for each x_i , either all or no symbols of w_j are included in the

solution. Indeed, if a set of symbols of w_j appears in a solution, it follows that we could add all the remaining symbols of w_i without shortening the resulting exemplar common subsequence. Furthermore, since $|w_i| = 10n$, all sequences w_i must be included in an exemplar common subsequence, otherwise the resulting solution is too short. Notice that each x_i appears exactly once in the reduction.

It remains to replace the mandatory symbols associated with edges, each with a sequence of unique symbols. Replace each edge symbol $s(e_{ij})$ with a sequence $z(e_{ij})$ of new optional symbols, such that $|z(e_{ij})| = n$. Again either all or no edge symbols are included in the solution.

Now if edge e_{ij} is incident to vertices v_i and v_j , $z(e_{ij})$ will appear in blocks i and j of s_1 and s_2 . It follows that one of the two occurrences of $z(e_{ij})$ might be taken. Since all symbols of w_i are taken, either the occurrences of $z(e_{ij})$ in block i of both s_1 and s_2 or the occurrences of $z(e_{ij})$ in block j of both s_1 and s_2 are taken, that is the threading scheme of $z(e_{ij})$ cannot cross the threading scheme of w_i . Observe that at most one occurrence of $z(e_{ij})$ can be taken in a solution of 2-ELCS(*, ≤ 1). Still, at least one symbol of both occurrences of $z(e_{ij})$ must be taken, for otherwise the resulting sequence is too short, and it is always possible to take only the symbols of one of the occurrences of $z(e_{ij})$ without shortening the resulting exemplar common subsequence.

VII. FIXED-PARAMETER ALGORITHMS

In this section, we propose some fixed-parameter algorithms for the resolution of the 2-ELCS(1) and 2-ELCS(≥ 1) problems, where the parameter is the number of mandatory symbols. First we describe a naive approach, then we present two dynamic programming algorithms. In what follows we denote by s_1 and s_2 the two input sequences, by $A_m = \{\alpha_1, \alpha_2, \dots, \alpha_m\}$ the set of mandatory symbols and by n the maximum of $|s_1|$ and $|s_2|$.

A. Naive approach

We present a naive algorithm for 2-ELCS(1) which is based on two phases: the first step consists of guessing the exact ordering of all mandatory symbols in the optimal solution, the second step basically fills in the gaps between each pair of mandatory symbols. Since each mandatory symbol appears exactly once in a feasible solution, the correct ordering of the mandatory symbol is a permutation of A_m , which can be computed in $O(m!)$ time.

Assume that s is an optimal permutation of mandatory symbols, the second phase consists of computing a longest common subsequence s^* of $\{s_1, s_2\}$. Notice that each optional symbol can appear an unrestricted number of times in any solution. Let us denote by $s[i]$ the i -th character of the sequence s and by $s[i \dots j]$ the substring of s starting with $s[i]$ and ending with $s[j]$. The recurrence equation for $\text{EL}[i, j, k]$, that is the length of an optimal solution over $s_1[1 \dots i]$, $s_2[1 \dots j]$, which are both supersequences of the sequence $s[1] \dots s[k]$, is:

$$\text{EL}[i, j, k] = \max \begin{cases} \text{EL}[i-1, j-1, k] + 1 & \text{if } s_1[i] = s_2[j], s_1[i] \in A_o \\ \text{EL}[i-1, j-1, k-1] + 1 & \text{if } s_1[i] = s_2[j] = s[k] \\ \text{EL}[i-1, j, k], \text{EL}[i, j-1, k] & \text{always} \end{cases}$$

The boundary conditions are $\text{EL}[0, j, 0] = 0$ and $\text{EL}[i, 0, 0] = 0$ for $0 \leq i \leq |s_1|$ and $0 \leq j \leq |s_2|$. The value of an optimal solution can be read in $\text{EL}[|s_1|, |s_2|, |s|]$. Once the matrix EL has been completely filled in, the actual optimal subsequence can be constructed with standard backtracking techniques [3]. The recurrence equation described above can be easily modified for the 2-ELCS (≥ 1), by removing the requirement $s_1[i] \in A_o$ in the first condition of the equation.

B. Dynamic programming algorithms

The algorithm described above computes the maximum length of an exemplar common subsequence, by computing all the possible permutations of mandatory symbols. Observe that if the number of mandatory symbols is m then the number of permutations is $m!$ and the above algorithm has time complexity $O(m!n^2)$. Next we present dynamic programming algorithms to compute the maximum length of an exemplar common subsequence of time complexity $O(m2^m n^2)$.

First, we describe a dynamic programming algorithm to compute the existence of a feasible solution of 2-ELCS. Denote by $\text{ES}[i, j]$ where $1 \leq i \leq |s_1|$ and $1 \leq j \leq |s_2|$, a boolean function which is *true* iff there exists a feasible solution of 2-ELCS with input sequences $s_1[1 \dots i]$ and $s_2[1 \dots j]$, otherwise $\text{ES}[i, j]$ is *false*. Let z be a feasible solution of 2-ELCS, we call the *restriction* of z and denote it by z_r , the subsequence of z consisting only of the rightmost occurrence of each mandatory symbol.

Lemma 8: Let z_r be restriction of a feasible solution z of 2-ELCS and let $\alpha \in A_m$ be the rightmost mandatory symbol of z_r . Then there exist two occurrences j_1 and j_2 of α in s_1 and

s_2 respectively, such that $z_r[1 \dots m - 1]$ is a restriction of an exemplar common subsequence of $s_1[1 \dots j_1 - 1]$ and $s_2[1 \dots j_2 - 1]$ with set of mandatory symbols $A_m - \{\alpha\}$.

Proof: In order to obtain a feasible solution we have to guarantee that each mandatory symbol has at least one occurrence. Since α is the rightmost symbol in z_r , it follows that $z_r[1 \dots m - 1]$ must contain all mandatory symbols in $A_m - \{\alpha\}$. Now assume that $z_r[m]$ is taken from two occurrences j_1 and j_2 of α in s_1 and s_2 respectively. It follows that all the mandatory symbols in $z_r[1 \dots m - 1]$, that is in $A_m - \{\alpha\}$, must be taken from $s_1[1 \dots j_1 - 1]$ and $s_2[1 \dots j_2 - 1]$, thus $z_r[1 \dots m - 1]$ is a restriction of an exemplar common subsequence of $s_1[1 \dots j_1 - 1]$ and $s_2[1 \dots j_2 - 1]$ with set of mandatory symbols $A_m - \{\alpha\}$. ■

Observe that there must be a mandatory symbol $\alpha \in A_m$ which is the rightmost mandatory symbol in a feasible solution. Thus function $\text{ES}[n, m]$ is *true* if and only if there exists a feasible solution $\text{ES}[r(o_1(\alpha)) - 1, r(o_2(\alpha)) - 1]$ over the sets of mandatory symbols in $A_m - \{\alpha\}$, where $r(o_1(\alpha))$ (resp. $r(o_2(\alpha))$) represents the rightmost occurrence of α in s_1 (resp. s_2) with $r(o_1(\alpha)), r(o_2(\alpha)) \leq n$.

Denote by $\text{ES}[j_1, j_2, A']$, where $A' \subseteq A_m$ is a subset of the mandatory symbols, a boolean function which has value *true* iff there exists a feasible solution of 2-ELCS with input sequences $s_1[1 \dots j_1], s_2[1 \dots j_2]$ containing all the mandatory symbols in A' , otherwise it has value *false*.

$$\text{ES}[i, j, A'] = \bigvee_{\alpha \in A'} \begin{cases} \text{ES}[i - 1, j - 1, A' - \{\alpha\}] & \text{if } s_1[i] = s_2[j], s_1[i] \in A' \\ \text{ES}[i - 1, j - 1, A] & \text{if } s_1[i] = s_2[j], s_1[i] \notin A' \\ \text{ES}[i, j - 1, A'], \text{ES}[i - 1, j, A'] & \text{always} \end{cases} \quad (1)$$

The boundary conditions are $\text{ES}[i, j, \emptyset] = \text{true}$ for all $0 \leq i \leq |s_1|, 0 \leq j \leq |s_2|$; $\text{ES}[0, j, A'] = \text{false}$ and $\text{ES}[i, 0, A'] = \text{false}$ for $0 \leq i \leq |s_1|$ and $0 \leq j \leq |s_2|$ and for all subsets $A' \subseteq A_m, A' \neq \emptyset$. The existence of a feasible solution of 2-ELCS can be read in $\text{ES}[|s_1|, |s_2|, A_m]$.

The time complexity of the above algorithm is $O(m2^m n^2)$. Indeed, each partial solution is computed by evaluating at most $O(m)$ equations, since we have to choose a mandatory symbol $\alpha \in A', |A'| \leq m$. The number of partial solutions is $O(2^m n^2)$, since the possible subsets $A' \subseteq A_m$ are $O(2^m)$, while indices i, j range over $[1, |s_1|]$ and $[1, |s_2|]$ respectively.

Now we extend the approach to compute a feasible solution, in order to design an algorithm that computes an exemplar longest common subsequence, that is a solution of the optimization problem. Informally, since (1) computes the rightmost occurrence of a mandatory symbol of set

A' in a (possible) feasible solution, we have to add to the solution some symbols between a pair of consecutive mandatory symbols.

First, we discuss the case where the solution must contain exactly one occurrence of each mandatory symbol, while the occurrences of each optional symbol are unrestricted. Denote by $\text{EL}[j_1, j_2, A']$ where $A' \subseteq A_m$ is a subset of the mandatory symbols, a function which represents the length of a longest exemplar common subsequence with input sequences $s_1[1 \cdots j_1], s_2[1 \cdots j_2]$ containing one occurrence of each mandatory symbol in A' . Indeed, occurrences of mandatory symbols in $A' - \{\alpha\}$ occur at the left of i_1 and i_2 since α is the rightmost mandatory symbol by hypothesis, while symbols in $A_m - A' - \{\alpha\}$ have already an occurrence in the exemplar subsequence. The following is the recurrence to compute $\text{EL}[j_1, j_2; A']$.

$$\text{EL}[i, j, A'] = \max_{\alpha \in A'} \begin{cases} \text{EL}[i-1, j-1, A' - \{\alpha\}] & \text{if } s_1[i] = s_2[j] = \alpha, \alpha \in A' \\ \text{EL}[i-1, j-1, A'] & \text{if } s_1[i] = s_2[j], s_1[i] \in A_o \\ \text{EL}[i, j-1, A'], \text{EL}[i-1, j, A'] & \text{always} \end{cases} \quad (2)$$

Denote by $\text{LSO}[j_1, j_2]$ the size of a longest common subsequence with input sequences $s_1[1 \cdots j_1], s_2[1 \cdots j_2]$, where all mandatory symbols in A_m are removed from intervals $[1, j_1]$ and $[1, j_2]$. The boundary conditions are $\text{EL}[i, j, \emptyset] = \text{LSO}[i, j]$ for $0 \leq i \leq |s_1|$ and $0 \leq j \leq |s_2|$, $\text{EL}[0, j, A'] = -\infty$ and $\text{EL}[i, 0, A'] = -\infty$ for $0 \leq i \leq |s_1|$ and $0 \leq j \leq |s_2|$ and for each subset $A' \subseteq A_m, A' \neq \emptyset$. The value of the optimal solution can be read in $\text{EL}[|s_1|, |s_2|, A_m]$.

The time complexity of the algorithm is $O(m2^m n^2)$. Indeed, each partial solution is computed by evaluating at most $4m$ equations. The number of partial solutions is $O(2^m n^2)$, since the possible subsets $A' \subseteq A_m$ are $O(2^m)$, while indices i, j range over $[1, |s_1|]$ and $[1, |s_2|]$ respectively.

Next we consider the case of 2-ELCS when a solution contains at least one occurrence of each mandatory symbol, while the occurrences of each optional symbol are unrestricted. Once again, we assume α is the rightmost mandatory symbol of a longest exemplar common subsequence of length $\text{EL}[j_1, j_2, A']$. With respect to (2), observe that we can add to a solution also mandatory symbols that are not in A' , since each mandatory symbol can appear more than once in a solution.

$$\text{EL}[i, j, A'] = \max_{\alpha \in A'} \begin{cases} \text{EL}[i-1, j-1, A' - \{\alpha\}] & \text{if } s_1[i] = s_2[j] = \alpha, \alpha \in A' \\ \text{EL}[i-1, j-1, A'] & \text{if } s_1[i] = s_2[j], s_1[i] \in A_o \cup A_m - A' \\ \text{EL}[i, j-1, A'], \text{EL}[i-1, j, A'] & \text{always} \end{cases} \quad (3)$$

Denote by $\text{LSM}[j_1, j_2]$ the size of a longest common subsequence with input sequences $s_1[1 \cdots j_1], s_2[1 \cdots j_2]$. The boundary conditions are $\text{EL}[i, j, \emptyset] = \text{LSM}[i, j]$ for $0 \leq i \leq |s_1|$ and $0 \leq j \leq |s_2|$; $\text{EL}[0, j, A'] = -\infty$ and $\text{EL}[i, 0, A'] = -\infty$ for $0 \leq i \leq |s_1|$ and $0 \leq j \leq |s_2|$ and for each subset $A' \subseteq A_m, A' \neq \emptyset$. The value of the optimal solution can be read in $\text{EL}[|s_1|, |s_2|, A_m]$.

The time complexity of the algorithm is $O(m2^m n^2)$. Indeed, each partial solution is computed by evaluating at most $4m4$ equations. As before, the number of partial solutions is $O(2^m n^2)$, since the possible subsets $A' \subseteq A_m$ are $O(2^m)$, while indices i, j range over $[1, |s_1|]$ and $[1, |s_2|]$ respectively.

VIII. IMPLEMENTATION

The algorithm described in recurrence (2) has been implemented and tested on randomly generated data. More precisely, we have tested the algorithm with two input sequences of length 200 and with an alphabet of mandatory symbols A_m of size 10. The algorithm produces the output in a few seconds. However, the space complexity of the algorithm, which grows exponentially in the size of A_m , makes the algorithm not applicable when the size of A_m is 20 or more.

We have implemented and tested a different dynamic programming algorithm to deal with the problem. This second algorithm uses a different approach and it preprocesses subsequences of the input sequences consisting only of optional symbols. However, the first approach turns out to be much more efficient both in time and space than the latter one. Both implementations are freely available at <http://www.algo.disco.unimib.it/> and licenced under the GNU General Public Licence.

IX. OPEN PROBLEMS

In this paper we have investigated the computational and approximation complexity of several versions of the Exemplar Longest Common Subsequence problem. Some interesting cases concerning the computational complexity of the Exemplar Longest Common Subsequence problem

still need to be addressed. More precisely, we have shown that the 2-ELCS problem when each mandatory symbol appears in total at most three times in the input sequences admits a polynomial-time algorithm. Such an algorithm determines if a feasible solution exists, but different feasible solutions can lead to exemplar common subsequences of different length. Indeed, the computational complexity of the general problem of computing an exemplar longest common subsequence when each mandatory symbol appears in total at most three times in the input sequences is still not known. Furthermore, we have shown that the 2-ELCS problem is **NP**-hard when each mandatory symbol appears at least three times in both input sequences. Hence we do not know the computational complexity of the 2-ELCS problem when each mandatory symbol appears less than three times in at least one sequence, while it appears in total more than three times in the two input sequences.

We have proposed some fixed-parameter algorithms to compute an Exemplar Longest Common Subsequence. Observe that both the time and space complexity of these algorithms are exponential on the size of the set of mandatory symbols A_m . In particular, the space complexity makes the algorithm not applicable when the size of A_m is 20 or more. Hence an interesting issue concerning the implementation of these algorithms, is the reduction of the space complexity of such algorithms.

REFERENCES

- [1] P. Alimonti and V. Kann. Some APX-completeness results for cubic graphs. *Theoretical Computer Science*, 237(1–2):123–134, 2000.
- [2] B. Aspvall, M. F. Plass, and R. E. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979.
- [3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.
- [4] K. Hakata and H. Imai. The longest common subsequence problem for small alphabet size between many strings. In *Proc. 3rd International Symp. on Algorithms and Computation (ISAAC)*, pages 469–478, 1992.
- [5] W. Hsu and M. Du. New algorithms for the LCS problem. *Journal of Computer and System Sciences*, 19:133–152, 1984.
- [6] T. Jiang and M. Li. On the approximation of shortest common supersequences and longest common subsequences. *SIAM Journal on Computing*, 24(5):1122–1139, 1995.
- [7] D. Maier. The complexity of some problems on subsequences and supersequences. *Journal of the ACM*, 25:322–336, 1978.
- [8] A. Bergeron and J. Stoye. On the similarity of sets of permutations and its applications to genome comparison. In *Proc. 9th International Computing and Combinatorics Conference (COCOON)*, pages 68–79, 2005.
- [9] T. Uno and M. Yagiura. Fast Algorithms to Enumerate All Common Intervals of Two Permutations. *Algorithmica*, 2: 290–309, 2000.

- [10] D. Sankoff and L. Haque. Power Boosts for Cluster Tests. In *Proc. of Comparative Genomics, RECOMB International Workshop, RCG*, pages 121–130, 2005.
- [11] D. Sankoff. Genome rearrangement with gene families. *Bioinformatics*, 11:909-917, 1999.
- [12] G. Blin and C. Chauve and G. Fertin. The breakpoint distance for signed sequences. In *Proc. 1st Int. Conference on Algorithms and Computational Methods for Biochemical and Evolutionary Networks, CompBioNets*, pages 3-16, 2004.
- [13] D. Bryant. The complexity of calculating exemplar distances. In *Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment, and the Evolution of Gene Families* D. Sankoff and J. Nadeau editors, pages 207–212, 2000.
- [14] G. Blin and R. Rizzi. Conserved interval distance computation between non-trivial genomes. In *Proc. 11th Int. Computing and Combinatorics Conference (COCOON)*, pages 22–31, 2005.
- [15] C. Chauve, and G. Fertin, and R. Rizzi and S. Vialette. Genomes containing Duplicates are Hard to compare. In *Proc. International Workshop on Bioinformatics Research and Applications (IWBRA)*, pages 783–790, 2006.
- [16] R. A. Wagner and M. J. Fischer The String-to-String Correction Problem. *Journal of the ACM*, 21(1):168-173, 1974.

Paola Bonizzoni is full professor of Computer Science at the Università di Milano-Bicocca in Milan. She received her Master Degree in Computer Science from the Università di Milano in 1988 and her Ph. D in Computer Science from the Università di Milano-Torino in 1993. She has been associate professor in Computer Science at Università di Milano-Bicocca from 1999 to 2006 and assistant professor in Computer Science at the Università di Milano from 1993 to 1999. She has been Visiting Research Associate at the University of Colorado at Boulder (USA).

Her research interests are mainly in the area of theoretical computer science, and include computational complexity, models in Biomolecular computation, graph theory, algorithms on strings, trees and graphs and computational biology. Recently, she was involved in research topics concerning sequence comparison, tree reconstruction and gene finding algorithms.

On these subjects she has published several papers in international journals, contributed volumes and conference proceedings.

Gianluda Della Vedova has been appointed as an associate professor at the Department of Statistics, Università di Milano-Bicocca in 2005. He has been an Assistant professor He holds a Ph.D. and a M.Sc. in Computer Science (Università di Milano). His research interests focus on the design of combinatorial algorithms in Bioinformatics and Graph theory. He has published several papers in Bioinformatics and Theoretical Computer Science international journals.

Riccardo Dondi received a M.Sc. in Computer Science from Università di Milano in 1999, and a Ph.D in Computer Science from Università di Milano-Bicocca in 2005. He is currently an assistant professor at the Università di Bergamo. His research interests are in the areas of algorithm design and computational complexity of some combinatorial problems in Computational Biology, in particular clustering, haplotyping and reconstruction and comparison of evolutionary trees.

Guillaume Fertin Biography text here.

Raffaella Rizzi Biography text here.

Stéphane Vialette Biography text here.