J Comb Optim (2008) 16: 263–278
DOI 10.1007/s10878-007-9132-y

Alignments with non-overlapping moves, inversions and tandem duplications in $O(n^4)$ time

Christian Ledergerber · Christophe Dessimoz

Published online: 22 December 2007

Abstract Sequence alignment is a central problem in bioinformatics. The classical dynamic programming algorithm aligns two sequences by optimizing over possible insertions, deletions and substitutions. 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.

Keywords Dynamic programming · String to string comparison · Block operations · Scoring schemes · Biological sequence alignment

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

A preliminary version of this paper appeared in the Proceedings of COCOON 2007, LNCS, vol. 4598, pp. 151–164.

C. Ledergerber (✉) · C. Dessimoz
ETH Zurich, Institute of Computational Science, Zurich, Switzerland
e-mail: ledergec@student.ethz.ch

C. Dessimoz
e-mail: cdessimoz@inf.ethz.ch

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 (Needleman and Wunsch 1970), 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 (Fliess et al. 2002), 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 (Lopresti and Tomkins 1997). In particular an extension to general move operations was shown to be NP-complete (Shapira and Storer 2002) while the complexity of an extension to inversions is still unknown.

The hardness results from above led to the development of approximation algorithms for move operations. The results include a greedy algorithm presented in Shapira and Storer (2002) which achieves an approximation factor of $O(n^{0.69})$ as shown by Chrobak et al. (2005). Furthermore, Cormode and Muthukrishnan (2002) presented a $O(\log^* n \log n)$ factor approximation algorithm for general move operations which runs in sub-quadratic time.

Alternatively, the problems can be simplified through stronger assumptions. For alignments with inversions, Schoeninger and Waterman (1992) proposed the simplification hypothesis that none of the inversions overlap. They found that alignments with non-overlapping inversions can be computed in $O(n^6)$ time. This result was then further improved by Chen et al. (2004), do Lago and Muchnik (2005), Alves et al. (2005), Vellozo et al. (2006) where Vellozo et al. (2006) obtained an $O(n^3)$ algorithm for a restricted scoring scheme.

A related problem is the detection of (tandem) repeats resulting from duplications within a given sequence (Liu and Wang 2006).

In this paper, we extend the non-overlapping hypothesis to moves and tandem duplications and show that the problem of computing alignments with non-overlapping moves, inversions and tandem duplications can be solved in polynomial time. We 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 Apic et al. (2001). They show that protein domain order is highly conserved during evolution. It is established in Apic et al. (2001) that most domains cooccur with only zero, one or two domain families. Since a move operation of the more elaborate type such as $ABCD \to ACBD$ immediately implies that B cooccurs with three other domains, we conclude that move operations have to be

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

rare. Furthermore, exon shuffling is highly correlated to domain shuffling (Kaessmann et al. 2002; Liu et al. 2005; Vibranovski et al. 2005) and hence cannot lead to a large amount of move operations. Finally, a number of move operations can be found in the literature (Bashton and Chothia 2002; Shandala et al. 2004). 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 (Andrade et al. 2001; Marcotte et al. 1999).

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 for the first problem.

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). If $j \leq i$, we define $S[i..j] = \lambda$. Note, 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 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. That is, $S = S_1 \dots S_d$. 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 Maes (1990):

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 maximum. The optimal score is denoted by $\delta_c(S, T)$.

For constant indel penalties 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 Maes (1990).

Finally, we assume that the reader is familiar with the concept of edit graphs as defined for instance in Myers (1991) or Gusfield (1997).

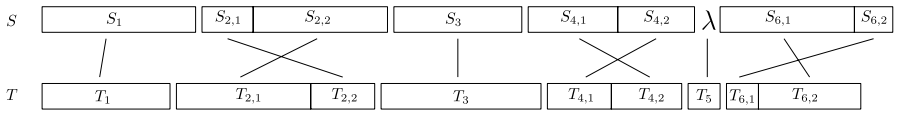

Fig. 1 Example of a non-overlapping move alignment of S with T

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 $l_{S_{i1}}, l_{S_{i2}}, l_{T_{i1}}, l_{T_{i2}}$ 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. Intuitively, in the above definition 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 Landau and Ziv-Ukelson (2001).

Note that although simplistic gap penalty schemes may not always yield bi- ologically meaningful alignments, they are interesting from the theoretical point of view.

2.3 Definition of alignment with non-overlapping moves, inversions and tandem-duplications

For the sake of simplicity, we assume constant indel penalties and constant penalties for block operations in the treatment of this problem. However, the scoring schemes of Sect. 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 Landau and Ziv-Ukelson (2001), Schmidt (1998) 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). □

We would like to note here that inverse total monotonicity of $OUT_{S,T}$ would suffice for the following conclusions. However, in the Appendix we show that $f(l)$ has to be a concave function unless we can prove stronger properties for $DIST_{S,T}$.

Using our observations and the results from Landau and Ziv-Ukelson (2001); Schmidt (1998), 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 Schmidt (1998). Then since OUT is inverse Monge, we can compute the output vector O by applying the algorithm of Aggarwal et al. (1987) 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 Schmidt (1998) 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 Sect. 6 of Schmidt (1998) and then compute the output vector O using the algorithm of Landau and Ziv-Ukelson (2001) 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 (as shown in the Appendix) these results cannot be easily extended to affine gap penalties. It is however open whether the algorithm in Landau and Ziv-Ukelson (2001) can be adapted to include more complex penalty functions for the restricted scoring scheme.

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.

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

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

where

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

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 $MOVE$.

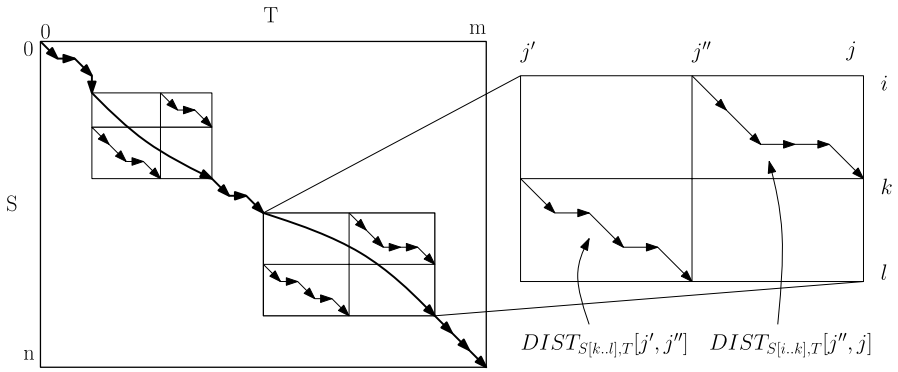

Fig. 2 An illustration of the computation of a move operation in DP_MOVE. Since the scores are additive: $SCO[l, j] = SCO[i, j'] + DIST_{S[k..l],T}[j', j''] + DIST_{S[i..k],T}[j'', j]$. In DP_MOVE this is maximized for all $i < k < l$, $j' < j'' < j$

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 Sect. 2.4 since both of them are output vectors as in Definition 7.

DP_MOVE

1: **for all** i, j **such that** $0 \le i \le n, 0 \le j \le m$ **do**
2: {base case}
3: $SCO[i, 0] := i \cdot \sigma_I$
4: $SCO[0, j] := j \cdot \sigma_I$
5: $SCO[i, j] := -\infty$ if $i \ne 0, j \ne 0$

6: **end for**
7: **for** i **from** 0 **to** n **do**
8: **for** j **from** 1 **to** m **do**
9: {standard alignment recurrence}
10: $SCO[i, j] := \max\{SCO[i, j], SCO[i-1, j] + \sigma_I, SCO[i, j-1] + \sigma_I, SCO[i-1, j-1] + \sigma(S[i], T[j])\}$
11: **end for**
12: **for** k **from** i **to** n **do**
13: {move operations}
14: $DIST_{S[i..k],T} := calcDist(DIST_{S[i..k-1],T})$
15: **for** l **from** k **to** n **do**
16: $DIST_{S[k..l],T} := calcDist(DIST_{S[k..l-1],T})$
17: $O_1 := calcOutput(OUT[j', j''] = SCO[i, j'] + DIST_{S[k..l],T}[j', j''] + \sigma_c(j'' - j') + \sigma_c(l - k))$
18: $O_2 := calcOutput(OUT[j'', j] = O_1[j''] + DIST_{S[i..k]}[j'', j] + \sigma_c(j - j'') + \sigma_c(k - i))$
19: **for** j **from** 0 **to** m **do**
20: $SCO[l, j] := \max\{SCO[l, j], O_2[j]\}$
21: **end for**
22: **end for**
23: **end for**
24: **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 (Myers 1991). 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.

For $i = 0$, the table is initialized with the base case of the recurrence. For $1 \leq i \leq n$, 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 lines 8 to 11 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 all edges due to move operations starting on that row on lines 12 to 23. Consequently, when we advance to the computation of row $i + 1$ the assumption is again true.

Using the results from Sect. 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* $O(n^3 m)$ *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.

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

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

where

$$MOVE = \max_{0 \le i' < i,\ 0 \le j' < j} \{SCO_{S,T}[i', j'] + \delta_c(S[i'..i], T[j'..j])\},$$

$$S\_DUPLICATE = \max_{0 \le i' < i, 0 \le j' < j'' < j} \{SCO_{S,T}[i', j'] + \delta(S[i'..i], T[j'..j''])$$
$$+ \delta(S[i'..i], T[j''..j]\},$$

$$T\_DUPLICATE = \max_{0 \le i' < i'' < i, 0 \le j' < j} \{SCO_{S,T}[i', j'] + \delta(S[i'..i''], T[j'..j])$$
$$+ \delta(S[i''..i], T[j'..j]\},$$

$$REVERSE = \max_{0 \le i' < i, 0 \le j' < j} \{SCO_{S,T}[i', j'] + \delta(\overline{S[i'..i]}, T[j'..j])\}.$$

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 Alves et al. (2005) it is shown how to handle *REVERSE*. *S_DUPLICATE* can be done as follows. We calculate $DIST_{S[i..k]}$. Then, we 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 illustrated in Fig. 3, we can now present our algorithm for this problem.

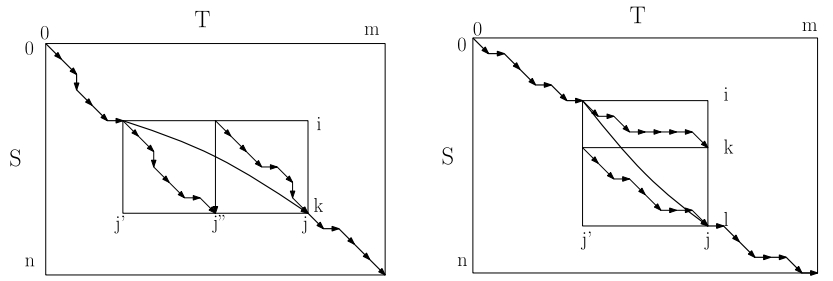

Fig. 3 An illustration on how duplications are treated

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 Vellozo et al. (2006)}
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: {S 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** j **from** 0 **to** m **do**
12: $SCO[l, j] := \max\{SCO[l, j], O_2[j] + \sigma_d\}$
13: **end for**
14: **end for**
15: **for** k **from** i **to** n **do**
16: {T duplication}
17: $DIST_{S[i..k],T} := calcDist(DIST_{S[i..k-1],T})$
18: **for** l **from** k **to** n **do**
19: $DIST_{S[k..l],T} := calcDist(DIST_{S[k..l-1],T})$
20: $O := calcOutput(OUT[j', j] = SCO[i, j'] + DIST_{S[i..k],T}[j', j] + DIST_{S[k..l],T}[j', j])$
21: **for** j **from** 0 **to** m **do**
22: $SCO[l, j] := \max\{SCO[l, j], O[j] + \sigma_d\}$
23: **end for**
24: **end for**
25: **end for**
26: **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* $O(n^3 m \log m)$ *time and* $O(mn + m^2)$ *space for* concave scoring schemes *and in* $O(n^3 m)$ *time and* $O(nm)$ *space for* restricted scoring schemes, *where* $n = |S|$ *and* $m = |T|$.

3.3 Asymmetric alignments with tandem duplications

If we restrict the alignment problem to tandem duplications it can be seen in DP_MOVE_INV_DUPL that considering duplication events of substrings of T is the bottleneck in terms of time. That is, if we restrict the problem to reversals and tandem duplication of substrings of S (e.g. no tandem duplications of substrings of T) we can solve the problem in $O(n^2 m)$ time. However, in this case the distance measure is no more symmetric e.g. $\delta_{rd}(S, T) \neq \delta_{rd}(T, S)$ which is undesirable in most applications. Furthermore, if the symmetric model reveals tandem duplications it is not guaranteed that the asymmetric model will do so too. Still, this is a heuristic approach for detecting tandem duplication events more efficiently.

4 Implementation and experiments

We have implemented an $O(n^5)$ version of the DP_MOVE with constant gap penalties in C.[2] Inversions and tandem duplications were not considered. 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 Fliess et al. (2002). In addition we run the algorithm on an example found in Shandala et al. (2004). This alignment is shown in Fig. 4 and is compared with a standard alignment obtained from Darwin (Gonnet et al. 2000).

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 $O(n^4)$ 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 restricted to non-overlapping moves, 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.

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

[2] Available from the authors upon request.

Fig. 4 (Color online) An example found in Shandala et al. (2004). In this figure we compare an alignment computed with our new algorithm and an alignment done with Darwin (Gonnet et al. 2000). 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 *red*

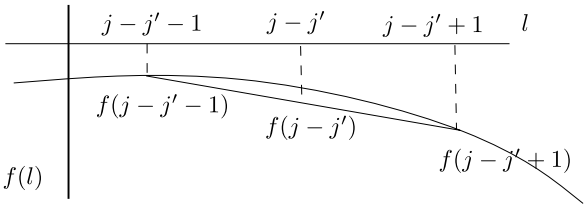

Fig. 5 For concave functions any point on a secant is below the function

Appendix

Concave scoring scheme

In this section we will first prove that the condition of a concave scoring scheme is sufficient by proving Lemmas 1 and 2. We also establish that the condition is necessary, that is, it is not guaranteed that the algorithm of Aggarwal et al. (1987) will compute the correct result without it.

For a proof of Lemmas 1 and 2 we have to define the notion of inverse Monge arrays (Monge 1781) first:

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

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

Furthermore we denote a matrix $M[0\ldots n; 0\ldots m]$ inverse totally monotone if for all $0 \leq i_1 < i_2 \leq n, 0 \leq j_1 < j_2 \leq m$

$$M[i_1, j_1] < M[i_1, j_2] \quad \Rightarrow \quad M[i_2, j_1] < M[i_2, j_2].$$

We note that total monotonicity is a weaker property than the Monge property. That is, inverse Monge implies inverse totally monotone, but the converse is not true.

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. □

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

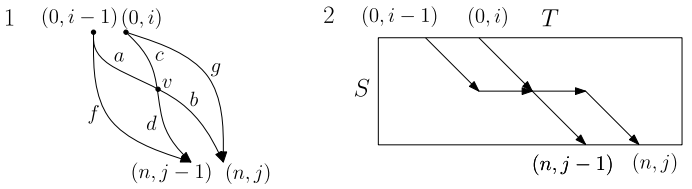

Fig. 6 1 illustrates that the path from vertex $(0, i)$ to $(n, j-1)$ and the path from vertex $(0, i-1)$ to (n, j) in the grid graph have to cross in a common vertex v. 2 gives a counter example for affine gap penalties

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 Schmidt (1998). However, we extended it to affine gap penalties.

Proof Let $\sigma_{INIT} < 0$ denote the gap opening penalty of an affine scoring scheme. Furthermore let $|a|$ be the length of path a and $a \cdot b$ be the concatenation of path a with path b.

The paths represented by $DIST_{S,T}[i-1, j] =: |a \cdot b|$ and $DIST_{S,T}[i, j-1] =: |c \cdot d|$ have to cross properly in a vertex v as shown in Fig. 6. Therefore, we have

$$\begin{aligned} &DIST_{S,T}[i-1, j] + DIST_{S,T}[i, j-1] \\ &\quad = |a \cdot b| + |c \cdot d| \\ &\quad \leq |a \cdot d| + |b \cdot c| - 2\sigma_{INIT} \\ &\quad \leq |g| + |f| - 2\sigma_{INIT} \\ &\quad = DIST_{S,T}[i, j] + DIST_{S,T}[i-1, j-1] - 2\sigma_{INIT} \end{aligned}$$

where the first inequality follows from the observation, that reconnecting the paths splits at most two gaps and hence increases the number of gaps at most by two. The second inequality follows since we are maximizing the lengths of the paths.

Hence *DIST* arrays are Monge for $\sigma_{INIT} = 0$. □

In Fig. 6(2) an example for affine gap penalties is given which illustrates that *DIST* arrays may not be Monge for large gap opening penalties.

As noted in Sect. 2.4 it would suffice to prove that $OUT_{S,T}$ is totally monotone. The following lemma however shows that if we drop one of the assumptions the resulting *OUT* array is not guaranteed to be totally monotone.

Lemma 3 *Let $A, B \in \mathbb{R}^{n \times m}$ be two arrays and let $C = A + B$. Then C is guaranteed to be inverse totally monotone iff A and B are inverse Monge. That is, if B is not inverse Monge we can always find A, A inverse Monge, such that C is not inverse totally monotone.*

Proof If A and B are inverse Monge then C is so too and hence C it also inverse totally monotone.

For the other direction first consider only the case that $A, B \in \mathbb{R}^{2\times 2}$ with

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \quad B = \begin{bmatrix} e & f \\ g & h \end{bmatrix} \quad \text{and hence} \quad C = \begin{bmatrix} a+e & b+f \\ c+g & d+h \end{bmatrix}.$$

Assuming B is not inverse Monge $e+h < g+f$ e.g. $(g-h)+(f-e) = \epsilon > 0$. We then choose A as $b = c = 0$ and $a = f - e - \epsilon/4, d = g - h - \epsilon/4$.

We thus have $a + d = g - h + f - e - \epsilon/2 = \epsilon/2 > 0 = b + c$ and hence A is inverse Monge. Further more we have $a + e = f - \epsilon/4 < f = b + f$ and $c + g = g > g - \epsilon/4 = d + h$ and hence C is not inverse totally monotone.

The general case follows since an array is Monge iff every 2×2 sub-array is Monge and an array is totally monotone iff every 2×2 sub-array is totally monotone and finally we can enlarge A by duplicating columns/rows. □

Finally, we would like to note, that it is still possible to allow affine gap penalties by relaxing one of the conditions in favor of the other. That is, if

$$\begin{aligned} &DIST_{S,T}[i-1, j] + DIST_{S,T}[i, j-1] + 2\sigma_{INIT} \\ &\leq DIST_{S,T}[i, j] + DIST_{S,T}[i-1, j-1] \end{aligned}$$

then we have to insist on

$$f_l(i-1, j) + f(i, j-1) - 2\sigma_{INIT} \leq f_l(i, j) + f_l(i-1, j-1)$$

such that the sum is still Monge. This may not be favorable in practical models. The converse, e.g. relaxing the condition on the penalty functions and allowing slightly convex functions, appears to be impossible because we cannot enforce a stronger condition on the *DIST* arrays.

Extension of *DIST* arrays

This simple algorithm is inspired by Schmidt (1998). 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.

References

Aggarwal A, Klawe MM, Moran S, Shor P, Wilber R (1987) Geometric applications of a matrix-searching algorithm. Algorithmica 2(1):195–208

Alves CER, do Lago AP, Vellozo AF (2005) Alignment with non-overlapping inversions in $o(n^3 \log n)$ time. In: Proceedings of GRACO 2005. Electronic Notes in Discrete Mathematics, vol 19. Elsevier, Amsterdam, pp 365–371

Andrade MA, Perez-Iratxeta C, Ponting CP (2001) Protein repeats: structures, functions, and evolution. J Struct Biol 134(23):117–131

Apic G, Gough J, Teichmann SA (2001) Domain combinations in archaeal, eubacterial and eukaryotic proteomes. J Mol Biol 310(2):311–325

Bashton M, Chothia C (2002) The geometry of domain combination in proteinsm. J Mol Biol 315(4):927–939

Chen ZZ, Gao Y, Lin G, Niewiadomski R, Wang Y, Wu J (2004) A space-efficient algorithm for sequence alignment with inversions and reversals. Theor Comput Sci 325(3):361–372

Chrobak M, Kolman P, Sgall J (2005) The greedy algorithm for the minimum common string partition problem. ACM Trans Algorithms 1(2):350–366

Cormode G, Muthukrishnan S (2002) The string edit distance matching problem with moves. In: SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics, pp 667–676

do Lago AP, Muchnik I (2005) A sparse dynamic programming algorithm for alignment with non-overlapping inversions. Theor Inform Appl 39(1):175–189

Fliess A, Motro B, Unger R (2002) Swaps in protein sequences. Proteins 48(2):377–387

Gonnet GH, Hallett MT, Korostensky C, Bernardin L (2000) Darwin v2.0: an interpreted computer language for the biosciences. Bioinformatics 16(2):101–103

Gusfield D (1997/1999) Algorithms on strings, trees, and sequences: computer science and computational biology. Press Syndicate of the University of Cambridge, Cambridge

Kaessmann H, Zöllner S, Nekrutenko A, Li WH (2002) Signatures of domain shuffling in the human genome. Genome Res 12(11):1642–1650

Landau GM, Ziv-Ukelson M (2001) On the common substring alignment problem. J Algorithms 41(2):339–354

Liu X, Wang L (2006) Finding the region of pseudo-periodic tandem repeats in biological sequences. Algorithms Mol Biol 1(1):2

Liu M, Walch H, Wu S, Grigoriev A (2005) Significant expansion of exon-bordering protein domains during animal proteome evolution. Nucleic Acids Res 33(1):95–105

Lopresti D, Tomkins A (1997) Block edit models for approximate string matching. Theor Comput Sci 181(1):159–179

Maes M (1990) On a cyclic string-to-string correction problem. Inf Process Lett 35(2):73–78

Marcotte EM, Pellegrini M, Yeates TO, Eisenberg D (1999) A census of protein repeats. J Mol Biol 293(1):151–160

Monge G (1781) Déblai et remblai. Mémoires de l'Académie Royale des Sciences

Myers EW (1991) An overview of sequence comparison algorithms in molecular biology. Technical report 91-29, Univ of Arizona, Dept of Computer Science

Needleman SB, Wunsch CD (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. J Mol Biol 48(3):443–453

Schmidt JP (1998) 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

Schoeninger M, Waterman MS (1992) A local algorithm for dna sequence alignment with inversions. Bull Math Biol 54(4):521–536

Shandala T, Gregory SL, Dalton HE, Smallhorn M, Saint R (2004) Citron kinase is an essential effector of the pbl-activated rho signalling pathway in drosophila melanogaster. Development 131(20):5053–5063

Shapira D, Storer JA (2002) Edit distance with move operations. In: CPM '02: Proceedings of the 13th annual symposium on combinatorial pattern matching, London, UK. Springer, Berlin, pp 85–98

Vellozo AF, Alves CER, do Lago AP (2006) Alignment with non-overlapping inversions in $o(n^3)$-time. In: WABI, LNCS, vol 4175. Springer, Berlin

Vibranovski MD, Sakabe NJ, de Oliveira RS, de Souza SJ (2005) 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