Suboptimal Local Alignments across Multiple Scoring Schemes

Morris Michael[1,2], Christoph Dieterich[2], and Jens Stoye[1]

[1] Technische Fakultät, Universität Bielefeld, 33594 Bielefeld, Germany
mmichael@techfak.uni-bielefeld.de, stoye@techfak.uni-bielefeld.de
[2] Computational Molecular Biology, Max Planck Institute for Molecular Genetics, 14195 Berlin, Germany, christoph.dieterich@molgen.mpg.de

Abstract. Sequence alignment algorithms have a long standing tradition in bioinformatics. In this paper, we formulate an extension to existing local alignment algorithms: *local alignments across multiple scoring functions.* For this purpose, we use the Waterman-Eggert algorithm for suboptimal local alignments as template and introduce two new features therein: 1) an alignment of two strings over a set of score functions and 2) a switch cost function δ for penalizing jumps into a different scoring scheme within an alignment.
Phylogenetic footprinting, as one potential application of this algorithm, was studied in greater detail. In this context, the right evolutionary distance and thus the scoring scheme is often not known *a priori*. We measured sensitivity and specificity on a test set of 21 human-rodent promoter pairs. Ultimately, we could attain a 4.5-fold enrichment of verified binding sites in our alignments.
Key words: Sequence alignment, non-parametric alignment, phylogenetic footprinting, comparative sequence analysis.

1 Introduction

Comparative sequence analysis is a powerful tool in bioinformatics for addressing a variety of issues. Applications range from grouping of sequences (e.g. protein sequences) into families to *de novo* pattern discovery of functional signatures. Thus, sequence comparison aims at detecting "biologically meaningful" similarities between sequences. Considering gene regulation, it has been known for a long time that there is considerable sequence conservation between species in non-protein-coding regions of the genome. Especially, sequence conservation within promoter regions of genes often stems from transcription factor binding sites that are under selective pressure (see [5] for a review). Duret and Bucher [4] give an overview on exploiting sequence conservation across species for the detection of regulatory elements. This concept is commonly referred to as *phylogenetic footprinting*.

Phylogenetic footprinting in a strict sense is carried out on orthologous promoter regions. Local sequence similarities can then be directly interpreted as related regions harboring conserved functional binding sites. Selecting suitable

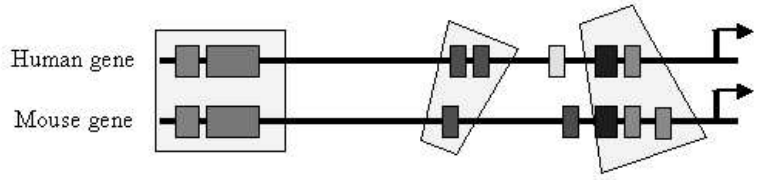


Fig. 1. The concept of phylogenetic footprinting. Local sequence similarities in orthologous promoter regions of genes (light "framed" regions) occur often due to selective pressure on transcription factor binding sites (shaded boxes).

sequence pairs and the choice of the right scoring parameters is crucial to the success of the footprinting approach.

Computational approaches to phylogenetic footprinting. If we recall the situation in Figure 1, an appropriate way of detecting local similarities is to retrieve many local alignments from the search space. Waterman and Eggert [14] proposed an extension of the Smith-Waterman [11] local alignment algorithm for finding non-trivial local similarities (non-intersecting suboptimal local alignments). In earlier work, we have employed an implementation of the Waterman-Eggert algorithm successfully in a large-scale study of man-mouse promoter regions [3].

Related approaches. Heuristic algorithms for large-scale comparison of genomic regions emerged as a new field in computational biology [9]. Recent reviews [12, 2] give a survey of the field and list all available "genome alignment" tools. These software packages are readily applicable to compare whole *syntenic* regions of genomes. BLASTZ [10] is the most similar heuristic solution to the Waterman-Eggert approach since it computes suboptimal local alignments with gaps, which have no constraints on their position. Although, BLASTZ is not guaranteed to find the optimal solution in our setting, it performs well in practice.

Alignment over several score functions. None of the previously mentioned solutions aligns sequence pairs over more than one score function. However, this is desirable in the context of *phylogenetic footprinting*. In maximizing local scores over more than one scoring scheme, sequence properties that can be reflected in the scoring scheme (e.g. local GC-content, bias in substitution patterns) are better captured by the alignment algorithm. As another application, Altschul [1] introduced the idea of multiple score functions for database searches where one does not know *a priori* the right evolutionary distance between two sequences. This is analogous to comparing two promoter regions where we do not know the evolutionary rates of neutrally diverging sequence and sequence elements under selective pressure. In this paper, we propose an extension to the Waterman-Eggert algorithm to meet these demands.

Structure of the paper. Firstly, we introduce the basic notation and edit operations of local alignment algorithms. Secondly, we formally extend the set of edit operations to alignments over several score functions. Subsequently, we present our implementation of an extended version of the Waterman-Eggert algorithm (as implemented by Huang and Miller [8]), and finally we evaluate the impact of our modifications on the problem of finding regulatory elements by comparative sequence analysis.

2 Definitions and Notation

The empty string is denoted by ε and an alphabet of symbols by $\mathcal{A}$. $\alpha_1\alpha_2\ldots\alpha_n$ is the concatenation of $\alpha_1, \alpha_2, \ldots, \alpha_n$, where α_i can be either a string or a symbol and ε is the neutral element: $\alpha\varepsilon = \alpha = \varepsilon\alpha$. $|w|$ denotes the length and w_i the ith symbol of the string w.

2.1 Basic Definitions

Here, we briefly review the standard terminology of sequence alignment, as we will use it throughout this paper.

Definition 1 (Edit Operation). *An* edit operation *is a pair*

$$(\alpha, \beta) \in (\mathcal{A} \cup \{\varepsilon\}) \times (\mathcal{A} \cup \{\varepsilon\}) \setminus \{(\varepsilon, \varepsilon)\}.$$

It is usually denoted by $\alpha \to \beta$.

Edit operations describe the step by step transformation of a source string into a target string. Three kinds of edit operations exist:

$\alpha \to \varepsilon$ denotes the *deletion* of the symbol α.
$\varepsilon \to \beta$ denotes the *insertion* of the symbol β.
$\alpha \to \beta$ denotes the *replacement* of the symbol α by β. Here we distinguish between two cases. If $\alpha = \beta$ it is called a *match*, otherwise it is called a *mismatch* or an *exchange*.

A maximal sequence of adjacent insertions and deletions forms a *gap*.

Definition 2 (Alignment). *An* alignment *A of two strings u and v is a sequence $(\alpha_1 \to \beta_1, \ldots, \alpha_h \to \beta_h)$ of edit operations such that $\alpha_1 \ldots \alpha_h = u$ and $\beta_1 \ldots \beta_h = v$.*

An alignment is usually displayed by placing the symbols of the two aligned strings in different lines, where ε is replaced by `-`.

Example 1. The alignment $(d \to \varepsilon, a \to a, \varepsilon \to i, r \to r, l \to l, i \to i, n \to n, g \to e)$ is displayed as follows:

u: `da-rling`
v: `-airline`

Definition 3 (Score Function). *A (similarity)* score function σ *assigns to each edit operation* $\alpha \to \beta$ *a score* $\sigma(\alpha \to \beta)$*, where similar pairs of symbols (matches or exchanges) are scored by positive values and dissimilar pairs by negative values. Using affine gap costs, the score of an alignment* $A = (\alpha_1 \to \beta_1, \ldots, \alpha_h \to \beta_h)$ *is the sum of the scores of all edit operations and an additional cost for each gap:* $\sigma(A) = \sum_{k=1}^{h} \sigma(\alpha_k \to \beta_k) + g \cdot \gamma$*, where* g *is the number of gaps in* A *and* $\gamma \leq 0$ *denotes the gap open cost.*

Definition 4 (Local Alignment Problem). *Let two strings* u *and* v *and a score function* σ *be given. The* local alignment problem *is to determine an alignment* A *of* u' *and* v' *such that* u' *and* v' *are substrings of* u *and* v *and the score of* A *is maximal:*

$$\sigma(u, v) = \max \left\{ \sigma(A) \,\middle|\, \begin{array}{l} A \text{ is an alignment of } u' \text{ and } v', \\ u' \text{ is a substring of } u,\ v' \text{ is a substring of } v \end{array} \right\}.$$

Each alignment that satisfies this condition is called an optimal local alignment *of* u *and* v*, and* $\sigma(u, v)$ *is called the* optimal local alignment score *of* u *and* v*.*

Definition 5 (Suboptimal Alignments). *A local alignment* A *of* u *and* v *is called* suboptimal *if* $\sigma(A)$ *is smaller than the optimal alignment score of* u *and* v*. If we are looking for suboptimal alignments, we want to find them ordered by decreasing score.*

2.2 Several Score Functions

Sequence alignment over several score functions can now be introduced as a direct generalisation of (standard) local alignment. Again, we first define edit operations and score functions before we state the alignment problem.

Definition 6 (Edit Operation over Several Score Functions). *An* edit operation over p score functions $\sigma_1, \ldots, \sigma_p$ *is a tuple*

$$(\alpha, \beta, i) \in (\mathcal{A} \cup \{\varepsilon\}) \times (\mathcal{A} \cup \{\varepsilon\}) \times \{1..p\} \setminus \{(\varepsilon, \varepsilon, j) \mid j \in \{1..p\}\}.$$

It is denoted by $\alpha \to_i \beta$*. Its score is* $\sigma_i(\alpha \to \beta)$*.*

Definition 7 (Alignment over Several Score Functions). *An* alignment A of two strings u and v over p score functions $\sigma_1, \ldots, \sigma_p$ *is a sequence* $(\alpha_1 \to_{i_1} \beta_1, \ldots, \alpha_h \to_{i_h} \beta_h)$ *of edit operations such that* $\alpha_1 \ldots \alpha_h = u$ *and* $\beta_1 \ldots \beta_h = v$*.*

An alignment over several score functions is displayed by placing the symbols of the two aligned strings in $p+1$ lines. For each edit operation $\alpha \to_i \beta$ the symbol α is placed in the first line and β in the $i+1$st line, where ε is replaced by - and the symbols of each edit operation are in the same column.

Example 2. The alignment $(d \to_2 \varepsilon, a \to_2 a, \varepsilon \to_2 i, r \to_1 r, l \to_1 l, i \to_1 i, n \to_1 n, g \to_2 e)$ is displayed as follows:

u: da-rling
v_1: rlin
v_2: -ai e

Definition 8 (Score of an Alignment over Several Score Functions). *The cost of switching between two score functions σ_1 and σ_2 is determined by a* switch cost function $\delta(\sigma_1 \to \sigma_2)$.

Using a given switch cost function δ and affine gap costs, the score of an alignment $A = (\alpha_1 \to_{i_1} \beta_1, \ldots, \alpha_h \to_{i_h} \beta_h)$ over $\sigma_1, \ldots, \sigma_p$ *is the sum of the scores of all edit operations, the gap open costs, and all switch costs:*

$$\sigma(A) = \sum_{k=1}^{h} \sigma_{i_k}(\alpha_k \to \beta_k) + \sum_{k=1}^{p} g_k \cdot \gamma_k + \sum_{k=1}^{h-1} \delta(\sigma_{i_{k-1}} \to \sigma_{i_k})$$

where g_k is the number of gap openings in A scored by σ_k and γ_k denotes the gap open cost for score function σ_k.

Note that according to this score function gap open costs are applied where the gap begins, although the score function may be switched within the gap.

Definition 9 (Local Alignment over Several Score Functions Problem). *Let two strings u and v, p score functions $\sigma_1, \ldots, \sigma_p$ and a switch cost function δ be given. The* local alignment over several score functions problem *is to determine an alignment A over $\sigma_1, \ldots, \sigma_p$ of u' and v' such that u' and v' are substrings of u and v and the score of A is maximal:*

$$\sigma(u, v) = \max \left\{ \sigma(A) \,\middle|\, \begin{array}{l} A \text{ is an alignment over } \sigma_1, \ldots, \sigma_p \text{ of } u' \text{ and } v', \\ u' \text{ is a substring of } u,\ v' \text{ is a substring of } v \end{array} \right\}.$$

Each alignment that satisfies this condition is called an optimal local alignment *of u and v over $\sigma_1, \ldots, \sigma_p$, and $\sigma(u, v)$ is called the* optimal local alignment score *of u and v over $\sigma_1, \ldots, \sigma_p$.*

2.3 Nonintersecting Alignments

In order to avoid redundancies, similar to Waterman and Eggert [14] we consider only nonintersecting alignments. Two alignments are nonintersecting if they share no replacement $u_i \to v_j$. More formally, we define:

Definition 10 (Projection). *Let two strings u and v, and substrings $u' = u_{b_u} \ldots u_{e_u}$ and $v' = v_{b_v} \ldots v_{e_v}$ be given. The* projection of an alignment $A = (\alpha_1 \to \beta_1, \ldots, \alpha_h \to \beta_h)$ *of u' and v' is*

$$\breve{A} = \{(b_u + |\alpha_1 \ldots \alpha_k| - 1, b_v + |\beta_1 \ldots \beta_k| - 1) \mid 1 \le k \le h, \alpha_k \ne \varepsilon \ne \beta_k\}.$$

The projection of an alignment over several score functions $A = (\alpha_1 \to_{i_1} \beta_1, \ldots, \alpha_h \to_{i_h} \beta_h)$ *of u' and v' is*

$$\breve{A} = \{(b_u + |\alpha_1 \ldots \alpha_k| - 1, b_v + |\beta_1 \ldots \beta_k| - 1, i_k) \mid 1 \le k \le h, \alpha_k \ne \varepsilon \ne \beta_k\}.$$

Definition 11 (Nonintersecting Alignments). *Two local alignments A_1 and A_2 of u and v over σ or over $\sigma_1, \ldots, \sigma_p$, respectively, are* nonintersecting *if and only if $\breve{A}_1 \cap \breve{A}_2 = \emptyset$.*

3 Algorithms

Various techniques have been developed to calculate optimal and suboptimal local alignments and have been improved to save resources. Huang and Miller [8] combined some of them to obtain an algorithm that calculates, for a given number K, the K best nonintersecting local alignments. In the following, we first sketch their algorithm, before we extend it to calculate the K best nonintersecting local alignments over p score functions.

3.1 Algorithm of Huang and Miller

Given two sequences u and v of lengths M and N, respectively, in the first phase of Huang and Miller's algorithm, a classical linear-space dynamic programming computation is performed to collect the K highest scores of local alignments of u and v in $O(M \cdot N)$ time and $O(M + N)$ space, together with the start and end position of each such alignment (see Algorithm 1). If some of these scores belong to intersecting alignments, only the highest score of mutually intersecting alignments is stored. Unfortunately, there is no guarantee that the K best scores found this way belong to the overall K best nonintersecting alignments, since some nonintersecting high-scoring alignments may be shaded by even higher scoring intersecting ones. Therefore, additional passes are required after a local alignment has been calculated.

More precisely, right after the first pass, using Hirschberg's [7] technique, the highest-scoring alignment (of length L_1) is computed in $O(L_1)$ space and $O(L_1^2)$ time by calculating a global alignment of the substrings determined by the start and end positions saved with the highest score. Thereby, the used replacements are recorded to be not used again.

Afterwards, the hidden high scoring alignments are discovered by a limited backwards dynamic programming pass determining the region of influence of the calculated alignment, and a forward pass to recompute the score matrix in this region. To determine the region of influence, some additional information is recorded. The possible local alignments are partitioned into equivalence classes. The K classes are stored in a data structure called LIST. An equivalence class S stored in LIST is represented by a tuple (C, F, u, T, B, L, R) where C is the score of the best alignment in S, F is the start position of all alignments in S, u is the end position of an alignment in S that gains score C, and $[T, B] \times [L, R]$ contains the end position of each alignment in S whose score is better than W, the lowest score of the K saved high scoring classes. The region of influence that needs to be recomputed is the part $[T', B] \times [L', R]$ of the score matrix that contains all entire alignments ending in $[T, B] \times [L, R]$ with a score greater than W.

An implementation of LIST must support the following operations:

- `find(f)`: returns the tuple whose $F =$ `f` or `null` if there is no such one.
- `insert(S)`: adds `S` to LIST.
- `maxtuple()`: removes a highest scoring tuples in LIST and returns it.
- `minscore()`: returns the lowest score (W) of all tuples in LIST.

Algorithm 1 Calculating alignment start position

The score for a local alignment ending at (i, j) is usually calculated by

1: $D(i,j) \leftarrow \max\{D(i-1,j), C(i-1,j)+\gamma\} + \sigma(u_i \rightarrow \varepsilon)$
2: $I(i,j) \leftarrow \max\{I(i,j-1), C(i,j-1)+\gamma\} + \sigma(\varepsilon \rightarrow v_j)$
3: $C(i,j) \leftarrow \max\{0, D(i,j), I(i,j), C(i-1,j-1)+\sigma(u_i \rightarrow v_j)\}$

To compute the start position the first line is refined to

1: **if** $D(i-1,j) > C(i-1,j)+\gamma$ **then**
2: $\quad D(i,j) \leftarrow D(i-1,j) + \sigma(u_i \rightarrow \varepsilon)$
3: $\quad \mathrm{StartD}(i,j) \leftarrow \mathrm{StartD}(i-1,j)$
4: **else if** $D(i-1,j) < C(i-1,j)+\gamma$ **then**
5: $\quad D(i,j) \leftarrow C(i-1,j)+\gamma+\sigma(u_i \rightarrow \varepsilon)$
6: $\quad \mathrm{StartD}(i,j) \leftarrow \mathrm{StartC}(i-1,j)$
7: **else** {*tie!*}
8: $\quad D(i,j) \leftarrow D(i-1,j) + \sigma(u_i \rightarrow \varepsilon)$
9: $\quad \mathrm{StartD}(i,j) \leftarrow \max_{\prec}\{\mathrm{StartD}(i-1,j), \mathrm{StartC}(i-1,j)\}$

$\mathrm{StartD}(i,j)$ denotes the start position of a highest scoring local alignment ending at (i,j) with a deletion. Similarily, StartI and StartC denote the start of an alignment ending with an insertion and replacement, respectively.
The other two lines are extended in the same way. If the maximum for $C(i,j)$ is 0, $\mathrm{StartC}(i,j) = (i,j)$.
In case of a tie the start position is chosen by an ordering $\prec$ of positions. This way, Huang and Miller showed that two alignments intersect if and only if they have the same start position.

- `replace(S)`: replaces a lowest scoring tuple in LIST by `S`.
- `size()`: returns the number of tuples in LIST.

The LIST is maintained by a function `enter` [8, Figure 3]: `enter(C,F,u,W,l)` first tests if there already is a class S in LIST with the same F. If there is one, its attributes are adjusted. Otherwise, a new class is added. If there are more than `l` classes in LIST, the class with the lowest score is deleted. `enter` returns the new minimum score W.

The steps – calculate alignment, determine its region of influence and search for hidden alignments – are repeated K times. If gaps and mismatches are not penalized too lightly by the used score function, Huang and Miller show that the algorithm takes $O(M \cdot N + \sum_{n=1}^{K} L_n^2)$ time and $O(M + N + \sum_{n=1}^{K} L_n)$ space in the expected case, where L_n is the length of nth reported alignment.

3.2 Extended Algorithm for Alignments over Several Score Functions

An outline of the extended algorithm that calculates the K best local alignments over several score functions $\sigma_1, \ldots, \sigma_p$ is shown in Algorithm 2. It differs from Huang and Miller's algorithm in additional **for** loops (lines 4 and 15) that iterate over the possible score functions, and in the calculations for the scores (lines 5 and 16), the alignment (line 10) and the region of influence (line 12). The function

Algorithm 2 Extension of Huang and Miller's alg. for several score functions

```
1: W ← 0
2: for i ← 0 to M do
3:    for j ← 0 to N do
4:       for r ← 1 to p do
5:          calculate C(i, j, r) and StartC(i, j, r).
6:          if C(i, j, r) > W then
7:             W ← enter(C(i, j, r), StartC(i, j, r), (i, j, r), W, K)
8: for n ← 1 to K do
9:    S ← maxtuple()
10:   alignment(S) {calculates and reports an optimal alignment for the equivalence
      class S that does not intersect with any already calculated alignment}
11:   if n ≠ K then
12:      calculate the region of influence [T', S.B] × [L', S.R].
13:      for i ← T' to S.B do
14:         for j ← L' to S.R do
15:            for r ← 1 to p do
16:               Calculate C(i, j, r) and StartC(i, j, r) relating to [T', S.B] × [L', S.R].
17:               if C(i, j, r) > W and (i, j) in [S.T, S.B] × [S.L, S.R] then
18:                  W ← enter(C(i, j, r), StartC(i, j, r), (i, j, r), W, K − n)
```

`enter` (lines 7 and 18) that maintains the LIST is almost unchanged. It is only adapted to consider the used score function as the third coordinate of start and end positions. Algorithm 3 shows exemplarily how the calculation of $C(i, j, r)$ and $\text{StartC}(i, j, r)$, of the alignment and of the region of interest is extended for several score functions.

The additional **for** loops (Algorithm 2, lines 4 and 15) and the nested loop (Algorithm 3, line 3) needed to compute the score, alignment and region of interest result in an additional factor of p^2 in the time complexity, yielding $O(p^2 \cdot (M \cdot N + \sum_{n=1}^{K} L_n^2))$ in the expected case. Regarding the space complexity, there is an additional factor of p for the intermediate results, yielding $O(p \cdot (M + N) + \sum_{n=1}^{K} L_n)$.

4 Proof of Concept

Now that we have presented our extension of the algorithm, we assess the impact of our modifications. Wasserman *et al.* [13] compiled a small test set of mammalian promoter regions where some binding sites had been verified experimentally. We retrieved 21 well annotated man-rodent sequence pairs from this set and investigated the effect of different parameter settings on the performance of the algorithm. We measured performance based on two factors: *sensitivity* in order to measure the ability of the method to recognize binding sites and *coverage* in order to measure the specificity of detected possible binding sites.

Definition 12 (Sensitivity). *The quotient of the number of found binding sites vs. the number of all annotated binding sites. We deem a binding site as found*

Algorithm 3 Extension of Algorithm 1 for several score functions

1: $D(i,j,r) \leftarrow \gamma_r$
2: $\text{StartD}(i,j,r) \leftarrow (i,j,r)$
3: **for** $rr \leftarrow 1$ **to** p **do**
4: **if** $C(i-1,j,rr) + \gamma_{rr} + \sigma(u_i \rightarrow_{rr} \varepsilon) + \delta(\sigma_{rr} \rightarrow \sigma_r) > D(i,j,r)$ **then**
5: $D(i,j,r) \leftarrow C(i-1,j,rr) + \gamma_{rr} + \sigma(u_i \rightarrow_{rr} \varepsilon) + \delta(\sigma_{rr} \rightarrow \sigma_r)$
6: $\text{StartD}(i,j,r) \leftarrow \text{StartC}(i-1,j,rr)$
7: **else if** $C(i-1,j,rr) + \gamma_{rr} + \sigma(u_i \rightarrow_{rr} \varepsilon) + \delta(\sigma_{rr} \rightarrow \sigma_r) = D(i,j,r)$ **then**
8: $\text{StartD}(i,j,r) \leftarrow \max_{\prec}\{\text{StartD}(i,j,r), \text{StartC}(i-1,j,rr)\}$
9: **if** $D(i-1,j,rr) + \sigma(u_i \rightarrow_{rr} \varepsilon) + \delta(\sigma_{rr} \rightarrow \sigma_r) > D(i,j,r)$ **then**
10: $D(i,j,r) \leftarrow D(i-1,j,rr) + \sigma(u_i \rightarrow_{rr} \varepsilon) + \delta(\sigma_{rr} \rightarrow \sigma_r)$
11: $\text{StartD}(i,j,r) \leftarrow \text{StartD}(i-1,j,rr)$
12: **else if** $D(i-1,j,rr) + \sigma(u_i \rightarrow_{rr} \varepsilon) + \delta(\sigma_{rr} \rightarrow \sigma_r) = D(i,j,r)$ **then**
13: $\text{StartD}(i,j,r) \leftarrow \max_{\prec}\{\text{StartD}(i,j,r), \text{StartD}(i-1,j,rr)\}$

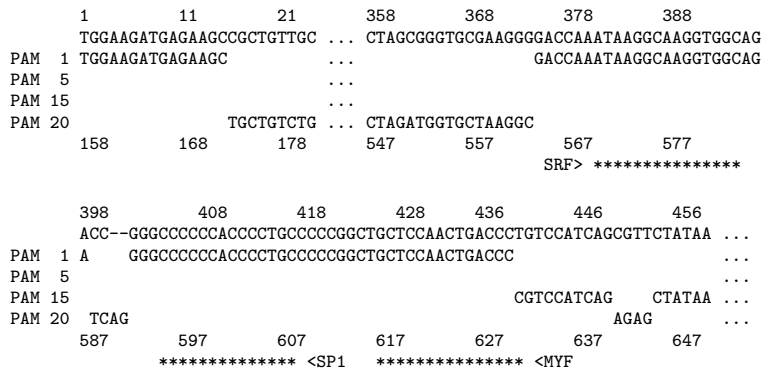
if at least 70% of the site (core region) are covered by an alignment in the lowest employed PAM distance.

Definition 13 (Coverage). *The length of all alignment parts in the lowest employed PAM distance divided by the arithmetic mean of the two sequence lengths.*

Our test scenario is as follows: The first 10 local alignments are computed for each sequence pair across all combinations of jump costs and scoring functions. Alignment gap open and extension costs are set to 11 and 0.1 times the match score, respectively (see [3]). All scoring matrices are derived from the *HKY model* of sequence evolution [6], which takes single nucleotide frequencies into account, and assume a transition to transversion ratio of 3 : 1. An enumeration of all parameter settings follows below:

1. **Score function sets L (all data in PAM):** $\{1,5,15,20\}$, $\{1,5,10,15,20\}$, $\{5,20,40,80\}$, $\{1,20\}$, $\{5,80\}$, $\{10,80\}$, $\{20,80\}$, $\{30,80\}$, and $\{40,80\}$.
2. **Switch cost factors F:** 1, 3, 5, 7, 9, 12, 16, 20, 24, and 99999 (no switch in score function within alignment). The switch cost function $\delta(\sigma_1 \rightarrow \sigma_2)$ is then given by $F \cdot |\sigma_1(\mathtt{A} \rightarrow \varepsilon) - \sigma_2(\mathtt{A} \rightarrow \varepsilon)|$.

Before the systematic evaluation here we present parts of an alignment in order to illustrate how the output of our algorithm may look like. Shown are the 5' untranslated regions of human cardiac actin gene (first row) and mouse alpha-cardiac actin gene (other rows). Switch cost factor is $F = 7$, and the set of score functions is $L = \{1,5,15,20\}$. The annotated binding sites (SRF, SP1, MYF), showing up nicely in the `PAM 1` row, are marked by asterisks in the last row:



4.1 Influence of Score Function Set

Figure 2 depicts how the performance of the algorithm is affected by the set of employed score functions. As expected, sets that include small PAM distances (1 and 5) perform generally better with respect to "specificity", whereas larger PAM distances are a bit more sensitive. Note that the variance along the x-axis is substantially larger than for the y-axis. This means that the choice of the scoring scheme mainly affects "specificity".

4.2 Influence of Switch Costs

Figure 3 demonstrates how switch costs that penalize jumping between different score functions alter the performance of the algorithm. Evidently, the coverage drastically increases if switch costs are low ($F \leq 3$) and alignments simply grow by alternating between score functions. Other than that, we could not observe any general trend with respect to sensitivity or specificity. The key data on the test set of 21 promoter pairs is shown in the following table:

Table 1 - maximal values		
sensitivity	coverage	sensitivity : coverage
87.3 %	**14.3 %**	**4.46**
F=99999; L=30.80	F=20; L=1.5.10.15.20	F=16; L=1.5.10.15.20

5 Discussion

The technique of sequence alignment is vital to bioinformatics. Sequence alignment is used in various fields for tasks as diverse as functional annotation, evolutionary parameter estimation and motif discovery. In this paper, we have presented a versatile algorithm for computing suboptimal local alignments over multiple score functions. Our implementation does not impose any constraints or prior assumptions on the position and segmentation of alignments. Consequently, two basic alignment problems are addressed by our solution: 1) Local alignments often show a "mosaic" structure (the alternation of regions of high

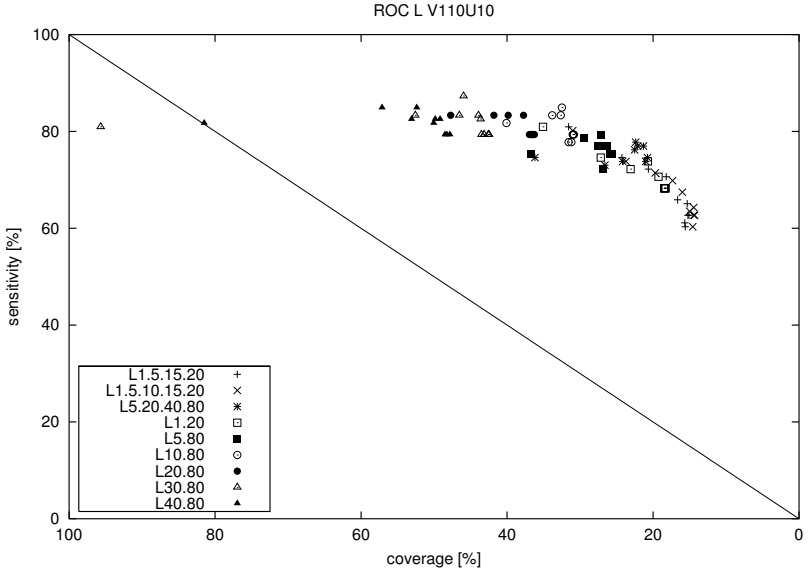


Fig. 2. Performance over all tested score function sets. For each set of PAM distances L with each switch cost factor F a point is plotted. Points with the same value of L are displayed as the same symbol. For a definition of the axis labels see Defs. 12 and 13.

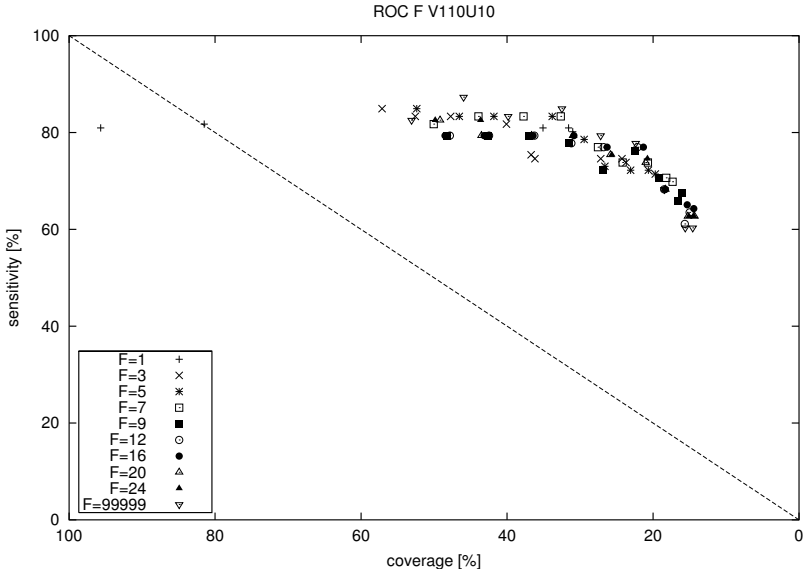


Fig. 3. Performance over all switch cost settings. The same graph as in Fig. 2, but points with the same value of F are displayed as the same symbol.

and low similarity, cf. [15]). We avoid this problem by aligning sequences with more than one scoring functions and thus capture the problem in a better way. 2) Often one does not know the proper scoring scheme in advance. This challenge is solved by employing a set of scoring schemes (e.g. for different evolutionary distances, protein domains, or secondary structures).

In this paper, the algorithm was exemplarily employed for the identification of transcription factor binding sites. For the test data, a 4.5-fold improvement of the sensitivity-to-coverage ratio was attained (see Table 1). This is just an estimation since we do not know how many binding sites escaped experimental validation so far.

References

1. S. F. Altschul. A protein alignment scoring system sensitive at all evolutionary distances. *J. Mol. Evol.*, 36:290–300, 1993.
2. P. Chain, S. Kurtz, E. Ohlebusch, and T. Slezak. An applications-focused review of comparative genomics tools: Capabilities, limitations and future challenges. *Briefings in Bioinformatics*, 4:105–123, 2003.
3. C. Dieterich, B. Cusack, H. Wang, K. Rateitschak, A. Krause, and M. Vingron. Annotating regulatory DNA based on man-mouse genomic comparison. *Bioinformatics*, 18(Suppl 2):S84–S90, 2002. (Proceedings of ECCB 2002).
4. L. Duret and P. Bucher. Searching for regulatory elements in human noncoding sequences. *Curr. Opin. Struct. Biol.*, 7:399–406, 1997.
5. R. C. Hardison. Conserved noncoding sequences are reliable guides to regulatory elements. *Trends Genet.*, 16:369–372, 2000.
6. M. Hasegawa, Y. Iida, T. Yano, F. Takaiwa, and M. Iwabuchi. Phylogenetic relationships among eukaryotic kingdoms inferred from ribosomal RNA sequences. *J. Mol. Evol.*, 22:32–38, 1985.
7. D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, 18:341–343, 1975.
8. X. Huang and W. Miller. A time-efficient, linear-space local similarity algorithm. *Adv. Appl. Math.*, 12:337–357, 1991.
9. W. Miller. Comparison of genomic DNA sequences: solved and unsolved problems. *Bioinformatics*, 17:391–397, 2001.
10. S. Schwartz, W. J. Kent, A. Smit, Z. Zhang, R. Baertsch, R. C. Hardison, D. Haussler, and W. Miller. Human-mouse alignments with BLASTZ. *Genome Res.*, 13:103–107, 2003.
11. T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, 147:195–197, 1981.
12. A. Ureta-Vidal, L. Ettwiller, and E. Birney. Comparative genomics: genome-wide analysis in metazoan eukaryotes. *Nat. Rev. Genet.*, 4:251–262, 2003.
13. W. W. Wasserman, M. Palumbo, W. Thompson, J. W. Fickett, and C. E. Lawrence. Human-mouse genome comparisons to locate regulatory sites. *Nature Genetics*, 26:225–228, 2000.
14. M. S. Waterman and M. Eggert. A new algorithm for best subsequence alignments with application to tRNA-rRNA comparisons. *J. Mol. Biol.*, 197:723–728, 1987.
15. Z. Zhang, P. Berman, T. Wiehe, and W. Miller. Post-processing long pairwise alignments. *Bioinformatics*, 15:1012–1019, 1999.