Avoiding Ambiguity and Assessing Uniqueness in Minisatellite Alignment

Benedikt Löwes and Robert Giegerich

Bielefeld University
Faculty of Technology and Center for Biotechnology
33501 Bielefeld, Germany
{bloewes,robert}@techfak.uni-bielefeld.de

Abstract

Several algorithms have been suggested for minisatellite alignment. Their time complexity is high—close to $O(n^3)$—due to the necessary reconstruction of duplication histories. We investigate the uniqueness of optimal alignments computed under the common single-copy duplication model. To this extent, it is necessary to avoid ambiguity in the algorithm employed. We re-code the ARLEM algorithm in the form of a grammar, and apply a disambiguation technique which uses a mapping to a canonical representation of minisatellite alignments. Having arrived at a non-ambiguous algorithm this way, we demonstrate that the underlying model—independent of the algorithm—gives rise to an exorbitant number of different, co-optimal alignments when applied to real-world data. We conclude that alignment-free methods should be considered for minisatellite comparison.

1998 ACM Subject Classification J.3.a [Life and Medical Sciences]: biology and genetics

Keywords and phrases minisatellite alignment, dynamic programming, ambiguity

Digital Object Identifier 10.4230/OASIcs.GCB.2013.110

1 Introduction

1.1 Background

The minisatellite comparison problem

Minisatellites are repetitive DNA sequences that have been used in population genetics and forensic studies [14,17,23]. They consist of short sequence motifs (6 – 100 bases), called units, which can spread over several kilobases as tandem repeats. The main mechanism behind their generation is unequal chromosomal crossover, which creates an extra copy of a unit in one chromosome, and a loss of such a unit in the other. In the course of evolution, units can pick up point mutations, which are then inherited by further copies. Thus, minisatellite alleles in a population differ not only in length, but also show microheterogeneities that make them useful—or even dangerous—genetic markers. For example, Jobling *et al.* have shown significant correlations between minisatellites on the human Y chromosome and the most common family names in England [16]. Recently, it has become of concern that these markers allow to re-personalize patient genome data [15].

Algorithms for minisatellite alignment

The minisatellite alignment problem has attracted substantial interest in bioinformatics. Analysis of highly repetitive sequences is difficult in general, as there is little information hidden in large data sets. Minisatellite sequences are compacted into "maps", i.e. sequences

German Conference on Bioinformatics 2013 (GCB'13).
Editors: T. Beißbarth, M. Kollmar, A. Leha, B. Morgenstern, A.-K. Schultz, S. Waack, E. Wingender; pp. 110–124

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of characters, where each character denotes a unit that differs from others by point mutations. Compacting microsatellites, characterized by a shorter unit length, leads to the same representation. These maps are then aligned as one does with protein or DNA sequences, but placement and scoring of gaps receives special attention. A gap in a minisatellite alignment has the interpretation that at this point, the two alleles have undergone a divergent duplication history, and these duplication histories are reconstructed and scored in order to make optimal gap placements.

We only recall previous work that leads directly to our present approach. Bérard and Rivals proposed an $O(n^4)$ time and $O(n^3)$ space algorithm [6]. This was improved to $O(|\Sigma|n^3)$ and $O(|\Sigma|n^2)$ by Behzadi and Steyaert [5]. Here, Σ is the alphabet size, i.e. the number of different unit types. They also introduced run-length encoding, where a minisatellite map such as `aaabbbbbccacccc` is encoded as `a3b5c2a1c4`. Abouelhoda *et al.* gave an algorithm with a slightly more general model that runs in $O(n^3)$ time and $O(n^2)$ space, and also has a variant for run-length encoded maps [2]. This algorithm is the basis of the ARLEM web server[1] [1], and the starting point for the present work. A further improvement has been made by Pinhas *et al.* by using fast matrix multiplication techniques, achieving a running time of $O(\frac{|\Sigma|n^3 \log^3 \log n}{\log^2 n})$ [18]. While the underlying model of minisatellite evolution has become slightly more general along the way, the main concern of this line of work (including our own contributions) has been the improvement of efficiency. We do not pursue any further this intention here, but instead we raise the question: Should we align minisatellites at all?

1.2 Motivation of the present study

Significance of alignments

Alignment is a powerful method in biosequence analysis for two reasons: First of all, it gives us a quantitative measure of sequence similarity, which is taken as an indicator of evolutionary relatedness. But second, aside from such a distance or similarity score, the alignment from which this score is derived also gives us some qualitative information. It denotes homologous residues, elucidates sequence motifs more conserved than their context, and points out compensating base changes in RNA, which preserve secondary structure in the presence of sequence variation. There are many scenarios, however, where detailed information of the second type is not needed or considered too expensive to compute. In this case, alignment-free methods of sequence comparison are employed, such as k-mer profiles or word metrics [20, 24]. Hybrid methods are also common, the most prominent example being BLAST, where high-scoring segment pairs are found from k-mer matches, but in the output, sequences are aligned [3].

Statistical significance of alignments is commonly rated by P-values, but there is also the independent concern of uniqueness. Detailed information conveyed by an alignment is only dependable when the alignment is unique in the sense that there are no other alignments of similar score, or if so, they differ from the optimal one only in minor aspects. While it is common practice to speak of "the" optimal alignment or "the" minimum free energy structure found by dynamic programming, bioinformaticians are generally aware that the optimal answer to a combinatorial problem does not need to be unique, and co-optimal answers need not be similar. Although few programs do, it appears fair to ask a dynamic programming algorithm to report, together with a solution, the total number of co-optimal solutions (or even report all co-optimal answers on demand). In our case, we ask: How

[1] `http://www.nubios.nileu.edu.eg/tools/`, but currently unavailable due to political circumstances.

many co-optimal alignments are there for a given pair of minisatellites? How many different, but co-optimal duplication histories can be reconstructed? Only when the answer to either question is a very small number (such as 1), it makes sense to interpret the alignments in detail. However, there lies an intrinsic difficulty in this request, which is less widely known.

Ambiguity in combinatorial optimization

Dynamic programming algorithms used in biosequence analysis often explore a search space of exponential size in polynomial time. The candidates in the search space of the algorithm represent the features of our interest – homology assignments to residues, RNA structures, duplication histories. But they need not do so in a unique fashion. An algorithm may construct several candidates which *mean* the same feature of interest – this phenomenon is called *semantic* ambiguity. For example, with context-free grammars modeling RNA secondary structure, two different parses of an RNA sequence may indicate the same structure. This becomes apparent by mapping parse trees to dot-bracket strings, which constitute a canonical representation of structures. In a reasonable model, all candidates with the same meaning achieve the same score. If one of them is optimal, they all are. This is the crux of the question about co-optimal answers: The algorithm has no way to decide whether a large number of co-optimal candidates designates many different features of interest, or is merely a technical artifact of the algorithm. The meaning we associate with the candidates is not represented within the algorithm itself.

This problem has been defined formally and evaluated empirically, its undecidability was shown, and methods for testing or avoiding semantic ambiguity were suggested in [8, 9, 11, 12, 19]. We build on this work below when we eliminate semantic ambiguity from minisatellite alignment algorithms in order to correctly assess the uniqueness of their results.

1.3 Goals and preview of results

These are the goals and results of the present study:

- We raise the question: "How unique are minisatellite alignments constructed by the algorithms mentioned above?" and take the ARLEM algorithm as their representative. This question is relevant to decide whether the effort of constructing such alignments is well spent.
- We observe that the answer to this question is obscured by a high degree of semantic ambiguity in the ARLEM algorithm. We modify the algorithm such that it evaluates the same search space in a unambiguous fashion. The technique we use here is interesting in its own right.
- With the unambiguous algorithm we observe that the number of co-optimal alignments and duplication histories is extraordinarily large, leading to the conclusion that alignment-free methods should be considered as an alternative mode of minisatellite comparison.

Note that focusing on one particular algorithm does not limit the generality of our conclusion: While semantic ambiguity is a property of a particular algorithm, the question of result uniqueness is not: After weeding out semantic ambiguity, all algorithms implementing the same model must agree not only on the optimal score, but also on the number of co-optimal solutions. When ARLEM is plagued with co-optimals, so are all other algorithms implementing the same model. This stringency of the method employed here may render it useful also with other combinatorial optimization problems in biosequence analysis.

2 Abstracting the ARLEM algorithm

The ARLEM algorithm is given in [2] in the traditional form of dynamic programming recurrences, which makes it rather difficult to reason about ambiguity. We develop here a more abstract presentation of this algorithm in the form of a tree grammar. This representation is helpful to explain the underlying alignment and duplication history model, and serves as the starting point of disambiguation in the next section. Better than the recurrences, the grammar reflects the algorithm's division in two logically independent parts: (i) the alignment itself, with the well-known edit operations match, insertion and deletion, and (ii) the computation of duplication histories, which refer to units where the alleles have undergone a divergent development. Incorporated into the alignment, the duplication histories describe a more sophisticated gap model, which can be computed for either sequence individually.

The alignment model

Grammar ARLEM

Alignment

$$A^* \rightarrow \texttt{match}(a, A, b) \mid \texttt{lDupS}(\overleftarrow{L_o}, A) \mid \texttt{lDupT}(A, \overleftarrow{L_o}) \mid \texttt{rDup}(R_o, A, R_o) \mid \texttt{empty}$$

Duplication history

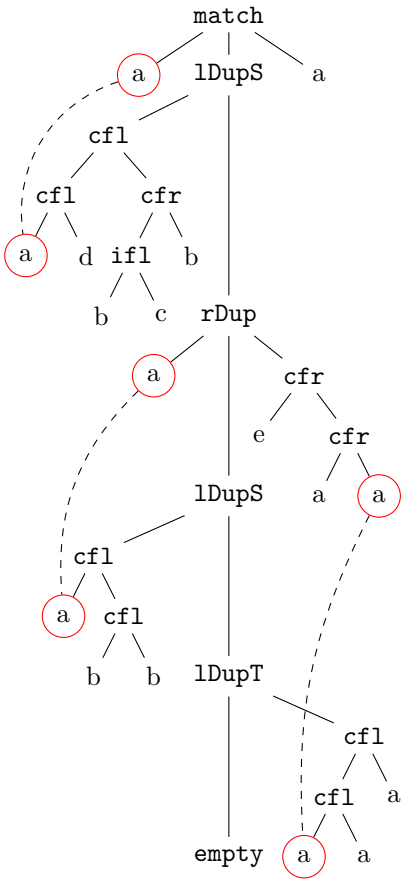
$$\begin{aligned} L_o &\rightarrow \texttt{cfl}(L_o, U) \mid \texttt{ifl}(L_o, U) \mid a \\ R_o &\rightarrow \texttt{cfr}(U, R_o) \mid \texttt{ifr}(U, R_o) \mid a \\ U &\rightarrow L \mid R \mid a \\ L &\rightarrow \texttt{cfl}(U, U) \mid \texttt{ifl}(U, U) \\ R &\rightarrow \texttt{cfr}(U, U) \mid \texttt{ifr}(U, U) \end{aligned}$$

The first rule of grammar ARLEM shows the classical edit distance model, adapted to the alignment of two minisatellite maps `S` and `T`. An asterisk marks the axiom of the grammar. It provides five edit operations: `match` (of two units in `S` and `T`), `lDupS` (left duplication in `S`), `lDupT` (left duplication in `T`), and `rDup` (right duplications in `S` and `T`).[2] Finally, `empty` represents the alignment of two empty minisatellites. The characters a and b are terminal symbols and denote arbitrary units from the unit alphabet. L_o and R_o denote duplications histories with their origin in the left/rightmost unit. They are further explained in the subgrammar for duplication histories. The left arrow indicates that the origin of duplication is one unit to the left and therefore also takes part in a previous `match` or `rDup` operation.

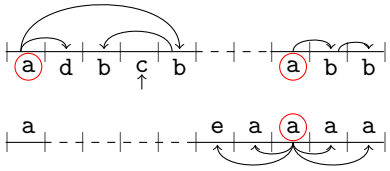
Duplication history model

A left (right) duplication is a sequence of units which originate from their leftmost (rightmost) unit. These sequences can be considered as gaps in the alignment. There is no need for duplication history rules with the origin in the middle of the sequence, since these cases can be constructed by simply using a right duplication followed by a left one. The `rDup` operation internally accounts for a match between the origin units of the two right duplications in `S` and `T`, where either right duplication may actually be empty. This asymmetric treatment of left and right duplications is a technical property of the ARLEM algorithm.

[2] Duplications in `S` could be called deletions, duplications in `T` insertions, but we avoid the directional bias of this terminology here.



(a) Tree structure showing sequence of used grammar rules



(b) Arched-arrow representation of the alignment

S	operations	T
a	(a) $\xleftrightarrow{\texttt{match}}$ a	**a**
a	$\xleftarrow{\texttt{lDupS}}$	a
a**b**	(a) $\xrightarrow{\texttt{cfl}}$ b	a
a**d**b	(a) $\xrightarrow{\texttt{cfl}}$ d	a
ad**b**b	b $\xleftarrow{\texttt{cfr}}$ b	a
adb**c**b	b $\xrightarrow{\texttt{ifl}}$ c	a
adbcb**a**	(a) $\xleftrightarrow{\texttt{rDup}}$ (a)	a**a**
adbcba	e $\xleftarrow{\texttt{cfr}}$ (a)	a**e**a
adbcba	a $\xleftarrow{\texttt{cfr}}$ (a)	ae**a**a
adbcba	$\xleftarrow{\texttt{lDupS}}$	aeaa
adbcba**b**	(a) $\xrightarrow{\texttt{cfl}}$ b	aeaa
adbcbab**b**	b $\xrightarrow{\texttt{cfl}}$ b	aeaa
adbcbabb	$\xrightarrow{\texttt{lDupT}}$	aeaa
adbcbabb	(a) $\xrightarrow{\texttt{cfl}}$ a	aeaa**a**
adbcbabb	(a) $\xrightarrow{\texttt{cfl}}$ a	aeaa**a**a
adbcbabb	empty	aeaaaa

(c) Table showing the sequence of events together with the growing sequences of `S` and `T`.

Figure 1 Example of a minisatellite alignment of two sequences `S="adbcbabb"` and `T="aeaaaa"`. The red-circled units and dashed lines connect all units which occur twice, but in our implementation are only "read" once.

Duplication histories follow the single-copy model of ARLEM. They can be constructed for (sub)-sequences of `S` and `T` individually and use the following operations: `cfl` (copy from left) and `cfr` (copy from right) describe duplications from left to right (or vice versa) followed by an optional mutation of the new unit. The operations `ifl` (insertion from left) and `ifr` (insertion from right) seem syntactically similar, but describe an insertion of a single unit which is a foreign DNA fragment, most likely due to a transposition event. Since the unit is inserted, it has no unit of origin in the duplication history. But for the technical purpose of integrating this event in the duplication history, it takes a neighboring unit as its origin, but this unit does not contribute to the insertion score. In the grammar, nonterminals L_o and R_o mark histories that entail the left/right origin, whereas arbitrary histories are derived from nonterminal symbols L and R. Again, a is a terminal symbol and denotes an arbitrary unit.

Representation of alignments and histories

An example of an alignment with the incorporation of duplication histories is given in Figure 1. Figure 1(a) shows a tree-like representation of the alignment. Along the middle axis, we see the edit operations, noting that only two pairs of units are matched. The first is the

explicit `match` operator near the root of the tree, the second is implicit in the `rDup` operator. All others units arise from duplications, which branch from the stem in the tree towards the left for `S`, and towards the right for `T`, and are built from left and right duplication operators. Providing concrete scoring functions for the edit operations, such a tree can be directly evaluated to its alignment score. Figure 1(b) shows a graphical representation of an alignment and embedded duplication histories. Figure 1(c) shows a possible history of events that produces the two minisatellite sequences from a joint ancestor `aa`. (Note that one cannot reconstruct the exact order in time.)

Implementation in Bellman's GAP

A grammar like the above can be written in GAP-L, the language of the Bellman's GAP system [22], which generates recurrences automatically and functions as a programming system for Algebraic Dynamic Programming (ADP) [10, 13]. This was used to create a faithful emulation of the original ARLEM algorithm, and afterwards its dis-ambiguated version. Important for our intentions, Bellman's GAP allows to augment scoring schemes with "counting algebras", enabling us to evaluate the number of co-optimal solutions. This feature will be used in Section 4 to compare ARLEM with our dis-ambiguated version.

Simplifications

The implementation handles a number of subtleties that have been abstracted away in our presentation. Certain units take part in several edit operations. For instance, take the first left duplication in `S` from position 1 to 5 in Figure 1. The first unit takes part in the `match` operation as well as in a left duplication, as it is the leftmost unit and origin of a duplication history in S. So, the tree appears to hold this element twice. The dashed line connecting both instances indicates that in our implementation, the grammar "reads" this element just once. In the grammar, this is indicated by a red arrow above the nonterminal symbol.

The second problem arises while using the `cfl` and `cfr` operations of the duplication history part of the grammar. Consider the uppermost `cfl` operation in the tree of the first left duplication in `S` and the scenario that the score for the `cfl` operation is the sum of the scores of its two children plus the score of duplicating the unit `a` and mutating it to `b`. Here, it may be possible that there are two (sub-)trees with better scores than the ones of the uppermost `cfl` in Figure 1(a), but since this alignment is optimal, the mutation from the duplicated `a` to `c` has to score worse than the chosen one from `a` to `b`. In general, the optimization chooses the two best scoring (sub-)trees of `cfl` and `cfr` from the underlying sequence interval without considering the cost for the eventual duplication and mutation of the origin, which could lead to suboptimal results at this stage of the alignment and would therefore violate Bellman's Principle of Optimality. This problem is solved by a slightly different decomposition of duplication history intervals.

A third simplification concerns efficiency. The treatment of `rDup` in the grammar would lead to an $O(n^4)$ time algorithm. Using two separated operations like `rDupS` and `rDupT`[3], one can reduce the runtime to $O(n^3)$. This is a common speed-up technique in dynamic programming and is applied in ARLEM as well as in our Bellman's GAP implementations.

Finally, ARLEM allows inserted units to duplicate. We consider this an oversight in ARLEM, as foreign DNA inserts do not support unequal crossover. As this artificially blows up the search space, it will be disallowed in our next version of the algorithm.

[3] Both `rDupT` and `rDupS` include their origin (at the rightmost position), in contrast to `lDupS` and `lDupT`.

Table 1 Examples for the ambiguity of the ARLEM algorithm shown as tree structures as well as canonical representations of the alignments and the corresponding ambiguity type.

Type	Alignment tree structure	Canonical representation
1	$\texttt{match}(a, \texttt{lDupS}(\texttt{cfl}(a, b), \texttt{lDupT}(\texttt{empty}, \texttt{cfl}(a, c))), a)$	$a > (b) >$ $a > (c) >$
	$\texttt{match}(a, \texttt{lDupT}(\texttt{lDupS}(\texttt{cfl}(a, b), \texttt{empty}), \texttt{cfl}(a, c)), a)$	
	$\texttt{rDup}(a, \texttt{lDupS}(\texttt{cfl}(a, b), \texttt{lDupT}(\texttt{empty}, \texttt{cfl}(a, c))), a)$	
2	$\texttt{rDup}(e, \texttt{empty}, \texttt{cfr}(\texttt{cfl}(\texttt{cfr}(a, b), c), d))$	e $< ((a)b(c)) < d$
	$\texttt{rDup}(e, \texttt{empty}, \texttt{cfr}(\texttt{cfr}(a, \texttt{cfl}(b, c)), d))$	
3	$\texttt{match}(a, \texttt{lDupS}(\texttt{cfl}(\texttt{ifl}(a, b), c), \texttt{empty}), a)$	$a > [b](c) >$ a
	$\texttt{match}(a, \texttt{lDupS}(\texttt{cfl}(a, \texttt{ifr}(b, c)), \texttt{empty}), a)$	$a > ([b]c) >$ a

3 Dis-ambiguation of the ARLEM algorithm

Semantic ambiguity in ARLEM

Running our emulation of the ARLEM algorithm, in addition to the original scoring scheme we employ a counting algebra to report the number of co-optimal alignments for each pair of minisatellites. Even for very short sequences, we obtain a large number of co-optimals. For example, aligning `adbcbabb` and `aeaaaa` from Figure 1 gives 1052 co-optimal answers.[4] Closer inspection shows that the same optimal alignment is returned several times. The algorithm is semantically ambiguous. For sources of ambiguity, consider Table 1.

1. Ambiguity type 1 shows three candidates that refer to the same "real" situation. Between the first two, the order of `lDupS` and `lDupT` is reversed, while in reality, these are individual duplication histories in `S` and `T` and do not happen in any order. The third entry uses the `rDup` operation with two empty histories, which is equivalent to a plain `match`.
2. Ambiguity type 2 shows a match between two `d` units, which is preceded by a right duplication in `T` showing a `b` unit originating from `d` and creating an `a` to the left and a `c` to the right. In the first example, the `a` is copied from `b` after the copy of `c`, while the second example reverses this order. This is within the duplication history of the same minisatellite. There is no evidence of which event happened first. Hence, we want to avoid reporting this situation twice.
3. Ambiguity type 3 shows an insertion of a unit `b`, which is modeled both as insertion from the left (unit `a`) and from the right (unit `c`). Remember that the insertion score depends neither on `a` nor on `c`. It is merely a technical requirement of the algorithm that the inserted unit has an origin, and hence one of the two cases should be ruled out.

These are just examples – note that there may be further sources of ambiguity!

Canonical alignment representation

Controlling ambiguity requires formalizing the "real world meaning" of the candidate alignments produced by the algorithm by means of a canonical representation [11], together with a semantic mapping μ that maps candidates to their meanings. For this mapping, we must show that $c \neq c'$ implies $\mu(c) \neq \mu(c')$ – then the algorithm is semantically unambiguous.

[4] For larger examples, numbers get so large that the counting algebra provided by Bellman's GAP overflows, and we had to use unlimited integers.

This is our canonical representation: Minisatellite alignments are represented as strings on two lines. Left duplications are enclosed in two "<"s, right duplications are enclosed in two "<"s. For empty right duplications, the "<"s may be omitted. Matched units of `S` and `T` are written below each other on the two lines; spaces have no meaning. For the duplication histories, we have to distinguish between the duplication (`cfl` and `cfr`) and insertion (`ifl` and `ifr`) operations. In case of duplications, the newly copied units are enclosed in parentheses ("()") and written to the right (left) of their origin. The same holds for insertions except that square brackets ("[]") are used. Examples of the canonical representation are given in the rightmost column of Table 1. Note that all the ambiguous cases there map to the same canonical representation, as intended. Formally, the canonical mapping μ is defined as follows:

Alignment

$$\mu(\texttt{lDupS}(S,A)) = \left\{\begin{matrix}\text{" > " } \texttt{~+} \ \sigma(S) \ \texttt{++} \text{ " > "}\end{matrix}\right\} \ \ddagger \ \mu(A)$$

$$\mu(\texttt{lDupT}(T,A)) = \left\{\begin{matrix}\text{" > " } \texttt{~+} \ \sigma(T) \ \texttt{++} \text{ " > "}\end{matrix}\right\} \ \ddagger \ \mu(A)$$

$$\mu(\texttt{rDup}(S,T,A)) = \left\{\begin{matrix}\text{" < " } \texttt{++} \ \sigma(S) \ \texttt{+~} \text{ " < "} \\ \text{" < " } \texttt{++} \ \sigma(T) \ \texttt{+~} \text{ " < "}\end{matrix}\right\} \ \ddagger \ \mu(A)$$

Duplication history

$$\begin{aligned}\sigma(\texttt{cfl}(A,B)) &= \sigma(A) \ \texttt{++} \text{ "(" } \texttt{++} \ \sigma(B) \ \texttt{++} \text{ ")"} \\ \sigma(\texttt{ifl}(A,B)) &= \sigma(A) \ \texttt{++} \text{ "[" } \texttt{++} \ \sigma(B) \ \texttt{++} \text{ "]"} \\ \sigma(\texttt{cfr}(A,B)) &= \text{"(" } \texttt{++} \ \sigma(A) \ \texttt{++} \text{ ")" } \texttt{++} \ \sigma(B) \\ \sigma(\texttt{ifr}(A,B)) &= \text{"[" } \texttt{++} \ \sigma(A) \ \texttt{++} \text{ "]" } \texttt{++} \ \sigma(B) \\ \sigma(a) &= \text{"}a\text{"}\end{aligned}$$

String concatenation operators

$$\begin{array}{llll}
\left\{\begin{matrix}A\\C\end{matrix}\right\} & \ddagger & \left\{\begin{matrix}B\\D\end{matrix}\right\} & = \left\{\begin{matrix}AB\\CD\end{matrix}\right\} \\
A & \texttt{++} & B & = AB \\
A & \texttt{~+} & xB & = AB \\
A \text{ " < " } x & \texttt{+~} & \text{" < " } B & = AxB \\
A \text{ " < " } \tilde{C}x & \texttt{+~} & \text{" < " } B & = A \text{ " < " } \tilde{C} \text{ " < " } xB
\end{array}$$

where capital letters A, B, C and D denote strings of length ≥ 0, x denotes a single unit and $\tilde{C}$ designates a string with length > 0 in which no "<" character is allowed.

Dis-ambiguating ARLEM

Grammar ARLEM-NONAMB

Alignment

$$\begin{aligned}
A^* &\rightarrow \texttt{rDup}(R_o, B, R_o) \mid \texttt{empty} \\
B &\rightarrow \texttt{lDupS}(L_o, C) \mid C \\
C &\rightarrow \texttt{lDupT}(A, L_o) \mid A
\end{aligned}$$

Duplication history

$$\begin{aligned}
L_o &\rightarrow \texttt{cfl}(L, R) \mid L_i \\
L_i &\rightarrow \texttt{ifl}(L_i, a) \mid \texttt{ifl}(a, a) \\
L &\rightarrow L_o \mid a \\
R_o &\rightarrow \texttt{ifr}(a, R_o) \mid R_h \\
R_h &\rightarrow \texttt{cfr}(R, R_h) \mid a \\
R &\rightarrow \texttt{cfr}(R, R) \mid L
\end{aligned}$$

The grammar ARLEM-NONAMB avoids the aforementioned types of ambiguity by the following means:

1. Type 1 ambiguity is avoided because the new grammar forces `lDupS` from nonterminal B before `lDupT` from nonterminal C, and the reverse order is not allowed. Also, we simply abandon the `match` operation, since it is covered by the more general `rDup`.
2. To avoid type 2, it is sufficient to restrict the left side of the `cfl` operation to the nonterminal L so that all `cfr` operations of one unit occur "before" the `cfl` operations.
3. We insist that the (merely technical) origin of an insertion is always on the left (unless it happens at the beginning of the minisatellite). If the direct left neighbor itself is inserted, the nearest left neighbor that is not inserted becomes origin of the `ifl` operation. This policy is reflected by the nonterminal L_i, whereas R_o holds the `ifr` operation in case the leftmost units of a right duplication have to be inserted. Finally, the nonterminal R_h is necessary to ensure that all right duplications starting with R_o have a right origin and L_o guarantees that a left duplication is at least of length two.

A proof is required to show that the grammar ARLEM-NONAMB is semantically unambiguous for the chosen canonical representation. We construct a context-free grammar that generates canonical representations in 1:1 correspondence with candidates constructed by ARLEM-NONAMB. Then the unambiguity of this grammar was proven with an automated ambiguity checker [8]. The full construction is included in Appendix B.

4 Assessing the co-optimal alignment search space

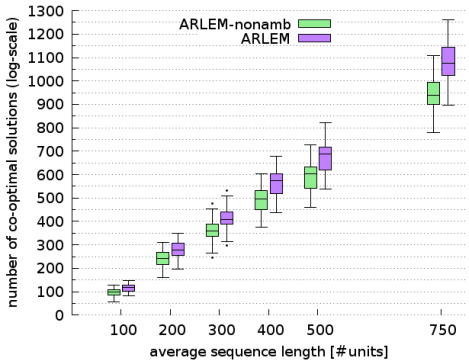
Using ARLEM-NONAMB, we investigate the number of co-optimal alignments, using three data sets: A set of random data (270 alignments), and the two "real-world" sets MSY1 and MS205 (59340 resp. 91806 alignments). See Appendix A for details. We use the same scoring system as in [2]. For comparison, we also describe the higher numbers reported by ARLEM due to its ambiguity. We find that while disambiguation has reduced the numbers reported, the co-optimal alignment space is still extremely large. See Figure 2. For sequences of average length 100, we find about 10^{100} co-optimal alignments, and 10^{500} for sequence length 400. The maps in the "real-world" data set are rather short, but still, 10^{20} co-optimal alignments discourage the idea that choosing a particular, optimal alignment for biological interpretation would be a meaningful effort.

We further investigated if the above situation might be alleviated by a version of the algorithm making use of run-length encoding. This has the effect that all duplication histories without modifications (a run) are considered equivalent and reported as one. While the counts are reduced further, an exponential pattern still prevails. See Appendix C for details.

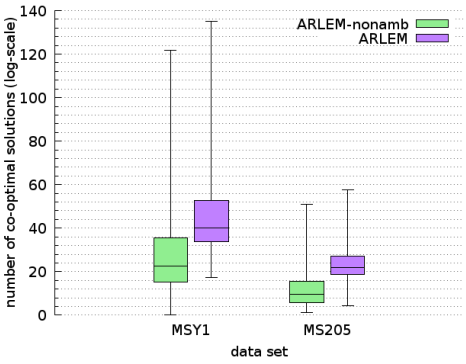
5 Conclusion

Dynamic programming algorithms as used in minisatellite alignment perform exact (rather than heuristic) combinatorial optimization. When such an algorithm is unambiguous and still reports a large number of co-optimal answers, the underlying problem is ill-posed. From our experiments with ARLEM-NONAMB, we conclude that the commonly used model of duplication histories lacks the distinctive power to designate a most plausible individual duplication history and pairwise minisatellite alignment.

A more fine-grained scoring scheme might produce sharper peaks for the same model. However, our own experiments with training `match`(a, A, b) from data sets has not led to an improvement. From this point, one could also move forward to advocate more sophisticated models of repeat evolution, such as the VNTR model in [21], where multiple copies are allowed and copies need not match unit boundaries. Ambiguity in such models has not yet



(a) Number of co-optimal alignments using random sequences



(b) Number of co-optimal alignments in MSY1 (length 48 – 114) and MS205 (length 23 – 75)

Figure 2 Numbers of co-optimal alignments reported by ARLEM-NONAMB and the emulated ARLEM algorithm. The y-axes show the logarithms ($\log_{10}$) of the actual values.

been addressed; a more elaborate model *may* lead to a smaller number of co-optimal results. Computational effort, however, will rise above $O(n^3)$.

We are not saying that minisatellite alignment methods have lost their merits. There may be data sets where alignments are unique. Observations like the directional bias of minisatellite growth, seen in [2], can only be made with an alignment method. Still, with large scale data and when only a similarity measure is required, we propose that minisatellite alignment should be abandoned in favor of alignment-free methods [20, 24]. These methods are based on k-mer composition and are often used for read clustering in meta-genomics. However, there is no off-the-shelf solution, as minisatellites are repetitive sequences and their k-mer profiles are dominated by a small number of k-mers that carry little information.

References

1 M. I. Abouelhoda, M. El-Kalioby, and R. Giegerich. WAMI: a web server for the analysis of minisatellite maps. *BMC Evolutionary Biology*, 10:167, 2010.

2 M. I. Abouelhoda, R. Giegerich, B. Behzadi, and J. M. Steyaert. Alignment of minisatellite maps based on run-length encoding scheme. *Journal of Bioinformatics and Computational Biology*, 7(2):287–308, April 2009.

3 S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, October 1990.

4 J. A. Armour, P. C. Harris, and A. J. Jeffreys. Allelic diversity at minisatellite MS205 (D16S309): evidence for polarized variability. *Human Molecular Genetics*, 2(8):1137–1145, August 1993.

5 B. Behzadi and J.-M. Steyaert. The Minisatellite Transformation Problem Revisited: A Run Length Encoded Approach. In Inge Jonassen and Junhyong Kim, editors, *WABI*, volume 3240 of *Lecture Notes in Computer Science*, pages 290–301. Springer, 2004.

6 S. Bérard and E. Rivals. Comparison of minisatellites. *Journal of Computational Biology*, 10(3-4):357–372, 2003.

7 N. Bouzekri, P. G. Taylor, M. F. Hammer, and M. A. Jobling. Novel mutation processes in the evolution of a haploid minisatellite, MSY1: array homogenization without homogenization. *Human Molecular Genetics*, 7(4):655–659, April 1998.

8 C. Brabrand, R. Giegerich, and A. Møller. Analyzing Ambiguity of Context-Free Grammars. *Science of Computer Programming*, 75(3):176–191, March 2010.

9 R. D. Dowell and S. R. Eddy. Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction. *BMC Bioinformatics*, 5:71, June 2004.

10 R. Giegerich. A systematic approach to dynamic programming in bioinformatics. *Bioinformatics*, 16(8):665–677, 2000.

11 R. Giegerich. Explaining and Controlling Ambiguity in Dynamic Programming. In Raffaele Giancarlo and David Sankoff, editors, *CPM*, volume 1848 of *Lecture Notes in Computer Science*, pages 46–59. Springer, 2000.

12 R. Giegerich and C. Höner zu Siederdissen. Semantics and ambiguity of stochastic RNA family models. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 8(2):499–516, 2011.

13 R. Giegerich, C. Meyer, and P. Steffen. A discipline of dynamic programming over sequence data. *Science of Computer Programming*, 51(3):215–263, June 2004.

14 P. Gill, A. J. Jeffreys, and D. J. Werrett. Forensic application of DNA 'fingerprints'. *Nature*, 318(6046):577–579, 1985.

15 M. Gymrek, A. L. McGuire, D. Golan, E. Halperin, and Y. Erlich. Identifying personal genomes by surname inference. *Science*, 339(6117):321–324, January 2013.

16 M. A. Jobling, N. Bouzekri, and P. G. Taylor. Hypervariable digital DNA codes for human paternal lineages: MVR-PCR at the Y-specific minisatellite, MSY1 (DYF155S1). *Human Molecular Genetics*, 7(4):643–653, April 1998.

17 M. A. Jobling and C. Tyler-Smith. The human Y chromosome: an evolutionary marker comes of age. *Nature Reviews Genetics*, 4(8):598–612, August 2003.

18 T. Pinhas, D. Tsur, S. Zakov, and M. Ziv-Ukelson. Edit Distance with Duplications and Contractions Revisited. In Raffaele Giancarlo and Giovanni Manzini, editors, *CPM*, volume 6661 of *Lecture Notes in Computer Science*, pages 441–454. Springer, 2011.

19 J. Reeder, P. Steffen, and R. Giegerich. Effective ambiguity checking in biosequence analysis. *BMC Bioinformatics*, 6:153, 2005.

20 G. Reinert, D. Chew, F. Sun, and M. S. Waterman. Alignment-free sequence comparison (I): statistics and power. *Journal of Computational Biology*, 16(12):1615–1634, December 2009.

21 M. Sammeth and J. Stoye. Comparing Tandem Repeats with Duplications and Excisions of Variable Degree. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(4):395–407, 2006.

22 G. Sauthoff, M. Möhl, S. Janssen, and R. Giegerich. Bellman's GAP–a language and compiler for dynamic programming in sequence analysis. *Bioinformatics*, 29(5):551–560, March 2013.

23 B. Sykes and C. Irven. Surnames and the Y chromosome. *American Journal of Human Genetics*, 66(4):1417–1419, April 2000.

24 S. Vinga and J. Almeida. Alignment-free sequence comparison-a review. *Bioinformatics*, 19(4):513–523, March 2003.

A Data sets used in evaluation

In the evaluation of the different algorithms, we used three data sets, a random one and two from human. We used random sequences of lengths 100, 200, 300, 400, 500 and 750 for the pairwise comparison of 10 sequences per length, resulting in 270 different alignments. In order to create some variance, the sequences were allowed to be 20% longer or shorter than the default length and the maximal repeat length of a unit was set to 25% of the complete length. The alphabet size of the 10 sequences per length was allowed to be in the range of 5% to 10% of the sequence length.

The MSY1 data set [7, 16] is based on a locus which lies on the human Y chromosome and shows sequence lengths varying between 48 and 114 units with a single unit having a length of 25 nt. Initially, five different unit types were identified, which differ only by few base substitutions at fixed sites. By using an extended approach, three additional unit types, which are mutated versions of the previous ones, could be identified, resulting in an alphabet size of 8. Overall, the data set consists of 345 distinct sequences which results in 59340 different pairwise alignments.

The locus of the MS205 data set [4] maps to the subtelomeric part of the short arm of the 16^{th} human chromosome. The complete length of the tandem repeat region is less than 5 kb and consists of 23 to 75 repeats with a unit length ranging from 45 to 54 bp. Overall, the data set consists of 429 different sequences with an alphabet size of only two, but with several sites of variation between the two types. This results in 91806 different pairwise alignments.

B Proof of unambiguity

Grammar ARLEM-NONAMB-STRING

Alignment

$$
\begin{array}{rcll}
A^* & \to & \text{" < "}\ R_o\ \text{" < "}\ B\ \text{" < "}\ R_o\ \text{" < "} & \mid \text{"\$"} \\
B & \to & \text{" > "}\ L_o\ \text{" > "}\ C & \mid C \\
C & \to & A\ \text{" > "}\ L_o\ \text{" > "} & \mid A
\end{array}
$$

Duplication history

$$
\begin{array}{rcll}
L_o & \to & L\ \text{"("}\ R\ \text{")"} & \mid L_i \\
L_i & \to & L_i\ \text{"["}\ U\ \text{"]"} & \mid U\ \text{"["}\ U\ \text{"]"} \\
L & \to & L_o & \mid U \\
R_o & \to & \text{"["}\ U\ \text{"]"}\ R_o & \mid R_h \\
R_h & \to & \text{"("}\ R\ \text{")"}\ R_h & \mid U \\
R & \to & \text{"("}\ R\ \text{")"}\ R & \mid L \\
U & \to & \text{"}a\text{"} \mid \ldots \mid \text{"}z\text{"} &
\end{array}
$$

In order to show that the (tree) grammar ARLEM-NONAMB is semantically unambiguous for the canonical representation μ, we transform the given grammar into a context-free (string) grammar ARLEM-NONAMB-STRING. Productions of both grammars are isomorphic, such that for a minisatellite alignment $t \in L(\text{ARLEM-NONAMB})$, $\mu(t)$ is generated by ARLEM-NONAMB-STRING by the same derivation. If ARLEM-NONAMB was semantically ambiguous, then ARLEM-NONAMB-STRING would be syntactically ambiguous. We use an ambiguity analyzer for context-free grammars [8] to prove that ARLEM-NONAMB-STRING is unambiguous, i.e. there is only one derivation for each valid canonical alignment string. This establishes the desired result.

A few technical problems must be solved before we can encode $\mu(L(\text{ARLEM-NONAMB}))$ as a context-free grammar. First, the two alignment strings for `S` and `T` are now written on a single line, separated by a "$" character. The string for `T` shows the alignment operations in reverse order, while between them, duplication histories are in their original order. For instance,

$$a > (b) > d$$
$$a > (a) > c$$

is now produced as

$$< a <> a(b) >< d < \$ < c <> a(a) >< a <,$$

and describes the alignment starting with a match of two `a` units, followed by a left duplication in `S` and `T`. It ends with a match between units `c` and `d`.

The example also shows two further deviations from the original canonical representation: right duplications include their origin within the <-brackets, rather than showing it outside to their right. Since for the sake of unambiguity, matches are treated as right duplications of length zero in ARLEM-NONAMB, the matched units are now shown as singletons between < and <.

With respect to left duplications, the string grammar is allowed to produce the same unit twice, first as a match between S and T and then again as the origin inside the duplication history. A match of unit `a` followed by a left duplication in S now reads

$$< a <> a(b) > \$ < a <$$

where the a in S actually exists only once. Still, this notation is in 1:1 correspondence with that generated by μ, although it is less readable for the human eye.

However, being context-free, the grammar also generates additional strings such as

$$< a <> c(b) > \$ < a <$$

with $a \neq c$. This does not correspond to a canonical representation generated by μ, so we have a language inclusion: $L(\text{ARLEM-NONAMB-STRING}) \supset \mu(L(\text{ARLEM-NONAMB}))$ modulo the aforementioned notation changes.

Syntactic ambiguity of context-free grammars, in general, is an undecidable problem. However, there are powerful semi-decision procedures. Finally, grammar ARLEM-NONAMB-STRING is submitted to the ACLA ambiguity checker at `http://www.brics.dk/grammar`, which confirms that it is unambiguous.

C Co-optimal alignment search space using run-length encoding

Presentation of new algorithms using run-length encoding

In addition to the two algorithms ARLEM and ARLEM-NONAMB, we used the idea introduced by Behzadi *et al.* [5] to incorporate run-length encoding (RLE) into the design of the algorithm. In Abouelhoda *et al.* [2], the initial goal was to use RLE to speed up the computation while ensuring that the optimal overall score is found. Speed plays only a minor role in our study; we want to use RLE to reduce the number of co-optimal alignments. We have realized this idea in two different ways. First, the algorithm RLE V1 uses the same approach as ARLEM-NONAMB, but with RLE sequences as input, such as `aaabbcddd` now represented as `a3b2c1d3`. Obviously, the basic algorithm ARLEM-NONAMB can be easily

Table 2 Scoring statistics of the algorithms RLE v1 and RLE v2 using the random, MSY1 and MS205 data set. Here, v1 and v2 are used synonymously for the scores of the algorithms RLE v1 and RLE v2.

	Random	**MSY1**	**MS205**
# alignments	270	59340	91806
v1 = v2	4%	34%	25%
mean $\frac{\mathbf{v1}}{\mathbf{v2}}$	1.04	1.20	1.64
max. $\frac{\mathbf{v1}}{\mathbf{v2}}$	1.29	7.66	7.66

modified to parse a unit as a character and an integer instead of just a character. Additionally, the scoring functions have to take the changes into account. For example, a run of length m scores as $m-1$ duplications, with no mutations involved. Overall, the basic approach of the algorithm remains unchanged. Taking advantage of the RLE input significantly speeds up the algorithm, but introduces the problem that we can no longer guarantee that the optimal alignment score will be found. This occurs when optimality requires to split runs, e.g. when aligning `a41` to `a20b1a20`.

Therefore, we tried a second approach, resulting in the algorithm RLE v2, to use the RLE idea and to guarantee that the optimal score can be found. Such an algorithm was already given by Abouelhoda *et al.*, but with the problem that it introduces yet another source of semantic ambiguity. A further issue is that this algorithm is not exclusively based on dynamic programming, but also makes us of combinatorics to compute the scores for the left/right duplications of the whole sequences from the scores of left/right duplications of the RLE sequences. In general, this approach uses the full-length sequences as input, but exploits the advantages of the RLE approach in the matrix recurrences. Despite these problems, we have also implemented RLE v1 and the slower but correct RLE v2 using the Bellman's GAP system and have eliminated the semantic ambiguity in similar fashion as before. In fact, our RLE v2 algorithm considers all duplication histories without modification equivalent and reports them as one.

Scoring statistics of RLE v1 versus RLE v2

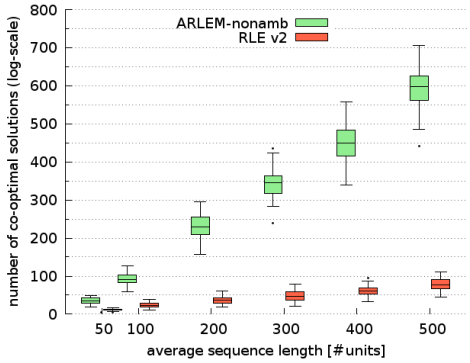
Using these two new grammars, we have investigated in what percentage the RLE v1 algorithm was able to compute the optimal alignment score. The result can be seen in Table 2. While the scores for the random sequences show no major differences between the two algorithms, the "real-word" data sets show large differences in cases of their maximum difference. In some cases the calculated score is not even near the optimum, which discourages the use of the RLE v1 algorithm.

Co-optimal alignments under run-length encoding

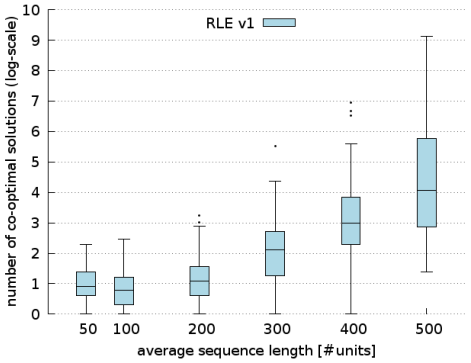
To evaluate if RLE reduces the number of co-optimal alignments significantly, see Figure 3.

For the RLE v2 algorithm, we observe that for random sequences of 300 units still 10^{50} different alignments exist. The same holds for the MSY1 and MS205 data sets where the median of RLE v2 is only a fraction smaller than the one of ARLEM-NONAMB. While the measured maximum value has decreased drastically, the median value is still high and discourages the use of this algorithm.

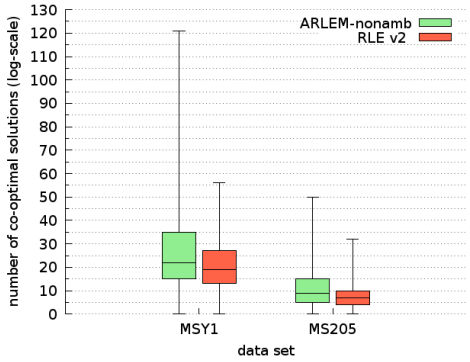
In contrast, the RLE v1 algorithm is characterized by a relatively small number of co-optimal alignments since the medians for the random sequences of lengths 200 and 300 are



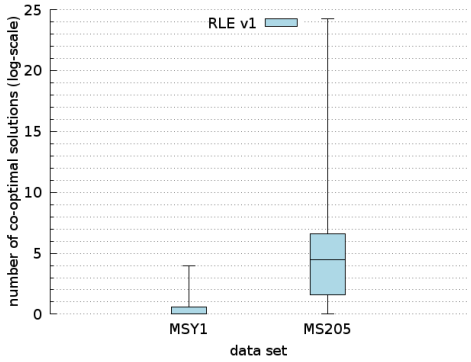
(a) Number of co-optimal alignments using random sequences and the ARLEM-NONAMB and RLE V2 algorithms



(b) Number of co-optimal alignments using random sequences and the RLE V1 algorithm



(c) Number of co-optimal alignments using the MSY1 and MS205 data sets and the ARLEM-NONAMB and RLE V2 algorithms



(d) Number of co-optimal alignments using the MSY1 and MS205 data sets and the RLE V1 algorithm

Figure 3 Numbers of co-optimal alignments reported by the ARLEM-NONAMB, RLE V1 and RLE V2 algorithm. The y-axes show the logarithms ($\log_{10}$) of the actual values.

about 10 and 200. This impression is confirmed when looking at the MSY1 data set, whereas the MS205 record shows a higher number of co-optimal alignments. This may be due to the small alphabet size of MS205, which inevitably leads to fewer different alignment scores.

In the end, the RLE V2 algorithm is not able to eliminate the exponential pattern of co-optimal alignments, while RLE V1 performs well in this regard, but strongly diverges from the optimal alignment score.