

Repetition-free longest common subsequence

Said S. Adi¹ Marília D.V. Braga² Cristina G. Fernandes³
Carlos E. Ferreira³ Fábio Viduani Martinez¹ Marie-France Sagot²
Marco A. Stefanos¹ Christian Tjandraatmadja³ Yoshiko Wakabayashi³

Universidade Federal do Mato Grosso do Sul, Brazil

Université Claude Bernard, Lyon I, France

Universidade de São Paulo, Brazil

Abstract

We study the problem of, given two sequences x and y over a finite alphabet, finding a repetition-free longest common subsequence of x and y . We show several algorithmic results, a complexity result, and we describe a preliminary experimental study based on the proposed algorithms.

Keywords: Longest common subsequence, APX-hardness, approximation algorithms.

1 Introduction

In the genome rearrangement domain, gene duplication is rarely considered as it usually makes the problem at hand harder. Sankoff [6] 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 [2,4].

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

¹ Email: said|fhvm|marco@dct.ufms.br

² Email: marilia@biomserv.univ-lyon1.fr|Marie-France.Sagot@inria.fr

³ Email: cris|cef|christj|yw@ime.usp.br

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 the length of w . The problem we are interested, denoted by RFLCS , consists of the following: given two sequences x and y , find a repetition-free LCS of x and y . We write $\text{RFLCS}(x, y)$ when we refer to RFLCS for a generic instance consisting of a pair (x, y) . We denote by $\text{opt}(\text{RFLCS}(x, y))$ the length of an optimal solution of $\text{RFLCS}(x, y)$.

Bonizzoni et al. [3] considered some variants of the RFLCS . For instance, 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.

2 Algorithmic results

We first mention some polynomially solvable cases of $\text{RFLCS}(x, y)$. 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 $\text{RFLCS}(x, y)$. There are polynomial algorithms for LCS, so this case is polynomially solvable.

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 LCSs. 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 $\text{RFLCS}(x, y)$.

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. Instead of choosing independently one of the occurrences of each symbol in the sequences, A3 picks uniformly at random only one number in the interval $[0, 1]$ and uses it to decide which occurrence of each symbol will remain. The same number is used to select each of the occurrences of all repeated symbols. The rest of the algorithm is the same as in Algorithm A2.

Theorem 2.1 *Algorithms A2 and A3 are m -approximations for $\text{RFLCS}(x, y)$, where m is the maximum of $m_a(x, y)$, over all symbols a .*

Sketch of the proof. Fix x, y , and a repetition-free LCS w of x and y . Sequence w can be thought of as a specific subsequence of x and y . Roughly speaking, each symbol in w has a chance of at least $1/m$ to be picked in the random process of both algorithms. So the expected length of the LCS between x' and y' is at least $1/m$ of $|w|$. \square

3 Hardness result

We show that RFLCS is APX-hard. This is done by presenting an L-reduction [5] 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. [3], as there are no “mandatory” symbols.

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

The problem $\text{MAX 2,3-SAT}(V, C)$ consists of, given a set C of 2-clauses over V , where each literal may appear 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 [1,5]. We assume that, for any v in V , no clause is of the form $\{v, \bar{v}\}$. For an assignment a , denote by $\text{val}(\text{MAX 2,3-SAT}(V, C), a)$ the number of clauses in C that are satisfied by a . Let $\text{opt}(\text{MAX 2,3-SAT}(V, C)) = \max\{\text{val}(\text{MAX 2,3-SAT}(V, C), a) : a \text{ is an assignment for } V\}$.

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 $\text{opt}(\text{RFLCS}(x, y)) \leq \alpha \text{opt}(\text{MAX 2,3-SAT}(V, C))$; **(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 $a = g(V, C, w)$ is an assignment for V , and $\text{opt}(\text{MAX 2,3-SAT}(V, C)) - \text{val}(\text{MAX 2,3-SAT}(V, C), a) \leq \beta (\text{opt}(\text{RFLCS}(x, y)) - |w|)$.

Theorem 3.1 *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.*

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 $V = \{v_1, v_2, \dots, v_n\}$ and C is a set of 2-clauses over V , we describe an instance $(x, y) = f(V, C)$ of RFLCS. Let $\{c_1, c_2, \dots, c_m\}$ 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 $\{c_1, c_2, \dots, c_m\}$.

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 a for V , the sequence $s(v)$ contains the clauses of C that would be satisfied if $a(v) = \mathbf{T}$ and the sequence $s(\bar{v})$ contains the clauses of C that would be satisfied if $a(v) = \mathbf{F}$. Observe that, since we do not have a clause of the form $\{v, \bar{v}\}$, then $s(v)$ and $s(\bar{v})$ 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 $D = \{d_1, d_2, \dots, d_k\}$, such that $k = 6(n - 1)$ and $D \cap C = \emptyset$, and construct the sequences x and y as follows.

$$x = s(v_1)s(\bar{v}_1)d_1 \cdots d_6s(v_2)s(\bar{v}_2)d_7 \cdots d_{12}s(v_3)s(\bar{v}_3) \cdots d_k s(v_n)s(\bar{v}_n) \text{ and}$$

$$y = s(\bar{v}_1)s(v_1)d_1 \cdots d_6s(\bar{v}_2)s(v_2)d_7 \cdots d_{12}s(\bar{v}_3)s(v_3) \cdots d_k s(\bar{v}_n)s(v_n).$$

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 $C = \{c_1, \dots, c_9\}$, with $c_1 = \{v_1, v_2\}$, $c_2 = \{\bar{v}_1, v_2\}$, $c_3 = \{\bar{v}_1, \bar{v}_2\}$, $c_4 = \{v_1, v_3\}$, $c_5 = \{v_1, \bar{v}_3\}$, $c_6 = \{\bar{v}_1, \bar{v}_3\}$, $c_7 = \{v_2, v_3\}$, $c_8 = \{\bar{v}_2, v_3\}$ and $c_9 = \{\bar{v}_2, \bar{v}_3\}$, then $D = \{d_1, \dots, d_{12}\}$ and

$$x = \overbrace{c_1 c_4 c_5}^{s(v_1)} \overbrace{c_2 c_3 c_6}^{s(\bar{v}_1)} d_1 d_2 d_3 d_4 d_5 d_6 \overbrace{c_1 c_2 c_7}^{s(v_2)} \overbrace{c_3 c_8 c_9}^{s(\bar{v}_2)} d_7 d_8 d_9 d_{10} d_{11} d_{12} \overbrace{c_4 c_7 c_8}^{s(v_3)} \overbrace{c_5 c_6 c_9}^{s(\bar{v}_3)}$$

$$y = \overbrace{c_2 c_3 c_6}^{s(\bar{v}_1)} \overbrace{c_1 c_4 c_5}^{s(v_1)} d_1 d_2 d_3 d_4 d_5 d_6 \overbrace{c_3 c_8 c_9}^{s(\bar{v}_2)} \overbrace{c_1 c_2 c_7}^{s(v_2)} d_7 d_8 d_9 d_{10} d_{11} d_{12} \overbrace{c_5 c_6 c_9}^{s(\bar{v}_3)} \overbrace{c_4 c_7 c_8}^{s(v_3)}.$$

In this case, the subsequence $c_1 c_4 c_5 d_1 d_2 d_3 d_4 d_5 d_6 c_3 c_8 c_9 d_7 d_8 d_9 d_{10} d_{11} d_{12} c_7$ is an optimal solution for RFLCS(x, y) and corresponds to an optimal solution for MAX 2,3-SAT(V, C), through the assignment $a(v_1) = \mathbf{T}$, $a(v_2) = \mathbf{F}$, and $a(v_3) = \mathbf{T}$.

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 $\text{opt}(\text{RFLCS}(x, y)) \leq m + 6(n - 1) \leq 12m$. On the other hand, we can easily set an assignment a for V such that $\text{val}(\text{MAX 2,3-SAT}(V, C), a) \geq m/2$ (see the appendix for further details), so $\text{opt}(\text{MAX 2,3-SAT}(V, C)) \geq m/2$. Putting the two together, we conclude that $\text{opt}(\text{RFLCS}(x, y)) \leq 24 \text{opt}(\text{MAX 2,3-SAT}(V, C))$, and **(C1)** holds with $\alpha = 24$.

Let (V, C) be an instance of $\text{MAX } 2,3\text{-SAT}$ 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 a for V that satisfies at least $q = p - |D|$ clauses of C . From this, one can deduce that **(C2)** holds with $\beta = 1$. The complete proof is in the appendix. \square

4 An IP based exact algorithm for the problem

We show in this section an IP formulation for $\text{RFLCS}(x, y)$. 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 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.

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

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. Using a general purpose IP solver, we were not able to solve instances of size over 250. However, with a specific branch-and-cut algorithm that we implemented, we could solve most of the instances to optimality.

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 tables with the experimental results are in the appendix. A first observation is that Algorithm A3 produces the worst results. Also, Algorithm A2 outperforms A1 for small length (under 50) sequences. For larger sequences, in both experiments, Algorithm A1 is the best. We also considered the algorithm that runs A1, A2 and A3 and outputs the best of their solutions. We refer to it as MAX. 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. 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 $5/4$.

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 instance sizes tested. However, it would be nice to test their performance on larger instances. For them, especially when the sequences have many repetitions (small alphabet) 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 would like to thank the financial support of FAPESP (Proc. 2003/09925-5, 2004/14335-5), CNPq (Proc. 490333/2004-4, 478329/2004-0), Fundect (Proc. 41/100.149/2006), and Alβan (Proc. E05D053131BR).

References

- [1] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 1999.
- [2] G. Blin, G. Fertin, and C. Chauve. The breakpoint distance for signed sequences. In *Proceedings of CompBioNets - Text in Algorithms*, volume 3, pages 3–16, 2004.
- [3] P. Bonizzoni, G. Della Vedova, R. Dondi, G. Fertin, and S. Vialette. Exemplar longest common subsequence. In *Proceedings of ICCS*, volume 3992 of *Lecture Notes in Computer Science*, pages 622–629. Springer, 2006.
- [4] D. Bryant. The complexity of calculating exemplar distances. In D. Sankoff and J.H. Nadeau, editors, *Comparative Genomics*, pages 207–212. Kluwer, 2001.
- [5] C.H. Papadimitriou and M. Yannakakis. Optimization, approximation and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.
- [6] D. Sankoff. Genome rearrangement with gene families. *Bioinformatics*, 15(11):909–917, 1999.

Appendix

A Complexity proof

Let (V, C) be an instance of $\text{MAX } 2,3\text{-SAT}$. We can easily set an assignment a for V , such that $\text{val}(\text{MAX } 2,3\text{-SAT}(V, C), a) \geq m/2$. Indeed, sequentially, for $i = 1, 2, \dots, n$, define $C_i \subseteq C$ as $C_i = \{c \in C : c \text{ contains either } v_i \text{ or } \bar{v}_i\}$ and $C = C \setminus C_i$; then make $a(v_i) = \mathbf{T}$ if v_i is more common than \bar{v}_i in the clauses of C_i , otherwise $a(v_i) = \mathbf{F}$. Note that the final C is empty and a 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 a satisfies at least $m/2$ clauses from the initial C . So $\text{opt}(\text{MAX } 2,3\text{-SAT}(V, C)) \geq m/2$.

Claim A.1 *Let (V, C) be an instance of $\text{MAX } 2,3\text{-SAT}$ 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 a 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 . From z , we describe an assignment a that satisfies at least $|z| - |D| \geq p - |D|$ clauses of C .

The following procedure constructs z from w , so that z is a supersequence of $d_1 d_2 \dots d_k$ and is at least as long as w . Sequentially, for $i = 1, 2, \dots, n - 1$, remove any symbol of w that comes from the alignment of a symbol of $s(v_i)s(\bar{v}_i)$ in x (of $s(\bar{v}_i)s(v_i)$ in y) and a symbol of $s(\bar{v}_{i+1})s(v_{i+1})d_{6(i+1)-5} \dots d_k s(\bar{v}_n)s(v_n)$ in y (of $s(v_{i+1})s(\bar{v}_{i+1})d_{6(i+1)-5} \dots d_k s(v_n)s(\bar{v}_n)$ 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 it. Observe that $|s(v_i)s(\bar{v}_i)| \leq 6$, thus at each step we replace at most six symbols by exactly six 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, we proceed to describe the assignment a . Since z contains all symbols from D , the other portions of z are subsequences of $s(v_i)s(\bar{v}_i)$ in x and $s(\bar{v}_i)s(v_i)$ in y , for each $i = 1, 2, \dots, n$. Moreover, because all symbols in $s(v_i)$ differ from those in $s(\bar{v}_i)$, the subsequence z does not align simultaneously symbols from both $s(v_i)$ and $s(\bar{v}_i)$. So we define an assignment a as $a(v_i) = \mathbf{T}$ if z aligns a symbol from $s(v_i)$, otherwise $a(v_i) = \mathbf{F}$. Set $g(V, C, w) = a$. Observe that assignment a satisfies at least $q = |z| - |D| \geq p - |D|$ clauses.

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

B Experimental results

Below we describe the tables with the experimental results. In Figure B.1, each row corresponds to the average results for ten instances. The two first columns show the alphabet size and the sequences length. The next three columns show the average solution length of the approximations A1, A2, and A3. In parenthesis, it is shown for how many of the ten instances this algorithm is the best of the three and how many 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 Max found an optimal solution. The last column shows the average length of the optimal value over the ten instances, obtained by our branch and cut code. When the time required for the algorithm to find the optimal solution exceeded two hours we interrupted the execution (therefore we do not have the optimal value).

$ \Sigma $	n	A1	A2	A3	Max	Opt
$n/8$	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
	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 (-)	—
$n/4$	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
	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 (-)	—
$3n/8$	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
	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
$n/2$	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
	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
$5n/8$	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
	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
$3n/4$	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
	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
$7n/8$	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
	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

Fig. B.1. First experiment.

In Figure B.2, each row corresponds to the average results for ten instances. The two first columns show the alphabet size and the maximum number of repetitions. The next columns are just like in the previous table.

$ \Sigma $	# Repts	A1	A2	A3	Max	Opt
4	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
	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
8	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
	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
16	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
	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
32	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
	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
64	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
	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

Fig. B.2. Second experiment.