ITAT 2013 Proceedings, CEUR Workshop Proceedings Vol. 1003, pp. 36–43
http://ceur-ws.org/Vol-1003, Series ISSN 1613-0073, © 2013 M. Višňovská, T. Vinař, B. Brejová

DNA Sequence Segmentation Based on Local Similarity

Martina Višňovská, Tomáš Vinař, Broňa Brejová

Faculty of Mathematics, Physics, and Informatics,
Comenius University, Mlynská Dolina, 842 48 Bratislava, Slovakia

Abstract: DNA sequences evolve by local changes affecting one or several adjacent symbols, as well as by large-scale rearrangements and duplications. This results in mosaic sequences with various degrees of similarity between regions within a single genome or in genomes of related organisms. Our goal is to segment DNA to regions and to assign such regions to classes so that regions within a single class are similar and there is low or no similarity between regions of different classes. We provide a formal definition of the segmentation problem, prove its NP-hardness, and give a practical heuristic algorithm. We have implemented the algorithm and evaluated it on simulated data. Segments found by our algorithm can be used as markers in a wide range of evolutionary studies.

1 Introduction

In this paper, we study the problem of sequence segmentation arising in computational biology. We are given a string over a finite alphabet, and our task is to identify non-overlapping segments in this string and to partition these segments into classes so that segments within each class are similar to each other, and there are no significant similarities between segments from different classes. We will call the resulting segments atomic segments or simply *atoms*. This task is somewhat similar to clustering, where we also aim to split input data into classes (clusters) so that the similarity between items in the same cluster is higher than the similarity between items from different clusters. The main difference is that in clustering, we are already given a fixed set of items to partition into clusters, whereas in our problem, the atomic segments can be located anywhere along the input string, and thus we need to simultaneously look for atoms themselves, as well as their clustering.

This problem arises in comparative genomics, where we compare genomes (DNA sequences) of related species. These sequences have evolved from a DNA sequence of their common ancestor by a series of mutations, which include local changes affecting one or several adjacent symbols, as well as by large-scale events, such as rearrangements and duplications. This results in mosaic sequences with various degree of similarity between regions within a single genome or in genomes of related organisms.

Small-scale and large-scale evolutionary changes are typically studied separately. Substitutions and short insertions and deletions can be described by various well-studied probabilistic models of evolution (Felsenstein, 2004). Application of these models typically starts with a set of *homologous sequences*, that is, sequences that have evolved from a common ancestor by local changes. These homologous sequences can represent atoms from one class in our segmentation. Studies of large-scale events are more often based on the parsimony principle, where the goal is to find an evolutionary history leading to present-day sequences, which contains the smallest possible number of large-scale evolutionary events, such as rearrangements, duplications, or deletions (Fertin et al., 2009). These studies typically represent the genome as a sequence of markers, and again, each atomic segment in our segmentation can be used as a marker in this context.

One frequently used method for obtaining segments or markers for both types of evolutionary analyses is to use genes, that is, portions of the genome encoding functional protein or RNA molecules. However, these approaches ignore information present in the intergenic regions of the genomes, and are sensitive to errors in gene annotation. In our work, we define atomic segments by considering only information about local sequence similarities between parts of the input sequences. Such similarities can be readily found by one of many sequence alignment programs; in our work we use lastz (Harris, 2007). In our previous work (Brejova et al., 2011), we have informally defined the goals of the segmentation problem and provided a practical heuristic algorithm attempting to cover the whole sequence by atoms. However, we were not able to design a suitable formalization of the problem. Here, we provide a formal definition of the problem as an optimization problem, prove its NP-hardness, and provide a new heuristic algorithm. The main difference from our previous goal is that we do not require that the whole sequence is covered by atoms; some problematic parts may be left out. We have implemented the algorithm and evaluated it on simulated data.

Our algorithm can provide fine-scale segmentations, with atom lengths in hundreds or thousands, in contrast to some earlier approaches concentrating on much longer segments. Multiple authors have for example studied methods for constructing so called synteny blocks from conserved chains of genes in two or multiple genomes (Choi et al., 2007; Hachiya et al., 2009); similar approach was used already by Nadeau and Taylor (1984). Our work is also related to the area of multiple sequence alignment, where the goal is to arrange k sequences into a matrix with k rows by padding them with special gap symbols as necessary so that each column contains homologous symbols.

In the presence of rearrangements and duplications, such linear representation is not possible, and thus multiple alignment programs split the genome into smaller blocks and align symbols in each block separately (Blanchette et al., 2004; Brudno et al., 2003; Ovcharenko et al., 2005). These blocks can be also considered atomic segments in our setting, but most multiple alignment algorithms do not consider duplications and concentrate on recovering correct alignment within individual blocks rather than on optimal division of sequence into blocks.

2 Problem Definition

The main goal of this section is a formal definition of the segmentation problem as a combinatorial optimization problem. The input will be a set of local similarities between pairs of regions in the string, which we wish to segment. In the rest of the paper, we will discuss segmentation of a single string; in case that we have several strings, such as DNA sequences from related species, we will first concatenate them to one string.

True segmentation. We start by defining the *true segmentation* for a string with a known evolutionary history. In this history, we ignore all substitutions, that is operations changing one symbol into another, and consider only events that insert, delete, move or duplicate parts of the string. Each such operation can be viewed as cutting the original string at some places known as breakpoints, adding, discarding, copying or changing the order of the pieces, and finally joining the pieces into one string. The true segmentation will use these breakpoints as atom boundaries, and will group to one class those atoms that have evolved by duplication from a common ancestor. In our simplified abstract view, DNA of multiple species also arose by duplication of the whole sequence from one species to another. However, if a breakpoint is created in a part of the string which has multiple copies elsewhere, we will also propagate this breakpoint to all these copies (see Figure 1). This ensures that if we take two atoms, they are either homologous (evolutionarily related) across their whole length, or not homologous at all.

For real biological sequences, the evolutionary history and the true segmentation are not known. Indeed, our motivation for segmenting the sequence is to use the result as an input for evolutionary history inference. We want to formulate an optimization problem, which will allow us to recover the true segmentation or its approximation based on information about sequence similarity. The resulting algorithms can be tested on sequences generated by a probabilistic model of evolution, where we know the true segmentation. In practice, we do not want to create an atom boundary at the position of each short insertion or deletion, because these events are quite frequent. We will thus only consider breakpoints created by longer events. As we will see, our problem uses a parameter L for minimum atom length related to the granularity at which we consider an event to be large-scale.

X: $ABCDEFG$
Y: $ABCDEB'C'D'FG$
Z: $ABCDEB'D'FC'G$

Figure 1: True segmentation of sequence Z with a known evolutionary history. Ancestral sequence X contains one atom $A \dots G$. In the first step, region BCD was copied to another place, obtaining sequence Y. Sequence Y has true atoms $\{A, BCD, B'C'D', E, FG\}$, with BCD and $B'C'D'$ belonging to the same class. Finally, region C' was moved to a different location, resulting in the final sequence Z, in which every symbol is a separate atom (breakpoints between B' a C' and C' and D' were propagated to the other copy BCD). Atoms denoted by the same letter (with or without prime) belong to the same class.

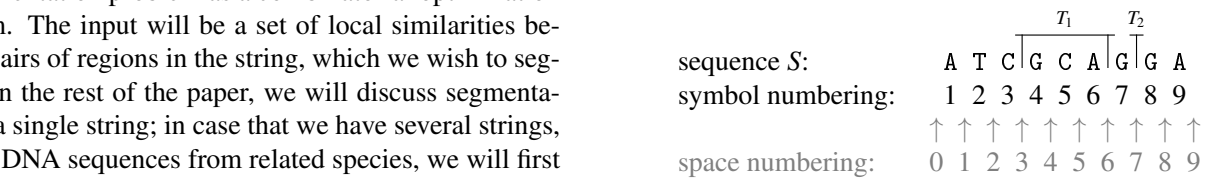

Figure 2: Sequence S and its regions $T_1 = S(3,6)$ and $T_2 = S(7,7)$.

Input data and notation. We now formally describe input data for our computational problem and introduce useful notation. As an input data for segmentation, we have DNA sequence $S = s_1 s_2 \dots s_n$ and a set α containing alignments between pairs of similar regions within S. Ideally, these alignments would reflect true homology, but in reality, some alignments might be spurious, missing, or locally incorrect. We will define an alignment more formally below.

We will frequently refer to positions and regions within S. To do so, we number the spaces between consecutive symbols of S so that the space between s_i and s_{i+1} is labeled i, space to the left of s_1 is labeled zero, and space to the right of s_n is labeled n. By $S(i,j)$ we denote the contiguous region of S starting in the space labeled by i and ending in the space labeled by j, where $0 \le i \le j \le n$. In this notation, we can refer to an empty region at specific position i in S by $S(i,i)$ (see Figure 2). We will consider regions $S(i,j)$ as open intervals. Region $S(i,j)$ then overlaps $S(k,\ell)$, if the intersection of intervals (i,j) and (k,ℓ) is non-empty. We will say that $S(i,j)$ covers $S(k,\ell)$ if interval (k,ℓ) is a subset of interval (i,j).

Recall that an alignment is a way of arranging two (or more) sequences by inserting a special gap symbol at some locations, so that both sequences have the same length (Figure 3). It is typically used to represent significant sequence similarities: the two aligned sequences are assumed to be homologous and symbols in the same column

sequence T_1: A T C - - G C G G A
sequence T_2: A T C A A G T G - A

Figure 3: An alignment of T_1 and T_2.

of the alignment are also hypothesized to share evolutionary origin. To define the segmentation problem, we represent an alignment as a relation between positions of two sequences.

Definition 1. *An* alignment *a with source $S(i,j)$ and destination $S(k,\ell)$ is a partial mapping from $\{i+1,\ldots,j\}$ to $\{k+1,\ldots,\ell\}$, such that $a(i+1) = k+1$, $a(j) = \ell$, and if for some $f < g$ both $a(f)$ and $a(g)$ are defined, then $a(f) < a(g)$.*

Informally, mapping $t = a(f)$ corresponds to the situation when symbols s_f and s_t are aligned in one column. Note that we require that the first and the last symbols of the two aligned regions map to each other (the alignment does not have a gap symbol in the first or the last column).

We will extend the notion of mapping from individual symbols to whole regions. Let a be an alignment between regions T_1 and T_2, and let U be a region such that T_1 covers U. Then, the *mapping of region U* through alignment a is the shortest region within T_2 that contains mapping of every symbol from U, for which such mapping exists. We will denote this regions as $a(U)$.

Note that our definition of an alignment is asymmetric. Typically for every alignment a mapping T_1 to T_2, the input will also contain its *reverse* counterpart a' mapping T_2 to T_1 so that $a(x) = y$ if and only if $a'(y) = x$.

Problem definition. We are now ready to formally define our computational problem.

Definition 2. *A* segmentation *of sequence S with respect to a set of alignments α and a length parameter L is a set of regions $\mathscr{A}$ (termed* atoms*) for which the following conditions hold:*

A1) No two atoms from set $\mathscr{A}$ overlap.

A2) The length of each atom is at least L.

A3) If a source or a destination of an alignment $a \in \alpha$ overlaps some atom A, it also covers A.

A4) If the source of alignment $a \in \alpha$ covers some atom A, then the region $a(A)$ overlaps with exactly one atom from $\mathscr{A}$.

Condition A3 ensures that no atom contains boundaries of any alignment. This is desirable, because alignment boundaries intuitively correspond to breakpoints in the evolutionary history.

If sequence S contains several homologous copies of a region d, we wish to avoid the situation where one copy of d forms an atom by itself and another copy is only a small part of a longer atom. This is enforced by condition A4, which implies that atom A can be aligned to a smaller part within another atom B, or to some region which contains not only atom B, but also some surrounding symbols. The second case is possible if the symbols surrounding B are not covered by any other atom of $\mathscr{A}$. We call such areas *waste regions*.

Definition 3. *Let $A_1,\ldots,A_p$ be the atoms of a segmentation $\mathscr{A}$ ordered by their position in S, where $A_i = S(f_i,t_i)$. The* waste region *between atoms A_i and A_{i+1} is the region $S(t_i,f_{i+1})$. Two additional waste regions $S(0,f_1)$ and $S(t_p,n)$ are located at sequence ends.*

Depending on a situation, we can specify a segmentation either by its atoms, or by the waste regions between the atoms. Alignments, which cover the same region of S usually do not have boundaries at identical positions, because it is difficult to detect true homology boundaries in local alignment search. As the true boundary has an uncertain position, we consider the region containing several alignment boundaries close to each other to be a waste region.

To partition atoms to classes, we will represent alignments in the form of a graph. In the *alignment graph* of segmentation $\mathscr{A}$, each vertex corresponds to an atom of $\mathscr{A}$ and two vertices A and B are connected by an edge if some alignment maps A to a region overlapping B. Then each atomic class of $\mathscr{A}$ corresponds to one connected component of the alignment graph.

To choose the best segmentation, we introduce a cost function. As we prefer to have a high coverage of input sequence S by atoms, the cost function penalizes a segmentation for every symbol located in a waste region. We also prefer a segmentation with a lower number of atoms to avoid unnecessary fragmentation. Therefore we penalize a segmentation for the number of atoms, but this second penalty has a low weight $\varepsilon = 1/n$, where n is the length of S. For simplicity, our cost function does not take atom distribution to classes into account. Thus a segmentation $\mathscr{A}$ with p atoms of total length r has cost $c(\mathscr{A}) = (n-r) + \varepsilon p$.

We can now approach the sequence segmentation problem as an optimization problem of searching for the minimum cost segmentation for a given sequence S, set of alignments α, and length parameter L.

3 NP-Hardness of the Segmentation Problem

In this section, we will prove NP-hardness of the segmentation problem. We will consider a decision version, in which we are given a threshold B, and we ask if there is a segmentation with cost at most B. It is easy to see that this problem is in NP, because we can non-deterministically guess a set of atoms and calculate its cost, as well as verify the requirements of Definition 2 in polynomial time.

We will prove NP-hardness of this problem by a reduction from the classical one-in-three 3SAT problem (Garey and Johnson, 1979). The instance of this problem consists of a set U of variables, and a collection C of clauses, each clause containing three literals. The problem is to determine, if there is a truth assignment for U such that each clause from C contains exactly one true literal. Our goal is to transform such an instance (U,C) to an instance (S,α,L,B) of the segmentation problem.

In our construction, we will for simplicity create many short sequences and alignments between these sequences or their parts. To get one input sequence S, we can concatenate these short sequences and adapt the created alignments so that they refer to the corresponding parts of S. Length threshold L can be an arbitrary constant greater than two.

The basic building block of our proof is a gadget consisting of $k+3$ sequences $S_0,\ldots,S_{k+2}$, all of the same length m. Sequence S_0 will be called the interface of the gadget, because its parts align to sequences outside this set, but sequences $S_1,\ldots S_{k+2}$ align only with other sequences within the gadget. In particular, for every i and j such that $0 \le i < j \le k+2$, we create an alignment $a_{i,j}$. This alignment maps the first and the last symbol of S_i to the first and the last symbol of S_j, respectively. It also maps the r-th symbol of sequence S_i to the $(r+1)$-st symbol of sequence S_j, for $2 \le r \le m-2$. The second symbol of S_i and the $(m-1)$-st symbol of S_j remain unaligned in this alignment. For every alignment $a_{i,j}$, we also create its reverse $a_{j,i}$ mapping S_j to S_i.

The goal of this gadget is to ensure that the interface sequence S_0 will contain at most one atom; otherwise the cost of the segmentation will be higher than B. This follows from the following lemma if we set $k = \lceil B \rceil + 1$ (the value of B will be determined later).

Lemma 1. *If sequence S_0 contains at least two different atoms, then the gadget $S_0,\ldots,S_{k+2}$ contains at least k symbols in waste regions.*

Proof. Let us assume that S_0 contains at least two different atoms, but the overall number of symbols in waste regions is at most $k-1$. This implies that at least three sequences S_i, S_j and S_ℓ for $1 \le i < j < \ell \le k+2$ do not contain any wasted symbols. However, each of these sequences aligns to S_0, and since S_0 contains at least two atoms, each of these sequences needs to be split into at least two atoms due to condition A4 of Definition 2. These atoms need to be separated by a waste region of length 0. Let us assume that $S_i(x,x)$ is such a waste region of length 0 in S_i. Symbols at positions $x+1$ and $x+2$ in S_j align to symbols at positions x and $x+1$ in S_i through $a_{j,i}$, and thus they cannot occur in the same atom in S_j. This means that $S_j(x+1,x+1)$ is a waste region as well. By the same reasoning, $S_\ell(x+1,x+1)$ is a waste region due to alignment $a_{\ell,i}$ and $S_\ell(x+2,x+2)$ is a waste region due to alignment $a_{\ell,j}$ mapping symbols $x+2$ and $x+3$ to two different atoms in S_j. However, this is a contradiction, because waste regions $S_\ell(x+1,x+1)$ and $S_\ell(x+2,x+2)$ create an atom $S_\ell(x+1,x+2)$ of length one, and each atom needs to have a length at least $L > 2$. As $S_\ell(x+1,x+2)$ cannot be an atom, it has to be a waste region, which is a contradiction. Therefore the set $S_1,\ldots,S_{k+2}$ contains at least k sequences containing at least one wasted symbol each, and the total number of wasted symbols in the gadget is at least k. □

Our construction will contain for each variable $y \in U$ two sequences X_y and $X_{\neg y}$, each of length L. It will also contain a gadget $\mathscr{U}_y$ as described above, with each sequence of the gadget having length $2L$. Finally, for each clause $c_i \in C$ we create a gadget $\mathscr{C}_i$ containing sequences of length $3L$. In addition to the alignments within gadgets, we will add gapless alignments between sequences X_y and parts of interfaces of gadgets $\mathscr{U}_y$ and $\mathscr{C}_i$. We say that an alignment is *gapless*, if it aligns regions T_1 and T_2 of the same length and maps the r-th symbol of T_1 to the r-th symbol of T_2. In particular, we will align sequence X_y by a gapless alignment to the first half of the interface sequence of $\mathscr{U}_y$ and we will align sequence $X_{\neg y}$ to the second half of this interface. For a clause $c_i = \ell_1 \vee \ell_2 \vee \ell_3$ we align X_{ℓ_1} to the first third of the interface for $\mathscr{C}_i$ and similarly align X_{ℓ_2} to the second third and X_{ℓ_3} to the last third of the interface. All these gapless alignments are created in both directions. Finally, we set $B = 2L(|U|+|C|)+1/2$.

An example of the construction is shown in Figure 4. Clearly the construction is polynomial in terms of $|U|+|C|$, and its correctness is summarized in the following theorem.

Theorem 1. *One-in-three 3SAT instance (U,C) is satisfiable if and only if the corresponding instance of the segmentation problem has a solution with cost at most B.*

Proof. Let us first assume that instance (U,C) is satisfied by a truth assignment $t : U \to \{\mathrm{T},\mathrm{F}\}$. We will create a segmentation $\mathscr{A}$ with cost $c(\mathscr{A}) \le B$.

Consider sequences X_y and $X_{\neg y}$ representing the positive and negative literal of variable $y \in U$. One of these sequences corresponds to the satisfied literal in the truth assignment t; this sequence will be included as an atom in $\mathscr{A}$. The other sequence will be completely covered by a waste region.

Each sequence in each gadget $\mathscr{U}_y$ and $\mathscr{C}_i$ will contain one atom. Each non-interface sequence will be completely covered by its atom. Interface sequences will contain one atom of length L in the region aligned to sequence X_ℓ for the satisfied literal ℓ and waste region elsewhere. Note that there is always exactly one satisfied literal ℓ for each variable y and for each clause c_i.

It is easy to see that this segmentation satisfies all conditions. The total number of wasted symbols in $\mathscr{A}$ is $2L(|U|+|C|) = B - 1/2$. Since each atom has length at least $L > 2$, the total number of atoms is less than $n/2$, where n is the total length of the concatenated sequence for the segmentation problem. Therefore the contribution

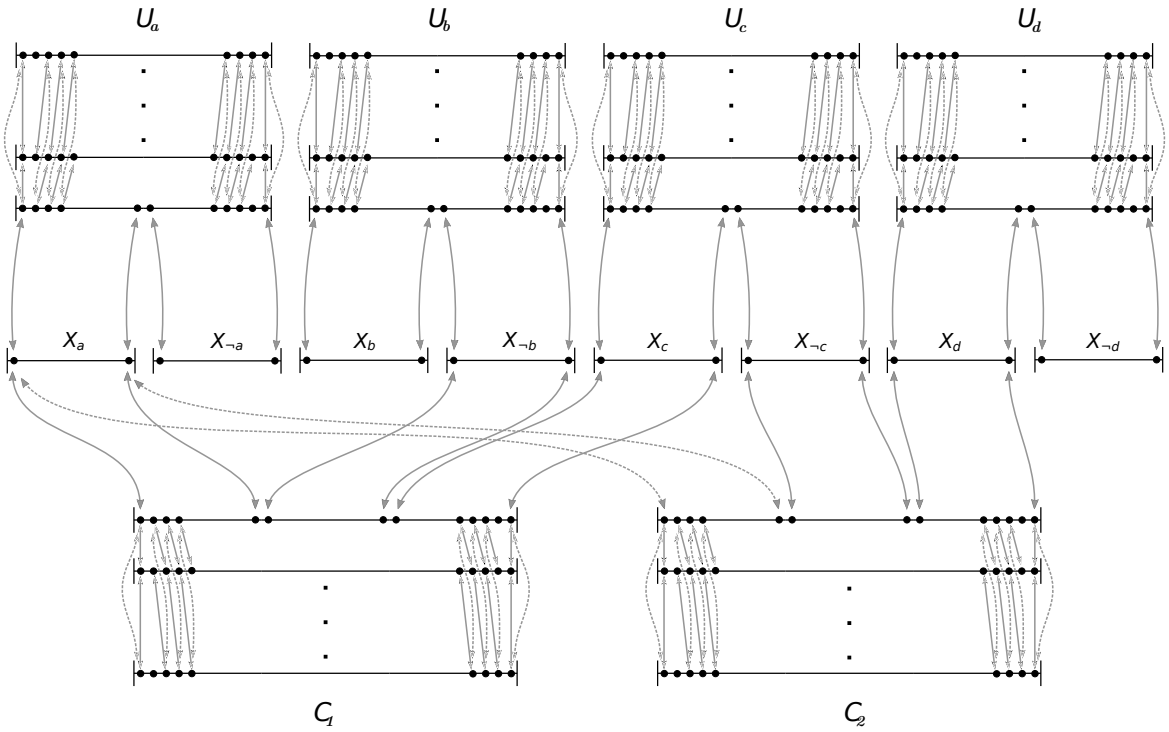

Figure 4: The instance of the segmentation problem for boolean formula $(a \vee \neg b \vee c) \wedge (a \vee \neg c \vee d)$. The instance contains eight sequences of length L, each representing one literal. It also contains four gadgets of length $2L$, each representing one variable, and two gadgets of length $3L$, each representing one clause.

of the atom number to the total cost of $\mathscr{A}$ is less than $\varepsilon \cdot n/2 = 1/2$ and the total cost $c(\mathscr{A})$ is at most B.

Now assume that we have a segmentation $\mathscr{A}$ with cost at most B, and we will prove that then the instance (U,C) of one-in-three 3SAT is satisfiable.

Each interface sequence has alignment endpoint every L symbols, and thus each atom in this sequence can have length of at most L. However, by Lemma 1, each gadget can have at most one atom in its interface. The remaining regions of length L within interface will be therefore waste. If some interface did not contain any atom, then all sequences in the corresponding gadget will have to be covered with waste as well, raising the cost above B. This implies that there will be exactly one atom in each interface.

Let A_y be the atom within interface of $\mathscr{U}_y$. This atom is aligned to sequence X_ℓ, where ℓ is y or $\neg y$. We will assign truth value $t(y)$ so that literal ℓ is satisfied. Since atom A_y aligns to X_ℓ, and X_ℓ has length L, it must be an atom as well. Similarly, $X_{\neg\ell}$ aligns to a waste region in the interface of $\mathscr{U}_y$, and thus $X_{\neg\ell}$ is also covered with waste. Therefore, $t(\ell)$ is true for some literal ℓ if sequence X_ℓ is an atom, and $t(\ell)$ is false if X_ℓ is a waste region.

Interface of $\mathscr{C}_i$ also contains exactly one atom B_i and two waste regions of length L. Each of these three thirds aligns to some X_ℓ. The alignment for the atom must map to another atom, and an alignment for a waste region must map to a waste region. Therefore exactly one of the three literals in clause c_i must have its corresponding sequence X_ℓ covered by an atom, and this means that exactly one literal of the clause is satisfied by t. We have thus proved that the assignment t satisfies the corresponding one-in-three 3SAT instance (U,C). □

4 Practical Algorithm for Sequence Segmentation

Since the segmentation problem is NP-hard, we have designed a heuristic algorithm for practical use. This algorithm creates a segmentation satisfying Definition 2, but possibly not the optimal one.

Our algorithm gets as an input a set of evolutionarily related sequences or a single sequence which contains duplicated segments. We use the LASTZ program (Harris, 2007) to align every sequence to every other and also to itself. We use the same alignment preprocessing as in our earlier work (Brejova et al., 2011) to discard weak alignments or their parts.

Now we create initial waste regions of zero length at both ends of each alignment, as our definition requires that there are no alignment boundaries inside atoms. If any consecutive waste regions are closer to each other than L, they are joined to a single waste region, because no atom can be located between them. The resulting set of *proto-atoms* located between successive waste regions satisfies conditions A1, A2, and A3 of Definition 2. It is possible, though, that condition A4 is not satisfied, if some proto-atom maps to a region overlapping two or

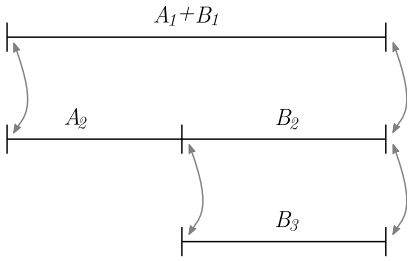

Figure 5: An example of a situation where alignment boundaries do not define a proper segmentation. Region B_2 is aligned to B_3, as a result of which we have an alignment boundary between A_2 and B_2. However, region A_1+B_1 is aligned as a whole to region A_2+B_2, and no alignment boundary separates A_1 from B_1. Proto-atom A_1+B_1 thus does not satisfy condition A4 of Definition 2.

more other proto-atoms, or to a region completely covered by waste (see Figure 5). Situations similar to the example in Figure 5 can be caused by the choice of similarity threshold, which we use to filter alignments. If alignments between pairs of atoms in some class are close to the similarity threshold, some of the alignments within the class pass the threshold, whereas others do not.

Our algorithm iteratively splits proto-atoms or extends existing waste regions until we get a proper segmentation. In a situation depicted in Figure 5, where a single proto-atom A_1+B_1 maps to two proto-atoms A_2 and B_2, we typically want to split the proto-atom A_1+B_1 into two. However, if atoms A_2 and B_2 are separated by a longer waste region, there could be several possible places, where A_1+B_1 could be split. One option would be to choose arbitrarily one of those places. However, this choice might lead to higher cost once we consider additional alignments. Instead, we collect such splitting requirements from all alignments covering the currently studied proto-atom and choose the optimal set of new waste regions to satisfy all of them. Instead of splitting the proto-atom, we may also expand the waste region at one of its ends.

We call our algorithm IMP (Inverse Mapping to a Proto-atom). In each step, we choose a proto-atom P, and consider each alignment a whose source covers P. We will map P through a and consider each waste region w within region $a(P)$. We will map w back to atom P by the alignment a' reverse to a, obtaining region w'. Let W be the set of all such waste regions inversely mapped to P. We will create a new set of waste regions W_{new}, which has to satisfy the following requirements:

C1) Each region in W overlaps with some waste region from W_{new}.

C2) No region between two successive waste regions from W_{new} is completely covered by a region from W.

C3) Every region between two successive waste regions from W_{new} has length at least L.

C4) The new set W_{new} has the lowest cost.

We apply this process to proto-atoms until no further changes are made, and thus the resulting segmentation satisfies conditions of Definition 2.

We construct the optimal set W_{new} for a proto-atom $P = p_1 p_2 \dots p_m$ by dynamic programming. For each prefix $P(0,i) = p_1 \dots p_i$ we compute the cost of the optimal waste region set with space labeled i being part of the last waste region; we denote this quantity as $\mathrm{cost}(i)$. In computing this cost, we consider all regions from W overlapping $P(0,i)$.

We will first describe details of dynamic programming for a simpler case when the set W does not contain any pair of regions such that one of them covers the other. The algorithm computes $\mathrm{cost}(i)$ only for those positions i that are covered by at least one region from W. Suppose that i is covered by a region from W and let $P(j,k)$ be the rightmost region in W such that $k < i$. Since there is a waste region at the space labeled i, all regions from W overlapping i have condition C1 satisfied. However, condition C1 needs to be satisfied also for $P(j,k)$, and therefore we have

$$\mathrm{cost}(i) = \min_{j \le \ell \le k} \mathrm{c}(\ell, i),$$

where value $\mathrm{c}(\ell,i)$ is the cost of the optimal set of new waste regions for prefix $P(0,i)$ given that both space i and space ℓ are part of waste regions. Value $\mathrm{c}(\ell,i)$ consists of the cost for prefix $P(0,\ell)$ and the cost added by enforcing a waste region at position i.

$$\mathrm{c}(\ell,i) = \begin{cases} \mathrm{cost}(\ell) + (i-l) & \text{if } i-\ell \le L \text{ or} \\ & \exists w \in W : P(i,\ell) \subseteq w \\ \mathrm{cost}(\ell) + \varepsilon & \text{otherwise.} \end{cases}$$

In the first case, the waste regions containing i and ℓ are joined together. Otherwise, we would create a new proto-atom which would be too short or which would align to a waste region. In the second case of the formula, we create a new proto-atom $P(\ell,i)$ and add ε to the cost.

If we include auxiliary region $P(m,m)$ in the set W, we will obtain the final cost of the optimal solution as $\mathrm{cost}(m)$. During computation, we store for each i also position ℓ for which $c(\ell,i)$ achieves the minimum. We use these stored positions to trace back the whole optimal solution W_{new}.

If the set of regions W contains some regions that cover other regions, we split the set into two parts. Set C will consists of the regions from W, which cover any other region of W, and set W' will contain the remaining regions. Clearly, if each region from W' overlaps some region from W_{new}, then also each region from C overlaps some region from W_{new}. We will therefore run the algorithm, considering only regions from W' for finding positions i and ℓ, but using the whole set W to determine if $P(i,\ell)$ is a subset of some $w \in W$ in computation of $c(i,\ell)$.

The dynamic programming algorithm runs in time $O(m^2)$ on a proto-atom of length m. We repeat this algorithm for all proto-atoms until there are no further changes. Clearly, the number of such iterations is polynomial, since in each iteration we either detect that no further changes were made for any atom, or we increase either the number of atoms or the number of wasted bases at least by one. In practice, the number of iterations is quite small, up to four in the experiments on simulated data described in the next section.

Note that this algorithm does not give optimal results, because it always tries to ensure condition A4 of Definition 2 by modifying the current atom P, but it might be better to instead modify destinations of alignments with source in P.

Our new IMP algorithm is somewhat similar to the original IHM algorithm introduced in Brejova et al. (2011) in the sense that we repeatedly map positions through input alignments and modify the segmentation, until no more changes are necessary. Unlike IMP, which maps whole waste regions, the IHM algorithm maps individual alignment endpoints and then modifies their location so that at most one endpoint is located within each window of size W. While our new algorithm produces a segmentation conforming to Definition 2, no such formal definition exists for IHM.

5 Experimental Evaluation

In this section, we evaluate the proposed algorithm on simulated sequences and compare the accuracy of its results with our earlier algorithm IHM (Brejova et al., 2011). Simulated data have the advantage that we know their true segmentation, and thus we can easily evaluate performance of the algorithms.

To evaluate a segmentation, we first compute *reciprocal best matches* (BRM) and *boundary fitting matches* (BFM) for its atoms. Suppose that atom A belongs to the true and atom B to the predicted segmentation. Atoms A and B are BRM to each other, if they overlap and no other atom overlaps with any of them by a larger or equal amount than A and B overlap each other. Atoms A and B are BFM to each other, if their start positions differ by at most some amount k and likewise their end positions. As threshold k we choose some fraction of the minimal atom length L, in the experiments we use $\lceil L/4 \rceil$. Note that at this setting, each atom has at most one BRM match and at most one BFM match, and that a BFM match, if it exists, is automatically also a BRM match. BRM-based accuracy measures were used already in Brejova et al. (2011); we propose BFM-based approach to measure if atom boundaries are approximately correct.

We use BRM and BFM to evaluate sensitivity and specificity of the predicted segmentation. In particular, let b be the number of BRM pairs, c be the number of BFM pairs, p be the number of predicted atoms, and t be the number of true atoms. Then BRM specificity is b/p, BRM sensitivity is b/t, BFM specificity is c/p, and BFM sensitivity is c/t.

The BRM specificity decreases for example when several predicted atoms overlap one true atom. The true atom creates a BRM pair with only one of the predicted atoms, while all the predicted atoms are included in p. In this situation, a BFM pair does not have to exist for the true atom, which causes even more significant decrease in the BFM specificity. Conversely, if a predicted atom covers several true atoms, we observe a decrease in BRM and BFM sensitivity.

We also evaluate correctness of the grouping of the predicted atoms into classes. Let A be a class of true atoms and B be a class of predicted atoms. We consider B to be the correct prediction of A, if each atom of B has its BRM/BFM pair in A and each atom of A has its BRM/BFM pair in B. Let d be the number of correctly predicted classes according to BRM, e be the number of correctly predicted classes according to BFM, p be the number of predicted classes, and t be the number of true classes. Then BRM class specificity is d/p, BRM class sensitivity is d/t, BFM class specificity is e/p, and BFM class sensitivity is e/t.

We have evaluated performance of the algorithms on ten simulated data sets taken from Brejova et al. (2011). The sets were produced by simulation of sequence evolution, allowing substitutions, short insertions and deletions, as well as large-scale deletions and duplications.

Unlike our algorithm, which introduces waste regions not covered by any atoms, the IHM algorithm covers the whole sequence by atoms of length at least W. In regions with many alignment boundaries this leads to many short atoms with lengths approximately W. To make the IHM comparable with the IMP algorithm, we have used $W = \lceil L/4 \rceil$, and we have subsequently filtered out all classes containing atoms shorter than the minimal required length L. Similar filtering was also used in Brejova et al. (2011).

Table 1 show the comparison of IMP and IHM on the simulated data with four different minimal atom lengths $L \in \{50, 100, 250, 500\}$. Column TRUE of the table shows the results for the true segmentation from which the atoms shorter than L have been discarded. This is an upper bound on the accuracy of any segmentation algorithm which does not overestimate lengths of atoms.

Both algorithms cover almost whole sequence by atoms and have very high BRM sensitivity and class sensitivity, close to the upper bound given by TRUE. Sensitivity of the IHM algorithm is slightly higher than sensitivity of IMP. However, IMP is much more specific for smaller values of L. For higher L the task becomes easier and both algorithm achieve good specificity. Under the BFM measures, the new algorithm is both more specific and sensitive with the exception of $L = 500$, where IHM slightly outperforms IMP. Overall, IMP performs better, particularly at short atoms lengths, but there is still room for improvement in specificity and correct prediction of atom boundaries.

	measure	$L = 50$ IMP	IHM	TRUE	$L = 100$ IMP	IHM	TRUE	$L = 250$ IMP	IHM	TRUE	$L = 500$ IMP	IHM	TRUE
	coverage	100 %	100 %	100 %	100 %	99 %	100 %	99 %	99 %	100 %	98 %	99 %	99 %
BRM	sp	75 %	46 %	100 %	98 %	67 %	100 %	100 %	90 %	100 %	100 %	100 %	100 %
BRM	sn	96 %	97 %	97 %	94 %	95 %	95 %	88 %	89 %	90 %	77 %	79 %	80 %
BRM	class sp	72 %	47 %	100 %	98 %	71 %	100 %	99 %	92 %	100 %	99 %	100 %	100 %
BRM	class sn	97 %	97 %	98 %	95 %	96 %	97 %	90 %	92 %	93 %	83 %	85 %	86 %
BFM	sp	47 %	19 %	100 %	83 %	42 %	100 %	93 %	77 %	100 %	98 %	99 %	100 %
BFM	sn	61 %	40 %	97 %	79 %	59 %	95 %	82 %	76 %	90 %	75 %	78 %	80 %
BFM	class sp	42 %	22 %	100 %	79 %	46 %	100 %	92 %	79 %	100 %	96 %	99 %	100 %
BFM	class sn	57 %	45 %	98 %	77 %	63 %	97 %	84 %	80 %	93 %	80 %	83 %	86 %

Table 1: Accuracy of the IMP and IHM algorithms on the generated data sets. Sensitivity (sn), specificity (sp) are measured on both atom and class levels with respect to BRM and BFM atom matching, as described in the text.

6 Conclusion

We have studied a sequence segmentation problem arising in comparative analysis of related DNA sequences. We have defined the problem formally, proved its NP hardness and provided a practical heuristic algorithm, which uses dynamic programming to select an optimal combination of several local changes. We have implemented and evaluated our algorithm on simulated data to show that our problem definition and algorithmic approach lead to reasonable results with respect to a simple model of evolution generating our data. Our tool can be used to prepare data for numerous algorithms operating on sets of markers extracted from multiple DNA sequences.

This area offers both practical and theoretical questions for further study. From a theoretical point of view, it would be interesting to study approximation or fixed-parameter algorithms for our problem. From a practical point of view, more detailed evaluation on both simulated and real biological data could discover weak points of our algorithm and to help to improve it further.

Acknowledgments. This research was supported by VEGA grant 1/1085/12.

References

Blanchette, M., Kent, W. J., Riemer, C., Elnitski, L., Smit, A. F., Roskin, K. M., Baertsch, R., Rosenbloom, K., Clawson, H., Green, E. D., et al. (2004). Aligning multiple genomic sequences with the threaded blockset aligner. *Genome Research*, 14(4):708–715.

Brejova, B., Burger, M., and Vinar, T. (2011). Automated segmentation of DNA sequences with complex evolutionary histories. In *Algorithms in Bioinformatics, 11th International Workshop (WABI)*, volume 6833 of *Lecture Notes in Computer Science*, pages 1–13. Springer.

Brudno, M., Malade, S., Poliakov, A., Do, C., Couronne, O., Dubchak, I., and Batzoglou, S. (2003). Glocal alignment: finding rearrangements during alignment. *Bioinformatics*, 19(1):i54–i62.

Choi, V., Zheng, C., Zhu, Q., and Sankoff, D. (2007). Algorithms for the extraction of synteny blocks from comparative maps. In *Algorithms in Bioinformatics (WABI)*, volume 4645 of *Lecture Notes in Computer Science*, pages 277–288. Springer.

Felsenstein, J. (2004). Inferring phylogenies. *Sinauer Associates*.

Fertin, G., Labarre, A., Rusu, I., Tannier, E., and Vialette, S. (2009). Combinatorics of genome rearrangements. *MIT Press*.

Garey, M. and Johnson, D. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co.

Hachiya, T., Osana, Y., Popendorf, K., and Sakakibara, Y. (2009). Accurate identification of orthologous segments among multiple genomes. *Bioinformatics*, 25(7):853–860.

Harris, R. (2007). *Improved pairwise alignment of genomic DNA*. PhD thesis, Pennsylvania State University.

Nadeau, J. H. and Taylor, B. A. (1984). Lengths of chromosomal segments conserved since divergence of man and mouse. *Proceedings of the National Academy of Science USA*, 81(3):814–818.

Ovcharenko, I., Loots, G. G., amd Minmei Hou, B. M. G., Ma, J., Hardison, R. C., Stubbs, L., and Miller, W. (2005). Mulan: Multiple-sequence local alignment and visualization for studying function and evolution. *Genome Research*, 15(1):184–194.