Laboratoire Bordelais de Recherche en Informatique
UMR 5800 - Université Bordeaux 1, 351, cours de la Libération,
33405 Talence CEDEX, France

Research Report RR-1443-08

New algorithms for aligning nested arc-annotated sequences

by Aïda Ouangraoua, Cedric Chauve, Valentin Guignon and Sylvie Hamel

April, 2008

New algorithms for aligning nested arc-annotated sequences

Aïda Ouangraoua[1], Cedric Chauve[2], Valentin Guignon[3,4], and Sylvie Hamel[4]

[1] LaBRI, Université Bordeaux I, 351 Cours de la Libération, 33405 Talence Cedex, France.
[2] Department of Mathematics, Simon Fraser University, 8888 University Drive, Burnaby, BC, V5A 1S6, Canada.
CGL and LaCIM, Université du Québec à Montréal, CP 8888 Succ. Centre-ville, Montréal, QC, H3C 3P8, Canada.
[3] LIRMM, Université Montpellier II, 161 rue Ada, 34392 Montpellier Cedex 5, France.
[4] DIRO, Université de Montréal, C.P. 6128 Succ. Centre-Ville, Montréal, QC, H3C 3J7, Canada.

Abstract. *Background.* We propose in this paper a new algorithm for the alignment of RNA secondary structures without pseudoknots, represented by nested arc-annotated sequences. We use a general edit distance model between arc-annotated sequences, that considers classical sequences edit operations, but also structural edit operations, such as the creation, deletion or modification of bonds between pairs of bases. The general edit distance problem is NP-hard in this model, and recently a hierarchy of arc-annotated sequence alignment problems was introduced in order to define tractable less general problems.
Results. We extend the hierarchy of alignment problems and describe a polynomial space and time algorithm that solves a more general alignment problem. Up to date the alignment problem we solve is the most general one that is known to be tractable in the considered edit distance model. This algorithm is efficient, as its asymptotic time and space complexities are the same as the complexities of the best previously published algorithm, that solved a less general problem.

1 Introduction

An RNA molecule is a polymer composed of a sequence of nucleotides which can be linked together by phosphodiester bonds determining its structural conformation. It has been shown that the conformation of an RNA molecule is correlated with its function. Thus, RNA structure comparison is essential in the comparative approach that relates structural similarity to functional similarity. With the increasing amount of known RNA molecules, especially non-coding RNAs [6, 19, 17], the development of precise, fast, and well understood algorithms to compare RNA structures is of primary importance. In this paper, we focus on the comparison of RNA secondary structures without pseudoknots.

From an algorithmic point of view, RNA secondary structures comparison was first considered in the framework of the edit distance between ordered trees [12, 20]. Recently, it has also been described as a problem of edit distances between *arc-annotated sequences*, that can also be used to represent RNA structures with pseudoknots [9, 14]. An arc-annotated sequence is a sequence, over a given alphabet, with additional structure described by a – possibly empty – set of arcs, each arc joining a pair of positions in the sequence. From a purely combinatorial point of view, arc-annotated sequences are then a natural extension of simple sequences. The problem of computing an edit distance between two arc-annotated sequences seems to have been introduced, at least in relation with the comparison of RNA structures, in [9, 21], where a simple model was introduced that used only three edit operations (insertion, deletion and substitution) and did not consider separately the sequence elements that belong to a same arc. This problem, in the case of nested arc-annotated sequences (*i.e.* where no pairs of arcs are crossing), is equivalent to the tree edit distance computation, that can be solved in polynomial time [7, 8, 20]. In [14], new edit operations were introduced, to account for structural evolutionary events, such as the creation, deletion or modification of bonds

between pairs of bases. Accounting for such events leads naturally to more realistic alignments between RNA secondary structures [18], but at the cost of computational tractability. Indeed, it was recently shown in [3] that computing the edit distance between two nested arc-annotated structures in the model introduced in [14] is NP-hard. Several groups have then defined less general alignment problems, by considering constraints either on the set of considered edit operations [13], or on the structure of possible alignments and edit sequences [4, 5, 11]. In particular, in [5], a hierarchy of several problems of distance and alignment computation between nested arc-annotated sequences was introduced, that enlightens the limit between hard and tractable problems. Up to date, the most general tractable distance model, using the edit operations introduced in [14], was presented in [4].

The main contribution of our paper is to extend the hierarchy of arc-annotated sequences alignments problems defined in [4, 5] and to introduce a new and more general, but still tractable, alignment problem, that can be applied in order to obtain more precise alignments between pairs of RNA secondary structures without pseudoknots. We propose an efficient dynamic programming algorithm solving this new problem. Our paper is structured as follows. In section 2, we formally describe the hierarchy of arc-annotated sequence alignment problems introduced in [5]. In section 3, we refine this hierarchy and describe some properties of the new classes of arc-annotated sequence alignment problems that we introduce. In particular we show that an algorithm that was described in [11] to compare RNA stem-loops is in fact an exact algorithm in this hierarchy of alignment problems. We also show that the new alignment problem we introduce fixes an important weakness of previously considered alignment problems as it allows to align two bases, belonging to arcs in two RNA secondary structures, without being forced to align together the two other extremities of these arcs. Finally, we show that this alignment problem is tractable, as it can be solved with the same complexity than the less general problem considered in [4]. In section 4, we present a dynamic programming algorithm to compute the distance in this new model. In section 5, we conclude with some experimental results and we outline some perspectives on using our algorithm to define more efficient methods for RNA secondary structures comparison.

2 Preliminaries: Arc-annotated sequences and their comparison

We now describe formally the different problems of comparison of nested arc-annotated sequences we consider, and the existing results.

2.1 A hierarchy of arc-annotated sequences

An arc-annotated sequence of length n on a finite alphabet Σ is a couple $A = (S, P)$ where S is a sequence of length n on Σ and P is a set of pairs (i_1, i_2), with $1 \leq i_1 < i_2 \leq n$. Arc-annotated sequences are used, among others, for representing RNA structures: in such a case Σ is the alphabet of nucleotide bases that compose an RNA molecule, $\Sigma = \{A, C, G, U\}$ (see [9, 14] for example). In this paper we consider arc-annotated sequences representing RNA structures and we will then call an element of S a *base*. We denote by $S[i]$ the i^{th} base of S and by $S[i_1..i_2]$ the sub-sequence containing bases $S[i_1], S[i_1 + 1], \ldots, S[i_2]$. A pair $(i_1, i_2) \in P$ represents an *arc* linking bases $S[i_1]$ (called the *origin* of the arc) and $S[i_2]$ (called the *end* of the arc) of S – a *base pair* in terms of RNA structures –; the bases $S[i_1]$ and $S[i_2]$ are said to belong to the arc (i_1, i_2) and are the only bases that belong to this arc; a base that does not belong to an arc is called an *unpaired base*. We denote by $U(A)$, $P_o(A)$, and $P_e(A)$ respectively the set of unpaired bases, origins of arcs and ends

of arcs of A. If $A = (S, P)$, we also use the notation $P(A) = P$ to denote the set of arcs of A. In an arc-annotated sequence, two arcs (i_1, i_2) and (i_3, i_4) are said to *cross*, or to be *crossing*, if $i_1 < i_3 < i_2 < i_4$ or $i_3 < i_1 < i_4 < i_2$.

Arc-annotated sequences can be classified according to the combinatorial structure of their arcs. We now present the classification of arc-annotated sequences defined in [9] and used in [5]. An arc-annotated sequence $A = (S, P)$ is said to be (Fig. 1):

- Unlimited (Unlim) if there is no restriction on P.
- Crossing (Cros) if every base belongs to at most one arc
- Nested (abbreviated Nest) if it belongs to Cros and it has no pair of crossing arcs
- Plain if P is empty

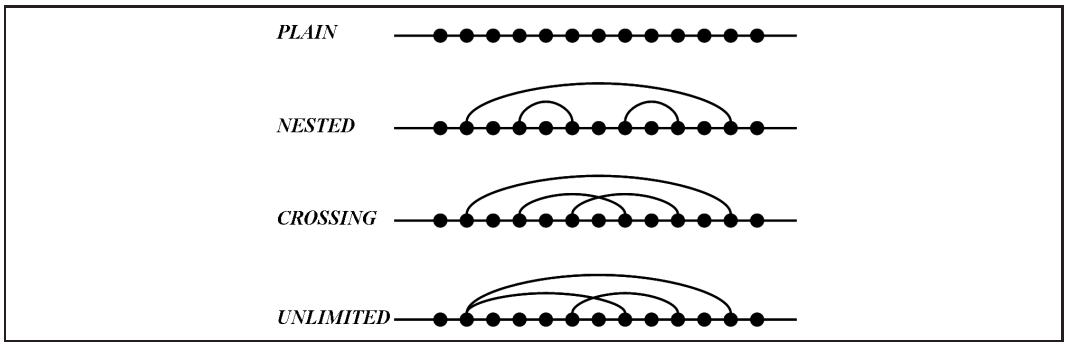

Fig. 1. Examples of arc-annotated sequences belonging to the classes of the hierarchy of arc-annotated sequences.

2.2 Edit operations, edit sequence and edit distance

We consider the set of edit operations on arc-annotated sequences that was introduced in [14]. In the following, a, b, c, d are elements of Σ.

- *Simple unpaired base operations* (Fig. 2):
  - Base-substitution (BS): substitution of a base a by a base b, denoted by $a \to b$.
  - Base-deletion (BD): deletion of a base a, denoted by $a \to \lambda$.
  - Base-insertion (BI): insertion of a base b, denoted by $\lambda \to b$.
- *Simple arc operations* (Fig. 2):
  - Arc-substitution (AS): substitution of an arc (a, b) by an arc (c, d), denoted by $(a, b) \to (c, d)$.
  - Arc-deletion (AD): deletion of an arc (a, b), denoted by $(a, b) \to \lambda, \lambda$.
  - Arc-insertion (AI): insertion of an arc (c, d), denoted by $\lambda, \lambda \to (c, d)$.
- *Complex arc operations* (Fig. 3):
  - Arc-breaking (AB): breaking an arc (a, b), denoted by $(a, b) \to a, b$.
  - Arc-creation (AC): creating an arc (c, d), denoted by $c, d \to (c, d)$.
  - Arc-altering (AA): altering an arc (a, b), denoted by $(a, b) \to a, \lambda$ or $(a, b) \to \lambda, b$.
  - Arc-completing (ACo): completing an arc (c, d), denoted by $c, \lambda \to (c, d)$ or $\lambda, d \to (c, d)$.

Each edit operation e has a cost depending on the operation and on the bases involved in it, denoted by $w(e)$. The set of costs associated to all possible edit operations is called the *cost scheme*. Note that a given cost scheme can implicitly prevent some edit operations to be considered in edit distance problems, if such operations can be replaced by a sequence of one or more edit operations for a lesser cost. If such a situation can not occur, a cost scheme is said to be *complete*. In this paper, we consider arbitrary cost schemes, including complete cost schemes.

Notation. We extend the cost notation defined above by identifying bases and arcs in an arc-annotated sequence by their position in the sequence. For example if x and y are the positions of two bases that form an arc in an arc-annotated sequence, the cost of breaking this arc is denoted by $w((x,y) \rightarrow x,y)$.

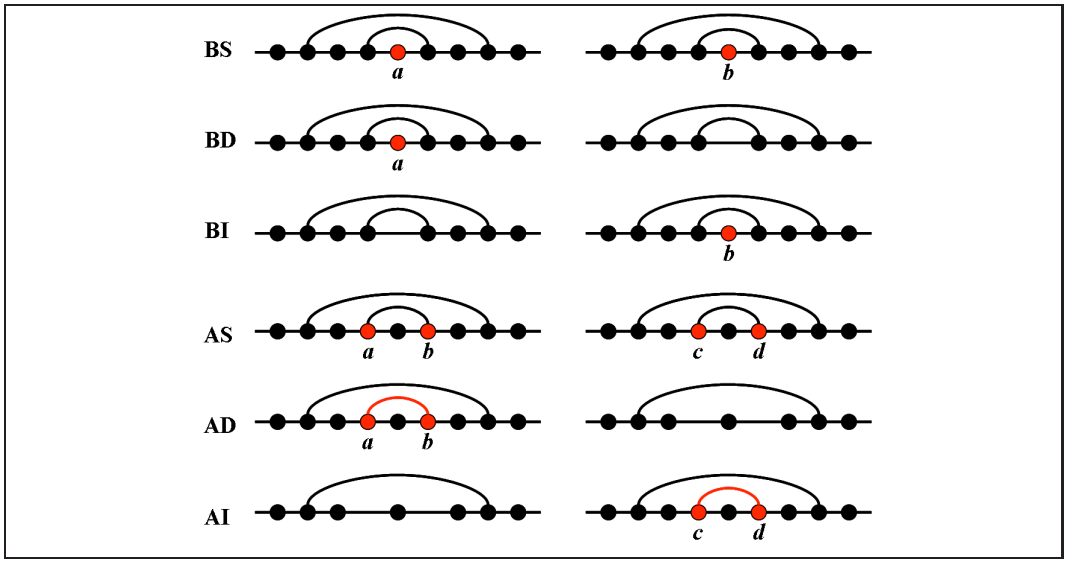

Fig. 2. Simple edit operations on arc-annotated sequence

Let $A_1 = (S_1, P_1)$ and $A_2 = (S_2, P_2)$ be two arc-annotated sequences of length n_1 and n_2 respectively. An *edit sequence* between A_1 and A_2 is a sequence E of edit operations that transforms A_1 into A_2. The cost of an edit sequence E denoted by $w(E)$ is the sum of the costs of the edit operations that compose it. An edit sequence between two arc-annotated sequences is said to be *optimal* if its cost is minimal among all edit sequences between these two arc-annotated sequences. The *edit distance* between two arc-annotated sequences is the cost of an optimal edit sequences between them.

Let C be a class of the hierarchy of arc-annotated structures. We denote by Edit(C,C) the general problem of computing the edit distance between two arc-annotated sequences belonging to C, for a given cost scheme.

Theorem 1. [3] Edit(Nest,Nest) *is NP-hard.*

The hardness of computing the edit distance between nested arc-annotated sequences naturally leads to define restrictions on the kind of considered of edit sequences, in such a way that computing an optimal edit sequences among this subset of edit sequences can be done efficiently. We present in

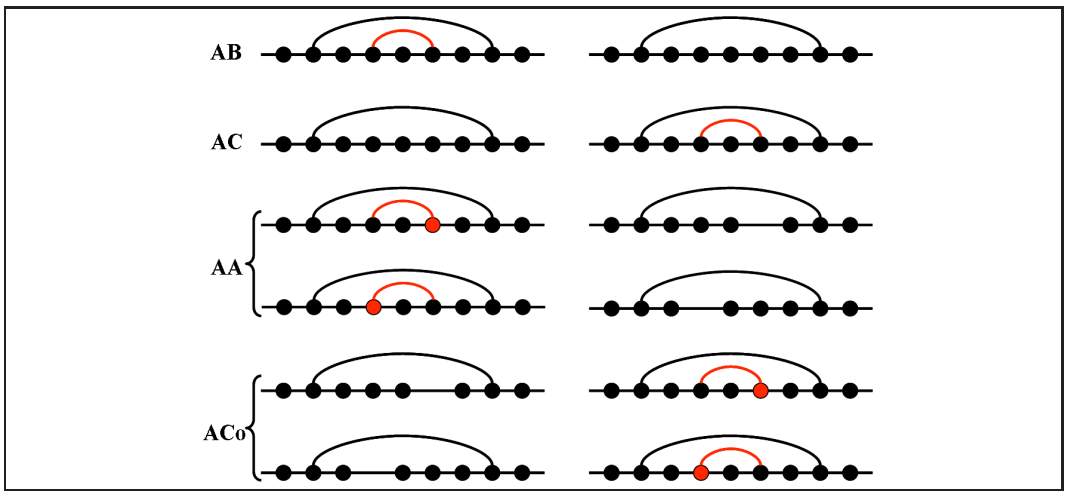

Fig. 3. Complex edit operations on arc-annotated sequence

Section 2.3 such an approach, introduced in [4, 5], that defines a hierarchy of comparison problems based on the notions of *super-sequence* and *alignment sequence*.

2.3 Alignment sequence and distances; a hierarchy of comparison problems

A *super-sequence* of an arc-annotated sequence $A = (S, P)$ is an arc-annotated sequence which can be obtained by applying on A an edit sequence composed of insertion and substitution operations only: BI, AI, AC, ACo, BS, and AS; symmetrically, an arc-annotated sequence can be obtained from any of its super-sequences using only deletion and substitution operations.

An edit sequence $E = e_1, \ldots, e_k$ composed of k edit operations between two arc-annotated sequences $A_1 = (S_1, P_1)$ and $A_2 = (S_2, P_2)$ is an *alignment sequence* if it can be decomposed into two sub-sequences, say $E_1 = e_1, \ldots, e_\ell$ and $E_2 = e_{\ell+1}, \ldots, e_k$ such that the edit operations of E_1 (resp. E_2) belong to the set {BI, AI, AC, ACo} (resp. {BS, BD, AS, AD, AB, AA}). The super-sequence of A_1 and A_2 induced by such an alignment sequence is the arc-annotated sequence obtained by applying on A_1 the sequence E_1 of edit operations (Fig. 4). E_1 and E_2 are called the *decomposition* of E.

Remark 1. In [4, 5], the definition of alignment sequence allows substitution operations BS and AS to occur in E_1. The restriction that they occur only in E_2 is intended to simplify the exposition and does not reduce the generality of our results.

For a given class C of the hierarchy of arc-annotated sequences, an alignment sequence E between two nested arc-annotated sequences A_1 and A_2 is said to be a C-alignment sequence if the super-sequence induced by E belongs to C. E is said to be an optimal C-alignment sequence if its cost is minimal among the set of all C-alignment sequences between A_1 and A_2. The cost of an optimal C-alignment sequence between A_1 and A_2 is called the C-alignment distance between A_1 and A_2, denoted by $d_{\mathrm{C}}(A_1, A_2)$. We denote by Align(Nest,Nest;C) the problem of computing the cost of an optimal C-alignment sequence between two nested arc-annotated sequences.

Theorem 2. [4, 5] Edit(Nest,Nest) = Align(Nest,Nest; Unlim).

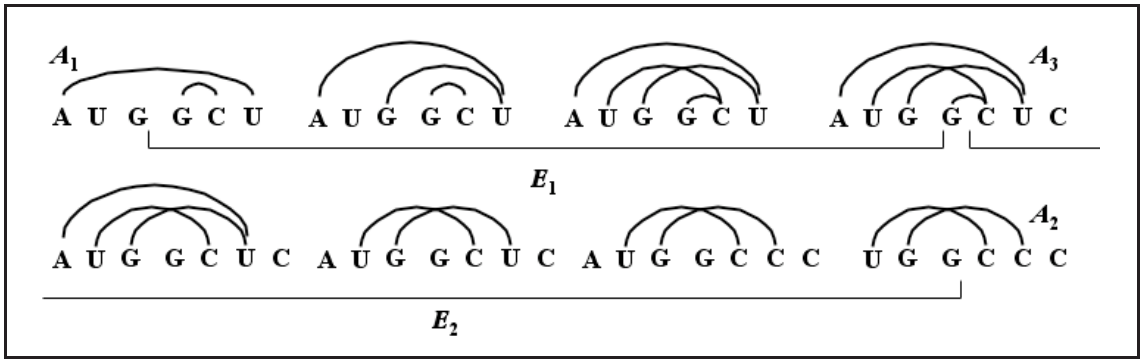

Fig. 4. An alignment sequence between two arc-annotated sequences – A_1 belongs to NEST and A_2 belongs to CROS –, and the corresponding super-sequence A_3 that belongs to UNLIM. The three first edit operations, that correspond to E_1 are respectively AC, AC and BI, and the last four, that correspond to E_2 are AB, AB, AS and BD.

Theorem 2 immediately suggests a natural way to restrict the set of considered edit sequences when comparing two nested arc-annotated sequences, in terms of the class that contains the corresponding super-sequence. Using such an approach, some alignment problems can be solved in polynomial time, generalizing previous results on the alignment of arc-annotated sequences with simple operations, that were described in terms of alignment of trees [15].

Theorem 3. [4, 5] *Let A_1 and A_2 be two nested arc-annotated sequences of respective lengths n_1 and n_2. $d_{NEST}(A_1, A_2)$ can be computed in $O(n^4)$ worst-case time and $O(n^3)$ space where $n = n_1 + n_2$.*

Theorem 4. [4, 5] ALIGN(NEST,NEST; CROS) *is NP-hard.*

Up to date, ALIGN(NEST,NEST; NEST) is the most general problem that is known to be tractable with a complete cost scheme. More general problems of alignment of nested arc-annotated sequences have been shown to be NP-hard, such as ALIGN(NEST,NEST; CROS) (Theorem 4) and ALIGN(NEST,NEST; UNLIM) (Theorem 2), but in both cases the hardness proofs assume non complete cost schemes that implicitly forbids some complex arc operations. On the other hand, for some non complete cost schemes, there exist exact and polynomial time algorithms for alignment problems with super-sequence that are more general than nested, like for example in [14] (AA, ACo, AI and AD are implicitly discarded) and in [13] (AB is the only considered complex arc operation).

2.4 Alignment of arc-annotated sequences.

We now relate the problem of computing the alignment distance to the actual alignment between arc-annotated sequences, as defined in [14].

An *alignment* between two arc-annotated sequences $A_1 = (S_1, P_1)$ and $A_2 = (S_2, P_2)$ is a couple $M = (A_1^M, A_2^M)$ where $A_1^M = (S_1^M, P_1^M)$ and $A_2^M = (S_2^M, P_2^M)$ are two arc-annotated sequences on the alphabet $\Sigma \cup \{-\}$ such that (Fig. 5):

- A_1^M and A_2^M have the same length $|S_1^M| = |S_2^M| = n$, (n is the length of the alignment M),
- removing symbols $-$ from A_1^M (resp. A_2^M) gives A_1 (resp. A_2), which implies that each base of S_1 (resp. S_2) corresponds to a unique base of S_1^M (resp. S_2^M),
- for any $1 \leq i \leq n$, $S_1^M[i] \neq -$ or $S_2^M[i] \neq -$,

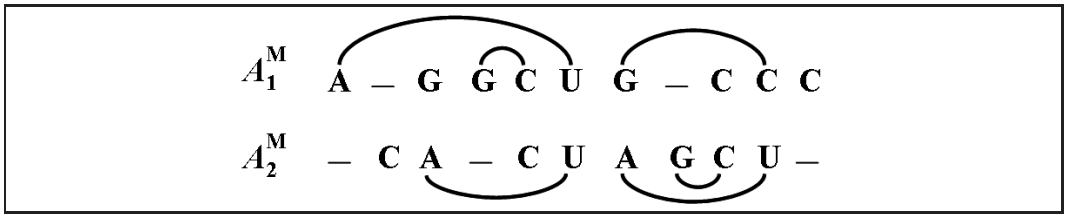

Fig. 5. An alignment between two nested arc annotated sequences $A_1 = (S_1, P_1)$ and $A_2 = (S_2, P_2)$ with $S_1 = AGGCUGCCC$, $P_1 = \{(1,5),(3,4),(6,8)\}$, $S_2 = CACUAGCU$ and $P_2 = \{(2,4),(5,8),(6,7)\}$.

– for any $(i_1, i_2) \in P_1^M$ (resp.$(j_1, j_2) \in P_2^M$), $S_1^M[i_1] \neq -$ and $S_1^M[i_2] \neq -$ (resp. $S_2^M[j_1] \neq -$ and $S_2^M[j_2] \neq -$).

For a base $S_1[i]$ of S_1, corresponding to the base $S_1^M[i']$, we say that it is aligned with $-$ if $S_2^M[i'] = -$ and with $S_2[j]$ if $S_2[j]$ corresponds to $S_2^M[i']$. A symmetrical definition applies for S_2. An alignment M between A_1 and A_2 defines an arc-annotated sequence $A_3^M = (S_3^M, P_3^M)$, on the alphabet Σ, called the super-sequence of A_1 and A_2 induced by M, as follows (Fig. 6):

– $|S_3^M| = n$,
– $P_3^M = P_1^M \cup P_2^M$,
– for any $1 \leq i \leq n$, if $S_1^M[i] \neq -$ then $S_3^M[i] = S_1^M[i]$ else $S_3^M[i] = S_2^M[i]$.

For a given class C of the hierarchy of arc-annotated sequences, an alignment M between A_1 and A_2 is said to be a C-alignment if the super-sequence A_3^M belongs to the class C. We denote by $\mathcal{M}_{\mathrm{C}}(A_1, A_2)$ the set of all C-alignments between A_1 and A_2.

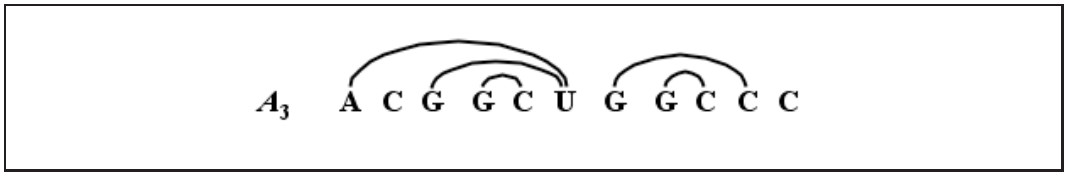

Fig. 6. The super-sequence induced by the alignment of Fig. 5

Following [14], we associate a cost to an alignment M between A_1 and A_2, denoted $w(M)$. M implicitly defines a set of edit operations, denoted $BS(M)$, $BD(M)$, $BI(M)$, $AS(M)$, $AD(M)$, $AI(M)$, $AB(M)$, $AC(M)$, $AA_l(M)$, $AA_r(M)$, $ACo_l(M)$ and $ACo_r(M)$ (the name obviously correspond to the edit operations defined in Section 2.2) as follows:

$$
\begin{array}{ll}
AS(M) & = P_1^M \cap P_2^M \\
BS(M) & = \{i \in [1 \ldots n] \mid S_1^M[i] \neq - \text{ and } S_2^M[i] \neq - \text{ and} \\
 & \quad \nexists\ j \text{ s.t. } (i,j) \in AS(M) \text{ or } (j,i) \in AS(M)\ \} \\
BD(M) & = \{i \in U(A_1^M) \mid S_2^M[i] = -\} \\
BI(M) & = \{j \in U(A_2^M) \mid S_1^M[j] = -\} \\
AD(M) & = \{(i_1,i_2) \in P_1^M \mid S_2^M[i_1] = - \text{ and } S_2^M[i_2] = -\} \\
AI(M) & = \{(j_1,j_2) \in P_2^M \mid S_1^M[j_1] = - \text{ and } S_1^M[j_2] = -\} \\
AB(M) & = \{(i_1,i_2) \in P_1^M \mid S_2^M[i_1] \neq - \text{ and } S_2^M[i_2] \neq - \text{ and } (i_1,i_2) \notin P_2^M\} \\
AC(M) & = \{(j_1,j_2) \in P_2^M \mid S_1^M[j_1] \neq - \text{ and } S_1^M[j_2] \neq - \text{ and } (j_1,j_2) \notin P_1^M\} \\
AA_l(M) & = \{(i_1,i_2) \in P_1^M \mid S_2^M[i_1] = - \text{ and } S_2^M[i_2] \neq -\} \\
AA_r(M) & = \{(i_1,i_2) \in P_1^M \mid S_2^M[i_1] \neq - \text{ and } S_2^M[i_2] = -\} \\
ACo_l(M) & = \{(j_1,j_2) \in P_2^M \mid S_1^M[j_1] = - \text{ and } S_1^M[j_2] \neq -\} \\
ACo_r(M) & = \{(j_1,j_2) \in P_2^M \mid S_1^M[j_1] \neq - \text{ and } S_1^M[j_2] = -\}
\end{array}
\tag{1}
$$

The cost of M, denoted by $w(M)$, is then defined by

$$
\begin{array}{rl}
w(M) = & \sum_{(i) \in BS(M)} w(S_1^M[i] \rightarrow S_2^M[i]) \\
 & + \sum_{i \in BD(M)} w(S_1^M[i] \rightarrow \lambda) \\
 & + \sum_{j \in BI(M)} (\lambda \rightarrow S_2^M[j]) \\
 & + \sum_{(i_1,i_2) \in AS(M)} w((S_1^M[i_1], S_1^M[i_2]) \rightarrow (S_2^M[i_1], S_2^M[i_2])) \\
 & + \sum_{(i_1,i_2) \in AD(M)} w((S_1^M[i_1], S_1^M[i_2]) \rightarrow \lambda, \lambda) \\
 & + \sum_{(j_1,j_2) \in AI(M)} w(\lambda, \lambda \rightarrow (S_2^M[j_1], S_2^M[j_2])) \\
 & + \sum_{(i_1,i_2) \in AB(M)} w((S_1^M[i_1], S_1^M[i_2]) \rightarrow S_1^M[i_1], S_1^M[i_2]) \\
 & + \sum_{(j_1,j_2) \in AC(M)} w(S_2^M[j_1], S_2^M[j_2] \rightarrow (S_2^M[j_1], S_2^M[j_2])) \\
 & + \sum_{(i_1,i_2) \in AA_l(M)} w((S_1^M[i_1], S_1^M[i_2]) \rightarrow \lambda, S_1^M[i_2]) \\
 & + \sum_{(i_1,i_2) \in AA_r(M)} w((S_1^M[i_1], S_1^M[i_2]) \rightarrow S_1^M[i_1], \lambda) \\
 & + \sum_{(j_1,j_2) \in ACo_l(M)} w(\lambda, S_2^M[j_2] \rightarrow (S_2^M[j_1], S_2^M[j_2])) \\
 & + \sum_{(j_1,j_2) \in ACo_r(M)} w(S_2^M[j_1], \lambda \rightarrow (S_2^M[j_1], S_2^M[j_2]))
\end{array}
\tag{2}
$$

A C-alignment between two arc-annotated sequences A_1 and A_2 is said to be optimal if it has minimum cost among the set of alignments $\mathcal{M}_{\mathrm{C}}(A_1, A_2)$. Proposition 1 below describes a natural relationship between optimal alignments and optimal alignment sequences, that was used implicitly in [4, 5, 14].

Proposition 1. *Let A_1 and A_2 be two nested arc-annotated sequences and* C *a class of the hierarchy of arc-annotated sequences. $d_{\mathrm{C}}(A_1, A_2) = \min_{M \in \mathcal{M}_{\mathrm{C}}(A_1,A_2)} w(M)$.*

Proof. Let M be a C-alignment between A_1 and A_2. By definition, A_3^M is a super-sequence of A_1 that can be obtained from A_1 by the edit operations implicitly defined by $BI(M)$, $AI(M)$, $AC(M)$, $ACo_l(M)$ and $ACo_r(M)$; these operations all belong to BI, AI, AC or ACo. Similarly, A_2 can be obtained from A_3^M by the edit operations defined by $BS(M)$, $AS(M)$, $BD(M)$, $AD(M)$, $AB(M)$, $AA_l(M)$ and $AA_r(M)$, that belong respectively to BS, AS, BD, AD, AB and AA. From the fact that A_3^M belongs to the class C we have that $d_{\mathrm{C}}(A_1, A_2) \leq \min_{M \in \mathcal{M}_{\mathrm{C}}(A_1,A_2)} w(M)$.

Let E be a C-alignment sequence between A_1 and A_2, and A_3 the corresponding super-sequence. It defines implicitly an alignment M between A_1 and A_2: bases that are never deleted or inserted (through base operations or arc operations) define pairs of bases of A_1 and A_2 that can be aligned, the induced gaps in A_1 and A_2 being filled with the symbol $-$. It is then immediate that $A_3 = A_3^M$ and $w(M) = w(E)$, and then $d_C(A_1, A_2) \geq \min_{M \in \mathcal{M}_C(A_1,A_2)} w(M)$. □

3 Extension of the hierarchy of arc-annotated sequences

We introduce new classes of arc-annotated sequences, together with some of the properties of the corresponding alignment sequences.

3.1 New classes of arc-annotated sequences.

A first extension: STEM. The first new class we describe is the class of arc-annotated sequences that correspond to the RNA secondary structures of stem-loops. We denote this class STEM. An arc-annotated sequence (S, P) belongs to the class STEM if

1. it belongs to NEST and
2. it has no pair of arcs (i_1, i_2) and (i_3, i_4) such that $i_1 < i_2 < i_3 < i_4$ or $i_3 < i_4 < i_1 < i_2$

Hence, we can remark that this new class induces the following inclusion relation: PLAIN ⊂ STEM ⊂ NEST ⊂ CROS ⊂ UNLIM.

The class STEM was considered in [11], where it was shown that, due to its simple structure, it is possible to compute the *conservative edit distance* between two arc-annotated sequences belonging to STEM (defined in Section 3.3, in polynomial time. It was also proposed to use a decomposition of RNA secondary structures in stem-loops and to use this algorithm to compare complete RNA secondary structures.

A second extension: MULT. Our second extension is inspired by the remark that there are two differences between NEST and UNLIM: in NEST, (1) a base can not belong to more than one arc and (2) arcs can not cross. In the class CROS the constraint (2) is relaxed, and then relaxing constraint (1) from the class CROS gives the class UNLIM. It is then natural to consider an alternative path from NEST to UNLIM, by first relaxing the constraint (2), then the constraint (1).

We define the extension MULT of a given class, other than UNLIM and PLAIN, by allowing a base to belong to more than one arc. For example the extension MULT of the class NEST, denoted by NMULT, contains the arc-annotated sequences such that arcs do not cross but a base can belong to more than one arc. Similarly we denote by SMULT the MULT extension of STEM. With this point of view, UNLIM can in fact be seen as CMULT, that is the extension MULT of CROS (Fig. 7).

In Section 4, we describe an algorithm that solves the problem ALIGN(NEST,NEST;NMULT). As far as we know, the only other alignment algorithm that considered an NMULT super-sequence did not consider all edit operations and a complete score scheme [13].

3.2 Properties of alignment sequences between nested arc-annotated sequences.

We present now a few properties of optimal alignment sequences between nested arc-annotated sequences. They allow to get a better understanding of the kind of alignments and alignment sequences that can be obtained in ALIGN(NEST,NEST;C) where C is a class of the hierarchy of arc-annotated sequences containing NEST (*i.e.* NEST, NMULT, CROS or UNLIM).

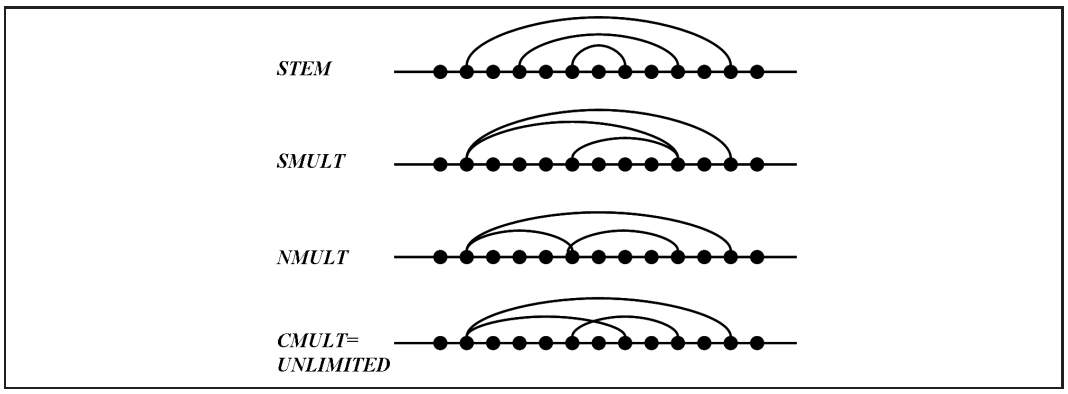

Fig. 7. Extension of the hierarchy of arc-annotated sequences.

Property 1. Let C be a class of the hierarchy of arc-annotated sequences containing NEST, E an optimal C-alignment sequence between two nested arc-annotated sequences $A_1 = (S_1, P_1)$ and $A_2 = (S_2, P_2)$, with a complete score scheme. Let E_1 and E_2 be the decomposition of E and x a base in A_1 that is not deleted in E_2.

1. x can be involved in at most two complex arc operations in E: one in E_1, that creates an arc containing x, and one in E_2, that does not delete the arc created by E_1 if any.
2. If C is either NEST or CROS (*i.e.* not a MULT extension), then x can be involved in at most one complex arc operation in E.

Proof. Let $A_3 = (S_3, P_3)$ be the super-sequence induced by E.

Assume the base x is involved in $k \geq 1$ complex arc operations in E_1 (AC or ACo). Then x belongs to at least k arcs in A_3 (and at most $k+1$ as A_1 is nested). As A_2 is nested, at least $k-1$ of these arcs have to be removed during E_2, either by AB or AA operations, that do not delete x by hypothesis. Let y be a base that defines an arc with x in A_3, that was created in E_1 and removed in E_2. If this arc was created by AC (resp. ACo) and removed by AB (resp. AA), then E is not optimal, as the second operation cancels the first one. If this arc was created by AC (resp. ACo), and is removed by an AA (resp. AB), then the same result would have been obtained by a single BD in E_2 (resp. BI in E_1). In all cases, with a complete score scheme E is not optimal, and then, if $k > 1$, there is a contradiction with the initial hypothesis that E is optimal. This proves that x can be involved in at most one complex operation, e_1, in E_1, that creates an arc, and one in E_2 that does not delete the arc created by e_1.

Assume now that A_3 belongs to NEST or CROS. If x is involved in a complex arc operation e_1 in E_1 (AC or ACo), then x was not belonging to an arc in A_1, and a complex operation in E_2 that would involve x (AB or AA) would delete the arc created by e_1. From our above argument, this contradicts the fact that E is optimal. If x is involved in a complex arc operation e_2 in E_2, then x was in an arc in A_1 and was not involved in a complex operation in E_1, as it belongs to a single arc in A_3. □

Example 1. Let $S_1 = abc$, $P_1 = \{(1,2)\}$, $S_2 = abc$ and $P_2 = \{(1,3)\}$. Depending of the cost scheme, an optimal alignment sequence between (S_1, P_1) and (S_2, P_2) could be an AC that creates the arc $(1,3)$ followed by an AB of the arc $(1,2)$ in (S_1, P_1). This alignment sequence induces a super-sequence where the base a is incident to two arcs, and then does not belong to NEST. An alignment

sequence that induces a nested super-sequence would be for example an ACo: $\lambda, c \rightarrow (a, c)$ followed by AA: $(a, b) \rightarrow \lambda, b$, which implies that the base a in S_1 was deleted and the base a in S_2 was inserted.

Property 2. Let C be a class of the hierarchy of arc-annotated sequences, E be an optimal C-alignment sequence between two nested arc-annotated sequences $A_1 = (S_1, P_1)$ and $A_2 = (S_2, P_2)$, with a complete score scheme. If C is NEST or NMULT (*i.e.* no crossing) then no operation in E that creates an arc can create a crossing between this arc and another arc.

Proof. This follows immediately from the fact that every arc created during E_1 belongs to A_3 the super-sequence induced by E. □

Example 2. Let $S_1 = abcde$, $P_1 = \{(2, 4)\}$, $S_2 = abcde$ and $P_2 = \{(1, 3)\}$. Depending of the cost scheme, an optimal alignment sequence between (S_1, P_1) and (S_2, P_2) could be an AC that creates the arc $(1, 3)$ followed by an AB that breaks the arc $(2, 4)$. This alignment sequence induces a crossing between two arcs. A sequence that does not involve a crossing would require to insert an a between b and c, then to create an arc between this a and c, then to insert a b after this a, then to alterate the arc (b, d) to delete the b and finally to delete the first a.

It follows from these two properties that optimal NEST-alignment sequences suffer from two limitations: bases can not be involved in more than one complex operation and no arc can be created that cross another arc. As shown in Property 1 considering NMULT-alignment sequences (or SMULT-alignment sequences when dealing with the comparison of two sequences belonging to the STEM class) overcomes the first limitation, while considering CROS-alignment sequences does not help with respect to this limitation. Conversely, CROS-alignment sequences do not suffer from the second limitation, unlike NMULT-alignment sequences.

3.3 Conservative edit distance between RNA stem-loops.

The notion of conservative edit distance between two RNA stem-loops was introduced in [11] together with a polynomial time and space algorithm to compute this distance. In this section we show that computing this distance is in fact equivalent to the problem ALIGN(STEM,STEM; STEM).

In [11], the conservative edit distance was defined using a representation of RNA stem-loops as ordered labeled trees where each base pair is represented by an internal node labeled by this base pair and each unpaired base by a leaf labeled by this base. Such trees are clearly equivalent to STEM arc-annotated sequences.

A *conservative edit sequence* between two STEM arc-annotated sequences A_1 and A_2 is an edit sequence between A_1 and A_2 satisfying the following additional constraints:

- every edit operation involves at least one base that belongs to A_1 or A_2,
- after the application of each edit operation, the resulting arc-annotated sequence belongs to STEM,
- each base is involved in at most one edit operation among {AB, AA, AC, ACo} and in at most one edit operation {BS, AS}.

The *conservative edit distance* between two STEM arc-annotated sequences A_1 and A_2 is the minimal cost of a conservative edit sequence between A_1 and A_2.

Proposition 2. *The conservative edit distance between two* STEM *arc-annotated sequences A_1 and A_2 is equal to $d_{\text{STEM}}(A_1, A_2)$.*

Proof. As a conservative edit sequence E_c between A_1 and A_2 is an edit sequence, there is an alignment sequence E_a between A_1 and A_2 such that $w(E_a) = w(E_c)$. E_a is necessarily a STEM-alignment sequence, since after the application of every edit operation of E_a the intermediate structure is still a STEM arc-annotated sequence.

Conversely, a given STEM-alignment between A_1 and A_2 implies an edit sequence between A_1 and A_2 that clearly satisfies the constraints of a conservative edit sequence. □

4 An algorithm for Align(Nest,Nest; NMult)

Theorems 3 and 4 indicate that ALIGN(NEST,NEST; NEST) is the most general problem of comparison between nested arc-annotated sequences that can be solved in polynomial time in the framework of the classical hierarchy of arc-annotated sequences. We introduced in Section 3 a new class, NMULT that generalizes the class NEST but is not comparable with CROS. Our main result is that ALIGN(NEST,NEST; NMULT) can be solved in polynomial time, with the same asymptotic worst-case time and space complexity than ALIGN(NEST,NEST; NEST). We describe in the remaining of this section a dynamic programming algorithm for solving the alignment problem ALIGN(NEST,NEST; NMULT). From now, all alignments we consider belong to the class of NMULT-alignments, and we call them only alignments in general.

4.1 Preliminaries

Indexing pairs. Similarly to other algorithms to compare arc-annotated sequences [11, 14], the dynamic programming tables we use in our algorithm are indexed by pairs of sub-sequences of the two considered arc-annotated sequences. These sub-sequences are defined in terms of *indexing pairs*, that can be related to the hierarchy of arc-annotated sequences: given an arc-annotated sequence $A = (S, P)$, with S of length n, and a class C of the hierarchy of arc-annotated sequences, an ordered pair of integers $I = (x, y)$, with $1 \leq x \leq y \leq n$, is an *indexing pair of type* C *for* A if the arc-annotated sequence $A' = (S, P \cup \{(x, y)\})$ belongs to C.

An indexing pair $I = (x, y)$ of $A = (S, P)$ defines an arc-annotated sub-sequence of A, denoted by $A^I = (S^I, P^I)$, obtained from A by deleting from S all the bases that do not belong to $S[x..y]$ and from P all the arcs that have at least one of their bases that does not belong to $S[x..y]$.

For technical reasons, we also introduce a special indexing pair, called the *empty indexing pair*, denoted $\emptyset$; $A^\emptyset$ is the empty arc-annotated sequence. The set of indexing pairs of type C of an arc-annotated sequence A, augmented with the empty indexing pair, is denoted by $I_{\text{C}}(A)$.

Lemma 1. *Let A_1 and A_2 be two nested arc-annotated sequences,* C *a class of the hierarchy of arc-annotated sequences and I an indexing pair that is not of type* C*. Any alignment sequence between A_1 and A_2 that contains an operation AC creating an arc between bases x and y of A_1 is not a* C*-alignment sequence.*

Proof. By definition of an alignment sequence, that starts with BI, AI, ACo and AC operations, as x and y are bases of A_1, we can assume without loss of generality that the AC operation creating an arc between x and y is the first edit operation performed on A_1. It then follows from the definition of the type of an indexing pair, that the resulting arc-annotated sequence does not belong to the

class C. As no base or arc will be deleted to obtain A_3, this implies that A_3 does not belong to C too, and then the whole alignment sequence is not a C-alignment sequence. □

Lemma 1 implies that to compute the C-alignment distance, we need to consider only indexing pairs of type C as more general indexing pairs would lead to wrong alignment sequences. From now we assume that all indexing pairs are of type NMult; in particular $I(A)$ means $I_{NMULT}(A)$.

Dynamic programming tables. Four dynamic programming tables , denoted by D, D_f, D_l and D_{fl} are used to compute an alignment between two nested arc-annotated sequences $A_1 = (S_1, P_1)$ and $A_2 = (S_2, P_2)$. The tables D, D_f, D_l and D_{fl} are two-dimensional tables indexed by pairs (I, J) such that $I \in I(A_1)$ and $J \in I(A_2)$. $J = \emptyset$.

- The cell $D[I, J]$ contains the minimal cost of an alignment between the arc-annotated sub-sequences A_1^I and A_2^J.
- Let $I = (x, y)$ and $J = (p, q)$. The cell $D_f[I, J]$ (resp. $D_l[I, J]$; $D_{fl}[I, J]$) contains the minimal cost of a alignment M between the arc-annotated sub-sequences A_1^I and A_2^J such that $S_1[x]$ is aligned with $S_2[p]$ in M (resp. $S_1[y]$ is aligned with $S_2[q]$ in M; $S_1[x]$ is aligned with $S_2[p]$ in M and $S_1[y]$ is aligned with $S_2[q]$ in M).
- $D_f[I, J]$, $D_l[I, J]$ and $D_{fl}[I, J]$ are not defined if exactly one of the two indexing pairs I and J is equal to $\emptyset$.
- $D_{fl}[I, J]$ is not defined if exactly one of the two indexing pairs I or J is of length 1.

We now discuss briefly why we use several dynamic programming tables. First, note that in an optimal alignment M, it follows from the definition of NMult that, for an arc (i_1, i_2) of P_1 and an arc (j_1, j_2) of P_2, it is possible that $S_1[i_1]$ is aligned with $S_2[j_1]$ in M while $S_1[i_2]$ is not aligned with $S_2[j_2]$ in M, which can not happen in an optimal Cros-alignment. This implies that when considering the possible configurations in an alignment for an arc (x, y), either in P_1 or in P_2, the following can happen: one of the bases of (x, y), say x, is aligned with a base of the other sequence that belongs to an arc, says the base p of the arc (p, q), and the second base of (x, y) is not aligned with the second base of (p, q). In such a configuration, we do not know, at this point, which edit operation will be induced for the arc (p, q) in the alignment, as long as we do not take a decision for its second base q. Moreover, when we take a decision for q, that is either to align it with a base or to align it with $-$, we need to remember the decision taken for p in order to evaluate the cost of the edit operation on the arc (p, q). The tables D_f, D_l and D_{fl} are used to record such partial alignment decisions taken on arcs.

4.2 Filling up the dynamic programming tables

We first describe how to fill the first values of the tables D, D_f, D_l, D_{fl}, using the value ∞ for the cells that are not defined.

Lemma 2. *For every non-empty indexing pair $I \in (A_1)$ and $J \in I(A_2)$*

$$D[I, \emptyset] = \sum_{i \in U(A_1^I)} w(i \rightarrow \lambda) + \sum_{(i_1, i_2) \in P(A_1^I)} w((i_1, i_2) \rightarrow \lambda, \lambda).$$
$$D[\emptyset, J] = \sum_{j \in U(A_2^J)} w(\lambda \rightarrow j) + \sum_{(j_1, j_2) \in P(A_2^J)} w(\lambda, \lambda \rightarrow (j_1, j_2)).$$
$$D_f[I, \emptyset] = D_f[\emptyset, J] = D_l[I, \emptyset] = D_l[\emptyset, J] = D_{fl}[I, \emptyset] = D_{fl}[\emptyset, J] = \infty.$$

Moreover

$$D[\emptyset, \emptyset] = D_f[\emptyset, \emptyset] = D_l[\emptyset, \emptyset] = D_{fl}[\emptyset, \emptyset] = 0.$$

Proof. Direct consequence of the definitions of D, D_f, D_l and D_{fl}. □

Before describing the main dynamic programming equations, we need to define the notion of partition of an indexing pair. Given an indexing pair $I = (x, y)$ of an arc-annotated sequence $A = (S, P)$, we say that two indexing pairs I_1 and I_2 partition I, denoted by $I_1 + I_2 = I$, if either one of them is $\emptyset$ and the other one is equal to I, or $I_1 = (x, z)$ and $I_2 = (z+1, y)$ with $x \leq z \leq y-1$.

In order to shorten the presentation of the equations, we use in Lemmas 3, 4, 5 and 6 the following notation, that hold for $I = (x, y)$ and $J = (p, q)$ two non-empty indexing pairs of two arc-annotated sequences A_1 and A_2: z_1 (resp. r_1) is such that $x \leq z_1 \leq y$ (resp. $p \leq r_1 \leq q$) and $(x, z_1) \in P(A_1)$ (resp. $(p, r_1) \in P(A_2)$). We also assume that if an indexing pair $I = (x, y)$ is such that $x > y$, then I takes value $\emptyset$.

Lemma 3. *Given $I = (x, y)$ and $J = (p, q)$ two indexing pairs respectively of A_1 and A_2.*

1. *If $x \in P_o(A_1^I)$ and $p \in P_o(A_2^J)$,*
   $D[I, J] = \min\{AS_1, AB_1, AB_2, AC_1, AC_2, AA_1, AA_2, AA_3, ACo_1, ACo_2, ACo_3, AD_1, AI_1\}$.
2. *If $x \in P_o(A_1^I)$ and $p \in U(A_2^J)$, $D[I, J] = \min\{AB_1, AB_2, AA_1, AA_2, AA_3, AD_1, BI_1\}$.*
3. *If $x \in U(A_1^I)$ and $p \in P_o(A_2^J)$, $D[I, J] = \min\{AC_1, AC_2, ACo_1, ACo_2, ACo_3, AI_1, BD_1\}$.*
4. *If $x \in U(A_1^I)$ and $p \in U(A_2^J)$, $D[I, J] = \min\{BS_1, BD_1, BI_1\}$.*

Where

$$
\begin{aligned}
BS_1 &= w(x \rightarrow p) + D[(x+1,y),(p+1,q)] \\
BD_1 &= w(x \rightarrow \lambda) + D[(x+1,y),(p,q)] \\
BI_1 &= w(\lambda \rightarrow p) + D[(x,y),(p+1,q)] \\
AS_1 &= w((x,z_1) \rightarrow (p,r_1)) + D[(x+1,z_1-1),(p+1,r_1-1)] + D[(z_1+1,y),(r_1+1,q)]\} \\
AD_1 &= \min_{(J_1,J_2)\in Q_1(J)}\{w((x,z_1) \rightarrow \lambda,\lambda) + D[(x+1,z_1-1),J_1] + D[(z_1+1,y),J_2]\} \\
AI_1 &= \min_{(I_1,I_2)\in Q_1(I)}\{w(\lambda,\lambda \rightarrow (p,r_1)) + D[I_1,(p+1,r_1-1)] + D[I_2,(r_1+1,q)]\} \\
AB_1 &= \min_{(J_1,J_2)\in Q_1(J)}\{w((x,z_1) \rightarrow x,z_1) + D_f[(x,z_1-1),J_1)] + D_f[(z_1,y),J_2]\} \\
AB_2 &= \min_{(J_1,J_2)\in Q_2(J)}\{D_{fl}[(x,z_1),J_1] + D[(z_1+1,y),J_2]\} \\
AC_1 &= \min_{(I_1,I_2)\in Q_1(I)}\{w(p,r_1 \rightarrow (p,r_1)) + D_f[I_1,(p,r_1-1)] + D_f[I_2,(r_1,q)]\} \\
AC_2 &= \min_{(I_1,I_2)\in Q_2(I)}\{D_{fl}[I_1,(p,r_1)] + D[I_2,(r_1+1,q)]\} \\
AA_1 &= \min_{(J_1,J_2)\in Q_1(J)}\{w((x,z_1) \rightarrow x,\lambda) + D_f[(x,z_1-1),J_1] + D[(z_1+1,y),J_2]\} \\
AA_2 &= \min_{(J_1,J_2)\in Q_1(J)}\{w((x,z_1) \rightarrow \lambda,z_1) + D[(x+1,z_1-1),J_1] + D_f[(z_1,y),J_2]\} \\
AA_3 &= \min_{(J_1,J_2)\in Q_2(J)}\{w((x,z_1) \rightarrow \lambda,z_1) + D_l[(x+1,z_1),J_1] + D[(z_1+1,y),J_2]\} \\
ACo_1 &= \min_{(I_1,I_2)\in Q_1(I)}\{w(p,\lambda \rightarrow (p,r_1)) + D_f[I_1,(p,r_1-1)] + D[I_2,(r_1+1,q)]\}\} \\
ACo_2 &= \min_{(I_1,I_2)\in Q_1(I)}\{w(\lambda,r_1 \rightarrow (p,r_1)) + D[I_1,(p+1,r_1-1)] + D_f[I_2,(r_1,q)]\} \\
ACo_3 &= \min_{(I_1,I_2)\in Q_2(I)}\{w(\lambda,r_1 \rightarrow (p,r_1)) + D_l[I_1,(p+1,r_1)] + D[I_2,(r_1+1,q)]\}
\end{aligned}
$$

and

$$
\begin{aligned}
Q_1(I) &= \{(I_1, I_2) \in I(A)^2 \mid I_1 + I_2 = I, \nexists (i_1, i_2) \in P(A^I) \mid i_1 \in I_1, i_2 \in I_2\} \\
Q_2(I) &= \{(I_1, I_2) \in Q_1(I) \mid I_1 = (x, z), z \in P_e(A^I)\}.
\end{aligned}
$$

Proof. The principle of these equations is similar to the one used to define the dynamic programming equations of algorithms computing an alignment between non-annotated sequences. Indeed, we we have to consider the three following configurations between x and p: x and p are aligned together, x is deleted (aligned with $-$) before p, or p is inserted (aligned with $-$) before x. The cases where x or p are aligned with bases but not together are considered through different indexing pairs. The main difference with non-annotated sequences alignment relies in the fact that x and/or p can belong to

an arc, which is why we consider four cases: both (resp. none) belong to an arc in case 1 (resp. case 4), only x (resp. p) belongs to an arc in case 2 (resp. case 3).

Let M be an alignment between the arc-annotated sub-sequences A_1^I and A_2^J, defined by $I = (x, y)$ and $J = (p, q)$. We describe below all possible configurations that can be found in M if either x and p are aligned together or one of them is aligned with $-$ before the other. These configurations are illustrated in Figure 8.

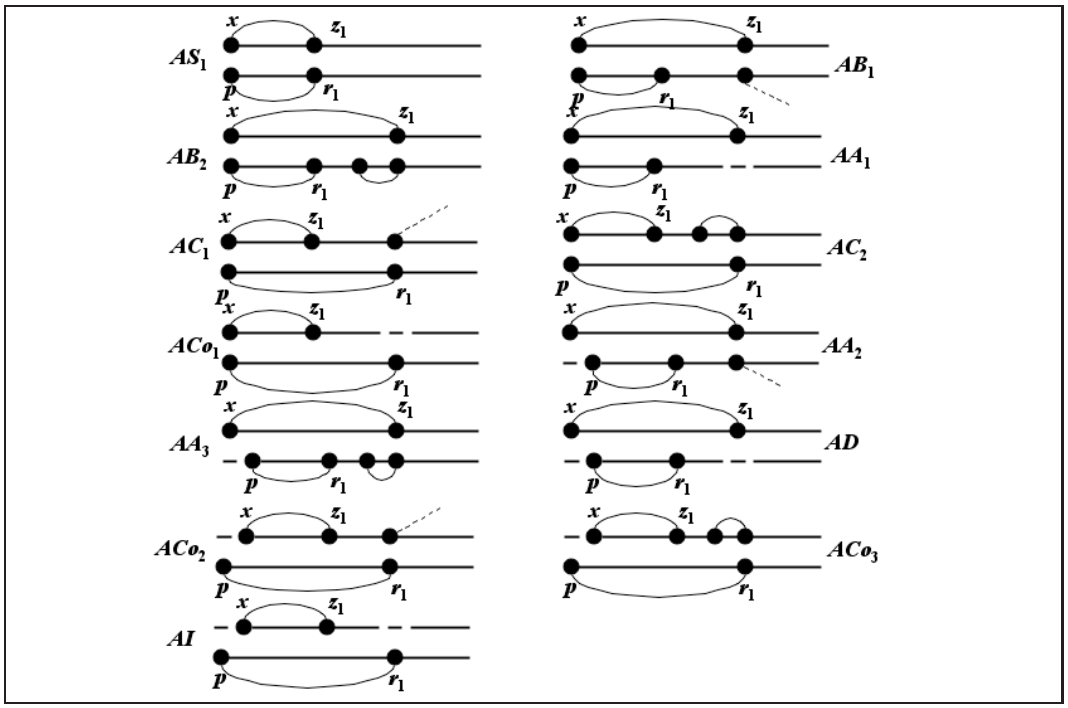

Fig. 8. Illustration of all possible configurations described in the proof of Lemma 3. A dashed line leaving a base indicates that this base can be either unpaired or belong to an arc, with the relative position (either on the left or on the right of the considered base) of the other extremity of this arc being indicated by the direction of the dashed line.

1. Assume $x \in P_o(A_1^I)$ and $p \in P_o(A_2^J)$.
   (a) Assume x and p are aligned together in M. There are three cases depending on the status of z_1 and/or r_1 in M.
      i. If z_1 is aligned with r_1 then $(x, z_1) \in AS(M)$, as defined in (1) and the inside (resp. outside) of the arc (x, z_1) has to be aligned with the inside (resp. outside) of (p, r_1), which is described by AS_1. In the case where $z_1 = y$ and/or $r_1 = q$, the initialization of table D described in Lemma 2 ensures that AS_1 still holds. Moreover, as (x, z_1) and (p, r_1) are arcs and I and J are indexing pairs of type NMult, the indexing pairs considered in AS_1 belong too to NMult.
      ii. If z_1 is aligned, either with $-$ or with a base of A_2, but after r_1 in M, here again there are three cases.
         A. z_1 is aligned with a base $r \notin P_e(A_2^J)$, which implies that $(x, z_1) \in AB(M)$.

This case is accounted by AB_1, where we add to the cost of M the cost of an arc-breaking and aligns the inside and outside of the arc (x, z_1) with all possible partitions of J into two NMULT indexing pairs, as defined by Q_1, recording that both x and z_1 are aligned with bases of A_2 by considering the table D_f in both cases. Note that, by considering D_f in $D_f[(z_1,y),J_2]$, we record that r is aligned with a base, which will be used if $r \in P_o(A_2^J)$ to account for the cost of the operation associated to the arc that contains r.

B. z_1 is aligned with a base $r \in P_e(A_2^J)$, which implies here again that $(x, z_1) \in AB(M)$. As M belongs to NMULT, the bases of A_1^I between x and z_1 (resp. after z_1) can only be aligned with the bases of A_2^J between p and r (resp. after r), and there can be no arc in A_2^J between these two sets of bases. This configuration corresponds to AB_2, where adding the cost induced by the fact that $(x, z_1) \in AB(M)$, will be done in the call to $D_{fl}[(x, z_1), J_1]$, that records that both bases x and z_1 are aligned with bases of A_2. By construction, all indexing pairs defined by Q_2 are of type NMULT.

C. z_1 is aligned with $-$ in M, still after r_1 (there is a base in A_2^J that is between p and r_1, r_1 included, which is aligned in M with a base of A_1^I that is before z_1), which implies that $(x, z_1) \in AA(M)$. This case is handled by AA_1 that records the cost of an arc-altering for (x, z_1), as this arc disappears from the two subsequent indexing pairs of A_1 defining the two considered sub-problems, and ensures that no arc of A_2^J will cross (x, z_1) in M, by definition of Q_1.

iii. The case where z_1 is aligned in M either with a base or with $-$ but before r_1 is handled in a symmetrical way by AC_1, AC_2 and ACo_1, where the cost of the operation on the arc of A_2 is accounted for, the cost of the operation on the arc (x, z_1) being postponed to a later sub-problem.

Clearly all previous cases hold when $z_1 = y$ and/or $r_1 = q$, due to the initialization of the tables described in Lemma 2.

(b) We now assume that x is then aligned with $-$ in M before p. There are three cases, that we discuss more briefly as the arguments for their correctness are very similar to the previous ones.

i. If z_1 is aligned with a base $r \in P_e(A_2^J)$, then the arc-altering that breaks the arc (x, z_1) is handled by AA_3.

ii. If z_1 is aligned with a base $r \notin P_e(A_2^J)$, then the arc-altering that breaks the arc (x, z_1) is handled by AA_2.

iii. If z_1 is aligned with a base $-$, then the arc deleting that deletes the arc (x, z_1) is handled by AD.

(c) The case where p is aligned with $-$ in M before x is handled in a symmetrical way to the previous one by ACo_2, ACo_3 and AI.

2. Assume $x \in P_o(A_1^I)$ and $p \in U(A_2^J)$. This case can be deduced from the previous one by removing all edit operations that assume that p belongs to an arc in A_2^J, namely arc-creation (AC_1 and AC_2), arc-substitution (AS_1), arc-completing (ACo_1, ACo_2 and ACo_3) and arc-inserting (AI_1).
3. Assume $x \in U(A_1^I)$ and $p \in P_o(A_2^J)$. This case can be deduced from the first one by removing all edit operations that assume that x belongs to an arc in A_1^I, namely arc-breaking (AB_1 and AB_2), arc-substitution (AS_1), arc-altering (AA_1, AA_2 and AA_3) and arc-deleting (AD_1).
4. Finally, if $x \in U(A_1^I)$ and $p \in U(A_2^J)$, there are three cases, as in the classical string alignment problem: x and p are aligned together in M (BS_1), x is aligned with $-$ before p (BD), and p is aligned with $-$ before x (BI).

□

Lemma 4. *Given $I = (x, y)$ and $J = (p, q)$ two indexing pairs respectively of A_1 and A_2.*

1. *If $x \in P_o(A_1^I)$ and $p \in P_o(A_2^J)$, $D_f[I, J] = \min\{AS_1, AB_1, AB_2, AC_1, AC_2, AA_1, ACo_1\}$.*
2. *If $x \in P_o(A_1^I)$ and $p \in U(A_2^J)$, $D_f[I, J] = \min\{AB_1, AB_2, AA_1\}$.*
3. *If $x \in U(A_1^I)$ and $p \in P_o(A_2^J)$, $D_f[I, J] = \min\{AC_1, AC_2, ACo_1\}$.*
4. *If $x \in U(A_1^I)$ and $p \in U(A_2^J)$, $D_f[I, J] = BS_1$.*

Where AB_1, AB_2, AA_1, AC_1, AC_2, ACo_1 and BS_1 are defined as in Lemma 3.

Proof. The proof follows directly from the proof of Lemma 3 by considering only the cases where x and p are aligned together. □

Lemma 5. *Given $I = (x, y)$ and $J = (p, q)$ two indexing pairs respectively of A_1 and A_2.*

1. *If $x \in P_o(A_1^I)$ and $p \in P_o(A_2^J)$,*
   $D_l[I, J] = \min\{AS_2, AB_3, AB_4, AC_3, AC_4, AA_4, AA_5, AA_6, ACo_4, ACo_5, ACo_6, AD_2, AI_2\}$.
2. *If $x \in P_o(A_1^I)$ and $p \in U(A_2^J)$, $D_l[I, J] = \min\{AB_3, AB_4, AA_4, AA_5, AA_6, AD_2, BI_2\}$.*
3. *If $x \in U(A_1^I)$ and $p \in P_o(A_2^J)$, $D_l[I, J] = \min\{AC_3, AC_4, ACo_4, ACo_5, ACo_6, AI_2, BD_2\}$.*
4. *If $x \in U(A_1^I)$ and $p \in U(A_2^J)$, $D_l[I, J] = \min\{BS_2, BD_2, BI_2\}$.*

Where

$$
\begin{aligned}
BS_2 &= w(x \to p) + D_l[(x{+}1,y),(p{+}1,q)] \\
BD_2 &= w(x \to \lambda) + D_l[(x{+}1,y),(p,q)] \\
BI_2 &= w(\lambda \to p) + D_l[(x,y),(p{+}1,q)] \\
AS_2 &= w((x,z_1) \to (p,r_1)) + D[(x{+}1,z_1{-}1),(p{+}1,r_1{-}1)] + D_l[(z_1{+}1,y),(r_1{+}1,q)] \\
AD_2 &= \min_{(J_1,J_2)\in Q_3(J)}\{w((x,z_1) \to \lambda,\lambda) + D[(x{+}1,z_1{-}1),J_1] + D_l[(z_1{+}1,y),J_2]\} \\
AI_2 &= \min_{(I_1,I_2)\in Q_3(I)}\{w(\lambda,\lambda \to (p,r_1)) + D[I_1,(p{+}1,r_1{-}1)] + D_l[I_2,(r_1{+}1,q)]\} \\
AB_3 &= \min_{(J_1,J_2)\in Q_1(J)}\{w((x,z_1) \to x,z_1) + D_f[(x,z_1{-}1),J_1)] + D_{fl}[(z_1,y),J_2]\} \\
AB_4 &= \min_{(J_1,J_2)\in Q_2(J)}\{D_{fl}[(x,z_1),J_1] + D_l[(z_1{+}1,y),J_2]\} \\
AC_3 &= \min_{(I_1,I_2)\in Q_1(I)}\{w(p,r_1 \to (p,r_1)) + D_f[I_1,(p,r_1{-}1)] + D_{fl}[I_2,(r_1,q)]\} \\
AC_4 &= \min_{(I_1,I_2)\in Q_2(I)}\{D_{fl}[I_1,(p,r_1)] + D_l[I_2,(r_1{+}1,q)]\} \\
AA_4 &= \min_{(J_1,J_2)\in Q_3(J)}\{w((x,z_1) \to x,\lambda) + D_f[(x,z_1{-}1),J_1] + D_l[(z_1{+}1,y),J_2]\} \\
AA_5 &= \min_{(J_1,J_2)\in Q_1(J)}\{w((x,z_1) \to \lambda,z_1) + D[(x{+}1,z_1{-}1),J_1] + D_{fl}[(z_1,y),J_2]\} \\
AA_6 &= \min_{(J_1,J_2)\in Q_2(J)}\{w((x,z_1) \to \lambda,z_1) + D_l[(x{+}1,z_1),J_1] + D_l[(z_1{+}1,y),J_2]\} \\
ACo_4 &= \min_{(I_1,I_2)\in Q_3(I)}\{w(p,\lambda \to (p,r_1)) + D_f[I_1,(p,r_1{-}1)] + D_l[I_2,(r_1{+}1,q)]\}\} \\
ACo_5 &= \min_{(I_1,I_2)\in Q_1(I)}\{w(\lambda,r_1 \to (p,r_1)) + D[I_1,(p{+}1,r_1{-}1)] + D_{fl}[I_2,(r_1,q)]\} \\
ACo_6 &= \min_{(I_1,I_2)\in Q_2(I)}\{w(\lambda,r_1 \to (p,r_1)) + D_l[I_1,(p{+}1,r_1)] + D_l[I_2,(r_1{+}1,q)]\}
\end{aligned}
$$

$$Q_3(I) = \{(I_1, I_2) \in Q_1(I) \mid I_2 \neq \emptyset\}$$

and Q_1 and Q_2 are defined as in Lemma 3

Proof. Similar to the proof of Lemma 3 but we impose that y and q are aligned together. This forbids, in some cases, to have a second indexing pair, in the partition of an indexing pair, that is empty, which is handled by the definition of Q_3. In the case where $z_1 = y$ and/or $r_1 = q$, the initialization of tables D_l and D_{fl} with ∞ when exactly one of the two indexing pairs is empty ensures that y and q are aligned together. □

Lemma 6. *Given $I = (x, y)$ and $J = (p, q)$ two indexing pairs respectively of A_1 and A_2.*

1. *If $x \in P_o(A_1^I)$ and $(x, y) \notin P(A_1^I)$ and $p \in P_o(A_2^J)$ and $(p, q) \notin P(A_2^J)$, $D_{fl}[I, J] = \min\{AS_2, AB_3, AB_4, AC_3, AC_4, AA_4, ACo_4\}$.*
2. *If $x \in P_o(A_1^I)$ and $(x, y) \notin P(A_1^I)$ and $p \in U(A_2^J)$, $D_{fl}[I, J] = \min\{AB_3, AB_4, AA_4\}$.*
3. *If $x \in U(A_1^I)$ and $p \in P_o(A_2^J)$ and $(p, q) \notin P(A_2^J)$, $D_{fl}[I, J] = \min\{AC_3, AC_4, ACo_4\}$.*
4. *If $x \in U(A_1^I)$ and $p \in U(A_2^J)$, $D_{fl}[I, J] = BS_2$,*
5. *If $(x, y) \in P(A_1^I)$ and $(p, q) \in P(A_2^J)$, $D_{fl}[I, J] = AS_2$.*
6. *If $(x, y) \in P(A_1^I)$ and $p \in P_o(A_2^J)$ and $(p, q) \notin P(A_2^J)$, $D_{fl}[I, J] = \min\{ABAC_1, ABAC_2, ABACo\}$.*
7. *If $(x, y) \in P(A_1^I)$ and $p \in U(A_2^J)$, $D_{fl}[I, J] = ABBS$.*
8. *If $x \in P_o(A_1^I)$ and $(x, y) \notin P(A_1^I)$ and $(p, q) \in P(A_2^J)$, $D_{fl}[I, J] = \min\{ACAB_1, ACAB_2, ACAA\}$.*
9. *If $x \in U(A_1^I)$ and $(p, q) \in P(A_2^J)$, $D_{fl}[I, J] = ACBS$.*

Where

$$
\begin{aligned}
ABAC_1 &= \min_{(I_1,I_2)\in Q_4(I)}\{w((x,z_1)\rightarrow x,z_1) + w(p,r_1\rightarrow(p,r_1)) + D_f[I_1,(p,r_1-1)] + D_{fl}[I_2,(r_1+1,q)]\}\\
ABAC_2 &= \min_{(I_1,I_2)\in Q_5(I)}\{w((x,z_1)\rightarrow x,z_1) + D_{fl}[I_1,(p,r_1)] + D_l[I_2,(r_1+1,q)]\}\\
ACAB_1 &= \min_{(J_1,J_2)\in Q_4(J)}\{w(p,r_1\rightarrow(p,r_1)) + w((x,z_1)\rightarrow x,z_1) + D_f[(x,z_1-1),J_1] + D_{fl}[(z_1,y),J_2]\}\\
ACAB_2 &= \min_{(J_1,J_2)\in Q_5(J)}\{w((p,r_1)\rightarrow p,r_1) + D_{fl}[(x,z_1),J_1] + D_l[(z_1+1,y),J_2]\}\\
ABACo &= \min_{(I_1,I_2)\in Q_4(I)}\{w((x,z_1)\rightarrow x,z_1) + w(p,\lambda\rightarrow(p,r_1)) + D_f[I_1,(p,r_1-1)] + D_l[I_2,(r_1+1,q)]\}\\
ACAA &= \min_{(J_1,J_2)\in Q_4(J)}\{w(p,r_1\rightarrow(p,r_1)) + w((x,z_1)\rightarrow x,\lambda) + D_f[(x,z_1-1),J_1] + D_l[(z_1+1,y),J_2]\}\\
ABBS &= w((x,z_1)\rightarrow x,z_1) + w(x\rightarrow p) + D_l[(x+1,y),(p+1,q)]\\
ACBS &= w(p,r_1\rightarrow(p,r_1)) + w(x\rightarrow p) + D_l[(x+1,y),(p+1,q)]
\end{aligned}
$$

and

$$
\begin{aligned}
Q_4(I) &= \{(I_1, I_2) \in I(A)^2 \mid I_1 + I_2 = I,\ \nexists (i_1, i_2) \in P(A^I)\backslash\{(x, y)\} \mid i_1 \in I_1, i_2 \in I_2\}\\
Q_5(I) &= \{(I_1, I_2) \in Q_4(I) \mid I_1 = (x, z), z \in P_e(A^I)\}
\end{aligned}
$$

Proof. Formulas 1 to 4 correspond to the cases where neither (x, y) is an arc of A_1^I nor is (p, q) an arc of A_2^J. They follow from Lemma 5 where we keep only the cases where x and p are aligned together.

Formulas 5 to 9 correspond to the cases where either only (x, y) is an arc of A_1^I (formulas 6 and 7), or (p, q) is an arc of A_2^J (formulas 8 and 9), or both (formula 5). They are illustrated (but formula 5 that corresponds to AS_2) in Figure 9.

The general principle for these formulas is that as one of the indexing pairs is an arc, and both its bases are matched, then we need to account for at least one arc operation, and possibly one other operation on one of the bases x and p and the arc that contains it any. Q_4 and Q_5 correspond to Q_1 and Q_2 when we allow an arc between the two indexing pairs resulting from a partition, composed of the two extremities of this indexing pair. From these points, the proof for formulas 5 to 9 is similar to the previous cases. □

Before describing the complete algorithm for the computation of the distance between two arc-annotated sequences A_1 and A_2, we study in the set of indexing pairs that need to be considered. Given an arc-annotated sequence A of length n, we denote $H(A)$ the set of indexing pairs of A (of type NMult) which end by n, by the end of an arc or by a base just before the end of an arc:

$$H(A) = \{(x, y) \in I(A) \mid y = n \text{ or } \exists\ w \leq x \text{ such that } (w, y) \in P(A) \text{ or } (w, y + 1) \in P(A)\}.$$

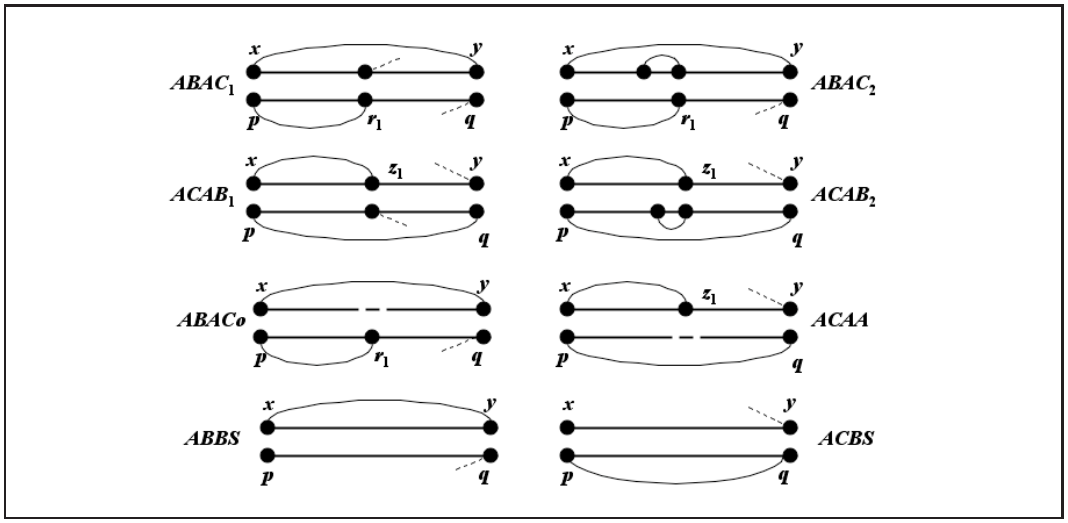

Fig. 9. Illustration of some possible configurations described in the proof of Lemma 6.

Lemma 7. *Let (I, J) be a couple of* NMult *indexing pairs of two arc-annotated sequences A_1 and A_2, that belong to $H(A_1)\times I(A_2)\cup I(A_1)\times H(A_2)$. Then, in Lemma 3, every couple of indexing pairs (I', J') that is involved in the computation of $D[I, J]$ belongs to $H(A_1) \times I(A_2) \cup I(A_1) \times H(A_2)$.*

Proof. We first consider the couples of indexing pairs that are immediately required to compute $D[I, J]$. It is easy to see, case by case and from the definition of Q_1 and Q_2, that they all belong to $H(A_1) \times I(A_2) \cup I(A_1) \times H(A_2)$. The fact that this property holds for the later needed indexing pairs follows by induction and from the fact that D_l, D_f and D_{fl} are computed using dynamic programming equations that are restrictions of the ones defined in Lemma 3. □

Theorem 5. *Given two nested arc-annotated sequences A_1 and A_2, of respective length n_1 and n_2, Algorithm 4.2 computes the cost of an optimal* NMult*-alignment between A_1 and A_2. It can be implemented to run in $O(n^4)$ worst-case time and $O(n^3)$ space where $n = n_1 + n_2$.*

Proof. The validity of this algorithm follows from three points.

- Lemmas 2, 3, 4, 5 and 6.
- The couples of indexing pairs which need to be considered in the dynamic programming algorithm are those of $H(A_1) \times I(A_2) \cup I(A_1) \times H(A_2)$ (Lemma 7).
- The fact that indexing pairs are considered in increasing order of length, and that $D_{fl}[I, J]$ is computed before $D_f[I, J]$ and $D_l[I, J]$ that are themselves computed before $D[I, J]$.

We now address the complexity of the algorithm. We can first notice that a nested arc-annotated sequence A of length n has $O(n^2)$ indexing pairs in $I(A)$ and $O(n)$ indexing pairs in $H(A)$. Hence, from Lemma 7, we have that the four dynamic programming tables require a space in $O(n^3)$. For the worst-case time complexity, then only point that needs to be addressed is that some dynamic programming equations consider a set of partitions of the current indexing pair in two indexing pairs. However, by definition of the partition of an indexing pair I into two indexing pairs I_1 and I_2, both I_1 and I_2, if non-empty, share an endpoint with I. The time complexity of the algorithm

Algorithm 1 Algorithm to compute the cost of an optimal NMULT-alignment between two nested arc-annotated sequences $A_1 = (S_1, P_1)$ and $A_2 = (S_2, P_2)$.

Algorithm : $ALIGN(A_1 = (S_1, P_1), A_2 = (S_2, P_2))$
Begin
Initialize tables D, D_f, D_l and D_{fl} using Lemma 2
For $i = 0 \rightarrow |\ S_1\ | - 1$ **Do**
For $x = 1 \rightarrow |\ S_1\ |$ **Do**
$y = x + i$
If $(x, y) \in I(A_1)$ **Then**
For $j = 0 \rightarrow |\ S_2\ | - 1$ **Do**
For $p = 1 \rightarrow |\ S_2\ |$ **Do**
$q = p + j$
If $(p, q) \in I(A_2)$ **Then**
If $(x, y) \in H(A_2)$ or $(p, q) \in H(A_2)$ **Then**
$I = (x, y),\ J = (p, q)$
Compute $D_{fl}[I, J]$ using Lemma 6
Compute $D_l[I, J]$ using Lemma 5
Compute $D_f[I, J]$ using Lemma 4
Compute $D[I, J]$ using Lemma 3
EndIf
EndIf
EndFor
EndFor
EndIf
EndFor
EndFor
End

Output: $D[(1, |\ S_1\ |), (1, |\ S_2\ |)]$

is then bounded by:

$$\left(|H(A_1)| \sum_{J \in I(A_2)} |J|\right) + \left(|I(A_2)| \sum_{I \in H(A_1)} |I|\right) + \left(|I(A_1)| \sum_{J \in H(A_2)} |J|\right) + \left(|H(A_2)| \sum_{I \in I(A_1)} |I|\right)$$

which is in $O(n_1 \times n_2^3 + n_1^2 \times n_2^2 + n_1^2 \times n_2^2 + n_1^3 \times n_2)$ or equivalently in $O(n^4)$. □

5 Discussion

Summary of our results. We proposed an extension of the hierarchy of arc-annotated sequences introducing three new classes named STEM, SMULT and NMULT. Based on this extension and on the arc-annotated sequences alignment framework introduced in [4, 5], we introduced new alignment problems and expressed the notion of conservative edit distance between RNA stem-loops introduced in [11]. The polynomial time and space algorithm we propose for computing the NMULT-alignment distance between two nested arc-annotated sequences solves the most general tractable problem in the extended hierarchy of alignment problems, when considering all the edit operations introduced in [14] and a complete score scheme. This extends previous results about ALIGN(NEST,NEST; NEST) [4, 5]. As far as we know, the only other alignment algorithm that considered an NMULT super-sequence did not consider all edit operations and a complete score scheme [13]. From the point of view of RNA secondary structures comparison, our work shows then that a more complex but tractable edit model can be considered. Using our algorithm one can then obtain possibly better alignments between pairs of RNA secondary structures.

Preliminary experimental results. We conducted experiments on a small set of secondary structures of antisense RNAs (CopA) taken from the RFAM database [10]. These RNAs are characterized by a long hairpin structure interrupted by several unpaired nucleotides or bulged loops. Figure 10 presents an example of comparison between two such structures (identified by X55895.1 and M16168.1 in the RFAM database) using two algorithms: ALIGN(NEST,NEST;NEST) and ALIGN(NEST,NEST;NMULT) with the following complete cost scheme: $w(BS) = 0$ (if the two bases are the same) or 25 (if they differ), $w(BD) = w(BD) = 100$, $w(AS) = 0$ (if the two arcs are identical) or 25 (if they differ by a single base) or 40 (if they differ on two bases), $w(AD) = w(AI) = 150$, $w(AB) = w(AC) = 100$ and $w(AA) = w(ACo) = 150$; these weights were used in [11].

We can see on this example that the problem ALIGN(NEST,NEST;NMULT), by allowing a base to be involved in two complex edit operations, leads to a an alignment that is worth to be considered and could not have been obtained using the algorithm for ALIGN(NEST,NEST;NEST): in the second part of the alignment, a base (A) which is the origin of an arc (A-U) in the first structure is aligned with a base (U) which is the origin of an arc (U-G) in the second structure whereas the ends of the arcs (U) and (G) are not aligned together. That could mean for example, that the base (A) in the first structure was first unpaired from the base (U), then substituted by a base (U) which was finally paired with a base (G) (Fig. 11). This configuration could not have been obtained using ALIGN(NEST,NEST;NEST).

Perspectives. The time complexity of RNA secondary structure alignments in the general edit distance model defined in [14] is relatively high, as the more general algorithms have an $O(n^4)$

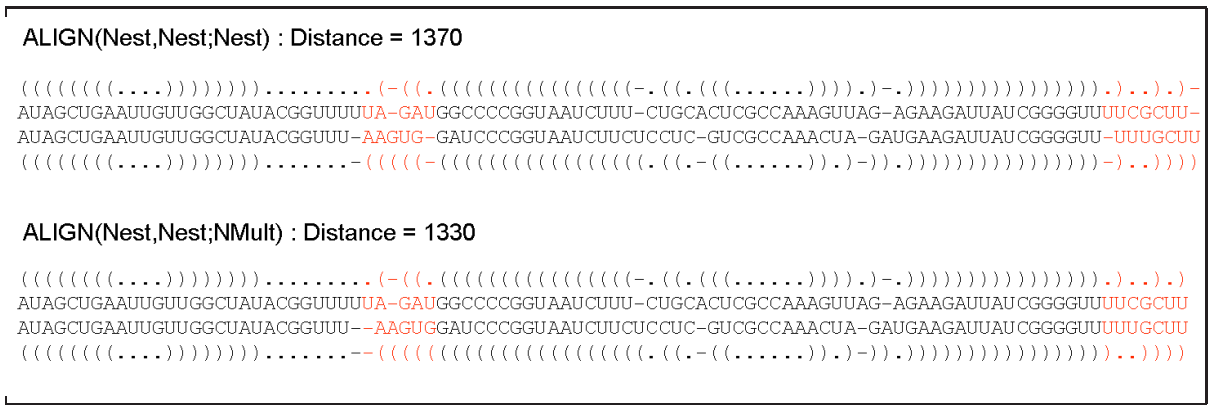

Fig. 10. Comparison of two secondary structures of antisense RNAs (CopA) using algorithms ALIGN(NEST,NEST;NEST) and ALIGN(NEST,NEST;NMULT). The parts of the arc-annotated sequences which are differently aligned by the two methods are colored in red color.

time complexity in the worst-case. Such issues have already been faced in the field of sequences alignment, where algorithms such as the Smith-Waterman algorithms have been considered too slow to be used directly for databases search. We think that the solutions that were proposed for the alignment of genomic sequences, or at least the general principles they are based on, can be used for RNA secondary structures alignment and database search. One could indeed think to a Blast-like approach [2] which could be defined as follows: the core of the methods would rely on finding highly similar seeds, in terms of RNA structures, for the local alignment of two arc-annotated sequences and then running a detailed comparison for the structures parts located between the identified seeds using a more general algorithm such as ALIGN(NEST,NEST;NMULT). Following another approach, in order to improve the comparison of RNA secondary structures some groups recently proposed to account for the multi-scale nature of RNA secondary structures for their comparison [1, 11, 16]. Following this approach, several levels of representation of an RNA secondary structures are considered and compared. In that framework, the problem ALIGN(NEST,NEST;NMULT) could be considered not at the level of whole structures, but to compare small substructures at the lower representation levels of structures.

From a theoretical point of view, Theorem 4, that states that ALIGN(NEST,NEST;CROS) is NP-hard, was proved in [3], in the case of a cost scheme that prevents to use the Arc-breaking operation. It then remains open to extend this hardness result to the case of a complete score scheme. Another complexity question that remains open is the complexity of ALIGN(STEM,STEM;CROS). It follows from [11] that the comparison of STEM arc-annotated can be used as an intermediate step to design an efficient RNA secondary structures comparison method. Hence, it would be interesting to see if a more general edit alignment model can be considered for this class of arc-annotated sequences.

References

1. J. Allali, M.-F. Sagot. A new distance for high level RNA secondary structure comparison. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(1):4–14. 2005.
2. S.F. Altschul, W. Gish, W. Miller, E.W. Myers, D.J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410. 1990.
3. G. Blin, G. Fertin, I. Rusu, C. Sinoquet. Extending hardness of RNA secondary structure comparison. In B. Chen, M. Paterson and G. Zhang (editors), *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies,*

First International Symposium, ESCAPE 2007, Hangzhou, China, April 7-9, 2007, Revised Selected Papers, volume 4614 of *Lecture Notes in Computer Science*, pages 140–151. Springer. 2007.
4. G.Blin, A. Denise, S. Dulucq, C. Herrbach, H. Touzet. Alignment of RNA structures. To appear in *IEEE/ACM Transactions on Computational Biology and Bioinformatics*. 2008.
5. G. Blin, H. Touzet. How to compare arc-annotated sequences: the alignment hierarchy. In F. Crestani, P. Ferragina and M. Sanderson (editors), *String Processing and Information Retrieval, 13th International Conference, SPIRE 2006, Glasgow, UK, October 11-13, 2006, Proceedings*, volume 4209 of *Lecture Notes in Computer Science*, pages. 291–303. Springer. 2006.
6. J. Couzin. Breakthrough of the year: small RNAs make big splash. *Science*, 298:2296–2297. 2002.
7. E. Demaine, S. Mozes, B. Rossman, O. Weimann. An optimal decomposition algorithm for tree edit distance. In L. Arge, C. Cachin, T. Jurdzinski and A. Tarlecki (editors), *Automata, Languages and Programming, 34th International Colloquium, ICALP 2007, Wroclaw, Poland, July 9-13, 2007, Proceedings*, volume 4596 of *Lecture Notes in Computer Science*, pages. 146–157. Springer. 2007.
8. S. Dulucq, H. Touzet. Decomposition algorithms for the tree edit distance problem. *Journal of Discrete Algorithms* 3(2-4):448-471. 2005.
9. P. Evans. Algorithms and complexity for annotated sequences analysis Ph.D. Thesis, University of Victoria, Canada. 1999.
10. S. Griffiths-Jones, S. Moxon, M. Marshall, A. Khanna, S.R. Eddy, A. Bateman. Rfam: annotating non-coding RNAs in complete genomes. *Nucleic Acids Research*, 33:D121–D124. 2005.
11. V. Guignon, C. Chauve, S. Hamel. An edit distance between RNA stem-loops. In M.P. Consens and G. Navarro (editors), *String Processing and Information Retrieval, 12th International Conference, SPIRE 2005, Buenos Aires, Argentina, November 2-4, 2005, Proceedings*, volume 3772 of *Lecture Notes in Computer Science*, pages. 335–347. Springer. 2005.
12. I.L. Hofacker. Vienna RNA secondary structure server. *Nucleic Acids Research*, 31(13):3429–3431. 2003.
13. M. Höchsmann, T. Töller, R. Giegerich, S. Kurtz. Local similarity in RNA secondary structures. In *2nd IEEE Computer Society Bioinformatics Conference (CSB 2003), 11-14 August 2003, Stanford, CA, USA*, pages 159–168. IEEE Computer Society. 2003.
14. T. Jiang, G.-H. Lin, B. Ma, K. Zhang. A general edit distance between RNA structures. *Journal of Computational Biology* 9(2):371-388. 2002.
15. T. Jiang, L. Wang, K. Zhang. Alignment of trees - an alternative to tree edit. *Theoretical Computer Science*, 143(1):137–148. 1995.
16. A. Ouangraoua, P. Ferraro, S. Dulucq, L. Tichit. Local similarity between quotiented ordered trees. *Journal of Discrete Algorithms* 5(1):23–35. 2007.
17. J.S. Pedersen, G. Berejano, A. Siepel, K. Rosenbloom, K. Lindblad-Toh, E.S. Lander, J. Kent, W. Miller, D. Haussler. Identification and classification of conserved RNA secondary structures in human genome. *PLoS Computational Biology*, 2(4):e33. 2006.
18. S. Siebert, R. Backofen. MARNA: multiple alignment and consensus structure prediction of RNAs based on sequence structure comparisons. *Bioinformatics*, 21(16):3352–3359. 2005.
19. S. Will, K. Reiche, I.L. Hofacker, P.F. Stadler, R. Backofen. Inferring non-coding RNA families and classes by means of genome-scale structure-based clustering. *PLoS Computational Biology*, 3(4):e65. 2007.
20. K. Zhang, D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262. 1989.
21. K. Zhang, L. Wang, B. Ma. Computing similarity between RNA structures. *Theoretical Computer Science*, 276(1-2):111–132. 2002.

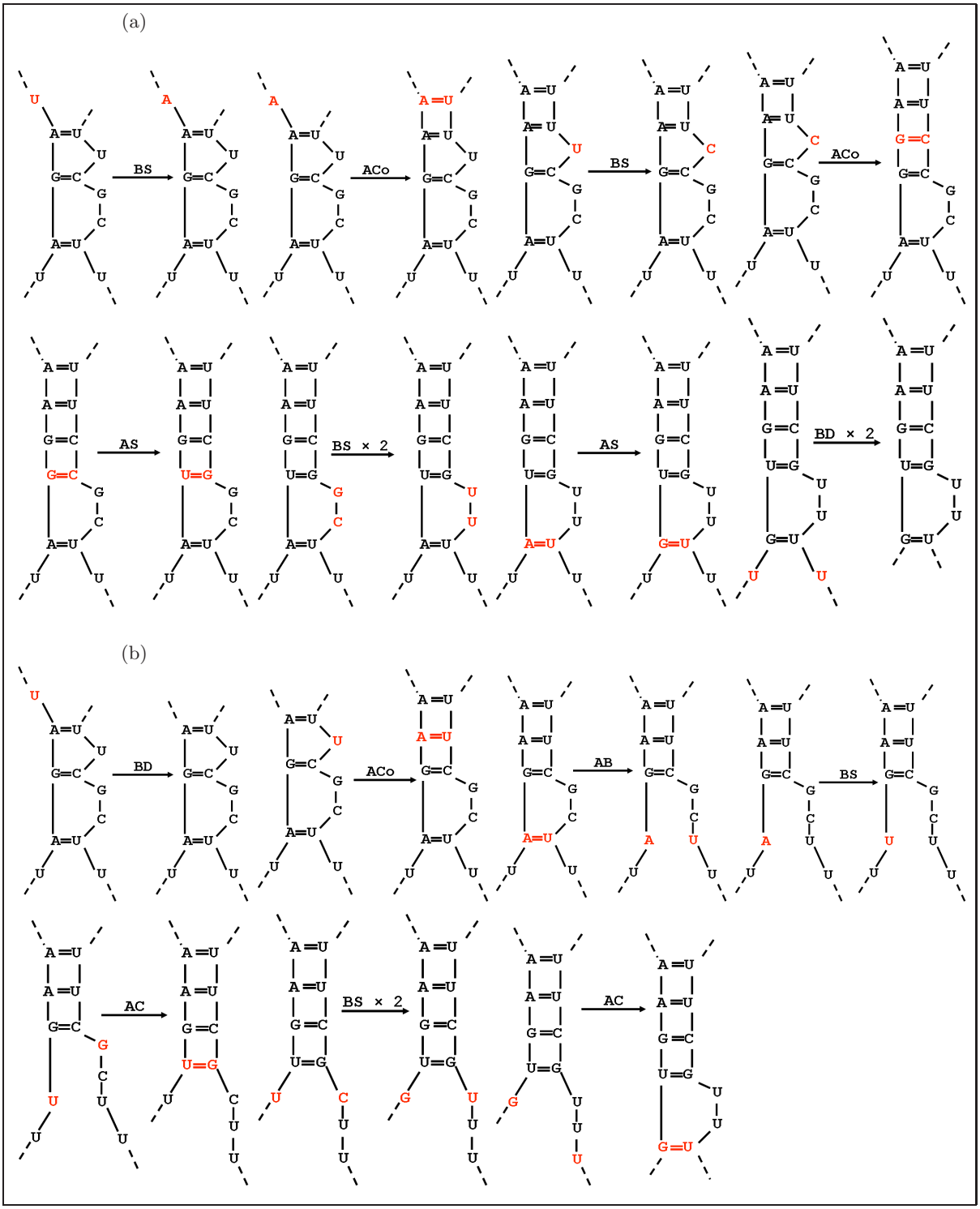

Fig. 11. Two examples of edit sequences which correspond to the alignments in red color in Figure 10 obtained using (a) ALIGN(Nest,Nest;Nest) and (b) ALIGN(Nest,Nest;NMult).