ARTICLE IN PRESS

Discrete Applied Mathematics ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

Repetition-free longest common subsequence

Said S. Adi [a], Marília D.V. Braga [b], Cristina G. Fernandes [c], Carlos E. Ferreira [c], Fábio Viduani Martinez [a], Marie-France Sagot [b], Marco A. Stefanes [a], Christian Tjandraatmadja [c], Yoshiko Wakabayashi [c,*]

[a] *Universidade Federal do Mato Grosso do Sul, Brazil*

[b] *Université Claude Bernard, Lyon I, France*

[c] *Universidade de São Paulo, Brazil*

ARTICLE INFO

Article history:
Received 29 March 2008
Received in revised form 30 September 2008
Accepted 20 April 2009
Available online xxxx

Keywords:
Longest common subsequence
APX-hardness
Approximation algorithms

ABSTRACT

We study the following problem. Given two sequences x and y over a finite alphabet, find a repetition-free longest common subsequence of x and y. We show several algorithmic results, a computational complexity result, and we describe a preliminary experimental study based on the proposed algorithms. We also show that this problem is APX-hard.

1. Introduction

In the genome rearrangement domain, gene duplication is rarely considered as it usually makes the problem at hand harder. Sankoff [11] proposed the so-called exemplar model, which consists in searching, for each family of duplicated genes, an exemplar representative in each genome. In biological terms, the exemplar gene may correspond to the original copy of the gene, which later originated all other copies. Following the parsimony principle, the choices of exemplars should be made so as to minimize the reversal distance between the two simpler versions of both genomes, composed only by the exemplar genes. An alternative to the exemplar model is the multigene family model, which consists in maximizing the number of paired genes among a family. Again, the gene pairs should be chosen so as to minimize the reversal distance between the genomes. Both exemplar and multigene models were proven to lead to NP-hard problems [4,6].

To compare two sequences, we propose a similarity measure that takes into account the concept of exemplar genes. The measure we propose is the length of a repetition-free *longest common subsequence* (LCS) between the two sequences. The concept behind the exemplar model is captured by the repetition-free requirement in the sense that at most one representative of each family of duplicated genes is taken into account. The length of an LCS is a measure of similarity between sequences, so the length of a repetition-free LCS can be seen as the edit distance between two sequences where only deletions are allowed and, furthermore, for each family with k duplicated genes, at least $k-1$ of them must be deleted.

An *alphabet* is a finite set and we refer to each of its elements as a *symbol*. All sequences considered in this paper are finite and over some alphabet usually implicit, as it may be considered to be the set of all symbols appearing in the involved sequences. For a sequence w, we use $|w|$ to denote its length. The problem we are interested, denoted by RFLCS, consists

* Corresponding address: Universidade de Sao Paulo, Rua do Matao, 1010 Cidade Universitaria, 05508-090 Sao Paulo, Brazil. Tel.: +55 1130916135 ; fax: +55 1130916134.
E-mail address: YWakabayashi@gmail.com (Y. Wakabayashi).

0166-218X/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2009.04.023

of the following: given two sequences x and y, find a repetition-free LCS of x and y. We write RFLCS (x, y) when we refer to RFLCS for a generic instance consisting of a pair (x, y). We denote by opt(RFLCS(x, y)) the length of an optimal solution of RFLCS (x, y).

Bonizzoni et al. [5] considered some variants of the RFLCS. Among others, they considered the case where some symbols are required to appear in the sought LCS, and possibly more than once. They showed that these variants are APX-hard and that, in some cases, it is NP-complete just to decide whether an instance of the variants is feasible. This second complexity result makes these variants less tractable.

We present some algorithmic and some hardness results for the RFLCS. We also report on some computational experiments with the algorithms proposed in this paper. We start by showing, in Section 2, some polynomial cases and three approximation algorithms for RFLCS. We describe c-approximations for the case where each symbol appears at most c times in at least one of the sequences. In Section 3, we prove that RFLCS is APX-hard even when each symbol appears at most twice in both sequences. Section 4 presents an integer linear programming formulation (IP) for RFLCS. Finally, in Section 5, we show some computational results we obtained for RFLCS, considering the three approximation algorithms and the use of an IP solver for the formulation presented in Section 4 for finding optimal solutions of the instances.

An extended abstract of this paper was presented at LAGOS 2007 (IV Latin-American Algorithms, Graphs, and Optimization Symposium) [1].

2. Algorithmic results

We first mention some polynomially solvable cases of RFLCS (x, y). If each symbol appears at most once either in x or in y then the problem is easy: it is enough to find an LCS of x and y. In this case, any LCS has no repetition and is therefore a solution of RFLCS (x, y). There are polynomial algorithms for LCS, so this case is polynomially solvable (see [7]).

For each symbol a and a sequence w, let $n(w, a)$ be the number of appearances of a in w. Let $m_a(x, y) = \min\{n(x, a), n(y, a)\}$. The case above is the one in which $m_a(x, y) \leq 1$ for all a. Consider the slightly more general case in which there is a constant bound k on the number of symbols a for which $m_a(x, y) > 1$. This case is also polynomially solvable. Indeed, let A_x be the set of symbols for which $m_a(x, y) = n(x, a)$, and A_y be the remaining symbols. Try each subsequence x' of x and each subsequence y' of y obtained in the following way. For each symbol a in A_x and each of the $m_a(x, y)$ occurrences of a in x, keep that occurrence and delete all the others from x, obtaining one x'. Do the same for y, obtaining one y'. For each x' and y', find an LCS of x' and y'. Return a longest one among all obtained LCS s. This method needs to solve $O(n^k)$ different LCS instances and therefore is polynomial.

Now we describe three simple approximation algorithms for the problem: A1, A2, and A3. Algorithm A1 consists of the following: given x and y, compute an LCS of x and y and remove all repeated symbols but one, in the obtained LCS. Return the resulting sequence. Let m be the maximum value of $m_a(x, y)$ taken over all a. It is not hard to see that Algorithm A1 is an m-approximation for RFLCS (x, y).

Algorithm A2 is probabilistic. It consists of the following: given x and y, for each symbol a, if $m_a(x, y) = n(x, a)$, pick uniformly at random one of the $m_a(x, y)$ occurrences of a in x, and delete all the others from x; if $m_a(x, y) \neq n(x, a)$, pick uniformly at random one of the $m_a(x, y)$ occurrences of a in y, and delete all the others from y. Let x' and y' be the resulting sequences after this clean-up. Compute an LCS w' of x' and y' and return w'.

Algorithm A3 is a variant of Algorithm A2 that uses less random bits. It works basically as Algorithm A2 in the sense that, for each repeated symbol a, the sequence (either x or y) that is chosen to keep an occurrence of a is the one with the least number of repetitions of a. The difference now is that A3 picks uniformly at random only one number, say r, in the interval $[0, 1]$ and uses r to select which occurrence of each symbol will remain. For each repeated symbol, the selection is done using the same number r. (If a symbol a occurs k times in the chosen sequence, we partition the interval $[0, 1]$ into k subintervals of the same size, each one corresponding to an occurrence of a, and then we keep the occurrence of a that corresponds to the subinterval that contains r.)

2.1. Analysis of the probabilistic algorithms

Algorithms A2 and A3 are obviously polynomial, so we concentrate on their approximation ratio.

Let A denote an arbitrary probabilistic algorithm for RFLCS (x, y) and let $A(x, y)$ represent the (probabilistic) output of A when given x and y as input. For a positive number α, we say A is an α-approximation for RFLCS (x, y) if $\mathbf{E}[|A(x, y)|] \geq \text{opt}(\text{RFLCS}(x, y))/\alpha$, for all x and y. The number α may depend on x and y.

Theorem 1. *Algorithm* A2 *is an m-approximation for* RFLCS (x, y)*, where m is the maximum of $m_a(x, y)$, over all symbols a.*

Proof. For a subsequence z' of a sequence z, denote by $I(z', z)$ a set of indices of z that corresponds to an occurrence of z' in z (an arbitrary one, if there is more than one). For instance, if $z' = abc$ and $z = cacbbac$, the set $\{2, 4, 7\}$ corresponds to an occurrence of z' in z. On the other hand, the set $\{2, 3, 5\}$ does not correspond to an occurrence of z' in z.

Fix x, y, and a repetition-free LCS w of x and y. Recall that x' and y' are, respectively, the sequences x and y after the random clean-up in Algorithm A2, and $w' = \text{A2}(x, y)$. Next we define a random variable Z that is the length of a common subsequence of x' and y' and therefore is a lower bound on $|w'|$.

Please cite this article in press as: S.S. Adi, et al., Repetition-free longest common subsequence, Discrete Applied Mathematics (2009), doi:10.1016/j.dam.2009.04.023

For each symbol a in w, let Z_a be a binary random variable that is 1 if and only if (i) $m_a(x, y) = n(x, a)$ and the index in $I(w, x)$ that corresponds to a is in $I(x', x)$ (that is, the random choice for a was from x and corresponds to the occurrence of a in w); or (ii) $m_a(x, y) \neq n(x, a)$ and the index in $I(w, y)$ that corresponds to a is in $I(y', y)$ (that is, the random choice for a was from y and corresponds to the occurrence of a in w). Let $Z = \sum Z_a$, where the sum is taken over every symbol a in w.

Note that the set of symbols a in w such that $Z_a = 1$ corresponds to a subsequence of both x' and y'. This implies that $Z \leq |w'|$ and therefore that $\mathbf{E}[|w'|] \geq \mathbf{E}[Z]$.

By linearity of expectation, $\mathbf{E}[Z] = \sum \mathbf{E}[Z_a]$. As Z_a is a binary random variable, $\mathbf{E}[Z_a] = \Pr[Z_a = 1]$. The random choice for a is always in a sequence that has $m_a(x, y) \leq m$ appearances of a and each choice is made uniformly at random, so $\Pr[Z_a = 1] \geq 1/m$. Hence,

$$\mathbf{E}[|w'|] \geq \mathbf{E}[Z] = \sum \mathbf{E}[Z_a] \geq |w|/m = \mathrm{opt}(\textsc{rflcs}(x, y))/m,$$

and A2 is an m-approximation for RFLCS (x, y). ■

Note that this proof does not depend on the choices for different symbols being independent from each other. It depends only on each of the choices being uniformly at random. For this reason, the same proof implies the following.

Theorem 2. *Algorithm* A3 *is an m-approximation for* RFLCS (x, y)*, where m is the maximum of* $m_a(x, y)$*, over all symbols a.*

2.2. Derandomization

It is not hard to see that Algorithm A3 can be derandomized. Indeed, denote by r the randomly chosen number in Algorithm A3. For each value of r, Algorithm A3 has a behavior, possibly different. There is however only a polynomial number of different behaviors that Algorithm A3 can have. Besides, from x and y, one can generate a polynomial number of values of r that, if given as input to Algorithm A3, make it behave in all possible different ways. The derandomization of Algorithm A3 consists of trying these polynomially many values of r and choosing the best of the outputs produced by Algorithm A3.

As for Algorithm A2, one can think of applying the method of conditional expectations to derandomize it (see for example [2,9]). Unfortunately, we did not succeed in doing that, because we do not know how to compute certain conditional expectations exactly. Each of these conditional expectations in question is basically the expected value of the output of Algorithm A2 for some input. In the previous subsection, we only computed a rough upper bound for this value.

3. Hardness result

We show that RFLCS is APX-hard. This is done by presenting an L-reduction [10] to RFLCS from a particular version of MAX 2-SAT, known to be APX-complete. Our result implies Theorems 1 and 2 of Bonizzoni et al. [5], as there are no "mandatory" symbols.

Let V be a set of Boolean variables. Denote by $\overline{v}$ the negation of a variable v. A *literal* (over V) is an element of $V \cup \{\overline{v} : v \in V\}$. A *clause* is a set of literals, and it is a *k-clause* if it has k literals. An *assignment* for V is a function $h : V \to \{\mathbf{T}, \mathbf{F}\}$. A literal ℓ is $\mathbf{T}$ according to h if, for some v in V, either $\ell = v$ and $h(v) = \mathbf{T}$, or $\ell = \overline{v}$ and $h(v) = \mathbf{F}$. A clause is *satisfied* by an assignment h if at least one of its literals is $\mathbf{T}$ according to h.

The problem MAX 2,3-SAT(V, C) consists of, given a set C of 2-clauses over V, where each literal appears in at most 3 clauses in C, finding an assignment for V that maximizes the number of satisfied clauses in C. This variant of MAX 2-SAT is APX-complete [3,10]. We assume that, for any v in V, no clause is of the form $\{v, \overline{v}\}$. For an assignment h, denote by val(MAX 2,3-SAT$(V, C), h)$ the number of clauses in C that are satisfied by h. Let opt(MAX 2,3-SAT$(V, C)) = \max\{\mathrm{val}(\textsc{max 2,3-sat}(V, C), h) : h \text{ is an assignment for } V\}$.

An L-*reduction* from MAX 2,3-SAT to RFLCS consists of a pair of polynomial-time computable functions (f, g) such that, for two fixed positive constants α and β, the following two conditions hold:

(C1) for every instance (V, C) of MAX 2,3-SAT, $f(V, C) = (x, y)$ is an instance of RFLCS, and

$$\mathrm{opt}(\textsc{rflcs}(x, y)) \leq \alpha\, \mathrm{opt}(\textsc{max 2,3-sat}(V, C));$$

(C2) for every instance (V, C) of MAX 2,3-SAT, and every repetition-free subsequence w of x and y, where $(x, y) = f(V, C)$, we have that $h = g(V, C, w)$ is an assignment for V, and

$$\mathrm{opt}(\textsc{max 2,3-sat}(V, C)) - \mathrm{val}(\textsc{max 2,3-sat}(V, C), h) \leq \beta\,(\mathrm{opt}(\textsc{rflcs}(x, y)) - |w|).$$

Theorem 3. *The problem* RFLCS *is* APX*-complete even when restricted to instances* (x, y) *in which the number of occurrences of every symbol in both x and y is bounded by two.*

Please cite this article in press as: S.S. Adi, et al., Repetition-free longest common subsequence, Discrete Applied Mathematics (2009), doi:10.1016/j.dam.2009.04.023

Proof. First we note that Algorithm A1 presented in Section 2 is a (deterministic) 2-approximation for RFLCS(x, y) when $m_a(x, y) \leq 2$. So the variant of RFLCS(x, y) addressed by this theorem is in APX. Next we show an L-reduction from MAX 2,3-SAT to RFLCS.

For an instance (V, C) of MAX 2,3-SAT, where $V = \{v_1, v_2, \ldots, v_n\}$ and C is a set of 2-clauses over V, we describe an instance $(x, y) = f(V, C)$ of RFLCS. Let $\{c_1, c_2, \ldots, c_m\}$ be a set of distinct labels, one for each of the clauses in C. For simplicity, we write c_i to refer both to the label c_i and to the clause whose label is c_i. So, in particular, we denote also by C the set of labels $\{c_1, c_2, \ldots, c_m\}$.

For each literal ℓ, we denote by $s(\ell)$ a sequence composed by the (labels of the) clauses in which ℓ is present, taken in an arbitrary order. Thus, for each v in V and an assignment h for V, the sequence $s(v)$ contains the clauses of C that would be satisfied if $h(v) = \mathbf{T}$ and the sequence $s(\overline{v})$ contains the clauses of C that would be satisfied if $h(v) = \mathbf{F}$. Observe that, since we do not have a clause of the form $\{v, \overline{v}\}$, then $s(v)$ and $s(\overline{v})$ have no common symbol. In addition, as each literal ℓ may appear in at most 3 clauses of C, we have that $|s(\ell)| \leq 3$. We also use a new set of symbols $D = \{d_1, d_2, \ldots, d_k\}$, such that $k = 6(n-1)$ and $D \cap C = \emptyset$, and construct the sequences x and y as follows.

$$x = s(v_1)s(\overline{v_1})d_1 \cdots d_6 s(v_2)s(\overline{v_2})d_7 \cdots d_{12}s(v_3)s(\overline{v_3}) \cdots d_k s(v_n)s(\overline{v_n}) \quad \text{and}$$
$$y = s(\overline{v_1})s(v_1)d_1 \cdots d_6 s(\overline{v_2})s(v_2)d_7 \cdots d_{12}s(v_3)s(\overline{v_3}) \cdots d_k s(\overline{v_n})s(v_n).$$

The alphabet adopted is the set $C \cup D$. By definition, the sets C and D are disjoint and each symbol of D occurs once in both x and y. In addition, as each clause c in C has two literals, and, for each literal ℓ, the corresponding sequence $s(\ell)$ appears once in either x or y, it follows that each symbol c occurs twice in x and also twice in y.

For instance, if $V = \{v_1, v_2, v_3\}$ and $C = \{c_1, \ldots, c_9\}$, with $c_1 = \{v_1, v_2\}$, $c_2 = \{\overline{v_1}, v_2\}$, $c_3 = \{\overline{v_1}, \overline{v_2}\}$, $c_4 = \{v_1, v_3\}$, $c_5 = \{v_1, \overline{v_3}\}$, $c_6 = \{\overline{v_1}, \overline{v_3}\}$, $c_7 = \{v_2, v_3\}$, $c_8 = \{\overline{v_2}, v_3\}$ and $c_9 = \{\overline{v_2}, \overline{v_3}\}$, then $D = \{d_1, \ldots, d_{12}\}$ and

$$x = \overbrace{c_1c_4c_5}^{s(v_1)}\overbrace{c_2c_3c_6}^{s(\overline{v_1})}\underline{d_1d_2d_3d_4d_5d_6}\overbrace{c_1c_2c_7}^{s(v_2)}\overbrace{c_3c_8c_9}^{s(\overline{v_2})}\underline{d_7d_8d_9d_{10}d_{11}d_{12}}\overbrace{c_4c_7c_8}^{s(v_3)}\overbrace{c_5c_6c_9}^{s(\overline{v_3})}$$
$$y = \underbrace{c_2c_3c_6}_{s(\overline{v_1})}\underbrace{c_1c_4c_5}_{s(v_1)}\underline{d_1d_2d_3d_4d_5d_6}\underbrace{c_3c_8c_9}_{s(\overline{v_2})}\underbrace{c_1c_2c_7}_{s(v_2)}\underline{d_7d_8d_9d_{10}d_{11}d_{12}}\underbrace{c_5c_6c_9}_{s(\overline{v_3})}\underbrace{c_4c_7c_8}_{s(v_3)}.$$

In this case, the subsequence $c_1c_4c_5d_1d_2d_3d_4d_5d_6c_3c_8c_9d_7d_8d_9d_{10}d_{11}d_{12}c_7$ is an optimal solution for RFLCS(x, y) that corresponds to an optimal solution for MAX 2,3-SAT(V, C), given by the assignment $h(v_1) = \mathbf{T}$, $h(v_2) = \mathbf{F}$, and $h(v_3) = \mathbf{T}$.

Note that the construction can be done in polynomial time. As all clauses have two literals, $n \leq 2m$, where $n = |V|$ and $m = |C|$. Also, each symbol of the adopted alphabet may appear at most once in a repetition-free subsequence of x and y, thus opt(RFLCS(x, y)) $\leq m + 6(n-1) \leq 12m$. On the other hand, we can easily set an assignment h for V such that val(MAX 2,3-SAT$(V, C), h) \geq m/2$. Indeed, sequentially, for $i = 1, 2, \ldots, n$, define $C_i \subseteq C$ as $C_i = \{c \in C : c$ contains either v_i or $\overline{v_i}\}$ and $C = C \setminus C_i$; then make $h(v_i) = \mathbf{T}$ if v_i is more common than $\overline{v_i}$ in the clauses of C_i, otherwise $h(v) = \mathbf{F}$. Note that the final C is empty and h satisfies at least $|C_i|/2$ clauses from C_i, for each i. Therefore, as $\cup C_i$ is equal to the initial C, the assignment h satisfies at least $m/2$ clauses from the initial C. So opt(MAX 2,3-SAT$(V, C)) \geq m/2$. Putting the two together, we conclude that opt(RFLCS$(x, y)) \leq 24$ opt(MAX 2,3-SAT$(V, C))$, and **(C1)** holds with $\alpha = 24$.

Let (V, C) be an instance of MAX 2,3-SAT and $(x, y) = f(V, C)$. To prove **(C2)**, essentially we show that there is a repetition-free subsequence w of x and y of length at least $p = q + |D|$ if and only if there is an assignment h for V that satisfies at least $q = p - |D|$ clauses of C.

Claim 1. *Let (V, C) be an instance of* MAX 2,3-SAT *and $(x, y) = f(V, C)$. There is a repetition-free subsequence w of x and y of length at least $p = q + |D|$ if and only if there is an assignment h for V that satisfies at least $q = p - |D|$ clauses of C.*

Proof. Let w be a repetition-free subsequence of x and y of length p. First we describe another repetition-free subsequence z of x and y of length at least p that contains all symbols in D. Roughly speaking, to construct z (from w) we substitute all symbol alignments that "cross" regions consisting of consecutive symbols d_i's by a complete alignment of the d_i's in these regions. The number of consecutive d_i's in each region was chosen to assure that with this substitution the resulting sequence is at least as large as the original one. After constructing z, we describe an assignment h that satisfies at least $|z| - |D| \geq p - |D|$ clauses of C.

The procedure to construct z from w, so that z is a sequence that contains all the symbols $d_1, d_2, \ldots, d_k$ (in this order) and is at least as long as w, is the following. Sequentially, for $i = 1, 2, \ldots, n-1$, remove any symbol of w that comes from the alignment of a symbol of $s(v_i)s(\overline{v_i})$ in x (of $s(\overline{v_i})s(v_i)$ in y) and a symbol of $s(\overline{v_{i+1}})s(v_{i+1})d_{6(i+1)-5} \cdots d_k s(\overline{v_n})s(v_n)$ in y (of $s(v_{i+1})s(\overline{v_{i+1}})d_{6(i+1)-5} \cdots d_k s(v_n)s(\overline{v_n})$ in x, respectively), and then add $d_{6i-5}d_{6i-4}d_{6i-3}d_{6i-2}d_{6i-1}d_{6i}$ (which are the symbols from D that are between $s(v_i)$ and $s(v_{i+1})$ in x or y). Call z the resulting subsequence after all these substitutions. At the end, add to z the symbols from D that are not already present in i, and call z the resulting sequence. Observe that $|s(v_i)s(\overline{v_i})| \leq 6$, thus at each step we replace at most 6 symbols by exactly 6 new symbols from D (that do not occur in w). Hence z is also a repetition-free subsequence of x and y, with $|z| \geq |w|$.

Now, let us describe the assignment h. Since z contains all symbols from D, the other portions of z are subsequences of $s(v_i)s(\overline{v_i})$ in x and $s(\overline{v_i})s(v_i)$ in y, for each $i = 1, 2, \ldots, n$. Moreover, because all symbols in $s(v_i)$ differ from those in $s(\overline{v_i})$, the

Table 1
First experiment.

$\|\Sigma\|$	n	A1		A2		A3		Max		Opt
	32	4.0	(10/10)	4.0	(10/10)	4.0	(10/10)	4.0	(10)	4.0
	64	7.8	(8/8)	8.0	(10/10)	7.9	(9/9)	8.0	(10)	8.0
$n/8$	128	15.3	(7/6)	15.7	(9/7)	14.2	(1/0)	15.8	(8)	16.0
	256	25.8	(9/–)	23.1	(1/–)	21.3	(0/–)	25.9	(-)	–
	512	52.1	(10/–)	40.5	(0/–)	36.5	(0/–)	52.1	(-)	–
	32	6.5	(4/4)	7.2	(10/10)	6.9	(7/7)	7.2	(10)	7.2
	64	12.7	(3/0)	13.9	(10/1)	12.9	(5/0)	13.9	(1)	15.3
$n/4$	128	21.7	(8/0)	20.5	(3/0)	19.2	(0/0)	22.0	(0)	26.2
	256	36.2	(10/0)	31.0	(0/0)	28.9	(0/0)	36.2	(0)	43.7
	512	58.2	(10/–)	46.2	(0/–)	43.2	(0/–)	58.2	(-)	–
	32	7.8	(3/3)	8.7	(9/7)	7.8	(2/2)	8.8	(8)	9.0
	64	13.9	(4/0)	14.7	(7/3)	13.3	(1/0)	15.0	(3)	16.1
$3n/8$	128	22.5	(8/0)	21.9	(5/0)	20.6	(1/0)	22.8	(0)	25.1
	256	35.7	(10/0)	31.6	(1/0)	30.3	(0/0)	35.7	(0)	39.6
	512	53.7	(10/0)	44.9	(0/0)	43.3	(0/0)	53.7	(0)	59.0
	32	8.2	(6/4)	8.6	(10/8)	7.9	(3/1)	8.6	(8)	8.8
	64	13.0	(2/1)	13.9	(9/3)	12.7	(1/0)	14.0	(3)	14.7
$n/2$	128	21.3	(7/0)	21.0	(5/1)	19.6	(1/0)	21.8	(1)	23.2
	256	33.5	(10/0)	30.7	(1/0)	29.3	(0/0)	33.5	(0)	35.8
	512	50.3	(10/0)	44.7	(0/0)	42.3	(0/0)	50.3	(0)	54.2
	32	7.6	(6/4)	7.8	(8/6)	7.5	(5/4)	8.1	(8)	8.3
	64	12.8	(6/4)	12.9	(7/3)	12.5	(4/4)	13.2	(6)	13.7
$5n/8$	128	20.4	(8/1)	19.8	(5/1)	19.3	(1/0)	20.6	(2)	21.6
	256	31.5	(9/2)	29.6	(2/0)	27.9	(0/0)	31.6	(2)	32.8
	512	46.2	(9/2)	42.4	(1/0)	41.3	(0/0)	46.4	(2)	48.3
	32	6.7	(1/1)	7.6	(10/9)	7.1	(5/4)	7.6	(9)	7.7
	64	11.9	(4/3)	12.5	(9/7)	11.9	(3/3)	12.6	(8)	12.8
$3n/4$	128	19.4	(8/6)	19.2	(6/3)	18.1	(2/1)	19.7	(7)	20.0
	256	28.4	(9/2)	28.0	(5/2)	26.9	(3/1)	28.7	(3)	29.9
	512	42.5	(10/2)	39.8	(0/0)	39.4	(1/0)	42.5	(2)	43.8
	32	7.2	(8/8)	7.4	(9/9)	7.1	(6/6)	7.5	(10)	7.5
	64	11.6	(6/5)	11.8	(8/7)	11.3	(4/4)	12.0	(9)	12.1
$7n/8$	128	18.4	(8/7)	18.4	(8/7)	17.9	(3/3)	18.6	(9)	18.8
	256	26.8	(9/6)	26.0	(4/1)	25.2	(1/0)	26.9	(6)	27.4
	512	39.2	(10/0)	37.4	(1/0)	36.6	(0/0)	39.2	(0)	40.7

subsequence z does not align simultaneously symbols from both $s(v_i)$ and $s(\overline{v_i})$. So we define the assignment h as follows: $h(v_i) = \mathbf{T}$ if z aligns a symbol from $s(v_i)$, otherwise $h(v_i) = \mathbf{F}$. Set $g(V, C, w) = h$. Observe that the assignment h satisfies at least $q = |z| - |D| \geq p - |D|$ clauses.

For the other direction, consider an assignment h for V that satisfies q clauses of C. Let w be a repetition-free subsequence of x and y obtained as follows. For $i = 1, 2, \ldots, n$, add to w the symbols that correspond to the clauses in $s(v_i)$ if $h(v_i) = \mathbf{T}$, otherwise add the symbols that correspond to the clauses in $s(\overline{v_i})$. After all the additions, eliminate repetitions and add to w, at the correct positions, all symbols from D. Then w is a repetition-free subsequence of x and y such that $|w| = q + |D|$.

From this, one can deduce that **(C2)** holds with $\beta = 1$, completing the proof of Theorem 3. ■

4. An IP based exact algorithm for the problem

We show in this section an IP formulation for RFLCS (x, y). For that, we need first to establish some notation. For each symbol a, let $E_a = \{(i, j) : x_i = y_j = a\}$. Moreover, set $E = \bigcup_a E_a$. The set E_a represents all possible alignments of the symbol a in x and y. Given (i, j) and (k, l) in E, we say that (i, j) and (k, l) *cross* if $i < k$ and $j > l$. We introduce, for each (i, j) in E, a

Please cite this article in press as: S.S. Adi, et al., Repetition-free longest common subsequence, Discrete Applied Mathematics (2009), doi:10.1016/j.dam.2009.04.023

Table 2
Second experiment.

$\|\Sigma\|$	# Repts	A1		A2		A3		Max		Opt
	3	3.3	(7/7)	3.6	(10/10)	3.6	(10/10)	3.6	(10)	3.6
	4	3.2	(5/5)	3.7	(10/10)	3.7	(10/10)	3.7	(10)	3.7
	5	3.5	(7/7)	3.9	(10/10)	3.8	(9/9)	3.9	(10)	3.9
4	6	3.5	(6/6)	3.9	(10/10)	3.9	(10/10)	3.9	(10)	3.9
	7	3.5	(6/6)	3.9	(10/10)	3.8	(9/9)	3.9	(10)	3.9
	8	3.7	(8/8)	3.9	(10/10)	3.9	(10/10)	3.9	(10)	3.9
	3	5.7	(6/6)	6.1	(10/10)	5.9	(8/8)	6.1	(10)	6.1
	4	6.5	(8/6)	6.6	(9/7)	6.5	(8/6)	6.7	(8)	6.9
	5	6.4	(6/6)	7.0	(10/10)	6.6	(6/6)	7.0	(10)	7.0
8	6	6.6	(4/3)	7.3	(9/7)	6.8	(5/3)	7.4	(8)	7.6
	7	6.8	(3/3)	7.5	(9/8)	7.3	(7/6)	7.6	(9)	7.7
	8	7.3	(6/5)	7.8	(10/9)	7.6	(8/7)	7.8	(9)	7.9
	3	9.6	(5/4)	10.2	(10/7)	9.2	(3/2)	10.2	(7)	10.5
	4	9.8	(5/1)	10.7	(9/2)	10.3	(5/1)	10.8	(2)	11.8
	5	10.8	(5/0)	11.6	(9/1)	11.1	(6/1)	11.7	(2)	12.7
16	6	11.9	(4/1)	12.7	(8/1)	12.0	(3/0)	12.9	(2)	14.2
	7	12.1	(5/1)	12.4	(7/1)	12.2	(6/2)	12.8	(2)	13.9
	8	12.2	(3/0)	13.4	(9/1)	12.3	(2/0)	13.5	(1)	14.9
	3	14.8	(8/3)	15.2	(10/5)	13.9	(1/0)	15.2	(5)	15.8
	4	18.1	(6/1)	17.7	(3/0)	16.7	(2/0)	18.7	(1)	20.3
	5	18.3	(6/0)	18.2	(6/0)	17.1	(2/0)	19.0	(0)	22.0
32	6	19.6	(6/0)	19.3	(5/0)	18.7	(2/0)	20.4	(0)	23.9
	7	22.1	(8/0)	20.8	(4/0)	19.6	(1/0)	22.3	(0)	26.8
	8	20.2	(5/0)	21.6	(7/0)	20.3	(1/0)	21.9	(0)	26.2
	3	23.1	(9/2)	22.1	(4/1)	21.5	(0/0)	23.4	(3)	24.4
	4	27.2	(9/1)	25.5	(4/0)	24.2	(2/0)	27.3	(1)	30.5
	5	31.8	(10/0)	27.8	(0/0)	25.9	(0/0)	31.8	(0)	35.0
64	6	31.9	(9/0)	29.4	(2/0)	28.0	(0/0)	32.0	(0)	38.8
	7	34.2	(10/0)	30.7	(1/0)	28.8	(0/0)	34.2	(0)	42.4
	8	39.6	(10/0)	32.7	(0/0)	30.6	(0/0)	39.6	(0)	47.8

binary variable z_{ij} and impose linear restrictions on z_{ij} so that $z_{ij} = 1$ if and only if x_i and y_j are aligned in a repetition-free LCS of x and y. The IP formulation is then as follows.

$$\begin{aligned} &\text{maximize} && \sum_{(i,j)\in E} z_{ij} \\ &\text{subject to} && \sum_{(i,j)\in E_a} z_{ij} \le 1 && \text{for each symbol } a, \\ &&& z_{ij} + z_{kl} \le 1 && \text{for each } (i,j) \text{ and } (k,l) \text{ in } E \text{ that cross}, \\ &&& z_{ij} \in \{0,1\} && \text{for each } (i,j) \text{ in } E. \end{aligned} \tag{1}$$

Indeed, the first constraint assures that the set $\{i : z_{ij} = 1 \text{ for some } j\}$ defines a repetition-free subsequence w_x of x and the set $\{j : z_{ij} = 1 \text{ for some } i\}$ defines a repetition-free subsequence w_y of y. The second constraint assures that the order of appearance of the symbols in w_x and w_y is the same, that is, $w_x = w_y$ and therefore we have a common subsequence. The objective function maximizes the length of such a subsequence.

We used this IP formulation to solve some instances of RFLCS, so that we could evaluate empirically our approximation algorithms. We used GLPK, a general purpose IP solver, and tested instances of size $n = 512$ and different alphabet sizes. For alphabet size 448 ($= 7n/8$) an optimal solution was found in around 12 min (on the average), but for alphabet size 256 ($= n/2$) the solver could not find an optimal solution within an hour. However, with a specific branch-and-cut algorithm (see [8] for more details) these same instances could be solved to optimality: it took (on the average) around 1 min for alphabet size 448 and 10 min for alphabet size 256.

Please cite this article in press as: S.S. Adi, et al., Repetition-free longest common subsequence, Discrete Applied Mathematics (2009), doi:10.1016/j.dam.2009.04.023

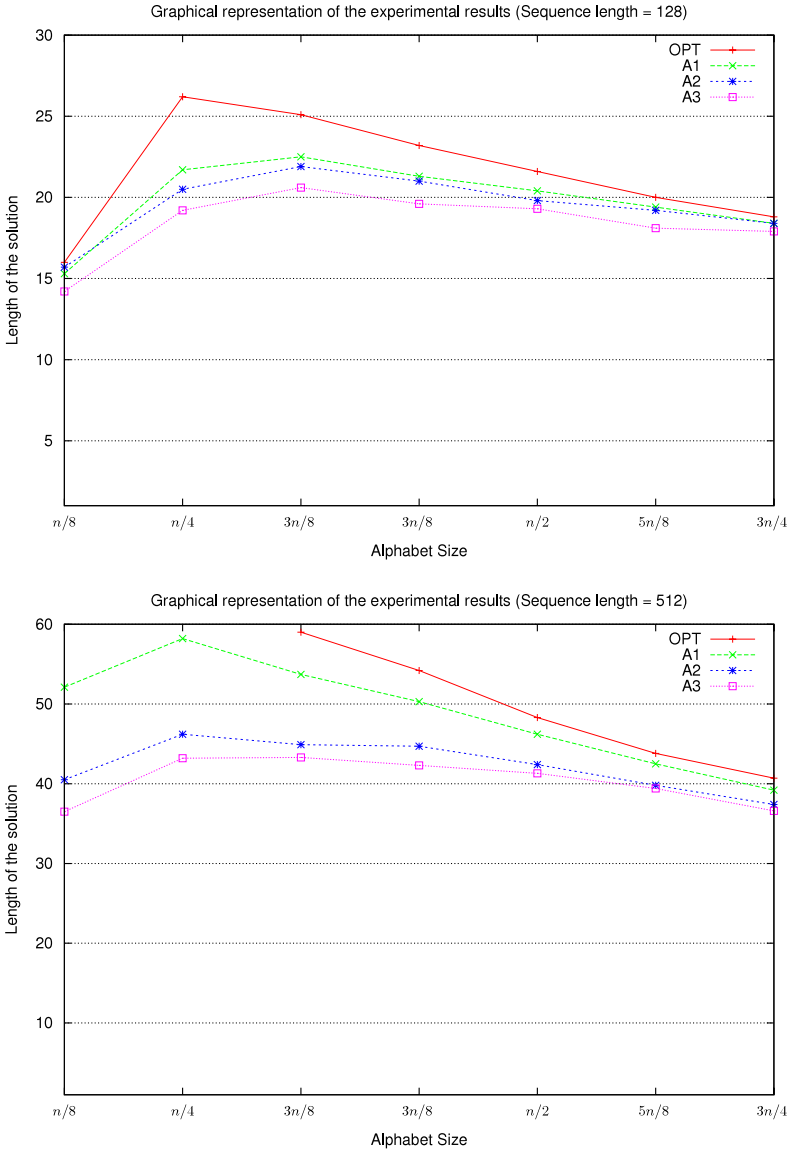

Fig. 1. Performance of algorithms A1, A2, A3 for sequences of length $n = 128$ and $n = 512$, and alphabet size $3n/4$.

5. Computational experiments

We tested the three approximation algorithms on two types of randomly generated instances. In the first type, we considered two parameters: the length of the sequences and the alphabet size as a function of the length. Each position of a randomly generated sequence is one of the symbols of the alphabet chosen uniformly at random. In these sequences, most of the symbols have approximately the same number of occurrences.

In the second type, we considered two parameters: the alphabet size and the maximum number of repetitions of each symbol. For each symbol, we pick, uniformly at random, the number of repetitions of this symbol in the sequence, respecting the given maximum. There is a linear-time (shuffling) procedure that produces, uniformly at random, a sequence with exactly this number of repetitions of each symbol. Note that the expected length of the generated sequence is half of the alphabet size times the maximum number of repetitions.

The experimental results are shown in Tables 1 and 2. In Table 1, each row corresponds to the average results for 10 instances. The two first columns show the alphabet size ($|\Sigma|$) and the sequences length (n). The next three columns show the average solution length of the approximations A1, A2, and A3. For each of these algorithms, a pair (x/y) indicates that x is the number of times (out of 10) that the corresponding algorithm is the best of the three, and y is the number of times it found an optimal solution. The next column shows the average solution length for the algorithm, denoted as Max, that runs A1, A2, and A3 and outputs the best solution found. In parenthesis, we show the number of times (out of 10) Max found

Please cite this article in press as: S.S. Adi, et al., Repetition-free longest common subsequence, Discrete Applied Mathematics (2009), doi:10.1016/j.dam.2009.04.023

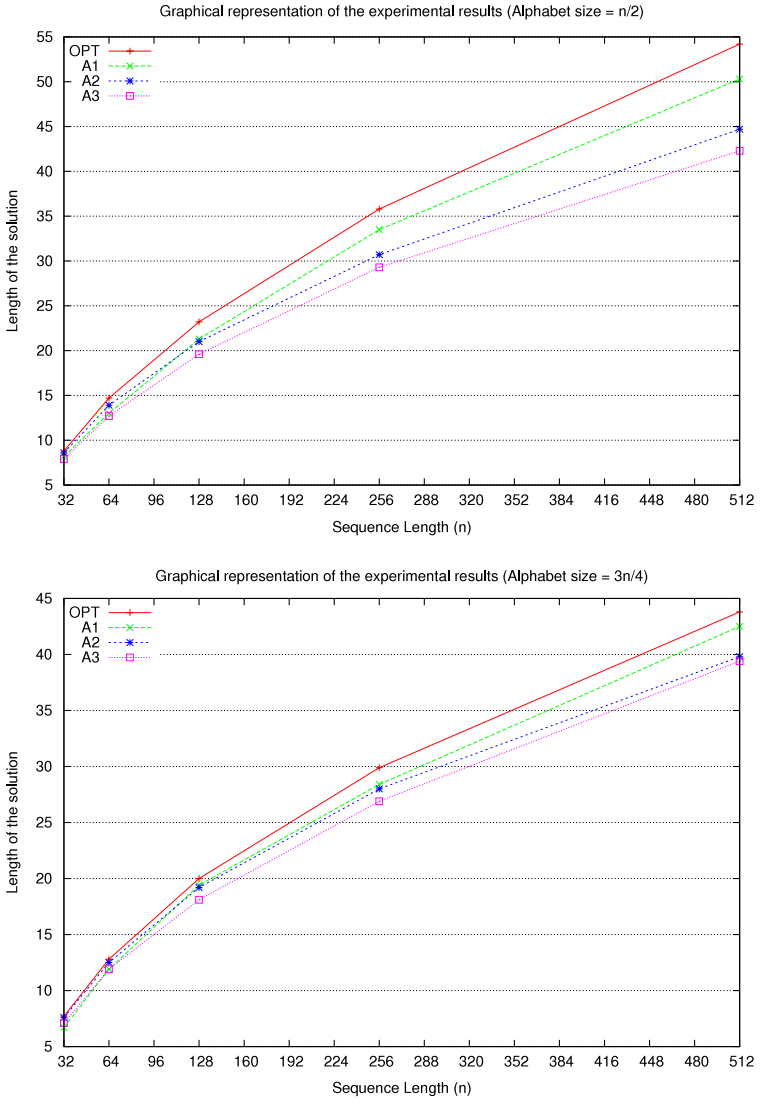

Fig. 2. Performance of algorithms A1, A2, A3 for sequences of length $n = 512$, and alphabet sizes $n/2$ and $3n/4$.

an optimal solution. The last column shows the average length of the optimal value over the 10 instances, obtained by our branch-and-cut code. When the time required to find an optimal solution exceeded two hours, we interrupted the execution (therefore in these cases we do not have the optimal value).

In Table 2, each row corresponds to the average results for 10 instances. The two first columns show the alphabet size and the maximum number of repetitions. The next columns are just like in Table 1. In both tables, the best results in each of the rows are indicated in a shaded box.

It is interesting to note that Max finds optimal solutions more often than A1, which means that A2 and A3 complement sometimes the behavior of A1. In terms of approximation, the ratio between the (average) optimal length and the (average) length of the solution produced by Max was always no more than 5/4 (for the instances where we had the optimal value).

We observe that instances with alphabet size between $n/4$ and $3n/8$ seem to become harder earlier (for shorter instances) in the sense that the approximation algorithms do not find an optimal solution so often (see Table 1). Indeed, except for these cases, in all other cases, the ratio above was no more than 11/10. Similar comments hold for the second type instances. For those, the ratio above is also always at most 5/4.

Some of the results shown in Table 1 are summarized in the graphics exhibited in Figs. 1 and 2. A first observation is that the performance of the approximation algorithms improves when the number of repetitions decreases (that is, when the alphabet size increases relatively to n). This can be seen in the graphics shown in Fig. 1. Another observation, now seen from the graphics shown in Fig. 2, is that, for a fixed (expected) number of repetitions of each symbol, the performance of the algorithms decays when the length of the sequences increases.

Please cite this article in press as: S.S. Adi, et al., Repetition-free longest common subsequence, Discrete Applied Mathematics (2009), doi:10.1016/j.dam.2009.04.023

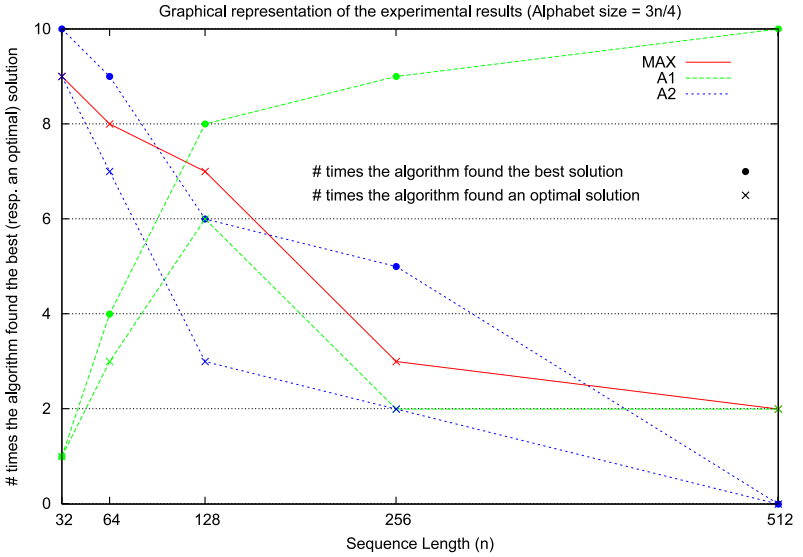

Fig. 3. Number of times the algorithms found the best (resp. an optimal) solution. Here 'best' means not worse than the solution found by the other algorithm.

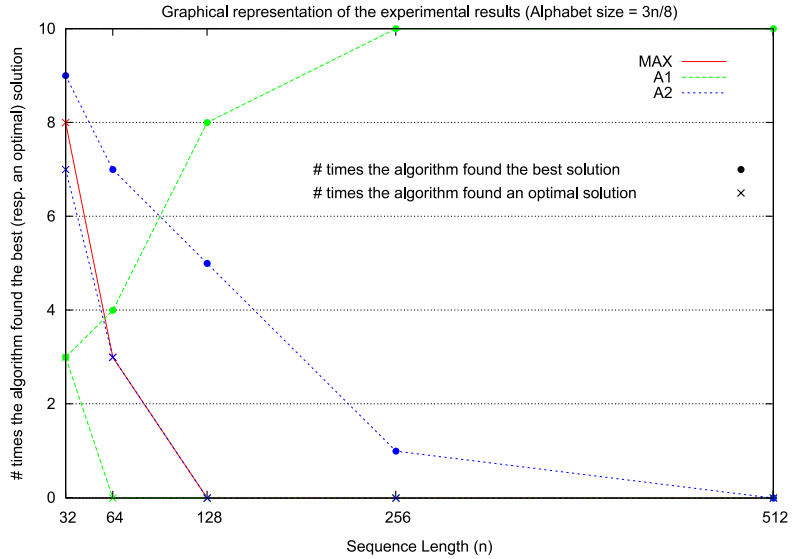

Fig. 4. Number of times the algorithms found the best (resp. an optimal) solution. Here 'best' means not worse than the solution found by the other algorithm.

We note that Algorithm A3 produces the worst results. Also, Algorithm A2 outperforms Algorithm A1 for small length (under 128) sequences. For larger sequences, in both experiments, Algorithm A1 is the best. This is more evident in the graphics shown in Figs. 1 and 2. We also point out the different behaviors of the algorithms when n grows. In Fig. 2, we can see that the solutions found by Algorithm A1 do not diverge much from the optimum, when n grows, while Algorithms A2 and A3 found solutions that diverge more and more from the optimum.

It is interesting to note in Figs. 3 and 4 the performance of Algorithms A1, A2 for alphabet sizes $3n/4$ and $3n/8$. For sequences of length $n < 128$, Algorithm A2 finds better solutions than Algorithm A1. But for $n \geq 128$, Algorithm A1 outperforms Algorithm A2. In these figures, we also note that as n gets larger, MAX finds fewer optimal solutions.

6. Final remarks

Despite of the not so good theoretical worst case ratio, the experimental results indicate that the performance of the approximation algorithms is quite satisfactory for the instances tested. However, it would be nice to test their performance on larger and different types of instances. For them, especially when the sequences have many repetitions (small alphabet)

Please cite this article in press as: S.S. Adi, et al., Repetition-free longest common subsequence, Discrete Applied Mathematics (2009), doi:10.1016/j.dam.2009.04.023

we can obtain the solution of the approximation algorithms very fast, but we are not always able to find the optimal value. We are working on the branch-and-cut algorithm to solve larger instances and hope to confirm the good performance of the approximation algorithms. In any case, it would be interesting to find out whether there is a constant approximation algorithm for RFLCS.

Acknowledgements

The authors thank an anonymous referee for the valuable comments and suggestions. They also thank the financial support received from FAPESP (Proc. 2003/09925-5, 2004/14335-5), CNPq (Proc. 490333/2004-4, 478329/2004-0, 305702/2007-6), Fundect (Proc. 41/100.149/2006), and Alßan (Proc. E05D053131BR).

References

[1] S.S. Adi, M.D.V. Braga, C.G. Fernandes, C.E. Ferreira, F.V. Martinez, M.-F. Sagot, M.A. Stefanes, C. Tjandraatmadja, Y. Wakabayashi, Repetition-free longest common subsequence, in: Proceedings of the IV Latin-American Algorithms, Graphs, and Optimization Symposium, in: Electron. Notes Discrete Math., vol. 30, 2008, pp. 243–248 (electronic).
[2] N. Alon, J. Spencer, The Probabilistic Method, John Wiley, 1992.
[3] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi, Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties, Springer, 1999.
[4] G. Blin, G. Fertin, C. Chauve, The breakpoint distance for signed sequences, in: Proceedings of CompBioNets - Text in Algorithms, vol. 3, 2004, pp. 3–16.
[5] P. Bonizzoni, G. Della Vedova, R. Dondi, G. Fertin, S. Vialette, Exemplar longest common subsequence, in: Proceedings of IWBRA, in: Lecture Notes in Computer Science, vol. 3992, Springer, Berlin, 2006, pp. 622–629.
[6] D. Bryant, The complexity of calculating exemplar distances, in: D. Sankoff, J.H. Nadeau (Eds.), Comparative Genomics, Kluwer, 2001, pp. 207–212.
[7] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, 2nd edition, MIT Press, 2001.
[8] C.G. Fernandes, C.E. Ferreira, C. Tjandraatmadja, Y. Wakabayashi, A polyhedral investigation of the LCS problem and a repetition-free variant, in: LATIN 2008: Theoretical Informatics, in: Lecture Notes in Computer Science, vol. 4957, Springer, Berlin, 2008, pp. 329–338.
[9] R. Motwani, P. Raghavan, Randomized Algorithms, Cambridge University Press, 1995.
[10] C.H. Papadimitriou, M. Yannakakis, Optimization, approximation and complexity classes, Journal of Computer and System Sciences 43 (1991) 425–440.
[11] D. Sankoff, Genome rearrangement with gene families, Bioinformatics 15 (11) (1999) 909–917.

Please cite this article in press as: S.S. Adi, et al., Repetition-free longest common subsequence, Discrete Applied Mathematics (2009), doi:10.1016/j.dam.2009.04.023