Alignments with Non-overlapping Moves, Inversions and Tandem Duplications in $O(n^4)$ Time

Christian Ledergerber and Christophe Dessimoz

ETH Zurich, Institute of Computational Science, Switzerland
ledergec@student.ethz.ch, cdessimoz@inf.ethz.ch

Abstract. Sequence alignment is a central problem in bioinformatics. The classical dynamic programming algorithm aligns two sequences by optimizing over possible insertions, deletions and substitution. However, other evolutionary events can be observed, such as inversions, tandem duplications or moves (transpositions). It has been established that the extension of the problem to move operations is NP-complete. Previous work has shown that an extension restricted to non-overlapping inversions can be solved in $O(n^3)$ with a restricted scoring scheme. In this paper, we show that the alignment problem extended to non-overlapping moves can be solved in $O(n^5)$ for general scoring schemes, $O(n^4 \log n)$ for concave scoring schemes and $O(n^4)$ for restricted scoring schemes. Furthermore, we show that the alignment problem extended to non-overlapping moves, inversions and tandem duplications can be solved with the same time complexities. Finally, an example of an alignment with non-overlapping moves is provided.

1 Introduction

In computational biology, alignments are usually performed to identify the characters that have common ancestry. More abstractly, alignments can also be represented as edit sequences that transform one sequence into the other under operations that model the evolutionary process. Hence, the problem of aligning two sequences is to find the most likely edit sequence, or equivalently, under an appropriate scoring scheme, the highest scoring edit sequence.

Historically, the only edit operations allowed were insertions, deletions and substitutions of characters, which we refer to as standard edit operations. The computation of the optimal alignment with respect to standard edit operations is well understood [1], and commonly used. But in some cases, standard edit operations are not sufficient to accurately model gene evolution. To take into account observed phenomena such as inversions, duplications or moves (intragenic transpositions) of blocks of sequences [2], the set of edit operations must be extended correspondingly. Such extensions have been studied in the past and a number of them turned out to be hard [3]. In particular an extension to general move operations was shown to be NP-complete [4]. For the simple greedy

G. Lin (Ed.): COCOON 2007, LNCS 4598, pp. 151–164, 2007.

algorithm presented in [4] it was shown by [5] that $O(n^{0.69})$ is an upper bound for the approximation factor. An efficient $O(\log^* n \log n)$ factor approximation algorithm for the general problem is presented in [6].

A number of results for the extension of the standard alignment including block inversions have been achieved. It is shown in [7] that sorting by reversals is NP-hard, but the complexity of alignment with inversions is still unknown. A restricted problem of non-overlapping inversions was proposed by [8] who found an $O(n^6)$ algorithm. This result was then further improved by [9,10,11,12] where [12] obtained an $O(n^3)$ algorithm for a restricted scoring scheme.

In this paper, we show that the alignment problem extended with non-overlapping moves and non-overlapping tandem duplications can be solved exactly in polynomial time, and provide algorithms with time complexity of $O(n^5)$ for general scoring schemes, $O(n^4 \log n)$ for concave scoring schemes and $O(n^4)$ for restricted scoring schemes.

Since the probability that k independent and uniformly distributed moves be non-overlapping decreases very rapidly[1], this restriction is only of practical interest for small k, that is, if such events are very rare. Convincing evidence that this is indeed the case can be found in [13]. They show that protein domain order is highly conserved during evolution. It is established in [13] that most domains cooccur with only zero, one or two domain families. Since a move operation of the more elaborate type such as $ABCD \rightarrow ACBD$ immediately implies that B cooccurs with three other domains, we conclude that move operations have to be rare. Furthermore, exon shuffling is highly correlated to domain shuffling [14,15,16] and hence cannot lead to a large amount of move operations. Finally, a number of move operations can be found in the literature [17,18]. As for tandem duplication events, articles on domain shuffling reveal that the most abundant block edit operations are tandem duplications where the duplicate stays next to the original [19,20].

In the next section, we present a rigorous definition of the two alignment problems solved here: an extension to non-overlapping moves and an extension to non-overlapping moves, inversions and tandem duplications. Then, we provide solutions to both problems. The last section presents the experimental results using the extension to non-overlapping moves.

2 Definition of the Problems and Preliminaries

2.1 Notation and Definitions

In the following, we will denote the two strings to be aligned with $S = s_1 \dots s_n$ and $T = t_1 \dots t_m$ where $|S| = n$ and $|T| = m$. The i-th character of S is $S[i]$ and $S[i..j] = s_{i+1} \dots s_j$ (note the indices). Thus, if $j \leq i$, $S[i..j] = \lambda$. By this definition, $S[i..j]$ and $S[j..k]$ are disjoint. $\overline{S} = s_n \dots s_1$ denotes the reverse of S and $\overline{S[i..j]} = \overline{S}[n-j..n-i]$ is the reverse of a substring, the substring of the reverse respectively (note the extension of the bar). Let us denote the score of

[1] For long sequences, this probability converges to $\frac{1}{(2k-1)!!} = \frac{1}{1 \cdot 3 \cdot 5 \cdots 2k-1}$.

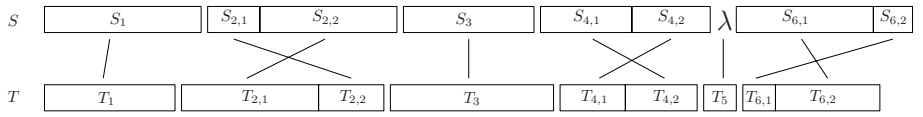

Fig. 1. Example of a non-overlapping move alignment of S with T

the standard alignment of S with T with $\delta(S,T)$. The score for substituting a character a with character b is denoted by an entry in the scoring matrix $\sigma(a,b)$. To simplify the definition of the alignment problems we introduce the concept of d-decompositions:

Definition 1. *Let a d-decomposition of a string S be a decomposition of S in d substrings such that the concatenation of all the substrings is equal to S. E.g. $S = S_1 \cdots S_d$. Let $\mathcal{M}_d(S)$ be the set of all d-decompositions of S.*

Note that S_i denotes a substring of a d-decomposition while s_i denotes a character. Let us further define the cyclic string to string comparison problem as introduced by [21]:

Definition 2. *The cyclic string comparison problem is to find the 2-decomposition $S_1S_2 \in \mathcal{M}_2(S)$ and $T_1T_2 \in \mathcal{M}_2(T)$ such that the score $\delta(S_1,T_2) + \delta(S_2,T_1)$ is maximal. The optimal score is denoted by $\delta_c(S,T)$.*

Due to the symmetry of the problem there exists always a two decomposition of $S = S_1S_2$ such that $\delta_c(S,T) = \delta(S_2S_1,T)$ as proven by [21].

Finally, we assume that the reader is familiar with the concept of edit graphs as defined for instance in [22] or [23].

2.2 Definition of Alignment with Non-overlapping Moves

Using d-decompositions and the cyclic string to string comparison problem we can now define the alignment with non-overlapping moves as follows.

Definition 3. *The problem of aligning S and T with non-overlapping moves is to find $d \in \mathbb{N}$ and d-decompositions of S and T such that the score $\sum_{i=1}^{d} \max\{\delta(S_i,T_i), \delta_c(S_i,T_i)+\sigma_c(l_{S_{i1}})+\sigma_c(l_{S_{i2}})+\sigma_c(l_{T_{i1}})+\sigma_c(l_{T_{i2}})\}$ is maximal for all $d \in \mathbb{N}, S_1 \ldots S_d \in \mathcal{M}_d(S)$ and $T_1 \ldots T_d \in \mathcal{M}_d(T)$. Where $l_{S_{i1}}, l_{S_{i2}}, l_{T_{i1}}$ and $l_{T_{i2}}$ are the lengths of the blocks involved in the move operation and $\sigma_c(l)$ is a penalty function for move operations. The optimal score is denoted by $\delta_m(S,T)$.*

Note that substrings S_i, T_i may be empty. However, a substring needs to have a length of at least 2 to contain a move. In other words, we align d pairs of substrings of S and T and allow for each aligned pair of substrings at most one swap of a prefix with a suffix as defined by the cyclic string comparison problem. $\sigma_c(l_{S_1})+\sigma_c(l_{S_2})+\sigma_c(l_{T_1})+\sigma_c(l_{T_2})$ is a penalty function for such a move operation and depends on the lengths of the four substrings involved in the move operation.

This decomposition in a sum will be required in the algorithm. An example of a non-overlapping move alignment is shown in Fig. 1. We now introduce different scoring schemes that will influence the time complexity of the results.

Definition 4. General scoring scheme: *the standard alignment of substrings is done with affine gap penalties, $\sigma_c(l)$ is an arbitrary function and the scoring matrix $\sigma(a,b)$ is arbitrary.* Concave scoring scheme: *the standard alignment of substrings is done with constant indel penalties, $\sigma_c(l)$ is a concave function and the scoring matrix $\sigma(a,b)$ is arbitrary.* Restricted scoring scheme: *the standard alignment of substrings is done with constant indel penalties and $\sigma_c(l)$ is a constant. The scoring matrix $\sigma(a,b)$ is selected such that the number of distinct values of $DIST[i,j] - DIST[i,j-1]$ is bounded by a constant ψ. For more details on the restricted scoring scheme, we refer to [24].*

2.3 Definition of Alignment with Non-overlapping Moves, Inversions and Tandem-Duplications

In favor of simplicity, we assume constant indel penalties and constant penalties for block operations in the treatment of this problem. However, the scoring schemes of section 2.2 could be used here as well.

Definition 5. *The problem of aligning S and T with non-overlapping moves, reversals and tandem duplications is to find $d \in \mathbb{N}$ and d-decompositions of S and T such that the score $\sum_{i=1}^{d} \max\{\delta(S_i,T_i), \delta_c(S_i,T_i)+\sigma_c, \delta_d(S_i,T_i)+\sigma_d, \delta_r(S_i,T_i)+\sigma_r\}$ is maximal for all $d \in \mathbb{N}, S_1 \dots S_d \in \mathcal{M}_d(S)$ and $T_1 \dots T_d \in \mathcal{M}_d(T)$, where $\delta_d(A,B) = \max\{\delta(AA,B), \delta(A,BB)\}$ and $\delta_r(A,B) = \delta(A,\overline{B})$. Where $\sigma_c, \sigma_d, \sigma_r$ are penalties for move operations, duplications or reversals respectively. The optimal score is denoted by $\delta_{drm}(S,T)$.*

2.4 Other Preliminaries

The notion of $DIST[i,j]$ arrays as used in [24,25] can be defined as follows.

Definition 6. *Let $DIST_{S,T}[i,j], 0 \le i \le j \le m$ denote the score of the optimal alignment of $T[i..j]$ with S.*

Let us further introduce input vectors I, output vectors O and a matrix OUT.

Definition 7. *Let $OUT_{S,T}[i,j] = I[i] + DIST_{S,T}[i,j] + \sigma_c(j-i), 0 \le i \le j \le m$. Then I is an arbitrary vector called input vector and $O[j] = \max_i OUT[i,j]$ is called output vector containing all the column maxima of OUT.*

Lemma 1. *$DIST_{S,T}[i,j]$ arrays are inverse monge arrays.*

The following lemma will become useful in the selection of the parameters.

Lemma 2. *If $f(l)$ is concave then $f_l(j',j) := f(j-j'), 0 \le j' \le m, 0 \le j \le m$ is inverse Monge.*

A proof of these lemmas can be found with the definition of inverse Monge in the *Appendix*.

Corollary 1. *$OUT_{S,T}[i,j] = DIST_{S,T}[i,j] + f(j-i) + I[i]$ is inverse Monge for f concave and constant indel penalties.*

Proof. Due to lemma 1 $DIST_{S,T}$ arrays with constant indel penalties are inverse Monge. The rest follows from definition 8 and lemma 2 (in *Appendix*).

Using our observations and the results from [24,25], we can conclude with the following results: (i) For arbitrary penalty functions σ_c and affine gap penalties as in the general scoring scheme, we can compute $DIST_{S[0..l],T}$ from $DIST_{S[0..l-1],T}$ in $O(m^2)$ as indicated in the *Appendix*. Then we can trivially compute the output vector O as in definition 7 in $O(m^2)$ time by inspecting all entries. (ii) For concave functions σ_c and constant indel penalties as in the concave scoring scheme, we can compute a representation of $DIST_{S[0..l],T}$ from $DIST_{S[0..l-1],T}$ in $O(m \log m)$ time using the data structure of [25]. Then since OUT is inverse Monge, we can compute the output vector O by applying the algorithm of [26] for searching all column maxima in a Monge array to OUT. This algorithm will access $O(m)$ entries of the array and hence the computation of O will take $O(m \log m)$ time since we can access an entry of $DIST$ in the data structure of [25] in $O(\log m)$ time. (iii) For constant functions σ_c, constant indel penalties and a restricted scoring matrix as in the restricted scoring scheme, we can compute a representation of $DIST_{S[0..l],T}$ from $DIST_{S[0..l-1],T}$ in $O(m)$ time due to section 6 of [25] and then compute the output vector O using the algorithm of [24] in $O(m)$ time.

Note that the $O(m \log m)$ and $O(m)$ results rely heavily on the fact that $DIST$ arrays are Monge. Since this is not true for affine gap penalties these results cannot be easily extended to affine gap penalties.

3 Algorithms

3.1 Alignment with Non-overlapping Moves

Let $SCO_{S,T}[i,j]$ be the score of the optimal alignment of $S[0..i]$ and $T[0..j]$ with non-overlapping moves. Then the following recurrence relation and initialization of the table will lead to a dynamic programming solution for the problem.

$$\text{Base Case: } SCO_{S,T}[i,0] = i \cdot \sigma_I \text{ and } SCO_{S,T}[0,j] = j \cdot \sigma_I$$

$$\text{Recurrence: } SCO_{S,T}[i,j] = \max \begin{cases} SCO_{S,T}[i,j-1] + \sigma_I \\ SCO_{S,T}[i-1,j-1] + \sigma(S[i],T[j]) \\ SCO_{S,T}[i-1,j] + \sigma_I \\ MOVE \end{cases}$$

where

$$MOVE = \max_{0 \le i' < i, 0 \le j' < j} \{ \quad SCO_{S,T}[i', j'] + \delta_c(S[i'..i], T[j'..j]) + \sigma_c(l_{S_{d1}}) + \sigma_c(l_{S_{d2}}) + \sigma_c(l_{T_{d1}}) + \sigma_c(l_{T_{d2}}) \}$$

Proof. Let us consider an optimal non-overlapping move alignment of $S[0..i]$ with $T[0..j]$. Let S_d and T_d be the last substrings of the optimal d-composition of $S[0..i]$ and $T[0..j]$. Then there are two cases: (1) S_d and T_d are aligned using the cyclic string comparison or (2) S_d and T_d are aligned by the standard alignment. In case (1), we know that $SCO_{S,T}[i, j] = SCO_{S,T}[i', j'] + \delta_c(S_d, T_d)$ which is considered in $MOVE$. In case (2), we are in the usual standard alignment cases. Hence, we consider all the cases and therefore find the optimal solution.

With the goal of economizing the computation of the table let us rewrite $MOVE$ as

$$\max_{\substack{0 \le i' < i'' < i \\ 0 \le j' < j'' < j}} \{SCO_{S,T}[i', j'] + DIST_{S[i''..i],T}[j', j''] + \sigma_c(j'' - j') + \sigma_c(i - i'') + DIST_{S[i'..i''],T}[j'', j] + \sigma_c(j - j'') + \sigma_c(i'' - i')\}.$$

To compute $MOVE$ for a given i' and i'' we can first maximize over j' and then over j''. That is, we can first compute the output row of the first $DIST_{S[i''..i],T}$ array and then, given that output, compute the output of the second $DIST_{S[i'..i''],T}$ array. This leads to the following definitions (illustrated in Fig. 2).

$$O_1[j''] = \max_{0 \le j' < j''} SCO_{S,T}[i', j'] + DIST_{S[i''..i],T}[j', j''] + \sigma_c(j'' - j') + \sigma_c(i - i'') \quad (1)$$

$$O_2[j] = \max_{0 \le j'' < j} O_1[j''] + DIST_{S[i'..i''],T}[j'', j] + \sigma_c(j - j'') + \sigma_c(i'' - i') \quad (2)$$

Given $DIST_{S[i''..i],T}[j', j'']$ and $DIST_{S[i'..i''],T}[j'', j]$, $O_1[j'']$ and $O_2[j]$ can be computed efficiently using the results from section 2.4 since both of them are output vectors as in definition 7.

DP_MOVE

for all i, j **such that** $0 \le i \le n, 0 \le j \le m$ **do**
  {base case}
  $SCO[i, 0] := i \cdot \sigma_I$
  $SCO[0, j] := j \cdot \sigma_I$
  $SCO[i, j] := -\infty$ if $i \ne 0, j \ne 0$
end for
for i **from** 0 **to** n **do**
  if $i \ge 1$ **then**
    for j **from** 1 **to** m **do**
      {standard alignment recurrence}
      $SCO[i, j] := \max\{SCO[i, j], SCO[i - 1, j] + \sigma_I, SCO[i, j - 1] + \sigma_I, SCO[i - 1, j - 1] + \sigma(S[i], T[j])\}$

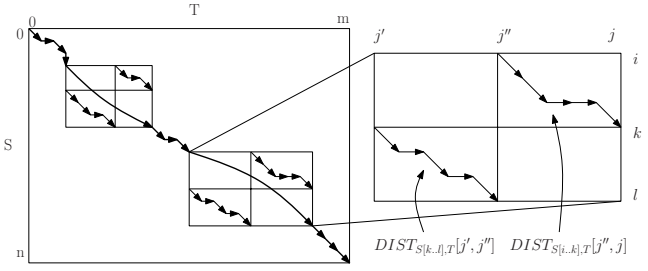

Fig. 2. An illustration of the computation of a move operation in DP_MOVE. Since the scores are additive: $SCO[l, j] = SCO[i, j'] + DIST_{S[k..l],T}[j', j''] + DIST_{S[i..k],T}[j'', j]$. In DP_MOVE this is maximized for all $i < k < l, j' < j'' < j$.

```
12:      end for
13:    end if
14:    for k from i to n do
15:      {move operations}
16:      DIST_{S[i..k],T} := calcDist(DIST_{S[i..k−1],T})
17:      for l from k to n do
18:        DIST_{S[k..l],T} := calcDist(DIST_{S[k..l−1],T})
19:        O_1 := calcOutput(OUT[j', j''] = SCO[i, j'] + DIST_{S[k..l],T}[j', j''] +
           σ_c(j'' − j') + σ_c(l − k))
20:        O_2 := calcOutput(OUT[j'', j] = O_1[j''] + DIST_{S[i..k]}[j'', j] + σ_c(j −
           j'') + σ_c(k − i))
21:        for j from 0 to m do
22:          SCO[l, j] := max{SCO[l, j], O_2[j]}
23:        end for
24:      end for
25:    end for
26:  end for
```

Where $calcDist(DIST_{S[0..l-1],T})$ computes $DIST_{S[0..l],T}$ from $DIST_{S[0..l-1],T}$ and $calcOutput(OUT[i, j])$ computes O as in definition 7.

Correctness. To show the correctness of the algorithm it suffices to show that we process all edges in the edit graph and whenever we process an edge $(u, v) \in E$ we have completed the computation of the score of u and any of its predecessors in topological order [22]. The computation of the score of a node u is completed iff all the incoming edges of u have been processed. This can be proven by induction. In our edit graph, the only edges are either due to the standard alignment, or due to move operations, as can be seen in the recurrence. Assuming that when computing the i-th row of SCO, all edges due to move operations starting in a row $i' < i$ have already been processed and the computation of any node (i', j) with $i' < i$ has been completed, we can see that the processing of the edges due to the standard alignment recurrence ending in the i-th row as done on line 9 to 12 is legitimate. After having processed those edges, we have completed

the computation of all edges ending on any node in the i-th row and hence can compute any edge due to move operations starting on that row which is done on line 14 to 24. We compute all such edges. Consequently, when we advance to the computation of row $i+1$ the assumption is again true.

Using the results from section 2.4 we can analyze the runtime of the algorithm and conclude with the following theorem.

Theorem 1. *The problem of aligning S and T, $|S| = n, |T| = m$, with non-overlapping moves can be solved in $O(n^3m^2)$ time and $O(nm + m^2)$ space for* general scoring schemes*, in $O(n^3 m \log m)$ time and $O(nm + m^2)$ space for* concave scoring schemes *and in $O(n^3m)$ time and $O(nm)$ space for* restricted scoring schemes.

3.2 Alignment with Non-overlapping Moves, Inversions, and Tandem Duplications

The dynamic programming recurrence of non-overlapping move operations extends nicely to this problem.

$$\text{Base Case: } SCO_{S,T}[i,0] = i \cdot \sigma_I \text{ and } SCO_{S,T}[0,j] = j \cdot \sigma_I$$

$$\text{Recurrence: } SCO_{S,T}[i,j] = \max \begin{cases} SCO_{S,T}[i,j-1] + \sigma_I \\ SCO_{S,T}[i-1,j-1] + \sigma(S[i],T[j]) \\ SCO_{S,T}[i-1,j] + \sigma_I \\ MOVE + \sigma_c \\ S\_DUPLICATE + \sigma_d \\ T\_DUPLICATE + \sigma_d \\ REVERSE + \sigma_r \end{cases}$$

where

$$\begin{aligned} MOVE &= \max_{0 \le i' < i, 0 \le j' < j} \{SCO_{S,T}[i',j'] + \delta_c(S[i'..i], T[j'..j])\} \\ S\_DUPLICATE &= \max_{0 \le i' < i, 0 \le j' < j'' < j} \{SCO_{S,T}[i',j'] + \delta(S[i'..i], T[j'..j'']) \\ &\quad + \delta(S[i'..i], T[j''..j]\} \\ T\_DUPLICATE &= \max_{0 \le i' < i'' < i, 0 \le j' < j} \{SCO_{S,T}[i',j'] + \delta(S[i'..i''], T[j'..j]) \\ &\quad + \delta(S[i''..i], T[j'..j]\} \\ REVERSE &= \max_{0 \le i' < i, 0 \le j' < j} \{SCO_{S,T}[i',j'] + \delta(\overline{S[i'..i]}, T[j'..j])\} \end{aligned}$$

A proof of this recurrence is analogous to the proof for non-overlapping moves and is omitted.

We have split tandem duplication into tandem duplication of a substring of S and tandem duplication of a substring of T. We have already shown how $MOVE$ can be treated and in [11] it is shown how to handle $REVERSE$. $S\_DUPLICATE$ can be done as follows. We calculate $DIST_{S[i..k]}$. Then, we

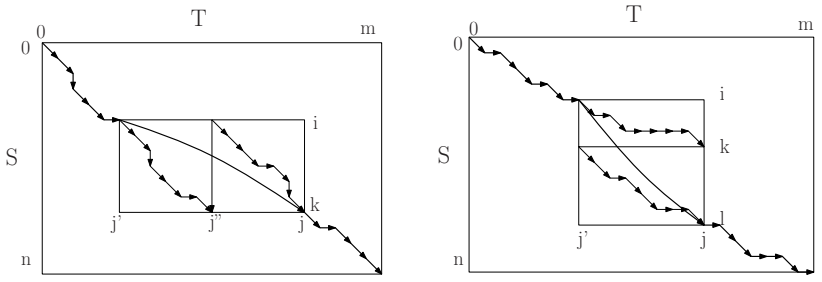

Fig. 3. An illustration on how duplications are treated

first use $SCO_{S,T}[i, j']$ as input vector for $DIST_{S[i..k]}$ to get $O_1[j']$ and then use $O_1[j']$ as input for $DIST_{S[i..k]}$ to get $O_2[j]$. $T\_DUPLICATE$ can be computed by computing the output vector of $DIST_{S[i..k],T}[j', j] + DIST_{S[k..l],T}[j', j]$ for the input vector $SCO_{S,T}[i, j']$. Note, that this array is well defined and is again inverse Monge because it is a sum of two inverse Monge arrays. Using these observations which are illustrated in Fig. 3 we can now present our algorithm for this problem.

DP_MOVE_INV_DUPL

1: {initialize the table as in DP_MOVE}
2: **for** i **from** 0 **to** n **do**
3: {compute REVERSE as done in [12]}
4: {compute the standard alignment recurrence as in DP_MOVE}
5: {treat MOVE as done in DP_MOVE}
6: **for** k **from** i **to** n **do**
7: {duplication}
8: $DIST_{S[i..k],T} := calcDist(DIST_{S[i..k-1],T})$
9: $O_1 := calcOutput(OUT[j', j''] = SCO[i, j'] + DIST_{S[i..k],T}[j', j''])$
10: $O_2 := calcOutput(OUT[j'', j] = O_1[j''] + DIST_{S[i..k],T}[j'', j])$
11: **for** l **from** k **to** n **do**
12: $DIST_{S[k..l],T} := calcDist(DIST_{S[k..l-1],T})$
13: $O := calcOutput(OUT[j', j] = SCO[i, j'] + DIST_{S[i..k],T}[j', j] + DIST_{S[k..l],T}[j', j])$
14: **for** j **from** 0 **to** m **do**
15: $SCO[l, j] := \max\{SCO[l, j], O[j] + \sigma_d, O_2[j] + \sigma_d\}$
16: **end for**
17: **end for**
18: **end for**
19: **end for**

A proof of the correctness of the algorithm is analogous to the proof for DP_MOVE. This proof however reveals that it is important to process edges due to REVERSE *before* the standard alignment recurrence.

Theorem 2. *The problem of aligning S and T with non-overlapping moves, reversals and tandem duplications can be solved in $O(n^3m^2)$ time and $O(nm + m^2)$ space for* general scoring schemes, *in $O(n^3m \log m)$ time and $O(mn + m^2)$ space for* concave scoring schemes *and in $O(n^3m)$ time and $O(nm)$ space for* restricted scoring schemes, *where $n = |S|$ and $m = |T|$.*

4 Implementation and Experiments

We have implemented an $O(n^5)$ version of the algorithm with constant gap penalties in C[2]. This implementation has proven useful for aligning sequences of up to about 400 AA, taking a few hours to compute the alignment. We have tested the algorithm on real data and were able to confirm a number of examples found in [2]. In addition we run the algorithm on an example found in [18]. This alignment is shown in figure 4 and is compared with a standard alignment obtained from Darwin [27].

```
DP_MOVE
Seq1: RPSTVPLP_NTQ__A_LAMA_[GTAYKGYVKVP_KPTGVK_KGWQRAYAVVCDCKLFLYDLPEGK_STQPGVIASQVLDLRDDEFAVSSVLA
Seq2: LSSADNDPEDSQHSSLLSLTQ[DSVFEGWLSVPNKQNRRRGHGWKRQYVIVSSRKIIFYNSDIDKHNTTDAVL___ILDL_SKVYHVRSVTQ

Seq1: SDVIHATRRDIPCIFRVT_ASLLG_S__PSKTSSL_L_ILTENENEKRK|GP_KPKAHQF_SIKSFPSPTQCSHCTSLMVGLIR__QGYACE
Seq2: GDVIRADAKEIPRIFQLLYAGE_GASHRPDEQSQLDVSVLHGNCNEERP|GTIVHKGHEFVHI_TYHMPTACEVCPKPLWHMFKPPAAYECK

Seq1: VCAFSCHVSCKDS_APQV_CPIPPE_QSKRP___LGVDVQ_RGI]WVGILEGLQAILHKNRLRSQVV_HVAQEAYD_S_SLPLI
Seq2: RCRNKIHKEHVDKHDPLAPCKLNHDPRSARDMLLLAATPEDQSL]WVARL__LKRI_QKSGYKAASYNNNSTDGSKISPSQSTR

DARWIN
Seq1: RPSTVPLPNTQALAMAGPKPKAHQFSIKSFPSPTQCSHCTSLMVGLIRQGYACEVCAFSCHVSCKDSAPQVCPIPPEQSKRPLGVDVQRGIG
Seq2: _____LSSADNDPEDSQHS__SLLSLTQ_____________________________________________________________D

Seq1: TAYKGYVKVPKPTGVKK__GWQRAYAVVCDCKLFLY__DLPEGKSTQPGVIASQVLDLRDDEFAVSSVLASDVIHATRRDIPCIFRV_____
Seq2: SVFEGWLSVPNKQNRRRGHGWKRQYVIVSSRKIIFYNSDIDKHNTTD____AVLILDL_SKVYHVRSVTQGDVIRADAKEIPRIFQLLYAGE

Seq1: _____________________________________________________________________________TASLLGS
Seq2: GASHRPDEQSQLDVSVLHGNCNEERPGTIVHKGHEFVHITYHMPTACEVCPKPLWHMFKPPAAYECKRCRNKIHKEHVDKHDPLAPCKLNHD

Seq1: PSKTSSLLILTENENEKRKWVGIL______EGLQAILHKNRLRSQVVHVAQEAYDSSLPLI
Seq2: PRSARDMLLLAATPEDQSLWVARLLKRIQKSGYKAASYNNN______STDGSKISPSQSTR
```

Fig. 4. An example found in [18]. In this figure we compare an alignment computed with our new algorithm and an alignment done with Darwin [27]. The brackets '[' and ']' indicate the boundary of the substrings containing a move operation and '|' marks the position of the split. Seq1: Q7TT49 AA 1005-1241; Seq2: Q9VXE3 AA 1112-1367. Using the annotation of SMART we have marked the domains involved in the move operation. Pleckstrin homology phospholipid binding domain is shown in blue, protein kinase C-type diacylglycerol binding domain is shown in yellow.

5 Conclusions

In this paper, we have presented a number of new alignment problems extending the notion of non-overlapping inversions to non-overlapping moves and tandem duplications. For all of them we found algorithms that solve the problems exactly and can be implemented to run in $O(n^2)$ space and $O(n^5)$, $O(n^4 \log n)$ or $O(n^4)$

[2] Available from the authors upon request.

time depending on the scoring scheme used. We believe that this approach may yield new insights by finding the best alignment of two sequences, and think that it is justifiable due to the rarity of such events in nature. Using the implementation of the $O(n^5)$ variant of the algorithm, we were able to align previously identified cases of pairs of sequences with move operations. Furthermore, these experiments also showed the necessity of an $O(n^4 \log n)$ implementation to be applicable to large sequences, which are more likely to contain a move.

Acknowledgments

We thank Manuel Gil, Gaston H. Gonnet, Gina M. Cannarozzi and three anonymous referees for helpful critiques and discussions.

References

1. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. 48(3), 443–453 (1970)
2. Fliess, A., Motro, B., Unger, R.: Swaps in protein sequences. Proteins. 48(2), 377–387 (2002)
3. Lopresti, D., Tomkins, A.: Block edit models for approximate string matching. Theor. Comput. Sci. 181(1), 159–179 (1997)
4. Shapira, D., Storer, J.A.: Edit distance with move operations. In: Apostolico, A., Takeda, M. (eds.) CPM 2002. LNCS, vol. 2373, pp. 85–98. Springer, Heidelberg (2002)
5. Chrobak, M., Kolman, P., Sgall, J.: The greedy algorithm for the minimum common string partition problem. ACM Trans. Algorithms 1(2), 350–366 (2005)
6. Cormode, G., Muthukrishnan, S.: The string edit distance matching problem with moves. In: SODA '02. Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA. Society for Industrial and Applied Mathematics, pp. 667–676. ACM Press, New York (2002)
7. Caprara, A.: Sorting by reversals is difficult. In: RECOMB '97. Proceedings of the first annual international conference on Computational molecular biology, pp. 75–83. ACM Press, New York (1997)
8. Schoeninger, M., Waterman, M.S.: A local algorithm for dna sequence alignment with inversions. Bull. Math. Biol. 54(4), 521–536 (1992)
9. Chen, Z.Z., Gao, Y., Lin, G., Niewiadomski, R., Wang, Y., Wu, J.: A space-efficient algorithm for sequence alignment with inversions and reversals. Theor. Comput. Sci. 325(3), 361–372 (2004)
10. do Lago, A.P., Muchnik, I.: A sparse dynamic programming algorithm for alignment with non-overlapping inversions. Theoret. Informatics Appl. 39(1), 175–189 (2005)
11. Alves, C.E.R., do Lago, A.P., Vellozo, A.F.: Alignment with non-overlapping inversions in $o(n^3 \log n)$ time. In: Proceedings of GRACO 2005. Electronic Notes in Discrete Mathematics, vol. 19, pp. 365–371. Elsevier, Amsterdam (2005)
12. Vellozo, A.F., Alves, C.E.R., do Lago, A.P.: Alignment with non-overlapping inversions in $o(n^3)$-time. In: Bücher, P., Moret, B.M.E. (eds.) WABI 2006. LNCS (LNBI), vol. 4175, pp. 186–196. Springer, Heidelberg (2006)

13. Apic, G., Gough, J., Teichmann, S.A.: Domain combinations in archaeal, eubacterial and eukaryotic proteomes. J. Mol. Biol. 310(2), 311–325 (2001)
14. Kaessmann, H., Zöllner, S., Nekrutenko, A., Li, W.H.: Signatures of domain shuffling in the human genome. Genome Res. 12(11), 1642–1650 (2002)
15. Liu, M., Walch, H., Wu, S., Grigoriev, A.: Significant expansion of exon-bordering protein domains during animal proteome evolution. Nucleic Acids Res. 33(1), 95–105 (2005)
16. Vibranovski, M.D., Sakabe, N.J., de Oliveira, R.S., de Souza, S.J.: Signs of ancient and modern exon-shuffling are correlated to the distribution of ancient and modern domains along proteins. J. Mol. Evol. 61(3), 341–350 (2005)
17. Bashton, M., Chothia, C.: The geometry of domain combination in proteins. J. Mol. Biol. 315(4), 927–939 (2002)
18. Shandala, T., Gregory, S.L., Dalton, H.E., Smallhorn, M., Saint, R.: Citron kinase is an essential effector of the pbl-activated rho signalling pathway in drosophila melanogaster. Development. 131(20), 5053–5063 (2004)
19. Andrade, M.A., Perez-Iratxeta, C., Ponting, C.P.: Protein repeats: structures, functions, and evolution. J. Struct. Biol. 134(2-3), 117–131 (2001)
20. Marcotte, E.M., Pellegrini, M., Yeates, T.O., Eisenberg, D.: A census of protein repeats. J. Mol. Biol. 293(1), 151–160 (1999)
21. Maes, M.: On a cyclic string-to-string correction problem. Inf. Process. Lett. 35(2), 73–78 (1990)
22. Myers, E.W.: An overview of sequence comparison algorithms in molecular biology. Technical Report 91-29, Univ. of Arizona, Dept. of Computer Science (1991)
23. Gusfield, D.: Algorithms on Strings, Trees, and Sequences: computer science and computational biology. Press Syndicate of the University of Cambridge, Cambridge (1997/1999)
24. Landau, G.M., Ziv-Ukelson, M.: On the common substring alignment problem. J. Algorithms 41(2), 338–354 (2001)
25. Schmidt, J.P.: All highest scoring paths in weighted grid graphs and their application to finding all approximate repeats in strings. SIAM J. Comput. 27(4), 972–992 (1998)
26. Aggarwal, A., Klawe, M.M., Moran, S., Shor, P., Wilber, R.: Geometric applications of a matrix-searching algorithm. Algorithmica 2(1), 195–208 (1987)
27. Gonnet, G.H., Hallett, M.T., Korostensky, C., Bernardin, L.: Darwin v. 2.0: An interpreted computer language for the biosciences. Bioinformatics 16(2), 101–103 (2000)
28. Monge, G.: Déblai et remblai. Mémoires de l'Académie Royale des Sciences (1781)

Appendix

Monge Properties

For a proof of lemma 1 and lemma 2 we have to define the notion of Monge arrays [28] first:

Definition 8. *A matrix* $M[0 \ldots n; 0 \ldots m]$ *is Monge if for all* $i = 1 \ldots n, j = 1 \ldots m$

$$M[i,j] + M[i-1,j-1] \leq M[i-1,j] + M[i,j-1]$$

and it is called inverse Monge if for all $i = 1 \ldots n, j = 1 \ldots m$

$$M[i,j] + M[i-1,j-1] \geq M[i-1,j] + M[i,j-1]$$

Then the proof for lemma 2 goes as follows.

Proof

$$\begin{aligned} f_l(j'-1,j-1)+f_l(j',j) &= 2f(j-j') \\ &\geq 2\frac{f(j-j'-1)+f(j-j'+1)}{2} \\ &= f_l(j'-1,j)+f_l(j',j-1) \end{aligned}$$

Where the inequality follows from the definition of concave. A function f is concave iff $f(tx+(1-t)y) \geq tf(x)+(1-t)f(y)$ holds for all $x, y \in \mathbb{R}, t \in [0,1]$. In other words every point on a secant is below the function. In the proof we used $t = 1/2$ as shown in Fig. 5.

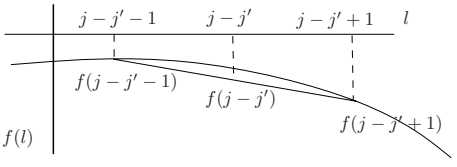

Fig. 5. For concave functions any point on a secant is below the function

Note that if any three points $f(j-1), f(j), f(j+1), 0 < j < m$ are not in concave position the resulting array will not be inverse Monge. That is, lemma 2 holds with equivalence if we restrict the definition of concave to values in $\{0 \ldots m+1\} \subseteq \mathbb{N}$.

The proof for Lemma 1 is analogous to the proof in [25].

Proof The paths represented by $DIST_{S,T}[i-1,j]$ and $DIST_{S,T}[i,j-1]$ have to cross properly in a vertex v as shown in figure 6. Therefore, we have

$$\begin{aligned} DIST_{S,T}[i-1,j]+DIST_{S,T}[i,j-1] &= (a+b)+(c+d) \\ &= (a+d)+(b+c) \\ &\leq g+f \\ &= DIST_{S,T}[i,j]+DIST_{S,T}[i-1,j-1] \end{aligned}$$

where the inequality follows from $a+d \leq f$ and $b+c \leq g$. Where $a+d \leq f$ holds since $a+d$ is the length of a path from $(0,i-1)$ to $(n,j-1)$ and the optimal path of length f can only be longer and analogously $b+c \leq g$.

In figure 6 2) a counter example for affine gap penalties is given. This is a counter example because the total number of gaps on the paths from $(0,i)$ to $(n,j-1)$ and $(0,i-1)$ to (n,j) is smaller than the total number of gaps on the paths from $(0,i)$ to (n,j) and $(0,i-1)$ to $(n,j-1)$. Hence $DIST_{S,T}$ arrays cannot be monge for large initial penalties.

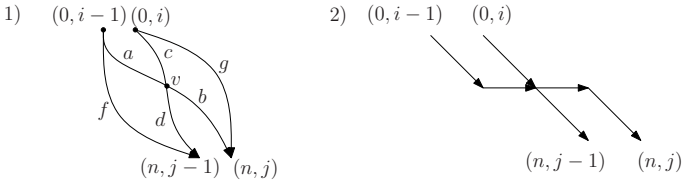

Fig. 6. 1) illustrates that the path from vertex $(0, i)$ to $(n, j-1)$ and the path from vertex $(0, i-1)$ to (n, j) in the grid graph have to cross in a common vertex v. 2) gives a counter example for affine gap penalties.

Extension of $DIST$ Arrays

This simple algorithm is inspired by [25]. It is repeated here to provide an idea on how to extend our algorithms to affine gap penalties.

For the base cases we observe that $DIST_{S,T}[i,j] = B_{S,T[i..j]}[n, j-i]$ in particular for constant indel penalties $DIST_{S[0..0],T}[i,j] = (j-i) \cdot \sigma_I$ and $DIST_{S[0..l],T}[i,i] = l \cdot \sigma_I$.

By mapping the standard alignment recurrence to $DIST$ arrays we obtain:

$$DIST_{S[0..l],T}[i,j] = \max \begin{cases} DIST_{S[0..l-1],T}[i,j] + \sigma_I \\ DIST_{S[0..l-1],T}[i,j-1] + \sigma(S[l], T[j]) \\ DIST_{S[0..l],T}[i,j-1] + \sigma_I \end{cases}$$

Therefore, we can compute $DIST_{S[0..l],T}$ given $DIST_{S[0..l-1],T}$ in $O(m^2)$ time. This recurrence can be extended to include affine gap penalties by mapping the more complicated recurrence for the standard alignment with affine gap penalties to $DIST$ arrays.