Algorithmique de l'alignement structure-séquence d'ARN : une approche générale et paramétrée

Philippe Rinaudo

THÈSE

pour l'obtention du grade de Docteur en Sciences

Spécialité informatique

Algorithmique de l'alignement structure-séquence d'ARN: une approche générale et paramétrée.

Philippe Rinaudo

Soutenance le 5 décembre 2012 devant la commission d'examen:

M.	Jean-Claude König	Professeur	Rapporteur
M.	Stéphane Vialette	Directeur de Recherche	Rapporteur
M^{me}	Alessendra Carbone	Professeur	Examinatrice
M.	Yannis Manoussakis	Professeur	Examinateur
M.	Dominique Barth	Professeur	Directeur de Thèse
M.	Alain Denise	Professeur	Directeur de Thèse

Laboratoire de Recherche en Informatique, U.M.R. CNRS 8623,
Université Paris-Sud, 91405 Orsay Cedex, France

Table des matières

Chapitre 1

Introduction

L'alignement de macromolécules biologiques comme les protéines, l'ADN ou encore l'ARN est une problématique biologique et bio-informatique qui a pour but de révéler une partie des mystères du fonctionnement des cellules, constituant des êtres vivants. Les approches bio-informatiques explorées par la recherche actuelle établissent un rapprochement étroit entre l'alignement de molécules et des problématiques informatiques ou mathématiques, liées par exemple à l'algorithmique du texte ou la théorie des graphes.

Le travail présenté dans ce mémoire apporte des éléments constructifs sur le problème appelé **l'alignement de structure-séquence d'ARN**. Ce problème se situant en plein cœur des problématiques biologiques va nous conduire à faire appelle à la **théorie des graphes** en vue de sa résolution.

Les ARN et la bio-informatique de l'ARN

Le premier chapitre (introduction mise à part) du mémoire est consacré aux ARN non-codants. Après une rapide description des molécules d'ARN, nous présenterons leur modélisation classique en bio-informatique. En conséquence de cette modélisation, nous commencerons à entrevoir pourquoi les différentes problématiques associées aux ARN sont très proches de problématiques de graphes.

Nous détaillerons ensuite deux des problèmes principaux de la bio-informatique des ARN : **la recherche d'ARN non-codant et la prédiction de structure**, en listant les principales approches tentant de les résoudre.

L'alignement structure-séquence en détail

Nous focaliserons ensuite notre attention sur l'une de ces approches : l'**alignement structure-séquence d'ARN**. Le problème d'alignement structure-séquence consiste comme son nom l'indique à aligner une structure connue d'un premier ARN (nous verrons plus tard ce que nous entendons par "structure") avec la séquence d'un deuxième ARN. Pour résoudre ce problème, nous cherchons à optimiser l'alignement selon une fonction de coût. C'est donc un problème d'optimisation, qui malheureusement se révèle **NP-Difficile** (en général). En conséquence différents travaux définissent des classes d'instances réduites pour lesquelles ils proposent des algorithmes à complexités polynomiales. La description précise du problème et son état de l'art forment le chapitre 3 du mémoire. Après l'introduction de notions qui seront reprises tout au long du mémoire, nous aborderons l'état de l'art du problème par une présentation générale de la **programmation dynamique**, qui est au cœur de toutes les méthodes que nous décrirons, et nous l'illustrerons par le célèbre algorithme de Smith et Waterman. Puis nous détaillerons les deux algorithmes principaux d'alignement structure-séquence : l'algorithme de Jiang et l'algorithme de Han. Enfin, mais de façon moins détaillée, nous présenterons les algorithmes de Wong, clôturant ainsi l'état de l'art du problème.

Les différents détails et réflexions apportés tout au long de l'état de l'art nous permettront de dresser un bilan positif : même si toutes ces approches résolvent l'alignement structure-séquence, de façon différentes, sur des familles d'instances différentes, nous pouvons entrevoir comment **unifier** tous ces travaux, et plus important encore, comment les **généraliser**.

La contribution : unification et généralisation par une approche à complexité paramétrée

L'unification et la génération des approches précédentes est la thématique du chapitre 4. C'est aussi la contribution principale du travail présenté dans ce mémoire. Nous montrerons comment, en utilisant un unique algorithme à complexité paramétrée et non spécifique à une classe d'instances, nous pouvons résoudre le problème d'alignement structure-séquence pour toutes les instances possibles, et aussi efficacement que les précédentes approches sur leur domaine de résolution respectif.

Pour réaliser ceci, nous utiliserons une technique empruntée à la théorie des graphes : la **décomposition arborescente**. Au final, nous obtiendrons un **algorithme paramétré**, dont le paramètre (entièrement lié à la décomposition arborescente) est raisonnablement petit pour les instances existantes dans des bases de données biologiques.

Vers une application

Nous détaillerons dans le chapitre 5 comment utiliser cet algorithme en vue d'une application efficace, comme par exemple en montrant qu'il peut être utilisé directement pour la recherche de solutions sous-optimales ou encore en montrant qu'il peut servir de base solide pour l'élaboration d'heuristiques adaptées.
Enfin, avant de conclure, nous terminerons le mémoire par un chapitre consacré aux fonctions de coûts, utilisées dans la recherche d'optimaux du problème d'alignement structure-séquence. Nous présenterons les fonctions standard avant de proposer une version personnelle, que nous voulons obtenir par apprentissage, et montrant déjà des signes d'améliorations.

Chapitre 2

Les ARN et la bio-informatique de l'ARN

Sommaire

Comprendre les mécanismes du vivant est une des grandes problématiques que la recherche moderne tente de résoudre. Le champ d'étude est très vaste et toutes les découvertes que les chercheurs peuvent faire donnent lieu à encore plus de fascinantes questions. De façon irréfutable, nous savons depuis l'apparition des microscopes que tous les organismes vivants sont composés d'une ou plusieurs cellules. De ce fait, pour comprendre en profondeur la biologie, nous devons visiter l'infiniment petit, c'est-à-dire décortiquer tous les mécanismes intervenant dans les cellules. C'est le domaine de la biologie moléculaire.

Dans la dynamique d'une cellule intervient un nombre incalculable de molécules de différentes natures, de différentes tailles et évidement avec des rôles différents. Les plus connues sont sans conteste les protéines et l'ADN.

Les premières ont obtenu leur renommée grâce à leur implication dans la majorité des processus cellulaires. Ainsi c'est grâce à leur existence qu'une cellule peut maintenir sa forme ou que les différentes molécules peuvent être transportées dans et en dehors des cellules. C'est aussi elles qui sont à l'origine des réactions chimiques cellulaires, on les nomme alors enzymes.

Les protéines sont synthétisées à partir des gènes qui sont contenus dans l'ADN. L'ADN est donc une molécule contenant le patrimoine génétique d'un organisme, se transmettant de générations en générations.

La synthèse d'une protéine est un mécanisme cellulaire que nous pouvons décomposer en deux étapes : la transcription d'un gène en une molécule d'acide ribonucléique (**ARN**), et la traduction de la molécule d'ARN en protéine. Dans certains cas, l'ARN issue de la transcription n'a pas pour vocation d'être traduit en protéine. On parle alors d'**ARN non-codants** (contrairement aux ARN codants, qui eux sont traduis en protéines). Ces ARN ont alors une fonction propre, et comme les protéines ils sont impliqués dans de nombreux processus cellulaire fondamentaux.

Cette conception de la biologie moléculaire fait partie de ce qu'on nomme le **dogme centrale de la biologie** (voir Figure 2.1). Même si cette dernière est aujourd'hui remise en question sur certains points, ce que nous retiendrons de cette courte présentation est que l'étude des molécules comme l'ARN et les protéines peut permettre de mieux comprendre les mécanismes cellulaires dont elles sont les principaux moteurs.

La problématique biologique que nous aborderons dans ce mémoire concerne l'étude des ARN. Plus particulièrement, nous allons nous intéresser aux ARN non-codants et à la façon dont nous pouvons obtenir des informations sur leur structure et comment reconnaitre et caractériser ces molécules.

Les molécules d'ARN se retrouvent dans tous les organismes vivants. L'ARN est aussi présent dans certains virus (comme le VIH) dont il représente le patrimoine génétique et sans lequel le virus n'aurait pas d'effet. Il est même envisagé qu'au début de l'apparition de la vie, les ARN constituaient l'unique molécule primordiale [24] (pas de protéines, pas d'ADN).

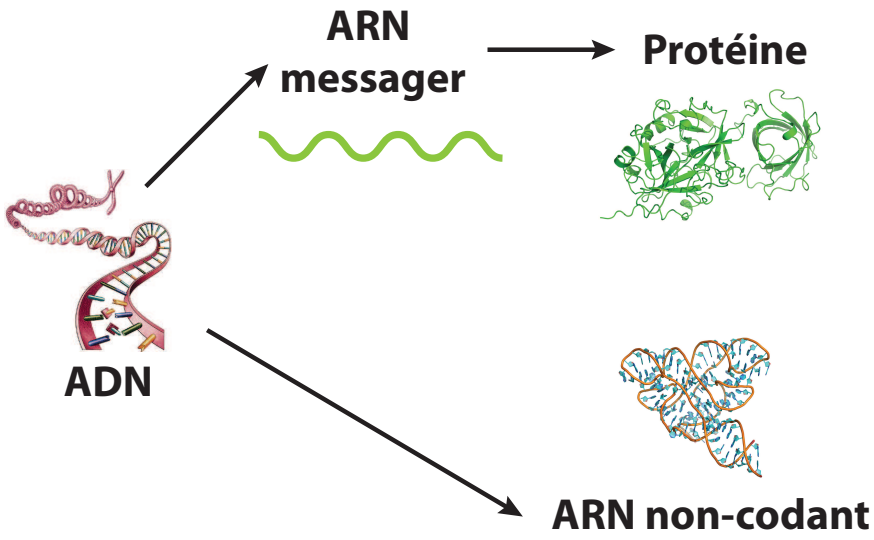


FIGURE 2.1 – *Un gène contenu dans l'ADN est transcris en ARN. Cet ARN est soit un ARN messager traduit ensuite en protéine, soit un ARN non-codant.*

Nous pouvons catégoriser les ARN dans deux grandes familles : les ARN codants (ou ARN messagers) et les ARN non-codants. Alors que les premiers servent de copies pour faire transiter l'information contenue dans l'ADN jusqu'aux molécules les traduisant en protéines, les ARN non-codants possèdent un cycle de vie différent. Ils peuvent par exemple réguler l'expression de gènes (ex : micro ARN), intervenir dans la traduction (ex : ARN ribosomiques, ARN de transfert), dégrader certains ARN messagers (ex : ARN interférents) ou encore avoir une action catalytique, c'est-à-dire provoquer une réaction chimique (ex : le ribozyme de la ribonucléase P).

Nous allons maintenant effectuer un rapide zoom sur ces molécules pour mieux connaître leur composition et en déduire une représentation manipulable par des approches algorithmiques.

2.1 Composition et structures 3D des ARN non-codants

Composition

L'ARN est un polymère, c'est-à-dire un enchainement de briques de base : les **nucléotides**. Sa représentation **primaire** capture cette conception selon laquelle l'ARN est comme un collier de perles où chaque perle représente un nucléotide.

Un nucléotide est constitué de trois sous-groupes d'atomes : la base (qui peut être une adénine (A), une cytosine (C), une guanine (G) ou un uracile (U)), le ribose et le groupement phosphate (voir Figure 2.2). L'enchainement de deux nucléotides consécutifs est assuré par une liaison phosphodiester, interaction covalente entre le groupement phosphate du premier nucléotide et le ribose du deuxième (voir Figure 2.2).

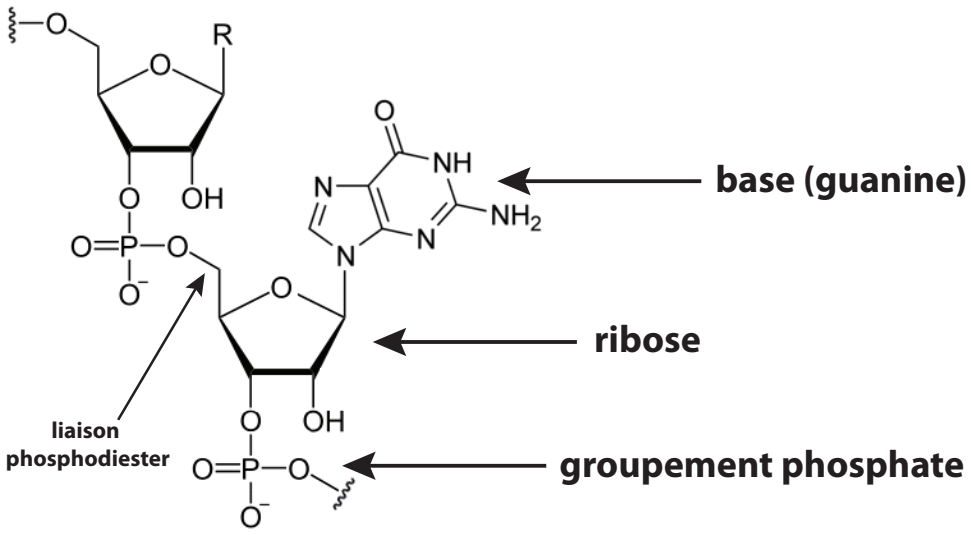


FIGURE 2.2 – *Représentation classique d'un nucléotide relié par un liaison phosphodiester à un autre nucléotide.*

Obtenir la succession des nucléotides formant un ARN, c'est-à-dire sa **séquence**, est aujourd'hui quelque chose d'accessible. On parle de séquençage d'ARN, et les dernière techniques mises au point (comme par exemple la RNAseq) permettent de séquencer avec un haut débit les ARN. En revanche, il est très difficile de reconnaitre un ARN à partir de sa séquence, c'est-à-dire de savoir si il est codant ou non-codant, et dans le dernier cas, à quelle famille il appartient.

Structures des ARN non-codants

Une fois transcrit (et même pendant la transcription) l'ARN ne reste pas à l'état d'un brin sans forme. Il se replie sur lui même pour adopter une conformation tridimensionnelle propre (voir Figure 2.3). C'est cette forme qui va conférer en grande partie la fonction à la molécule.

Le repliement est la conséquence de la formation d'interactions de différentes natures s'effectuant entre les atomes des nucléotides. Les plus représentées et les plus importantes sont les interactions hydrogène entre les bases (voir Figure 2.4).

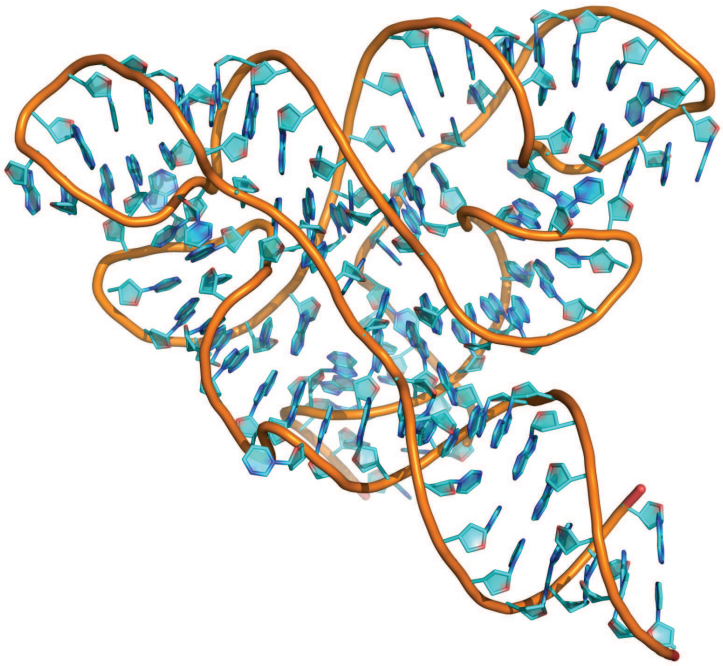


FIGURE 2.3 – *Structure 3D du riboswitch FMN.*

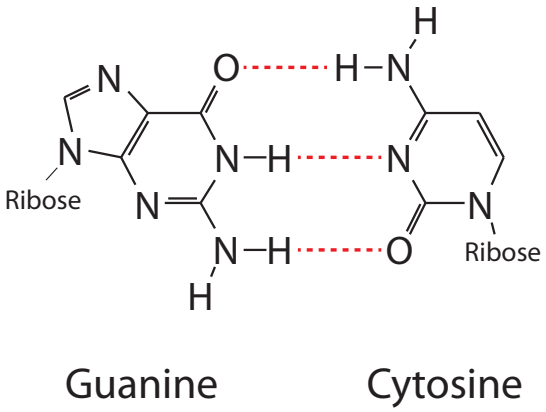


FIGURE 2.4 – *Interaction hydrogène entre une guanine et une cytosine. Les interactions hydrogène sont représentées en pointillés.*

Schématiquement, nous pouvons représenter une base comme un triangle, dont chaque côté porte un nom [36] : Hoogsteen (H), Watson-Crick (W), Sugar (S) (voir Figure 2.5). Les bases peuvent s'apparier deux à deux par ces côtés en formant des liaisons hydrogène. Il existe douze combinaisons possibles car chaque combinaison de côtés se décline en deux versions (*cis* et *trans*) selon l'orientation (voir Figure 2.5).

Il existe néanmoins des incompatibilités d'appariements en fonction de la nature des bases et des côtés considérés [35, 57]. Nous reviendrons plus précisément sur ce point à la fin de ce mémoire.

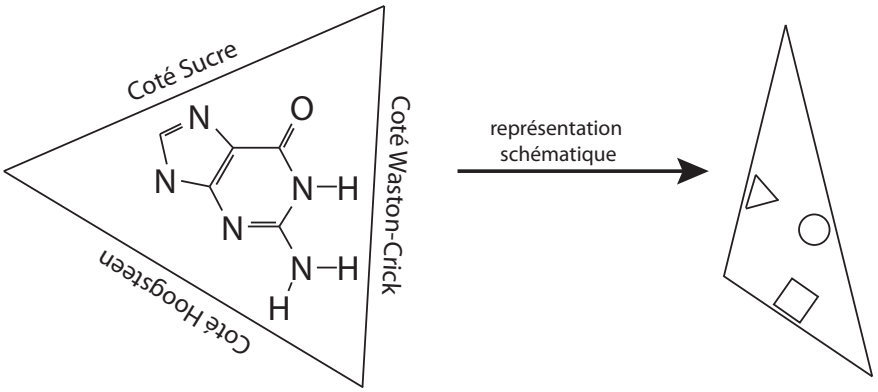


schéma	cotés en interaction	symbole
	Watson-Crick / Watson-Crick	
	Watson-Crick / Sucre	
	Watson-Crick / Hoogsteen	
	Hoogsteen / Hoogsteen	
	Hoogsteen / Sucre	
	Sucre / Sucre	

FIGURE 2.5 – *Illustration de la nomenclature Léontis-Westhof (interaction en cis uniquement[35].*

Nous pouvons classer les interactions en deux sous-groupes : les **interactions canoniques** et **interactions non-canoniques**. On nomme **interactions canoniques**, les interactions du type Watson-Crick / Watson-Crick, (c'est-à-dire que les deux bases sont en interaction par leur côté Watson-Crick) entre deux bases dites complémentaires. La complémentarité entre deux bases s'établit ainsi : A et U sont complémentaires et G et C sont complémentaires. Il est admis (mais pas systématiquement) que G et U sont aussi complémentaires. Toutes les autres interactions seront dites **non-canoniques**.

Il existe d'autres interactions (sucre-phosphate, empilements ...), néanmoins nous restreindrons les interactions aux interactions hydrogène car c'est principalement celles-ci qui déterminent la structure d'un ARN.

Pour déterminer expérimentalement la structure tridimensionnelle d'une molécule d'ARN, nous pouvons, comme pour d'autres molécules (par exemple les protéines pour ne citer qu'elles), avoir recours aux techniques de cristallographie ou celles basées sur la résonance magnétique nucléaire (RMN). Ces techniques donnent d'excellent résultats, avec une résolution très fine (moins de 4Å pour la cristallographie) mais sont longues, coûteuses et difficiles à mettre en place. Pire, pour énormément d'ARN, il n'est pas (encore) possible d'appliquer ces méthodes.

2.2 Modélisation bioinformatique de l'ARN

Afin de pouvoir utiliser des approches de bio-informatique pour résoudre des problèmes de biologie liés à l'ARN, il est nécessaire de posséder une bonne représentation d'une molécule d'ARN. Ainsi, il a été clairement défini cinq niveaux de structure, organisés hiérarchiquement. Le principal intérêt de ces niveaux de structure est de pouvoir contenir uniquement l'information nécessaire à la résolution d'un problème précis. Cette hiérarchie commence à la séquence de l'ARN et s'étend jusqu'à la structure $3D$ de la molécule.

Nous allons maintenant décrire chacun des niveaux de structure en détails.

2.2.1 La structure primaire

Le premier niveau de structure correspond à la séquence de l'ARN. C'est ce que nous avons déjà appelé la **structure primaire**. Pour beaucoup d'ARN, nous disposons uniquement de cette donnée car la détermination par méthodes expérimentales d'une structure primaire d'ARN est un processus maintenant bien maîtrisé, contrairement aux autres niveaux de structure. C'est donc bien souvent un des points de départ d'approches bio-informatiques.

Formellement, la structure primaire d'un ARN est une séquence S dont le $i^{ème}$ nucléotide, noté $S[i]$, est l'une des 4 lettres A,C,G,U. La séquence est écrite dans le sens dit $5' \mapsto 3'$, résultant d'une numérotation classique des atomes de carbone des nucléotides (on retiendra simplement que le sens n'est pas arbitraire).

2.2.2 Les niveaux de structures intermédiaires

Les niveaux de structure intermédiaires (comme la structure secondaire) peuvent être vus comme des graphes, contenant deux types d'information : la séquence des nucléotides et leurs interactions. Les nœuds de ces graphes sont les nucléotides (représentés uniquement par leur base) et les arêtes sont de deux types : un premier type pour les interactions hydrogènes, et un second pour les liaisons phosphodiester. Les arêtes du second type (parfois représentées par des arcs), forment par construction une chaîne hamiltonienne dans le graphe. Nous allons maintenant décrire trois

niveaux de structure (structure secondaire, structure secondaire avec pseudo-nœuds et structure secondaire étendue). Chacun de ces niveaux ne considère qu'un sous-ensemble des interactions hydrogène, défini selon des considérations biologiques.

La structure secondaire

La structure secondaire représente (ou tente de représenter) la principale information structurelle d'une molécule d'ARN. Elle contient un sous-ensemble important des interactions canoniques permettant de représenter les 4 briques de bases d'une structure d'ARN : **les hélices, les jonctions, les boucles terminales et les boucles internes** (voir Figure 2.6).

Généralement, les interactions canoniques ne sont pas isolées, c'est-à-dire qu'elles forment des empilements. Ces empilements forment à leur tour des hélices (voir Figure 2.6). L'agencement des hélices donne alors naissance aux jonctions et boucles. Ce niveau de structure est suffisamment stable (d'un point de vue énergétique) pour pouvoir se maintenir lui-même (sans l'aide des autres interactions), c'est en cela que l'on dit qu'il contient le cœur de la structure.

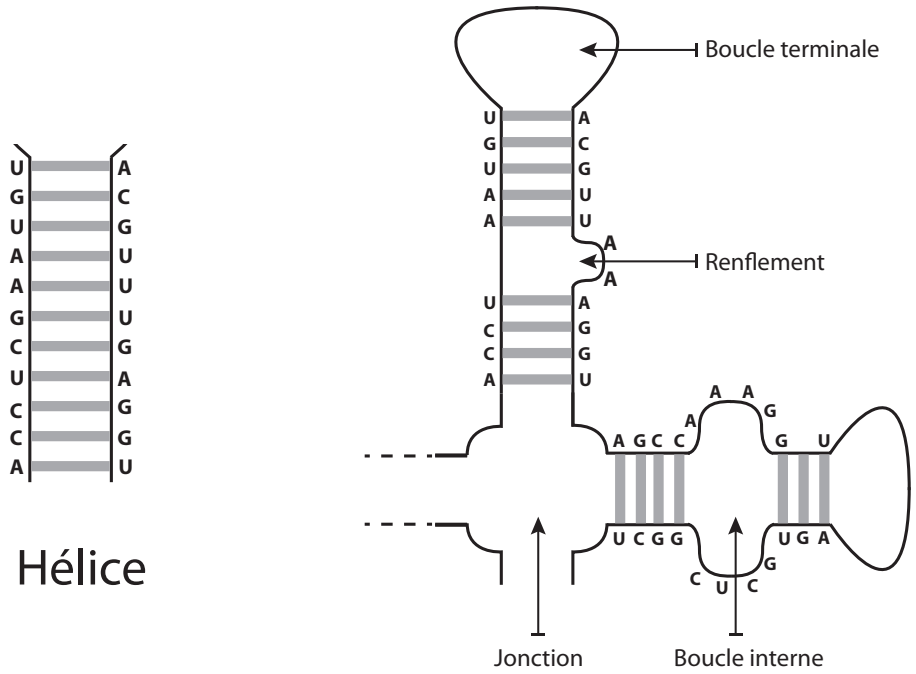


FIGURE 2.6 – *À gauche : une hélice, résultat d'un empilement d'interactions canoniques (en gris). À droite : illustrations des motifs de base des structures secondaires*

L'ensemble des interactions canoniques représentant l'ensemble des hélices d'un

ARN caractérise la structure secondaire. Cependant, la définition exacte d'une structure secondaire interdit que deux interactions se croisent. C'est-à-dire, si $(S[i], S[j])$ et $(S[i'], S[j'])$ sont deux interactions d'une structure secondaire telles que $i < i'$, alors $i < i' < j' < j$ ou $i < j < i' < j'$. Ces croisements sont appelés **pseudo-nœuds**, un exemple est donné dans la Figure 2.7. Par conséquent, la notion de structure secondaire est ambigüe, car à partir de l'ensemble des interactions canoniques, nous pouvons construire plusieurs structures secondaires. C'est souvent l'expertise biologique qui donnera "la" solution (il n'y a malheureusement pas de règle établie).

Enfin, remarquons qu'étant donné que seules les interactions canoniques sont prises en compte, un nucléotide est en interaction avec au plus un autre nucléotide (bornant ainsi le degré des sommets du graphe par 3 : une interaction maximum, et deux liaisons maximum).

La structure secondaire avec pseudo-nœuds

Comme annoncé et comme son nom l'indique, la structure secondaire avec pseudo-nœuds prend en compte toutes les interactions canoniques (voir Figure 2.7), même celles formant des pseudo-nœuds. Les pseudo-nœuds participent énormément à la fonction de la molécule (pour les molécules avec pseudo-nœuds). C'est la raison pour laquelle il est nécessaire de définir un niveau de structure les prenant en compte. La suppression de la contrainte de non-croisement des interactions va apporter un saut de complexité dans la plus part des algorithmes manipulant des structures secondaires (avec ou sans pseudo-nœud).

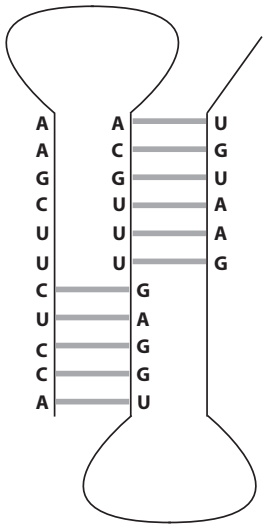


FIGURE 2.7 – *Illustration d'un pseudo-nœud (type H). Quelles interactions doit-on conserver pour la structure secondaire ? Le problème possède plusieurs solutions valident.*

La structure secondaire étendue

Reste à prendre en compte les autres types d'interactions, c'est-à-dire les interactions non-canoniques. La structure secondaire étendue (aussi appelée structure tertiaire, mais cette appellation étant aussi utilisée pour la structure $3D$ dans certains cas nous ne l'adopterons pas) prend en compte l'ensemble des interactions possibles entre bases. Les arêtes du graphe correspondant sont alors étiquetées (ou colorées), et un nucléotide peut être en interaction avec plusieurs bases (voir Figure 2.8). L'intérêt porté à ces interactions est récent et en grande partie motivé par le fait que la combinaison d'interactions canoniques et non-canoniques permet la formation de motifs structuraux, comme par exemple le Kink-Turn. Ces motifs, lorsqu'ils sont présents, sont indispensables à la molécule pour qu'elle assure sa fonction (comme pour les pseudo-nœuds).

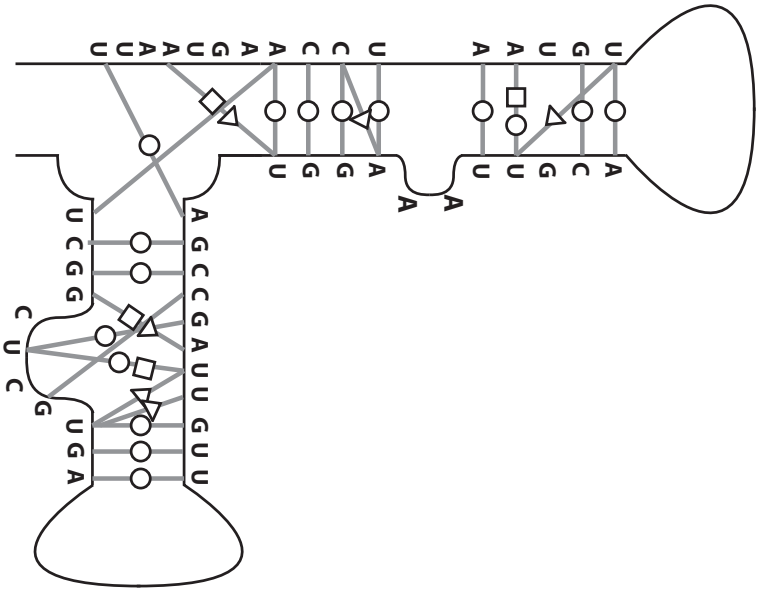


FIGURE 2.8 – *Illustration d'une structure secondaire étendue. Les interactions sont étiquetées selon les symboles donnés en Figure 2.5*

2.2.3 La structure $3D$

C'est bien souvent à partir de la structure $3D$ de la molécule que nous pouvons inférer les autres niveaux de structure (ceci est surtout vrai pour les structures secondaires étendues). On nomme ce procédé l'annotation de structure d'ARN. Les outils les plus classiques que sont RNAview [66] et MC-Annotate [23], calculent l'ensemble des interactions d'une molécule d'ARN dont la structure $3D$ est contenue dans un fichier PDB. Les fichiers PDB contiennent les coordonnées de tous les atomes

de la molécule et leur format constitue le standard pour la représentation $3D$ de molécules biologiques.

2.3 Évolution et homologie

Nous abordons ici un thème biologique très connu et qui motive le problème d'alignement structure-séquence que nous allons traiter en détail par la suite. Ce thème est l'évolution, et plus particulièrement l'homologie entre molécules d'ARN.

Nous savons tous (ou nous en sommes tous convaincus) que toutes les espèces vivantes sont issues d'ancêtres communs. Ces premiers organismes possédaient déjà la mécanique cellulaire indispensable à la vie et que tous les organismes vivants partagent. De ce fait, tous les organismes vivants ont un patrimoine génétique commun ou proche, plus ou moins grand en fonction de leur lien de parenté dans l'arbre de vie des espèces. Ainsi, tous les organismes vivants possèdent des molécules parentes, ayant bien souvent des fonctions proches. Lorsque deux molécules d'ARN de deux organismes différents sont issues de l'évolution d'une même molécule ancêtre, alors ces deux molécules sont dites **homologues**.

Par exemple l'ARN ribosomique $16s$ de l'homme est homologue à l'ARN ribosomique $16s$ de la levure, même si ils ne sont pas identiques, et il en va de même pour les ARN de transferts du rat, ceux de la rose et ceux d'une bactérie.

En conséquence, ce que nous savons d'une molécule, nous le savons aussi (en partie) pour tous ses homologues. À l'inverse, les différences de fonctionnement de deux molécules homologues peuvent s'étudier en étudiant les dissimilitudes entre les deux molécules.

L'étude des molécules d'ARN par homologie de part son grand intérêt et ses nombreuses applications est rapidement devenu l'un des plus important domaine de la biologie moléculaire.

Dans les deux problèmes de biologie moléculaire que nous allons maintenant aborder, nous allons voir que l'étude par homologie représente une, sinon la meilleure des solutions actuelle.

2.4 Recherche d'ARN non-codants

La recherche de gènes, d'ARN ou de protéines dans des bases de données est une très grande thématique de la bio-informatique. À titre d'exemple, le logiciel BLAST [3] qui effectue ce type de recherches est de très loin le logiciel bio-informatique le plus utilisé ([3] est aussi l'article le plus cité toutes disciplines confondues). L'idée fondamentale de toutes les approches existantes sur ce domaine est toujours la même : pour déterminer si deux molécules sont homologues, il faut construire un alignement entre les deux séquences des polymères. En fonction de l'alignement obtenu (souvent évalué par une fonction de scores ou de coûts), on déclare les deux molécules homologues ou non-homologues.

Cependant, l'alignement des ARN non-codants est beaucoup plus délicat que l'alignement d'autres molécules comme les protéines. Alors que deux protéines homologues sont presque toujours très proches en séquences, les séquences de deux ARN non-codants homologues peuvent être très différentes. Ceci est du en grande partie aux mutations dites compensatoires. Prenons un exemple. Soit deux nucléotides d'un même ARN, N_1 et N_2 et supposons qu'ils soient en interaction canonique. Supposons également que N_1 soit une adénine et que donc par conséquent, N_2 soit une uracile. Une mutation de N_1 en guanine ne provoquerait que très peu de changements dans la structure. L'interaction entre N_1 et N_2 serait toujours possible et toujours canonique. Supposons alors qu'une seconde mutation intervienne, modifiant N_2 en cytosine. Encore une fois l'interaction entre N_1 et N_2 serait toujours possible et toujours canonique, mais N_1 et N_2 seraient tous les deux modifiés alors que la structure resterait inchangée. Ce processus est propre à l'ARN, et rend la problématique de l'alignement d'ARN homologues spécifique.

Par conséquent, il est très fréquent d'avoir des ARN homologues avec des faibles similarités de séquences. Or les approches classiques comme BLAST [3] basent leur critère d'homologie sur cette similarité. D'autres approches plus évoluées, comme par exemple celle utilisé par HMMER [19] utilisant des chaînes de Markov cachées, ne prennent aussi en compte que les éléments de structures primaires pour effectuer la recherche d'homologues. Alors que leur efficacité pour la recherche de protéines est incontestable, elles sont presque inutilisables pour la recherche d'ARN non-codant homologues de similarité de séquences faibles.

À contre pied de ces méthodes, des approches considérant les structures secondaires ont été développées spécifiquement pour l'alignement d'ARN non-codants (et donc pour la recherche d'ARN non-codants).

La majeure partie de ces méthodes considèrent en entrée une collection d'ARN homologues préalablement alignés (un alignement **multiple**) associés à une structure secondaire consensus [20, 62, 1]. À partir de ces données, ces approches construisent automatiquement un système de score pour déterminer à partir de la séquence d'un ARN quelconque, si celui-ci est homologue aux ARN fournis en entrée. La plus connue de ces approches est le modèle de covariance, reposant sur une grammaire non-contextuelle stochastique. Les différentes implémentations de ces approches, comme par exemple le logiciel INFERNAL [45] pour le modèle de covariance, peuvent donner de bons résultats [22], avec pour les meilleurs un peu plus de 80% de précision (probabilité empirique qu'un ARN décrété homologue le soit vraiment).

Néanmoins, ce résultat doit être relativisé. Premièrement, toutes ces approches dépendent en grande partie de l'alignement multiple fourni en entrée. Celui-ci est souvent réalisé en considérant uniquement la similarité de séquences avec des approches comme ClustalW [13]. Par conséquent l'alignement multiple introduit un biais dans le système de score. Les alignements multiples peuvent être aussi obtenus par considération des informations structurelles, c'est-à-dire en construisant l'alignement multiple en l'optimisant par rapport à une structure consensus construite simultanément [20, 54]. Mais même ainsi, l'alignement multiple introduit un biais.

En effet les ARN homologues collectés pour l'alignement multiple ont été trouvés principalement par les approches utilisant la similarité de séquences, et on se retrouve alors confronté au problème précédent. Enfin, notons que la plupart de ces approches ont une limitation de taille : elles ne peuvent prendre en compte que les informations liées à la structure secondaire. Les alignements (ou les recherches) d'ARN non-codant possédant des pseudo-nœuds ou des motifs tertiaires seront alors effectués sans considérer une partie de ce qui fait leur fonction.

Un échappatoire existe à ces pièges, mais cela va nécessiter de connaitre au préalable la structure d'un ARN. Lorsque l'on connait la structure d'un premier ARN et que l'on veut déterminer si un deuxième ARN (dont on ne connait que la séquence) lui est homologue, nous pouvons alors calculer à quel point la structure du premier est compatible avec le deuxième. Le problème formel (que nous allons décrire dans le prochain chapitre) se nomme l'alignement structure-séquence. En ne considérant qu'un seul ARN (mais avec sa structure), on évite le biais engendré par les similarités de séquences. Alors que comme pour l'alignement à partir d'une structure consensus, les premières méthodes ne prenaient en compte que les structures secondaires [31, 33], aujourd'hui la direction principale des innovations sur cette approche s'oriente vers la généralisation à n'importe quel niveau de structure [31, 29, 64, 63, 8].

Notons enfin, que le problème de recherche d'homologue implique de devoir effectuer une quantité phénoménale d'alignements. Ainsi, des heuristiques se sont très vite mises en place [56, 7], offrant une véritable alternative et même si actuellement leur manque de visibilité leur fait défaut (logiciels associés non disponibles ou spécifique à certains systèmes, format de fichiers d'entrés différents des standards ...) c'est sans conteste aussi avec ces méthodes qu'il faudra compter dans l'avenir.

2.5 Prédiction de structures

La prédiction de structures est indissociable de la biologie moléculaire. En effet, pour comprendre et étudier la fonction d'une molécule, ou d'un complexe (groupement de molécules), le premier réflexe est d'étudier leur structure car le rôle d'une molécule dans une cellule est corrélé à sa forme. La plupart des mécanismes cellulaires se comportent selon le mode de fonctionnement clé/serrure, amplifiés par leurs propriétés chimiques. Ceci est encore plus vrai pour les ARN non-codants, dont les constituants de bases (les nucléotides) sont dépourvus de fortes propriétés bio-chimiques spécifiques (contrairement aux acides aminées des protéines). En cela, l'étude et la détermination des fonctions des ARN non-codants sont intimement liés à celles des structures.

Pour prédire la structure d'un ARN, deux grandes approches sont explorées : les méthodes ***ab initio*** et les méthodes **comparatives**, aussi appelées méthodes par **homologie**. Le but des deux méthodes est d'obtenir une structure, soit en partant uniquement de la séquence (méthodes *ab initio*), soit en utilisant des informations

supplémentaires (méthodes comparatives), comme des structures d'homologues ou encore des séquences d'homologues.

2.5.1 Les méthodes *ab initio*

Lorsque l'on fait référence à la prédiction de structure d'ARN *ab initio* en bio-informatique on parle généralement de prédiction de structure secondaire, car c'est l'approche la plus ancienne et la plus maîtrisée. Historiquement, le premier algorithme de prédiction de structure est l'algorithme de Nussinov [48]. Celui-ci prédit une structure secondaire d'un ARN à partir de sa séquence en calculant la structure maximisant le nombre d'interactions canoniques. L'algorithme est exact, de complexité cubique (en fonction de la longueur de la séquence) et basé sur la programmation dynamique (le même problème est aussi polynomial pour les structures secondaire avec pseudo-nœuds) . Même si cet algorithme ne donne pas de résultats concluants (les structures prédites étant en général loin de la réalité), le schéma de base de l'algorithme sera ensuite repris pour élaborer d'autres algorithmes donnant de très bons résultats comme par exemple les algorithmes sur la minimisation d'énergie libre selon le modèle de Turner [43].

Les programmes comme mfold [68], RNAfold [68, 44] ou encore RNAstructure [15] calcule une structure secondaire en optimisant selon un critère énergétique ayant fait ses preuves pour la prédiction de structures secondaires d'ARN. De façon très schématique, dans le modèle de Turner, les empilements d'interactions canoniques apportent de la stabilité (contrairement à une interaction canonique isolée), alors que les autres éléments de structures (les jonctions, les boucles ...) déstabilisent la structure. Les résultats obtenues sont alors plus proches de la réalité que ceux que l'on peut obtenir par l'algorithme de Nussinov.

Ces programmes sont tous plus ou moins dérivés de l'algorithme de Zuker *et al.* [68], lui même dérivé de l'algorithme de Nussinov. Cependant, alors que déterminer la structure secondaire d'énergie minimale (suivant un modèle du type de Turner) est polynomiale, déterminer la structure secondaire avec pseudo-nœuds d'énergie minimale est NP-Difficile, d'où la restriction aux structures secondaires.

Les précisions des prédictions que nous pouvons espérer obtenir par ces méthodes (c'est-à-dire la probabilité qu'une interaction prédite existe dans la vraie structure) se situent entre 60 et 90 % selon les benchmarks.

Enfin les dernières approches s'orientent maintenant vers les méthodes dites de "sampling" [16]. L'idée au cœur de ces méthodes n'est plus de calculer uniquement l'énergie libre de la structure optimale mais de toutes celles qui sont sous-optimales. Plus exactement, le principe est de déterminer pour chaque structure, la probabilité que l'ARN adopte cette structure grâce à la fonction de partition de Bolztman. Ensuite le but est de regrouper les structures en fonction de leurs ressemblances (nombre d'interactions différentes), les groupes de probabilité importante représentent alors une solution potentielle. Ces méthodes sont alors plus précises,

moins sujettes aux biais du modèle énergétique et surtout elles permettent de déterminer plusieurs structures pour un seul et même ARN, comme par exemple pour les riboswitches qui peuvent adopter plusieurs conformations [42].

La recherche actuelle s'oriente aussi aujourd'hui et de plus en plus sur la prédiction de structure avec pseudo-nœuds. Le succès de ces approches est encore un peu mitigé mais certains programmes (Cylofold [9], Pknots [51], RNAstructure [15]), obtiennent de bons résultats sur certains types précis de pseudo-nœuds.

La prédiction des structures secondaires étendues *ab initio* est encore vierge, et celle des structures $3D$ encore à l'état très expérimentale. On notera tout de même que les travaux accomplis ces dernières années laissent entrevoir une rapide amélioration de ces méthodes [14, 49, 34]

2.5.2 Les méthodes comparatives

Les méthodes de prédiction de structures par approche comparative se basent soit sur un alignement d'une structure (consensus ou non), soit sur la détermination d'un alignement multiple de séquences homologues basé sur la recherche d'une structure commune. En effet, il est évident que construire une structure consensus revient à prédire une structure pour un ensemble d'ARN et aligner une séquence d'ARN sur une structure (consensus ou non) permet d'inférer directement une structure de même niveau. Donc la prédiction de structure par homologie s'effectue par les même types de méthodes que la recherche d'homologues. La seule différence notable est le niveau de précision des alignements. Il n'est pas forcément utile de construire un alignement au nucléotide près pour déclarer deux ARN homologues (on peut envisager d'aligner les hélices par exemple), alors que pour déterminer la structure d'un ARN les nucléotides sont les éléments de base à considérer.

Les prédictions que l'on peut obtenir par ces méthodes (modèle de covariance ou alignement structure-séquence) ont une précision supérieure aux méthodes *ab initio*, mais nécessitent plus d'informations.

Chapitre 3

L'alignement structure-séquence d'ARN

Sommaire

3.1 Motivations et Enjeux

Nous allons maintenant nous intéresser au problème de l'alignement structure-séquence d'ARN. Comme nous l'avons déjà indiqué, l'alignement structure-séquence consiste à aligner deux molécules d'ARN, la première ayant une structure connue contrairement à la seconde dont seule la séquence est déterminée. Nous voulons donc aligner une structure (et sa séquence associée) avec une séquence.

Pour quelles raisons ce problème est-il intéressant ? Nous avons déjà vu que l'ARN peut évoluer très fortement en séquence tout en conservant sa structure et donc sa fonction. L'alignement structure-séquence, tel que nous allons le définir, vise à améliorer l'alignement de deux séquences lorsque celles-ci ont une faible similarité. Il a pour but de nous procurer un outil capable à la fois de détecter les homologues d'un ARN à partir de sa séquence et surtout de sa structure et aussi de reconstruire une structure par inférence à partir d'un alignement, c'est à dire résoudre les deux problématiques de biologie moléculaire que nous avons décris dans le chapitre précédent. Cet ambition est cautionné par le fait qu'il se positionne comme un problème exact en considérant les nucléotides et leurs interactions comme éléments de base, ainsi la précision des alignements que l'on souhaite obtenir sera suffisante pour résoudre les deux problèmes.

Comparée aux autres approches, comme le modèle de covariance, l'alignement structure-séquence n'est pas limité aux structures secondaires. Comme nous allons le voir, nous pouvons considérant n'importe quel niveau de structures (secondaire, secondaire avec pseudo-nœuds, secondaire étendue) même si de premier abord, la complexité du problème peut apparaitre un facteur limitant. Et c'est sur ce point que nous allons insisté, car si le problème de l'alignement structure-séquence possède de bonnes solution si la structure à aligner est une structure secondaire, il n'existe actuellement aucune solution généralisé acceptable pour les deux autres niveaux de structures.

Nous proposons alors une formalisation du problème d'alignement structure-séquence, permettant à la fois de considérer les pseudo-nœuds, les interactions multiples et non-canoniques. Dans une première partie nous allons définir rigoureusement le problème, puis nous analyserons les résultats de complexité établis depuis quelques années. Pour cela nous montrerons que le problème présenté est très proche (et même équivalent sur une partie des instances possibles) du problème d'édition de séquences arc-annotées dont nous rappellerons les fondamentaux. Ceci motivera l'utilisation des séquences arc-annotées pour la modélisation des structures d'ARN dans le cadre de l'alignement structure-séquence. Dans un deuxième temps, nous étudierons les principaux algorithmes d'alignement structure-séquence existant afin d'en extraire les idées caractéristiques et fondamentales que nous généraliserons et unifierons dans le chapitre suivant.

3.2 Définition du problème

Avant de définir formellement le problème d'alignement structure-séquence d'ARN nous allons choisir une représentation pour les structures d'ARN. Comme nous l'avons vu précédemment, une structure d'ARN secondaire, secondaire avec pseudo-nœuds ou secondaire étendue peut se représenter par un graphe. Du fait qu'il existe nécessairement un ordre intrinsèque, celui de la séquence nucléotidique, appelé $5' \rightarrow 3'$, il est naturel de vouloir représenter la séquence des nucléotides horizontalement. Partant de ce constat, Evans définit la notion de séquence arc-annotée [21], représentant une structure d'ARN comme une séquence associée à un ensemble d'interactions (Figure 3.1).

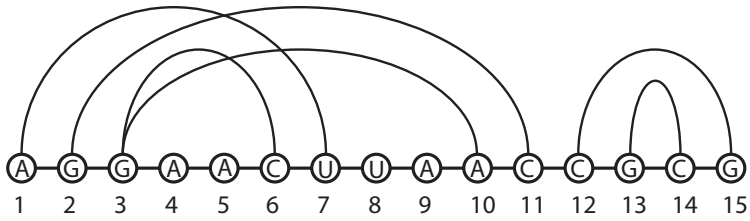


FIGURE 3.1 – *Représentation d'une séquence arc-annotée. La séquence en nucléotides de l'ARN est écrite horizontalement et les interactions entre nucléotides sont représentées par des arcs (géométriques).*

Définition 1. *(Séquence arc-annotée)*
Une **séquence arc-annotée** est un couple (S, P) où S est une séquence sur un alphabet Σ et P est un ensemble de couples de positions dans S.

Dans une séquence arc-annotée $A = (S, P)$, S représente naturellement la séquence en nucléotides de l'ARN et donc $\Sigma = \{A, U, G, C\}$. L'ensemble P représente l'ensemble des interactions entre nucléotides. C'est à dire, si $(i, j) \in P$, alors il existe une interaction entre le nucléotide en position i ($S_A[i]$) et le nucléotide en position j ($S_A[j]$), les nucléotides étant numérotées dans l'ordre usuel $5' \rightarrow 3'$. Afin de pouvoir distinguer les différents types d'interactions entre nucléotides, nous pouvons associer à chaque interaction un type. Dans toute la suite, lors que nous parlerons de structure, nous ferons systématiquement référence à une séquence arc-annotée (sauf mention particulière, par exemple une structure 3D).

Nous avons choisi de définir un alignement structure-séquence comme une **fonction de plongement** entre la structure (la séquence arc-annotée) et la séquence (Figure 3.2). Un autre formalisme possible aurait été de définir un alignement comme un couple de deux sur-séquences de même taille (une pour chacun des ARN) comme on représente classiquement (dans un fichier de données par exemple) un alignement. Ces différentes possibilités étant équivalentes, notre choix du plongement s'est fondé

sur sa simplicité d'écriture relative aux développements d'algorithmes et de preuves.

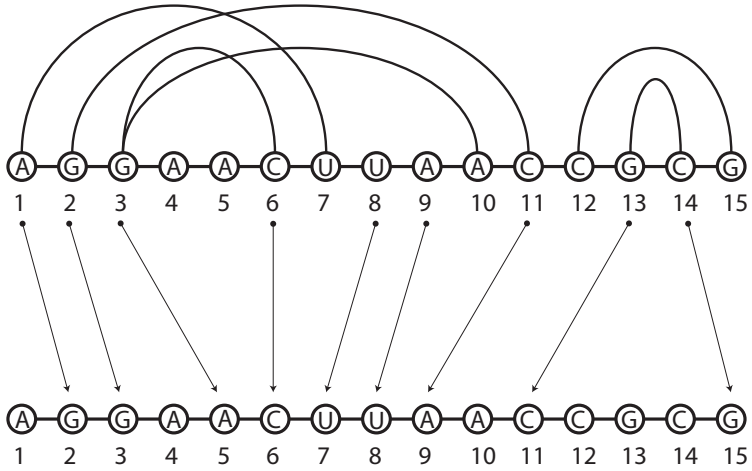


FIGURE 3.2 – *Représentation d'un alignement entre une séquence arc-annotée (partie haute) et une séquence (partie basse) comme fonction de plongement.*

Définition 2. *(Alignement structure-séquence)*
Un **alignement structure-séquence** entre une séquence arc-annotée $A = (S_A, P_A)$ de taille n et une séquence S_B de taille m est une fonction μ de $[1, n]$ dans $[1, m]$ tel que :
– μ est injective.
– μ préserve l'ordre : $\mu(i) < \mu(j) \Rightarrow i < j$

Nous dirons qu'une position i de A est **alignée** sur une position i' de S_B par μ si et seulement si $\mu(i) = i'$ (et inversement i' est aligné sur i). Cela signifie, de façon équivalente, que le nucléotide $S_A[i]$ est aligné sur le nucléotide $S_B[j]$.

L'injectivité implique qu'un nucléotide de l'un des deux ARN ne peut être en correspondance qu'avec au plus un seul autre nucléotide de l'autre ARN.

La préservation de l'ordre est quant à elle une contrainte d'ordre biologique.

Remarquons que certaines positions de la structure ou de la séquence peuvent ne pas avoir de correspondance. Dans ce cas, pour une telle position i de la structure on notera $\mu(i) = \perp$, et pour une position i' de la séquence on notera $\mu^{-1}(i') = \perp$. Nous dirons que ces positions sont **non-alignées**.

L'ensemble des alignements entre une séquence arc-annotée A et une séquence S_B est noté $\mathcal{F}(A, S_B)$, ou $\mathcal{F}$ si il n'y a pas d'ambiguïté.

Il nous reste maintenant à définir le problème à résoudre. Pour chaque alignement entre une structure A et une séquence S_B nous allons associer un **coût**. Ce coût va être calculé à partir d'**opérations élémentaires** ayant elles-mêmes un coût propre. Ainsi allons établir une liste d'opérations élémentaires caractérisant un alignement

(Figure 3.3). En premier lieu, nous définissons les opérations intervenant sur les éléments de séquences (S_A et S_B) :

- **Substitution** : alignement d'une position i de S_A sur une position j de S_B.
- **Suppression** : non-alignement d'une position i de S_A.
- **Insertion** : non-alignement d'une position i' de S_B.

Le coût d'une substitution dépend de la nature de la substitution. Il est évident que la substitution d'une adénine (A) en une autre adénine (A) doit avoir un coût inférieure ou égale à la substitution en une guanine (G). Les coûts de ces différentes substitutions peuvent être représentés (et stockés) par une matrice/fonction de coûts de substitutions, que nous nommerons γ. Ainsi le **coût d'une substitution** sera donné par $\gamma(S_A[i], S_B[\mu(i)])$, ou par abus de notation $\boldsymbol{\gamma(i, \mu(i))}$. Les **coûts d'une délétion** et **d'une insertion** sont quant à eux uniques et seront notés Gap.

Ce schéma de coûts (qui ne tient pas encore compte des interactions) est le schéma de base de l'alignement de deux séquences comme par exemple dans l'algorithme de Smith-Waterman. Dans les alignements de séquences utilisés en pratique, les positions non-alignées et consécutives sont généralement groupées ensemble pour l'attribution du coût.

Un **gap** (composé) est un ensemble maximum de positions consécutives non-alignées. **La longueur d'un gap** g notée $|g|$ est le nombre de positions qu'il contient et nous noterons $\boldsymbol{g \subset A}$ pour un gap s'effectuant dans la structure A et $\boldsymbol{g \subset B}$ pour un gap s'effectuant dans la séquence S_B. Nous pouvons ainsi considérer des **pénalités de gaps affines**, c'est-à-dire que le coût d'un gap est évalué par une fonction affine (fonction de sa longueur) $\lambda_A(|g|)$ (délétion) et $\lambda_B(|g|)$ (insertion). Si nous posons $\lambda_X(|g|) = a_X \times |g| + b_X$, $X \in \{A, B\}$, b_X est appelée **pénalité d'ouverture de gap**. Les algorithmes que nous allons analyser dans ce chapitre n'utilisent pas de pénalités d'ouverture de gaps. Néanmoins nous allons les prendre en compte dans la définition du problème d'alignement structure-séquence car nous les manipulerons dans le chapitre suivant.

Nous définissons ensuite les opérations intervenant sur les éléments de structure (P_A) :

Soit $i, j \in S_A$ tels que $(i, j) \in P_A$.

- **Substitution d'arc** : alignement de i et j de sur deux positions de S_B.
- **Altération d'arcs** : alignement de i sur une position de S_B et non-alignement de j ou non-alignement de i et alignement de j sur une position de S_B.
- **Suppression d'arcs** : non-alignement de i et j.

En considérant les interactions comme des entités propres, il est alors nécessaire de pouvoir attribuer un coût à chacun de leurs alignements (et non-alignements). De même que pour une substitution de nucléotide, le coût d'une substitution d'arc dépend de sa nature. Les coûts des différentes substitutions seront représentés par une matrice ArcSub. Ainsi le **coût d'une substitution d'arc** (i, j) par μ sera donné par $\boldsymbol{\mathsf{ArcSub}(i, j, \mu(i), \mu(j))}$. Si nous voulons considérer plusieurs types d'interactions,

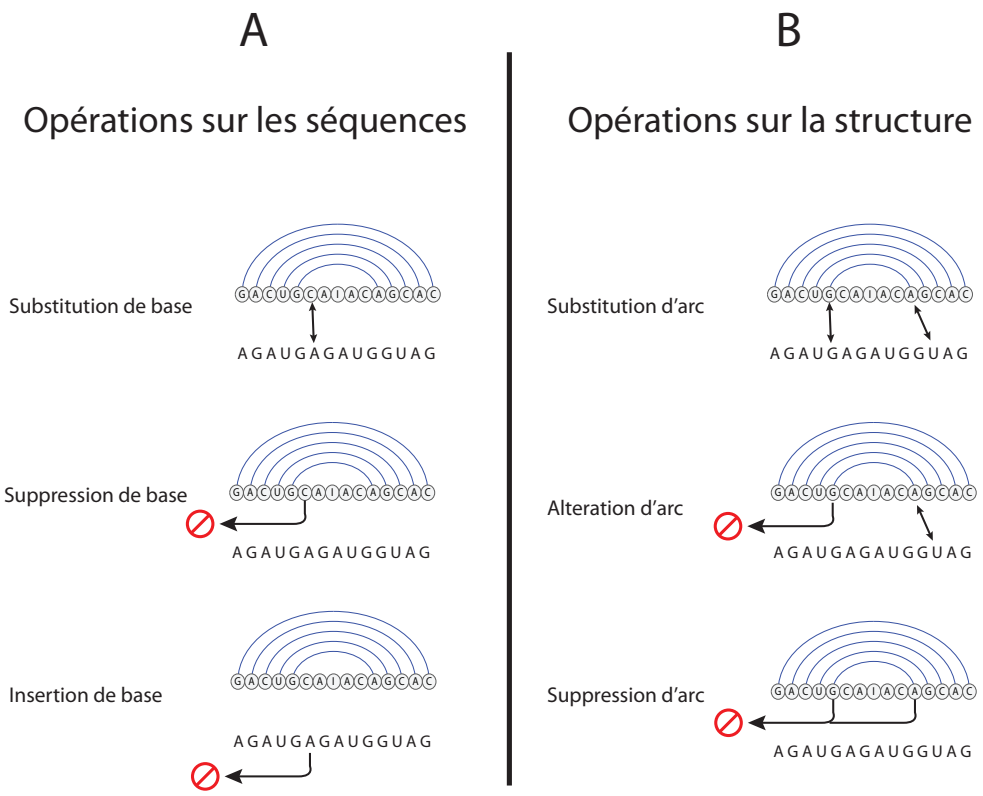


FIGURE 3.3 – *A. opérations sur les positions et B. opérations sur les interactions.*

il faudra bien évidement considérer autant de fonctions de substitutions d'arcs. Pour ne pas alourdir les notations, nous n'utiliserons qu'une seule et unique fonction, qu'il faudra alors replacer dans son contexte dans le cas général. Les **coûts des altérations d'arcs** seront représentés par deux fonctions différentes $\mathsf{ArcAltL}(\boldsymbol{i, j, \mu(i)})$ et $\mathsf{ArcAltR}(\boldsymbol{i, j, \mu(j)})$ (auxquels il faudra porter la même attention concernant les différentes natures d'interactions possibles). Enfin les **coûts des suppression** seront représentés par la fonction $\mathsf{ArcSup}(\boldsymbol{i, j})$.

Afin de factoriser toutes ces notations, nous noterons $\boldsymbol{\varphi(i, j, \mu(i), \mu(j))}$ le coût d'alignement de l'arc $(i, j) \in P_A$ par μ, qui sera égale à $\mathsf{ArcSub}(i, j, \mu(i), \mu(j))$ dans le cas d'une substitution, $\mathsf{ArcAltL}(i, j, \mu(i))$ ou $\mathsf{ArcAltR}(i, j, \mu(j))$ pour une altération ou encore $\mathsf{ArcSup}(i, j)$ dans le cas d'une suppression :

$$\varphi(i, j, \mu(i), \mu(j)) = \begin{cases} \mathsf{ArcSub}(i, j, \mu(i), \mu(j)) & \text{si } \mu(i) \textit{ et } \mu(j) \neq \perp \\ \mathsf{ArcAltL}(i, j, \mu(i)) & \text{si } \mu(i) \neq \perp \textit{ et } \mu(j) = \perp \\ \mathsf{ArcAltR}(i, j, \mu(j)) & \text{si } \mu(i) = \perp \textit{ et } \mu(j) \neq \perp \\ \mathsf{ArcSup}(i, j) & \text{si } \mu(i) = \mu(j) = \perp \end{cases}$$

Nous calculerons ensuite le coût d'un alignement en effectuant la somme de tous les coûts des opérations élémentaires le composant.

Définition 3. *(Coût d'un alignement)*
Le **coût d'un alignement** structure-séquence μ, entre une séquence arc-annotée A et une séquence S_B est défini par :

$$\mathsf{Coût}(\mu) = \sum_{i\in[1,n],\mu(i)\neq\perp} \gamma(i,\mu(i)) + \sum_{\text{gap } g\subset A} \lambda_A(|g|) + \sum_{\text{gap } g\subset S_B} \lambda_B(|g|) + \sum_{(i,j)\in P_A} \varphi(i,j,\mu(i),\mu(j)) \tag{3.1}$$

Parmi tous les alignements possibles entre une structure A et une séquence S_B, l'un d'entre eux (ou certains d'entre eux) possède un coût minimum. C'est cet ou ces alignements que nous devons rechercher pour résoudre le problème d'alignement structure-séquence.

Définition de problème 4. *Problème de l'alignement structure-séquence (*PASS*)*
Étant données une séquence arc-annotée A et une séquence S_B, le **problème d'alignement structure-séquence** consiste à trouver l'alignement de coût minimum entre A et S_B.

3.3 État de l'art

Le problème d'alignement structure-séquence d'ARN tel que nous venons de le définir ne fut au départ qu'indirectement étudié, et ce par l'intermédiaire du problème d'édition de séquences arc-annotées. Jiang *et al.* [31] ainsi que Blin *et al.* [10] notamment, étudièrent en détail ce problème et établirent des résultats de complexité très complets.

Étant données deux séquences arc-annotées, le problème d'édition consiste à trouver la suite d'opérations (semblables aux opérations que nous avons défini) la moins coûteuse transformant la première séquence arc-annotée en la deuxième. Ce problème peut ensuite s'étudier en fonction de la complexité des structures des séquences arc-annotées : structure primaire ⇔ PLAIN, structure secondaire sans jonctions ⇔ CHAIN, structure secondaire ⇔ NESTED, structure secondaire avec pseudo-nœuds ⇔ CROSSING, structure secondaire étendue ⇔ UNLIMITED. EDIT(*classe* 1, *classe* 2) représente alors le problème d'édition entre une structure appartenant à la *classe* 1 et une structure appartenant à la *classe* 2. Ainsi Jiang *et al.* montrèrent que le problème d'édition est MAX SNP-difficile dès lors que l'une des structures considérées appartient à la classe CROSSING (voir Table 3.1), quelle que soit la classe de la seconde structure [31]. Ils donnèrent aussi un algorithme polynomial pour la résolution du problème EDIT(NESTED,PLAIN), que nous nommerons

algorithme de Jiang.

	Complexité
EDIT(PLAIN, PLAIN)	$O(n \cdot m)$
EDIT(NESTED, PLAIN)	$O(n \cdot m^3)$
EDIT(CROSSING, PLAIN)	MAX SNP-difficile
EDIT(UNLIMITED, PLAIN)	MAX SNP-difficile

TABLE 3.1 – *Tableau récapitulatif des complexités du problème* EDIT *où au moins l'une des structures appartient à la classe* PLAIN.

La raison pour laquelle nous abordons ici l'édition de séquences arc-annotées est que ce problème est équivalent au problème d'alignement structure-séquence d'ARN pour des instances de structures appartenant aux classes PLAIN, NESTED, CROSSING comme l'ont montré Zhang *et al.* [67]. Pour la classe UNLIMITED, l'équivalence devient caduque. En fait, dès lors que nous considérons des interactions multiples l'équivalence ne tient plus. En effet, les opérations que nous avons définies ne suffisent plus pour construire un script d'édition, c'est-à-dire une succession d'opérations élémentaires transformant la structure en la séquence. Prenons l'exemple de l'alignement de la figure Figure 3.4. Quelle que soit l'opération que l'on voudrait effectuer en premier (sur un nucléotide ou une interaction) elle impacterait nécessairement sur une autre interaction, et l'opération ne serait donc plus, par définition, élémentaire. Néanmoins étant donné que CROSSING⊂UNLIMITED, le problème d'alignement structure-séquence sera aussiMAX SNP-difficile pour cette classe.

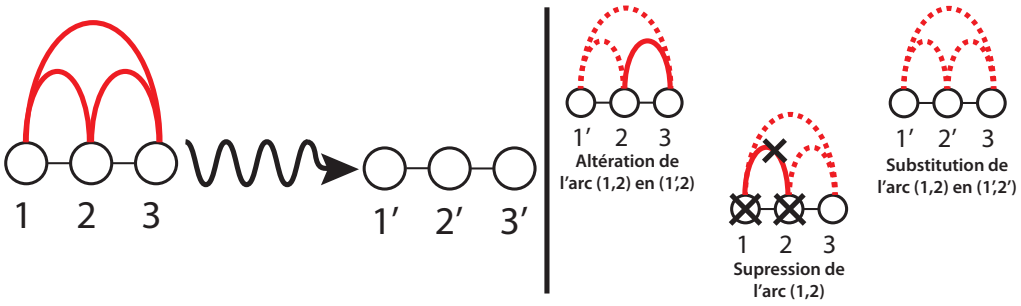


FIGURE 3.4 – *Alignement pour lequel le problème d'équivalence entre l'alignement structure-séquence et* EDIT *n'existe plus. Quelle que soit le script l'édition que nous voulons construire pour transformer la structure en la séquence (figure de gauche), nous devons utiliser des opérations sur les arcs. Quelle que soit l'opération (substitution, altération, suppression), celle-ci impliquera nécessairement deux arcs. Sur la figure de droite sont représentés en pointillés les arcs impliqués dans l'opération indiquée.*

Donc en fonction du niveau de structures, les résultats de complexité associés à l'édition de séquences arc-annotées peuvent ou ne peuvent pas nous servir de référence. Nous pouvons par conséquent, comme pour le problème d'édition, caractériser le niveau de complexité du problème d'alignement structure-séquence d'ARN en fonction de la classe des structures (voir Table 3.2) que l'on considère.

Plus qu'un résultat de complexité, cette équivalence nous permet aussi d'utiliser l'algorithme de Jiang pour résoudre notre problème pour les structures appartenant à la classe NESTED. De plus, cet algorithme a ensuite servi de référence et de base pour la construction des algorithmes de Han [29] et de Wong [64, 63]. Ces algorithmes, s'intéressant cette fois-ci directement à l'alignement structure-séquence, sont de complexités polynomiales (en temps) et résolvent le problème pour des sous classes de structures des classes CROSSING et UNLIMITED appelées pseudo-nœuds standard (SPK), pseudo-nœuds non-standard simple (SNP) et triple hélices standard (STH) voir Table 3.2).

	Complexité
PLAIN	$n \cdot m$
NESTED	$n \cdot m^3$
CROSSING	MAX SNP-difficile
UNLIMITED	MAX SNP-difficile

TABLE 3.2 – *Tableau récapitulatif des complexités du problème d'alignement structure-séquence.*

Ceci étant, alors que l'algorithme de Jiang est compréhensible et simple dans son écriture, les algorithmes dérivés de Han [29] et Wong [64, 63] sont quant à eux beaucoup plus complexes. Comme nous allons le voir, ils sont tous basés sur la programmation dynamique et donc le développement de formules de récurrence. La clé de ces algorithmes est (comme pour tout algorithme de programmation dynamique) de définir un type de sous-structures permettant de résoudre le problème de façon récursive. Or, si celle-ci semble naturelle pour les structures de type NESTED, cela devient très vite un casse tête lorsque l'on doit manipuler une sous classe de CROSSING ou même pire une sous classe de UNLIMITED.

La description des algorithmes qui suit a deux buts principaux. Le premier est de mettre en évidence l'évolution croissante de la difficulté de leur écriture, impliquant ainsi la nécessité de simplifier les approches. L'autre objectif majeur est d'extraire les idées principales des algorithmes afin de les généraliser et de les unifier.

3.3.1 Principe algorithmique et notions importantes

Tous les algorithmes que je vais maintenant décrire sont donc basés sur le principe de la programmation dynamique. Le premier développement pour l'alignement

de séquences biologiques d'un tel algorithme fut donné par Needleman et Wunsch pour leur algorithme connu sous le nom de **Needleman-Wunsch** [46]. Même si cet algorithme n'est plus utilisé dans la pratique en bioinformatique, il illustre néanmoins parfaitement le principe de base de la programmation dynamique appliqué à l'alignement de séquences.

Comme tout algorithme de programmation dynamique, il résout le problème récursivement. En l'occurrence, il utilise le coût du meilleur alignement de deux sous-séquences pour calculer le coût du meilleur alignement de deux autres sous-séquences plus grandes.

Supposons donc que l'on veuille aligner deux séquences d'ARN, S_A et S_B, en utilisant les opérations sur les séquences ainsi que les coûts associés que nous avons défini précédemment (sans pénalité d'ouverture de gap). Notons $\mathsf{Coût}(i, i')$ le coût de l'alignement optimum entre $S_A[1, i]$ et $S_B[1, i']$, où $S_A[1, i]$ représente la sous-séquence de S_A : $S_A[1]S_A[2] \dots S_A[i]$ (il en va de même pour $S_B[1, i']$). Pour calculer ce coût nous allons supposer que nous connaissons les coûts des alignements optimaux entre $S_A[1, i-1]$ et $S_B[1, i'-1]$, entre $S_A[1, i]$ et $S_B[1, i'-1]$, et enfin entre $S_A[1, i-1]$ et $S_B[1, i']$. Nous pouvons alors étendre en une opération chacun des trois sous-alignements optimaux afin de former des alignements entre $S_A[1, i]$ et $S_B[1, i']$ parmi lesquels se trouvera nécessairement l'alignement optimal (Figure 3.5).

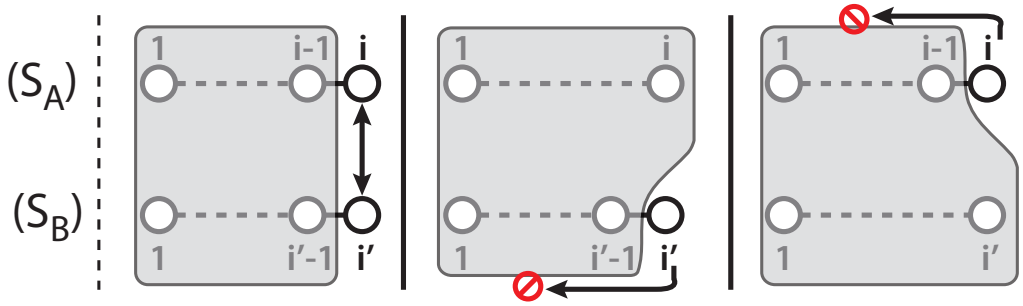


FIGURE 3.5 – *Pour calculer* Coût*(i,i'), nous pouvons utiliser le coût de trois sous-alignements optimaux (en grisé) :* Coût*(i-1,i'-1) (à gauche),* Coût*(i,i'-1) (au milieu) et* Coût*(i-1,i') (à droite). Dans chacun des cas, nous étendons en une opération (en non-grisé) le sous-alignement pour réaliser un alignement entre $S_A[1, i]$ et $S_B[1, i']$. Le coût de chacun des alignements est égal au coût du sous-alignement correspondant auquel on ajoute le coût de l'opération utilisée lors de l'extension. Parmi ces trois alignements se trouve nécessairement l'alignement optimal.*

De ces trois sous-alignements entre $S_A[1, i]$ et $S_B[1, i']$ (nouvellement formés) nous voulons bien-entendu choisir celui qui va minimiser le coût. Le premier choix correspond à une substitution, le deuxième une insertion et le troisième une suppression. Cet alignement sera alors (par construction) optimal. De ce fait, la formule de récurrence suivante permet de déterminer le coût de l'alignement optimal entre

$S_A[1,i]$ et $S_B[1,i']$.

$$\mathsf{Coût}(i,i') = \min \begin{cases} \mathsf{Coût}(i-1,i'-1) + \gamma(i,i') \quad (substitution) \\ \mathsf{Coût}(i,i'-1) + \mathsf{Gap} \quad (insertion) \\ \mathsf{Coût}(i-1,i') + \mathsf{Gap} \quad (suppression) \end{cases}$$

En initialisant en premier lieu $\mathsf{Coût}(i,0) = \mathsf{Gap} \times i$ et $\mathsf{Coût}(0,i') = \mathsf{Gap} \times i'$, on accède au coût du meilleur alignement en appliquant récursivement la formule à partir de $\mathsf{Coût}(n,m)$.

Par la suite, nous effectuons un **retour sur trace** pour construire l'alignement. Pour chaque $\mathsf{Coût}(i,j)$ que nous calculons, le choix découlant de la recherche du minimum (substitution, insertion, suppression) est enregistré. Ainsi, en partant du coût final $\mathsf{Coût}(n,m)$, nous pouvons remonter les choix effectués jusqu'aux premiers coûts et ainsi reconstruire l'alignement optimal dont nous avions calculé le coût. Cette opération s'appelle le retour sur trace. Elle est commune à tous les algorithmes de programmation dynamique, et donc à tous les algorithmes d'alignement structure-séquence présentés.

Pour cet algorithme de programmation dynamique nous effectuons, pour chaque $\mathsf{Coût}(i,i')$ un nombre constant d'opérations (3 additions et une recherche de minimum parmi 3 valeurs). Nous avons besoin de calculer toutes les valeurs des coût $\mathsf{Coût}(i,i')$ possibles, ce qui implique que l'algorithme Needleman-Wunsch est de complexité quadratique en temps : $O(n \cdot m)$. De même, nous avons besoin de conserver en mémoire chacun des $\mathsf{Coût}(i,i')$ (plus un pointeur pour le retour sur trace), ce qui aboutit à un algorithme de complexité quadratique en espace : $O(n \cdot m)$.

Ceci est le principe de base de la programmation dynamique que nous allons appliquer par la suite, non plus à deux séquences, mais à une structure et une séquence. Afin de faciliter la discussion, je vais définir ici quelques notions importantes.

Comme nous venons de le laisser entendre, pour chaque algorithme, nous allons établir un système de sous-structures (et non plus sous-séquences) pour résoudre le problème de façon récursive. Pour cela nous allons avoir besoin de la notion de **structure induite**.

Définition 5. *(Structure Induite)*
Soit $A = (S_A, P_A)$ une structure et S' une sous-séquence (pas nécessairement contiguë) de S_A. Alors la **structure induite** de S' est la structure (S', P') où $P' = \{(i,j) \in P_A | S_A[i], S_A[j] \in S'\}$.

3.3.2 L'algorithme de Jiang

L'algorithme de Jiang tel qu'il est décrit dans l'article d'origine résout le problème d'édition EDIT(NESTED,PLAIN). D'après l'équivalence soulignée précédemment, il résout donc aussi le problème d'alignement structure-séquence pour des structures secondaires. La difficulté supplémentaire introduite avec la considération des structures secondaires (par rapport à l'alignement de deux séquences) vient du fait que les nucléotides possédant une interaction ne peuvent plus être alignés de façon indépendante. En effet, nous devons maintenant considérer un nucléotide et son partenaire (si il existe) comme une entité propre.

Soit $\boldsymbol{A(i,j)}$ **la sous-structure de** A **induite par la sous-séquence** $\boldsymbol{S_A[i,j]}$, alors le calcul du coût du meilleur alignement par l'intermédiaire de sous-structures du type $A(1,i)$ comme précédemment n'est plus possible. En effet, supposons que nous ayons calculé le coût de l'alignement optimal entre $A(1,i)$ et $S_B[1,i']$, pour un i et un i' quelconque. Supposons aussi que i soit en interaction avec un nucléotide j tel que $i < j$. Le coût de l'alignement optimal entre $A(1,i)$ et $S_B[1,i']$ ne tient pas compte de l'interaction (i,j). Par conséquent, nous ne pouvons pas garantir que le choix effectué pour l'alignement de i restera le meilleur choix pour l'alignement de (i,j) (remise en cause de l'optimalité voir Figure 3.6). Il se pourrait qu'un choix d'alignement pour i paraissait mon bon lors de l'alignement entre $A(1,i)$ et $S_B[1,i']$ (coût supérieur), mais que celui-ci soit en fin de compte moins coûteux pour l'alignement de (i,j) (Figure 3.6). La source du problème est que lors de l'alignement de la position j (la position de droite d'une interaction), nous ne connaissons plus l'alignement de la position i. Il faudrait alors effectuer un retour sur trace, mais le fait d'utiliser un choix non effectué dans l'étape précédente de la récurrence est synonyme de remise en cause de l'optimalité. Donc le choix d'alignement pour i_1 et le coût qu'il engendre ne peut se calculer qu'en conjonction du choix de l'alignement pour i_2.

Nous abordons ici la notion que nous appellerons **notion de structure valide** et dont nous nous servirons tout au long de cet état de l'art. Nous appellerons **sous-structure valide** une sous-structure A' de A telle qu'il n'existe pas de nucléotide appartenant à A' ayant un partenaire n'appartenant pas à A' (Figure 3.7). Autrement dit, pour toute interaction (i,j) de A soit $S_A[i,j]$ contient $S_{A'}$, soit $S_{A'}$ contient $S_A[i,j]$ soit $S_{A'}$ et $S_A[i,j]$ sont disjoints. L'utilité d'une telle propriété est de permettre le calcul du coût d'un alignement optimal par récurrence en utilisant l'alignement de sous-structures valides en nombre limité à chaque appel récursif. La contrainte de non croisement d'interactions imposée aux structures secondaires donne intuitivement un caractère récursif aux sous-structures valides (soit elles sont indépendantes, soit l'une est contenue dans l'autre) et permet dans les faits de les utiliser pour résoudre récursivement le problème d'alignement structure-séquence.

En utilisant deux indices par séquence, et en reprenant le raisonnement de l'algorithme de Needleman-Wunsch nous allons alors sans trop de difficultés pou-

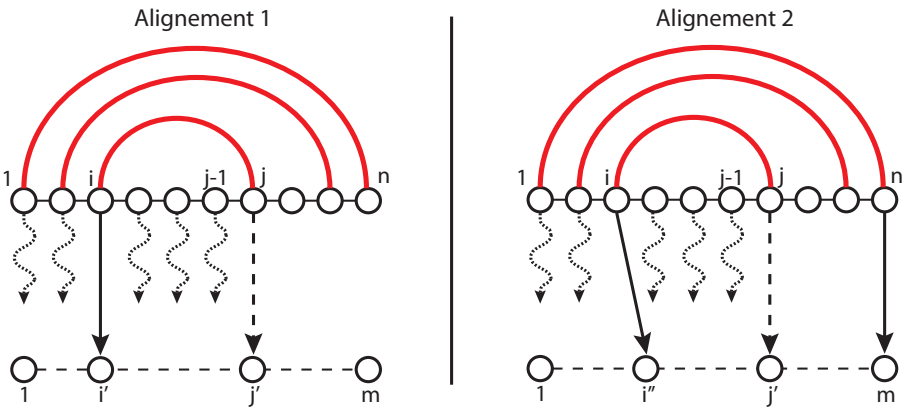


FIGURE 3.6 – *Exemple montrant la complexité supplémentaire introduite par la prise en compte des interactions comme entités propres. Deux alignements identique à l'alignement d'une position près (position i). Alignement* 1 *(à gauche) : alignement construit comme précédemment. Alignement* 2 *(à droite) : alignement construit en minimisant le coût de l'alignement de* (i, j)*. Rien ne peut garantir que le coût de l'alignement* 1 *soit inférieur à celui de l'alignement* 2*.*

voir construire une formule de récurrence adaptée aux structures secondaires. Soit $\mathsf{Coût}(i, j; i', j')$ le coût de l'alignement optimal entre une sous-structure valide $A(i, j)$ et la sous-séquence $S_B[i', j']$. Pour calculer ce coût nous devons utiliser les coûts des alignements optimaux de sous-structures contenues dans $A(i, j)$ (avec des sous-séquences de $S_B[i', j']$) qui doivent être elles aussi valides afin de ne jamais remettre en cause l'optimalité. En utilisant cette démarche, Jiang montre alors que l'on peut séparer les différentes possibilités en plusieurs sous-cas.

Cas1, $(i, j) \in P_A$

Si nous considérons le cas particulier où $(i, j) \in P_A$, alors $A(i, j)$ est valide (car A est une structure secondaire). De même $A(i + 1, j - 1)$ est elle aussi valide, ce qui implique que nous pouvons donc utiliser ses alignement optimaux pour le calcul de $\mathsf{Coût}(i, j; i', j')$ (non remise en cause de l'optimalité). Pour obtenir un alignement entre $A(i, j)$ et $S_B[i', j']$ à partir d'un alignement impliquant $A(i + 1, j - 1)$ nous avons trois possibilités, correspondant à la substitution d'arc, l'altération d'arc ou la destruction d'arc. Notons :

$$ARC - SUBSTITUTION = \mathsf{Coût}(i + 1, j - 1; i' + 1, j' - 1) + \mathsf{ArcSub}(i, j; i', j')$$

$$ALTERATION = \min \begin{cases} \mathsf{Coût}(i + 1, j - 1; i', j' - 1) + \mathsf{ArcAltL}(i, j; j') \\ \mathsf{Coût}(i + 1, j - 1; i' + 1, j') + \mathsf{ArcAltR}(i, j; i') \end{cases}$$

$$ARC - SUPPRESSION = \mathsf{Coût}(i + 1, j - 1; i', j') + \mathsf{ArcSup}(i, j)$$

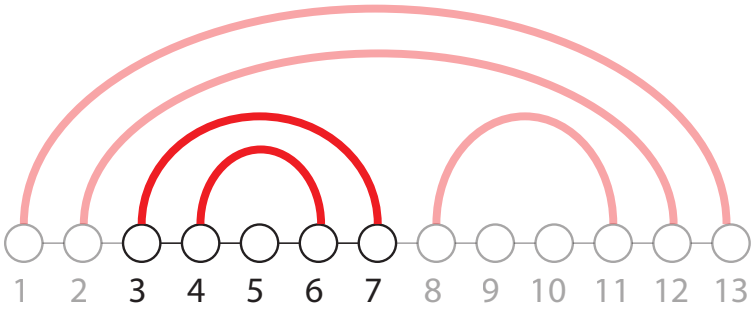


FIGURE 3.7 – *Exemple de sous-structure valide (en gras) d'une structure secondaire (grisée).*

En plus de ces trois cas, nous pouvons aussi construire un alignement entre $A(i,j)$ et $S_B[i',j']$ directement à partir d'un alignement entre $A(i,j)$ et $S_B[i'+1,j']$ ou $S_B[i',j'-1]$, correspondant à deux insertions différentes. Notons :

$$INSERTION1 = \min \begin{cases} \mathsf{Coût}(i,j;i',j'-1) + \mathsf{Gap} \\ \\ \mathsf{Coût}(i,j-1;i'+1,j') + \mathsf{Gap} \end{cases}$$

Cas2, $(i,j) \notin P_A$

Supposons maintenant que $(i,j) \notin P_A$. Sous cette condition, nous devons seulement considérer les cas où $A(i,j)$ est valide. Si tel est le cas, pour calculer $\mathsf{Coût}(i,j;i',j')$ comme précédemment nous devons une fois de plus utiliser les coûts de sous-alignements faisant intervenir des sous-structures valides, et pour cela nous allons distinguer deux différents cas, celui où j est en interaction et celui où il ne l'est pas.

Si j n'est pas en interaction alors $A(i,j-1)$ est valide et par conséquent $\mathsf{Coût}(i,j;i',j')$ peut se calculer à partir du coût d'alignements faisant intervenir $A(i,j-1)$ et en y ajoutant le coût d'une substitution ou le coût d'une suppression. Notons :

$$SUBSTITUTION = \mathsf{Coût}(i,j-1;i,j'-1) + \gamma(j,j')$$

$$SUPPRESSION = \mathsf{Coût}(i,j-1;i,j') + \mathsf{Gap}$$

À cela, nous pouvons aussi ajouter le cas où l'on construit un alignement entre $A(i,j)$ et $S_B[i',j']$ directement à partir d'un alignement entre $A(i,j)$ et $S_B[i',j'-1]$

correspondant à une insertion. Notons :

$$INSERTION2 = \mathsf{Coût}(i, j; i, j' - 1) + \mathsf{Gap}$$

L'hypothèse $(i, j) \notin P_A$ n'implique pas que j soit libre, et donc n'implique pas que $A(i, j-1)$ soit valide. Néanmoins, comme nous avons supposé que $A(i, j)$ l'était, si j est en interaction avec une position k, alors nécessairement $k \in]i, j[$ et par suite $A(i, k-1)$ et $A(k, j)$ sont deux sous-structures valides (car A est une structure secondaire). Pour calculer $\mathsf{Coût}(i, j; i', j')$ à partir de sous-structures nous n'avons donc pas d'autre choix que de considérer les coûts des alignements optimaux impliquant les deux sous-structures $A(i, k-1)$ et $A(k, j)$. Le coût de l'alignement optimal entre $A(i, j)$ et $S_B[i', j']$ sera égale à la plus petite somme des coûts des alignements optimaux de $A(i, k-1)$ avec $S_B[i', k'-1]$ et de $A(k, j)$ avec $S_B[k', j']$ pour tout $k' \in [i', j']$. Concrètement nous n'attribuons pas d'opérations (et donc pas de coût supplémentaire) à ce cas, mais divisons le problème en deux sous-problèmes. Ces cas correspondent aux jonctions multiples de la structure. Notons :

$$JONCTION = \min_{i' \leq k' \leq j'} \{\mathsf{Coût}(i, k-1; i', k'-1) + \mathsf{Coût}(k, j; k', j')\}$$

où k est en interaction avec j

Dans notre prise en compte du **Cas2**, nous avons raisonné uniquement sur la position j. Nous pourrions bien évidement effectuer le même travail avec la position i. Néanmoins nous n'en n'avons pas besoin, les appels récursifs que nous allons effectuer nous permettront systématiquement de nous retrouver dans un des cas traités.

Pour résumer, nous pouvons calculer $\mathsf{Coût}(i, j; i', j')$ selon la formule de récurrence :

Cas1. $(i, j) \in P_A$ et $1 \leq i' \leq j' \leq m$,

$$\mathsf{Coût}(i, i'; j, j') = \min \begin{cases} ARC - SUBSTITUTION \\ ALTERATION \\ DESTRUCTION \\ INSERTION1 \end{cases}$$

Cas2. $(i,j) \notin P_A$ et $1 \leq i' \leq j' \leq m$ et $A(i,j)$ valide
Nous distinguons deux sous cas :

-Si i' est libre, alors

$$\mathsf{Coût}(i,i';j,j') = \min \begin{cases} SUBSTITUTION \\ DELETION \\ INSERTION2 \end{cases}$$

-Si j n'est pas libre alors

$$\mathsf{Coût}(i,i';j,j') = JONCTION$$

Le coût de l'alignement optimal entre A et S_B est alors donné par $\mathsf{Coût}(1,n;1,m)$. Comme pour l'algorithme de Needleman-Wunsch il faut au préalable initialiser la récurrence.

Initialisation1. Pour tout $i \in [1,n]$ et $1 \leq j \leq j' \leq m$,

$$\mathsf{Coût}(i,i-1;j,j') = (j'-j+1) \times \mathsf{Gap}$$

Initialisation2. Pour tout $1 \leq i \leq i' \leq n$ et $j \in [1,m]$ tels que $A(i,j)$ valide,

$$\mathsf{Coût}(i,i';j,j-1) = \sum_{(i_1,i'_1) \in P_A[i,i']} \mathsf{ArcSup}(i_1,i'_1) + (i'-i+1) \times \mathsf{Gap}$$

Nous nous intéressons alors à la complexité de l'algorithme découlant de la récurrence ci-dessus. Toutes les formules de récurrence ainsi que les initialisations se calculent en temps constant, mis à part $JONCTION$ qui demande $O(m)$ comparaisons.

Par conséquent, le calcul des coûts correspondants au **Cas1** nécessite donc un temps de calcul de $O(n \cdot m^2)$ étant donné qu'il y a au plus $n/2$ interactions dans A. L'**Initialisation1** demande elle aussi un temps de calcul de $O(n \cdot m^2)$.

Pour déterminer la complexité (au pire) liée au **Cas2**, nous devons déterminer le nombre de sous-structures valides $A(i,j)$ telles que $(i,j) \notin P_A$ en séparant les deux sous cas (j est ou n'est pas impliqué dans une interaction). Dans les deux cas, au pire, il existe $O(n^2)$ sous-structures valides possibles, et donc $O(n^2)$ paires d'indices possibles. Donc le calcul des coûts correspondants au **Cas2** nécessite un temps de calcul de $O(n^2 \cdot m^2)$ pour le sous premier sous cas (j n'est pas impliqué dans une interaction) et de $O(n^2 \cdot m^3)$ pour le second sous-cas (j est pas impliqué dans une interaction). Par le même raisonnement, nous pouvons conclure que l'**Initialisation2**

nécessite un temps de calcul de $O(n^2 \cdot m)$.

Par conséquent, la complexité en temps et au pire d'un algorithme de programmation dynamique basé sur ces équations de récurrence est de $O(n^2 \cdot m^3)$. Pourtant, nous avions annoncé et même écrit dans la Table 3.2 que le problème pouvait être résolu en $O(n \cdot m^3)$. Si le **Cas1** et **Initialisation1** sont conforme à l'algorithme de Jiang, les contraintes (sur i et sur j) du **Cas2** et **Initialisation2** ne le sont pas.

Telles que ces contraintes sont écrites, nous allons calculer un nombre significatif de coûts qui ne vont jamais nous servir. En effet, seuls les coûts $\mathsf{Coût}(i, j; i', j')$ que nous pouvons atteindre (par récurrence) à partir de $\mathsf{Coût}(1, n; 1, m)$ doivent être calculés. Or formulées ainsi, les équations ne permettent pas d'atteindre un coût $\mathsf{Coût}(i, j; i', j')$ où ni i ni $i-1$ ne sont en interaction (mise à part le cas spécial $i = 1$). En effet, un tel $\mathsf{Coût}(i, j; i', j')$ ne pourrait pas être atteint par le **Cas1** (car dans ce cas $i-1$ serait en interaction, Figure 3.8) et ni par le **Cas2** puisque nous ne faisons pas évoluer l'indice i dans premier sous-cas et que dans le deuxième, les deux appels de récurrences se font sur i d'une part et sur k d'autre part, qui lui est en interaction (Figure 3.8). De fait, nous n'avons pas non plus besoin d'initialiser ni de calculer ces cas.

Donc sans changer la spécification de l'algorithme, mais en rajoutant la contrainte : $i-1$ est en interaction ou $i = 1$, au **Cas2** et à l'**Initialisation2** nous allons gagner en complexité. Avec cette contrainte supplémentaire, pour toute paire d'interactions distinctes $(i_1 - 1, i_1'')$ et $(i_2 - 1, i_2'')$ de P_A alors toute paire de couples (i_1, j_1), (i_2, j_2) sont nécessairement disjoints (c'est-à-dire que $j_1 \neq j_2$). Par conséquent, le nombre de $\mathsf{Coût}(i, j; i', j')$ que nous devons calculer est maintenant de $O(n \cdot m^2)$ pour le **Cas2** et de $O(n \cdot m)$ pour l'**Initialisation2**. Par suite, la complexité en temps de l'algorithme de Jiang est de $O(n \cdot m^3)$.

Nous pouvons cependant affiner encore un peu plus ce résultat de complexité. La composante cubique de la complexité (m^3) est due exclusivement au deuxième sous cas du **Cas2**. Or cette situation apparait seulement pour les indices i et j caractérisant la première interaction d'une hélice impliqué dans une jonction. Si en théorie ce nombre peut être de l'ordre de n, dans la pratique l, le nombre d'hélices impliquées dans des jonctions d'une structure est faible, et très largement inférieur à la longueur de la structure. Il est donc intéressant de paramétrer la complexité du problème par l : $O(l \cdot m^3 + n \cdot m^2)$.

La complexité en espace est ici entièrement corrélée au nombre de coûts d'alignements de sous-structure/sous-séquence que nous devons calculer puis stocker (plus des pointeurs pour le retour sur trace). Or, comme nous l'avons vu, nous calculons $O(n \cdot m^2)$ coûts différents, donc la complexité en espace est donc aussi égale à $O(n \cdot m^2)$.

Nous venons de voir que la prise en compte des structures secondaires pour l'ali-

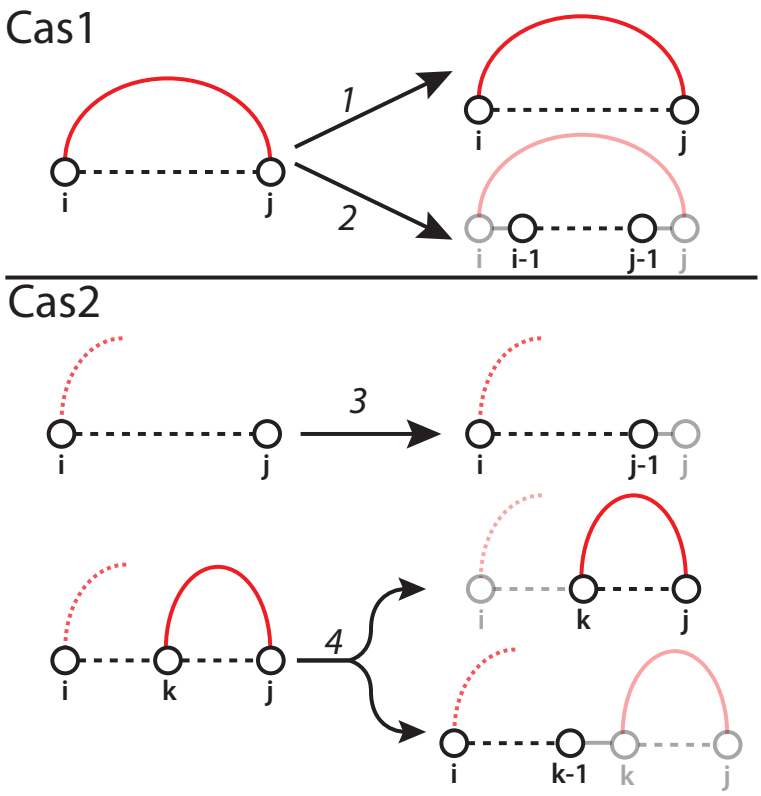


FIGURE 3.8 – *Exemple montrant les différentes possibilités de sous-structures valides utilisées lors des appels de récurrence. En commençant la récurrence avec la structure toute entière* $(i = 1, j = n)$ *il n'est pas possible de tomber sur un couple* (i, j) *(avec* $i \neq 1$*) tel que ni* i *ni* $i - 1$ *soit en interaction. 1 :* $INSERTION1$*,* $2 : ARC - SUBSTITUTION, ARC - ALTERATION, ARC - SUPPRESSION$*,* $3 : SUBSTITUTION, SUPPRESSION$*,* $4 : JONCTION$*.*

gnement structure-séquence complexifie l'élaboration des équations de récurrence. La première étape de cette élaboration est de définir un système de structures valides, qui dans les équations de récurrence est représenté par un jeu d'indices (i et j dans l'algorithme précédent). L'étape qui suit (la formalisation des équations) est ensuite un traitement au cas par cas des structures valides en fonction des différentes possibilités ((i, j) en interaction ou non, sinon j en interaction ...). Enfin, et après s'être assuré de la correction des formules de récurrence, il faut éliminer tous les cas qui n'ont pas besoin d'être calculés pendant la récurrence pour gagner encore en complexité.

Bien qu'il soit encore difficile de comprendre tous les tenants et aboutissants de cette dernière étape, elle est en fait au cœur de tous les algorithmes que nous allons décrire ci-dessous. La prise en compte de structures de plus en complexes va paradoxalement clarifier cette constatation. En effet la trop grande variété des différents cas possibles pour les structures valides que nous allons manipuler nécessite

une généralisation du concept stipulant que des cas n'ont pas besoin d'être calculés pendant la récurrence.

3.3.3 L'algorithme de Han

En 2009, Han *et al.* [29] proposèrent un nouvel algorithme polynomial d'alignement structure-séquence pour une sous classe de CROSSING. Ainsi, ils définirent la classe des pseudo-nœuds standard (SPK) que nous allons maintenant détailler.

La motivation de cette nouvelle classe, outre le fait que l'alignement structure-séquence soit polynomial pour ces structures, est qu'une grande majorité des pseudo-nœuds connus dans les structures réelles y appartiennent. C'est le cas par exemple des pseudo-nœuds de type H ou encore des kissing-hairpins. C'est d'ailleurs une généralisation de la classe de structures très connue définie par AKUSTU [2], elle aussi ayant étant été construite avec les mêmes motivations (mais pour la prédiction de structures *ab initio*).

Informellement, une structure appartient à la classe SPK si elle est constituée d'une succession d'hélices se chevauchant alternativement. De plus ce chevauchement est fortement contraint. Ainsi, en appelant pied gauche d'une hélice, les nucléotides de l'hélice se trouvant en 5', et pied droit d'une hélice, les nucléotides de l'hélice en 3', pour deux hélices successives H_1 et H_2, le pied droit de H_1 doit chevaucher uniquement le pied gauche de H_2 (Figure 3.9).

Formellement, on définit la classe SPK comme suit :

Définition 6. *(SPK)*
Une séquence arc-annotée $A = (S_A, P_A)$ appartient à la classe SPK si est seulement si il existe une suite croissante de positions (appelés pivots) : $1 = y_0 < y_1 < \ldots < y_k$ telle que : les sous ensembles de P_A, $P_t = \{(i,j),\ i \in S[y_{t-1}, y_t - 1]\ et\ j \in S[y_t, y_{t+1} - 1]\}$, $t \in [1, k-1]$ et $P_k = \{(i,j),\ i \in S[y_{k-1}, y_k - 1]\ et\ j \in S[y_k, n]\}$:
- les sous ensembles de P_A, $P_t = \{(i,j),\ i \in S[y_{t-1}, y_t - 1]\ et\ j \in S[y_t, y_{t+1} - 1]\}$, $t \in [1, k-1]$ et $P_k = \{(i,j),\ i \in S[y_{k-1}, y_k - 1]\ et\ j \in S[y_k, n]\}$, forme une partition P_A.
- Les structures A_t induit par P_t, $t \in [1, k-1]$, appartiennent toutes à la classe CHAIN.

Le **degré d'un pseudo-nœud standard** est égal au nombre de ses pivots.

Du fait que la classe SPK autorise certains croisements, il n'est donc plus envisageable de pouvoir utiliser directement l'algorithme de Jiang. Nous devons donc trouver un nouveau type de sous-structures à utiliser pour une approche de programmation dynamique. Théoriquement, l'algorithme décrit par Han est applicable pour un degré quelconque de pseudo-nœud standard. Néanmoins, étant donné que le nombre d'équations de récurrence qu'il met en jeu est proportionnel au degré, il est difficile de le décrire pour un degré suppérieur à 3 ou 4. Ainsi, comme dans

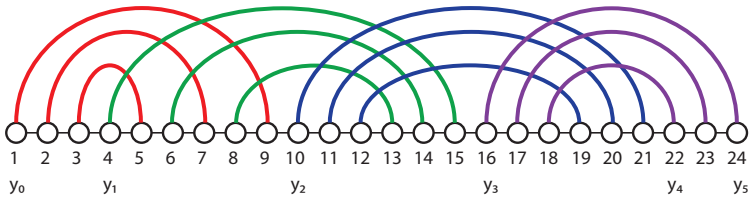


FIGURE 3.9 – *Pseudo-nœud standard de degré* 5. *Chaque jeu d'arcs de couleur représente une hélice.*

l'article [29], je prendrai le cas particulier d'un pseudo-nœud standard de degré 3 dans la suite.

Nous pouvons alors redéfinir spécifiquement un pseudo-nœud standard de degré 3, appelé pseudo-nœud simple (Figure 3.10) :

Définition 7. *(pseudo-nœud simple)*
Une séquence arc-annotée $A = (S_A, P_A)$ est un **pseudo-nœud simple** si est seulement si il existe deux positions y_1 et y_2 telles que :
- les sous-ensembles de P_A, $P_L = P_A[1, y_2]$, et $P_R = P_A[y_1, n]$ forment une partition P_A.
- A_L induit par P_L et A_R induit par P_R appartiennent à la classe CHAIN.

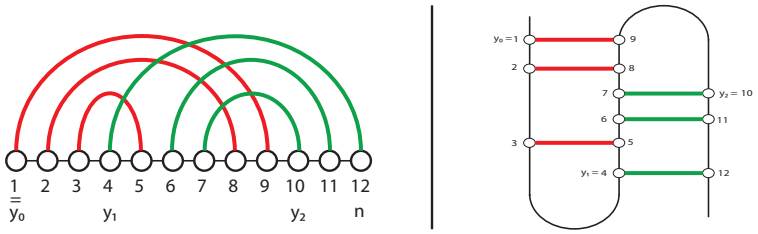


FIGURE 3.10 – *Pseudo-nœud simple. À gauche sa représentation arc-annotée, à droite une autre représentation de la même structure faisant apparaitre un ordre sur les interactions, représenté par leur hauteur sur le dessin.*

À partir de cette définition, Han introduit la notion de frontière. Une frontière est un triplet (i, j, k) de positions de A définissant la sous-structure induite par l'union des deux sous séquences $S_A[1, i]$ et $S_A[j, k]$ (Figure 3.11). Cette frontière définit un jeu d'indices qui va permettre de former des structures valides, utilisable pour résoudre le problème d'alignement structure-séquence pour les structures de la classe SPK.

Soit **A(i,j,k)** la sous structure induite par $S_A[1, i] \cup S_A[j, k]$. Comme précédemment, nous devons définir la notion de structures valides. Nous dirons que

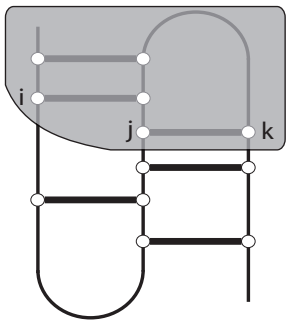


FIGURE 3.11 – *Sous-structure d'un pseudo-nœud simple défini par une frontière i, j, k (zone grisée).*

$A(i,j,k)$ est valide si et seulement si pour toute interaction $(i_1, i_2) \in P_A$ telle que $i_1 \in [1, i[\cup]j, k[$ (*i.e.*i_1 appartient à $A(i,j,k)$), alors $i_2 \in [1,i]\cup[j,k]$ (*i.e.* i_2 appartient aussi à $A(i,j,k)$).

Soit $\mathsf{Coût}(i,j,k,i',j',k')$ le coût du meilleur alignement entre la sous-structure $A(i,j,k)$ et la séquence $S_B[1,i'] \cup S_B[j',k']$. Les valeurs de $\mathsf{Coût}(i,j,k,i',j',k')$ peuvent alors être calculées récursivement si $A(i,j,k)$ est valide.

En effet, prenons le cas spécial où $(i,j) \in P_L$ et où de plus $A(i,j,k)$ est valide (le cas où $(j,k) \in P_R$ se gère symétriquement) et cherchons à calculer $\mathsf{Coût}(i,j,k,i',j',k')$. Nous devons alors considérer quatre sous-cas. Le cas $(MATCH)$ où on aligne (i,j) avec (i',j'), les cas $(ALTERATION)$ où on aligne seulement une des deux positions i ou j avec la position correspondante parmi i' et j', le cas $(INSERTION)$ où on n'aligne ni i ni j, et enfin le cas où l'on fait une suppression dans la structure $(SUPPRESSION)$. Nous pouvons alors montrer que le coût se calcule par la formule :

$$\mathsf{Coût}(i,j,k,i',j',k') = \min\{MATCH, INSERT, SUPPRESSION\}$$

$$MATCH = \mathsf{Coût}(i-1,j+1,k,i'-1,j'+1,k') + \mathsf{ArcSub}(i,j,i',j')$$

$$ALTERATION = \min \begin{cases} \mathsf{Coût}(i-1,j+1,k,i'-1,j',k') + \mathsf{ArcAltL}(i,j,i') \\ \mathsf{Coût}(i-1,j+1,k,i',j'+1,k') + \mathsf{ArcAltR}(i,j,i') \end{cases}$$

$$INSERTION = \mathsf{Coût}(i-1,j+1,k,i',j',k') + \mathsf{ArcSup}(i,j)$$

$$SUPPRESSION = \min \begin{cases} \mathsf{Coût}(i,j,k,i'-1,j',k') + \mathsf{Gap} \\ \mathsf{Coût}(i,j,k,i',j'+1,k') + \mathsf{Gap} \\ \mathsf{Coût}(i,j,k,i',j',k'-1) + \mathsf{Gap} \end{cases}$$

Les appels de récurrence s'effectueront eux aussi sur des structures valides et le coût du meilleur alignement entre A et S_B est alors donné par :

$$\min_{1 \le i' \le m} \mathsf{Coût}(1, y_2 - 1, y_2, i', i'+1, m)$$

Un algorithme naïf devra considérer $O(n^3 \cdot m^3)$ $\mathsf{Coût}(i, j, k, i', j', k')$ différents afin de calculer le coût optimal. Toutefois, nous remarquons que comme pour l'algorithme de Jiang, l'appel récursif du calcul du coût s'effectue sur un nombre limité de combinaison de sous-structures.

Ainsi $\mathsf{Coût}(i, j, k, i', j', k')$ se calcule uniquement en considérant les frontières $(i-1, j+1, k)$ $(MATCH, ALTERATION, INSERTION)$ et (i, j, k) $(DELETION)$. Nous pouvons alors en déduire qu'il n'est pas nécessaire de calculer $\mathsf{Coût}(i, j, k, i', j', k')$ pour toutes les frontières (i, j, k).

L'idée ingénieuse développée dans [29] est d'ordonner les interactions de la structure afin d'ordonner partiellement les sous-structures valides. Si A est un pseudo-nœud simple, si $(i, j) \in P_A$ alors nécessairement, soit $i \in [y_1, y_2[$ soit $j \in [y_1, y_2[$. De ce constat, nous pouvons donc directement établir un ordre (total) entre les interactions. On pose $(i, j)_{[y_1,y_2[} = i$ si $i \in [y_1, y_2[$ et $(i, j)_{[y_1,y_2[} = j$ sinon. Par suite, pour tout couple (i, j), $(i', j') \in P_A$, on pose $(i, j) \prec (i', j') \Leftrightarrow (i, j)_{[y_1,y_2[} \prec (i', j')_{[y_1,y_2[}$ (Figure 3.10). La relation $\prec$ définit un ordre total sur P_A. Cet ordre est ensuite utilisé pour guider la récurrence car il définit un ordre partiel sur les sous-structures valides.

Ainsi, pour une frontière donnée (i, j, k), il est possible de définir une fonction ζ afin de déterminer quelle frontière va être utilisée pour calculer l'ensemble des $\mathsf{Coût}(i, j, k, i', j', k')$:

$$\zeta(i,j,k) = \begin{cases} (i-1, j+1, k) & \text{si } (i,j) \in P_A \\ (i, j+1, k-1) & \text{si } (j,k) \in P_A \\ (i-1, j, k) & \text{si } i \text{ est libre et } 1 \leq i < y_1 \\ (i, j+1, k) & \text{si } j \text{ est libre et } y_1 \leq j < y_2 \\ (i, j, k-1) & \text{si } k \text{ est libre et } y_2 \leq j \leq n \\ \text{frontière vide} & \text{sinon} \end{cases}$$

Partant de la frontière $(1, y_2 - 1, y_2)$, et par appels récursifs de la fonction ζ, nous définissons ainsi l'ensemble des sous-structures valides que nous devons considérer pour effectuer l'alignement de la structure toute entière. Nous pouvons remarquer que nous ne considérons alors que les structures $A(i, j, k)$ avec $1 \leq i < y_1 \leq j < y_2 \leq k \leq n$. Mais surtout, nous ne considérons que $O(n)$ structures, car à chaque appels récursif de ζ nous modifions la frontière que seulement s'il est possible de remplacer au moins une des positions (de la frontière) par une autre position qui n'a jamais considérée.

Ainsi, la complexité en temps de l'algorithme de Han est de $O(n \cdot m^3)$. La complexité en espace de l'algorithme de Han est de $O(n \cdot m^3)$ car nous devons conserver les coûts des alignements optimaux des sous-structures valides que nous considérons (et ceci pour chaque sous-séquence de S_B possible). En généralisant le résultat, nous pouvons montrer que la complexité en temps et en espace de l'alignement structure-séquence d'un pseudo-nœud standard de degré k est de $O(n \cdot m^k)$.

3.3.4 Les algorithmes de Wong

L'algorithme de Han que nous venons de décrire pose les bases d'une approche algorithmique qui va être ensuite reprise par Wong *et al.*. En 2011 et 2012, ces auteurs élaborent un nouvel algorithme d'alignement structure-séquence d'ARN pour une sous classe de CROSSING et un autre pour une sous classe de UNLIMITED. Ces sous classes sont appelées respectivement **pseudo-nœuds simples non-standard** (SNP) et **triples hélices standard** (STH).

Alignement de pseudo-nœuds simples non-standard

La classe SNP est une sorte d'extension de la classe des pseudo-nœuds standard. Les structures associés à cette classe sont des pseudo-nœuds standard auxquels on ajoute une hélice supplémentaire spéciale dont un des pieds se trouve à la fin ou au début de la structure, l'autre se trouvant entre deux pivots du pseudo-nœud standard.

Ce genre de motif se retrouve très souvent dans les ribosomes par exemple. Ces structures ont (comme les pseudo-nœuds standard) un degré associé, et comme pour l'algorithme de Han, il est difficile de décrire l'algorithme pour un degré quelconque. Pour expliquer (plus rapidement cette fois-ci) l'algorithme nous allons décrire le cas particulier d'un pseudo-nœuds simple non-standard de degré 4 (Figure 3.12). Il y a deux types de tels pseudo-nœuds, voici la définition du type I.

Définition 8. *(pseudo-nœuds simple non-standard de degré 4)*
Une séquence arc-annotée $A = (S_A, P_A)$ est un pseudo-nœud simple non-standard de degré 4 (Type I) si est seulement si il existe trois positions : y_1, y_2, y_3 et un indice $\tau \in \{1, 2\}$ tels que :
- Les sous-ensembles de P_A, $P_t = \{(i, j),\ i \in S[y_{t-1}, y_t - 1]$ *et* $j \in S[y_t, y_{t+1} - 1]\}$, $t \in \{1, 2\}$ et $P' = \{(i, j),\ i \in S[y_{\tau-1}, y_\tau - 1]$ *et* $j \in S[y_3, n]\}$, forment une partition de P_A.
- Les sous-ensembles de P_A, $P_1 = \{(i, j),\ i \in S[y_0, y_1 - 1]$ *et* $j \in S[y_1, y_2 - 1]\}$, $P_2 = \{(i, j),\ i \in S[y_1, y_2 - 1]$ *et* $j \in S[y_2, y_3 - 1]\}$ et $P_\tau = \{(i, j),\ i \in S[y_{\tau-1}, y_\tau - 1]$ *et* $j \in S[y_3, n]\}$, forment une partition de P_A.
- Les structures A_t induit par P_t, $t \in \{1, 2\}$ et P', appartiennent toutes à la classe CHAIN.

Nous pouvons définir de la même façon un pseudo-nœud simple non-standard de degré 4 (Type II), en plaçant la sous-structure spéciale A_τ en début de séquence (Figure 3.12). Dans la suite, nous allons uniquement considérer les pseudo-nœuds simple non-standard de Type I. En plus de la distinction par leur Type, l'algorithme donné par Wong *et al.* traite différemment les structures selon que l'indice τ est égal à 1 ou à 2.

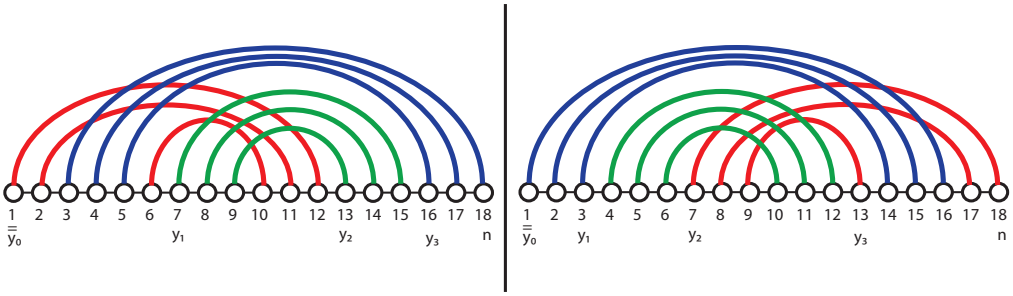


FIGURE 3.12 – *Pseudo-nœud simple non-standard Type I (à gauche) et Type II (à droite). Dans les deux cas, l'hélice supplémentaire spéciale apparait en bleu.*

Si $\tau = 1$, les sous-structures valides que nous allons définir ici vont être induites de l'union de deux sous séquences (comme pour l'algorithme précédent) mais n'ayant aucune extrémité commune avec la séquence de la structure toute entière (contrairement à l'algorithme précédent). Nous allons donc manipuler un jeu de quatre indices (deux par sous-séquence) pour définir nos sous-structures.

Soit $A(i, j, k, l)$ la sous-structure de A induite par $S_A[i, j] \cup S_A[k, l]$. Comme Han *et al.* l'ont fait avant eux, Wong *et al.* définissent un ordre partiel entre les sous-structures $A(i, j, k, l)$ et l'utilisent pour établir leur formule de récurrence. Celle est composé de plus d'une vingtaine de sous-cas (en fonction du type de sous-structure valide, par exemple : si $(i, j) \in P_A$ ou si $(i, l) \in P_A$ ...)

.

Si τ est pair, il faut alors considérer l'union de trois sous-séquences rendant le problème encore plus difficile encore (les structures valides se construise à partir de 5 positions) et même si la démarche est exactement la même, la mise en équation (et cela même pour un degré de 4 seulement).

La complexité s'analyse de la même façon que pour l'algorithme précédent (les approches étant presque identiques), d'où une complexité en temps et en espace de $O(n \cdot m^{k+1})$ pour l'alignement structure-séquence d'un pseudo-nœud simple non-standard de degré k.

Alignement des triples hélices standard

L'alignement des triples hélices est à ma connaissance (ainsi que celle des auteurs de l'article) la première tentative de prise en compte des triples hélices et plus généralement de prise en compte d'interactions multiples. Les triples hélices sont des éléments de structures que nous pouvons observer dans des ARN tels que les télomérases ou dans les pseudo-nœuds des ARN viraux impliqués dans le glissement de cadre de lecture des ribosomes. Pour ces ARN, casser certaines interactions non canoniques ou multiples détériorent l'activité de la molécule, montrant ainsi que ces interactions participent à la fonction de l'ARN. Dans [63], les auteurs définissent la classe de structures appelée triples hélices standard (STH), qui représente une

version simplifiée des triples hélices. Cette simplification est certainement issue d'un compromis entre une représentation exacte des triples hélices et une représentation permettant une manipulation aisée au sein d'un algorithme d'alignement. La définition de la classe des triples hélices standard tente ainsi de capturer des informations supplémentaires que ne capturer pas un pseudo-nœud simple. En effet, nous pouvons interpréter la classe STH comme une sur-couche des pseudo-nœuds simple auxquels nous autorisons certaines (et donc pas toutes) interactions multiples (Figure 3.13).

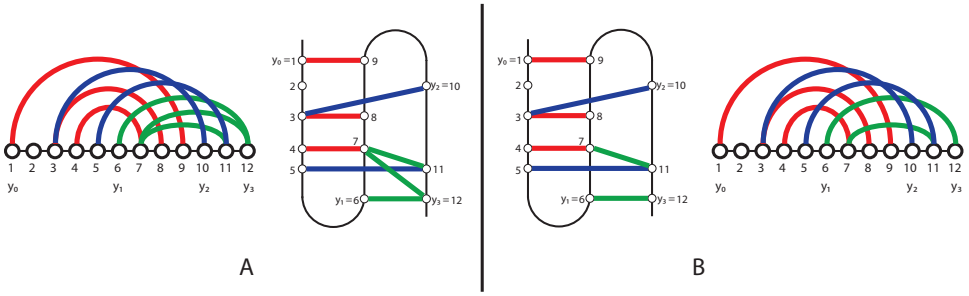


FIGURE 3.13 – *À gauche : une triple hélice standard, à droite : une structure que n'est pas une triple hélice standard.*

Nous nous contenterons de cette description succincte, l'énoncé formel de la définition de cette classe étant un peu fastidieux et peu intéressant pour le développement du reste de l'argumentation. Ce que nous retiendrons de l'algorithme donné par Wong *et al.* sur l'alignement triples hélices standard - séquence est que la formule de récurrence employée ici est composée d'un nombre encore plus impressionnant de cas particuliers différents à prendre en compte que pour l'algorithme précédent. Ainsi, il faut prendre en compte plus d'une trentaine de cas afin de garantir de couvrir l'ensemble des alignements possibles dans le calcul du coût de l'alignement optimal s'effectuant en $O(n \cdot m^3)$.

3.3.5 Combinaison des classes de structures

Pour former véritablement une structure secondaire d'ARN avec pseudo-nœuds ou secondaire étendue grâce aux classes SPK , SNP ou STH, nous devons presque nécessairement les associer avec la classe NESTED ou même entre elles. S'il est en théorie possible de modifier un algorithme d'alignement structure-séquence d'une classe pour le rendre compatible avec certaines combinaisons avec une autre classe, l'écrite concrète de la modification sera de plus en plus délicate au fur-et-à-mesure que nous voudrons ajouter des classes. Ainsi, que ce soit dans [29] ou dans [64], les auteurs décrivent (équations l'appui) respectivement qu'un cas particulier de combinaisons (et à chaque fois avec la classe NESTED). Cette limitation rend encore plus spécifique l'utilisation de ces approches.

Conclusion

S'il était encore raisonnable de traiter au cas par cas les différentes classes de structures (NESTED,SPK), nous voyons bien qu'il est plus que nécessaire d'unifier ces différentes approches. Plus que l'unification, c'est la généralisation du problème qui est en jeu, car comme nous allons le voir, faire l'effort de construire une approche unifiant les précédentes va nous donner accès à une généralisation du problème.

Chapitre 4

Unification et généralisation

Sommaire

Traiter l'alignement structure-séquence au cas par cas est viable lorsqu'on s'intéresse uniquement à un nombre très restreint de structures, en revanche ce type d'approches est voué à l'échec lorsque la variété des structures intéressantes est grande. C'est particulièrement vrai pour les structures secondaires étendues. En effet les motifs tertiaires sont variés et ce au sein même d'un motif. C'est aussi vrai (mais dans une moindre mesure tant les pseudo-nœuds de type H sont abondants) pour les pseudo-nœuds. Par exemple, les *pseudo-trefoils*, que l'on peut trouver dans plusieurs bases de données de structures d'ARN [59, 58, 4], ne sont gérés par aucun algorithme d'alignement.

Voici pourquoi je pense qu'il est indispensable de s'affranchir d'une définition de classe de structures particulière afin de résoudre le problème dans sa globalité.

Dans ce chapitre nous allons aborder ce qui constitue le cœur de ma thèse. Nous allons construire un algorithme capable d'aligner n'importe quelle structure d'ARN avec une séquence. Nous avons vu précédemment que le problème était NP-Difficile mais qu'il existait des classes de structures pour lesquelles nous savons effectuer l'alignement en temps polynomial. Beaucoup de problèmes NP-difficile peuvent être résolus par des algorithmes à complexité **paramétrée** [18, 6]. La complexité de ces algorithmes a une forme polynomiale, par exemple du type $O(f(k) \cdot p(n))$ où p représente un polynôme fonction n représentant la taille des données et où la fonction f peut prendre n'importe qu'elle forme possible (par exemple, $f(k) = 2^k$). Même si au pire la complexité reste exponentielle (par exemple si $k = n$ dans l'exemple précédent), dans la pratique le paramètre doit s'adapter aux instances fournis afin d'adapter en conséquence la complexité de l'algorithme.

La complexité des approches du chapitre précédent semble être corrélée à une forme de complexité structurale des ARN. Ne pourrait on pas alors envisager une approche à complexité paramétrée ? Dans ce qui suit, je vais exposer comment paramétrer efficacement l'alignement structure-séquence pour l'ensemble complet des structures d'ARN.

Le point de départ de ce travail est sans aucun doute l'algorithme de Han [29] et plus particulièrement la fonction ζ. Celle-ci définit une façon de décomposer un pseudo-nœud standard ; décomposition que nous allons retravailler pour ensuite pour établir les formules de récurrence. C'est d'ailleurs cette partie de la méthode qui a été reprise ensuite par Wong *et al.* [64, 63], le reste du travail étant surtout de l'analyse au cas par cas.

Ce qui peut choquer à la lecture de ces papiers est que la fonction ζ semble être dédiée uniquement à amélioration de la complexité. Elle y est décrite en tant que telle et toujours après la formalisation du système de structures valides qui va être employé dans la récurrence. Or si les récurrences qu'ils définissent peuvent résoudre le problème c'est dû uniquement à l'existence d'une telle fonction ζ.

Concrètement, cette fonction représente une possibilité de définir un ordre total

sur les positions de la structure. Cet ordre qui doit tenir compte des interactions peut être mis en parallèle avec la notion de triangulation de graphe. Pour tester si un graphe est cordal (triangulé) il suffit de rechercher si celui-ci possède un ordonnancement d'élimination parfaite de ses sommets. Or, l'ordre total représenté par ζ est en réalité un ordonnancement d'élimination parfaite d'un sur graphe cordal de la structure. Cette constatation doit nous amener à la question suivante : existe t-il une approche visant à décomposer un graphe (une structure) basée ou en relation avec la notion de graphes cordals et d'ordonnancement d'élimination parfaite ? Une réponse à cette question nous guide dans les sections suivantes vers **la décomposition arborescente**.

C'est grâce à l'utilisation de décompositions arborescentes que nous allons généraliser la résolution de l'alignement structure-séquence. La complexité de l'algorithme que nous allons établir va être paramétré par une caractéristique calculer à partir des décompositions arborescentes que nous allons créer.

Après avoir défini la décomposition arborescente, j'expliquerai comment l'utiliser dans le cadre de l'alignement structure-séquence d'ARN. Nous nous intéresserons ensuite à la construction de décompositions arborescentes efficaces pour la résolution de notre problème. Dans tous les cas, la méthode décrite égale ou améliore les résultats de complexités précédents et permet en plus de traiter n'importe quelle autre classe de structure.

4.1 Décomposition arborescente

Dans cette section nous allons définir un moyen de décomposer les structures d'ARN sur lequel nous baserons nos équations de récurrence. La décomposition arborescente répond parfaitement à cette problématique. À l'origine, cette décomposition s'effectue sur des graphes et non sur des séquences arc-annotées, mais étant donné la très grande proximité de ces deux objets, nous pouvons très simplement effectuer une adaptation.

Les décompositions arborescentes ont été introduites pour la première fois par Halin [28], mais c'est véritablement à Seymour et Robertson que l'on doit les plus gros travaux fédérateurs sur le sujet [52]. Depuis, bien d'autres travaux ont été réalisés [6, 11], utilisant les décompositions arborescentes pour paramétrer des problèmes NP-difficile classiques des graphes comme par exemple la recherche du stable maximum. Elles ont même déjà été utilisées en bio-informatique, et pour des problèmes proches de l'alignement structure-séquence [65, 56].

Informellement, une décomposition arborescente d'une séquence arc-annotée $A = (S_A, P_A)$ est un arbre dont les sommets sont des sous-ensembles des positions de A que l'on appelle **sacs** (Figure 4.1). De plus, l'arbre doit vérifier certaines (quatre en tout) propriétés. Voici la définition formelle :

Définition 9. *(Décomposition arborescente)*
Étant donné une séquence arc-annotée $A = (S_A, P_A)$ de longueur n, une **décomposition arborescente** de A est une paire (X, T) où :
- $X = \{X_1, \ldots, X_N\}$ est l'ensemble des **sacs**, où un sac est un sous-ensemble de positions $\{i, i \in [1, n]\}$ de A.
- T est un arbre dont les sommets sont des éléments de X et qui vérifie les conditions suivantes :

1. Toute position appartient à un sac :
   $\bigcup_{l \in [1,N]} X_l = [1, n]$.
2. Chaque interaction est présente dans un sac :
   $\forall (i, j) \in P, \exists l \in [1, N], \{i, j\} \subset X_l$.
3. Chaque paire de positions consécutives est présente dans un sac :
   $\forall i \in [1, n-1], \exists l \in [1, N], \{i, i+1\} \subset X_l$.
4. Pour toute paire de sacs X_l et X_s, $l, s \in [1, N]$, $X_l \cap X_s \subset X_r$ pour tout sac X_r sur le chemin entre X_l et X_s.

Une caractéristique très importante des décompositions arborescentes est leur **largeur**. Comme nous allons le voir, c'est cette caractéristique qui va paramétrer le problème d'alignement structure-séquence et sur laquelle nous allons ultérieurement concentrer nos efforts.

Définition 10. *(Largeur arborescente)*
La **largeur** d'une décomposition arborescente est la taille de son plus grand sac moins un, c'est-à-dire avec les notations précédentes : $Largeur((T, X)) = \max\{|X_l| - 1, l \in [1, N]\}$.
La **largeur arborescente** $t_w(A)$ d'une séquence arc-annotée A est la plus petite largeur parmi toutes les décompositions arborescentes possibles de A.

Une décomposition arborescente d'une séquence arc-annotée est illustrée par la Figure 4.1. En général, les décompositions arborescentes ne sont pas enracinées. Néanmoins dans un souci de clarté de la description des algorithmes, nous enracinerons systématiquement les décompositions et désignerons par X_0 la racine.

Il a été montré que toute décomposition arborescente peut être transformée (en temps linéaire) en une décomposition arborescente binaire (tout sac a au plus deux fils) [11]. D'autre part, nous interdisons qu'une décomposition possède un sac X_l qui soit totalement inclus dans son père X_r (sinon il suffit de supprimer le sac X_l et ses les fils deviennent les fils de X_r). Toutes les décompositions arborescentes que nous utiliserons et construirons respecterons ces deux propriétés.

Calculer la largeur arborescente d'une séquence arc-annotée (et donc trouver sa meilleure décomposition) est un problème NP-difficile [5]. Il existe des algorithmes

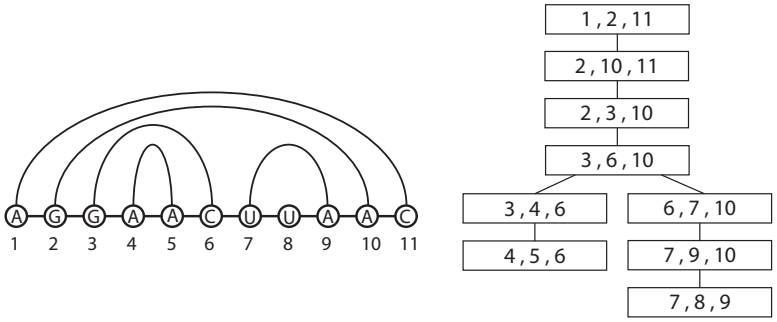


FIGURE 4.1 – *Une séquence arc-annotée et une décomposition arborescente associée. Chaque sac (figure de droite) contient trois positions, donc la largeur de la décomposition arborescente est de* 2.

théoriques de décompositions arborescentes, qui étant donné un graphe de largeur arborescente au plus t, construisent une décomposition arborescente optimale en $O(c^t \cdot n)$ [53]. Cependant, la constante c est extrêmement trop élevée pour pouvoir les utiliser en pratique. Pour cela, l'approche dominante, voir l'unique approche lorsque les graphes sont de grandes tailles, est d'utiliser des heuristiques ne garantissant pas une décomposition optimale. Or la puissance de l'approche que je vais présenter dépend entièrement de la capacité à savoir décomposer des structures d'ARN en décompositions arborescentes de faibles largeurs. Nous verrons dans les prochaines sections comment répondre à ce problème. Pour l'instant nous allons focaliser nos efforts sur la façon d'utiliser les décompositions arborescentes pour résoudre le problème d'alignement structure-séquence.

4.2 Aligner avec les décompositions arborescentes

L'approche que je vais maintenant présenter est basée (comme les algorithmes précédents) sur un algorithme de programmation dynamique. Et comme nous l'avons vu dans le chapitre précédent avec les autres approches, notre algorithme va résoudre le problème de façon récursive en résolvant le problème sur des sous-structures de la structure totale. Contrairement aux approches précédentes, nous n'allons pas définir directement et manuellement le système de sous-structures à utiliser. Cette étape sera entièrement déterminée lors de construction de la décomposition arborescente que nous verrons dans les prochaines sections.

Une fois que nous obtenue la décomposition arborescente, nous ne chercherons plus à aligner la structure avec une séquence, mais à aligner la décomposition arborescente avec une séquence. Ainsi, le caractère arborescent de la décomposition va

guider l'algorithme en nous offrant un schéma de programmation dynamique.

Pour définir cela concrètement nous allons avoir besoin de quelques notions supplémentaires. Dans toute la suite de la section, nous allons considérer déjà construite une décomposition arborescente $D = (X, T)$ possédant N sacs d'une séquence arc-annotée quelconque $A = (S_A, P_A)$ de longueur n que nous voulons aligner avec une séquence S_B de longueur m.

4.2.1 Définitions et notations

Nous définissons les sous-structures que nous utilisons à partir de la décomposition arborescente. Comme dans le chapitre précédent, les sous-structures vont être induites par un jeu d'indices. Ces indices nous sont donnés par les sacs de la décomposition. Ainsi, pour chaque sac, nous définissons une sous-structure.

Définition 11. *(Sous-séquence descendante)*
La **sous-séquence descendante** d'un sac $X_l \in X$ d'une décomposition arborescente D est la sous-séquence formée par l'ensemble des positions apparaissant dans la descendance de X_l dans D (positions de X_l comprises).

Soit X_l le sac de la 4.1 contenant les positions $3, 6, 10$ (celui possédant deux fils). Alors la sous-séquence descendante de X_l est la séquence contenant les positions $3, 4, 5, 6, 7, 8, 9, 10$.

Définition 12. *(Sous-structure descendante)*
La **Sous-structure descendante** d'un sac $X_l \in X$ d'une décomposition arborescente D, notée $\mathcal{D}(X_l)$, est la sous-structure (sous-séquence arc-annotée) induite par la sous-séquence descendante de X_l.

En reprenant le même sac X_l que pour l'illustration précédente, la sous-structure descendante de X_l contient alors les interactions $(3, 6), (4, 5), (7, 9)$. Remarquons que la sous-structure descendante de la racine est égale à la structure toute entière.

Définition 13. *(Sous-Ensemble des alignements)*
Soit X' un sous-ensemble de positions de A. On note $\boldsymbol{\mathcal{F}|_{X'}}$ l'ensemble des alignements de la sous-structure induite par X' avec la séquence S_B.

Pour deux sacs X_l et X_r, on note $\boldsymbol{X_{l,r} = X_l \cap X_r}$.

Définition 14. *(Compatibilité)*
Soit X' et X'' deux sous-ensemble de positions de A. Les alignements $\mu' \in \mathcal{F}|_{X'}$ et $\mu'' \in \mathcal{F}|_{X''}$ sont **compatibles** si et seulement si μ' et μ'' vérifient les propriétés suivantes :
– $\forall i \in X' \cap X''$, $\mu'(i) = \mu''(i)$.
– $\forall i \in X' - X''$, $\forall j \in X'' - X'$, $i < j \Leftrightarrow \mu'(i) < \mu''(j)$.

Enfin, nous utilisons la notation usuelle $\mu_{|X'}$ pour désigner la restriction de μ à X'.

4.2.2 Équation de récurrence

Quelle que soit la décomposition arborescente $D = (X, T)$ dont nous disposons, il est possible d'utiliser systématiquement une unique équation de récurrence pour calculer le coût de l'alignement optimal entre A et S_B. Cette équation de récurrence calcule le coût des différents alignements de toutes les structures descendantes des sacs de la décomposition arborescente. Nous allons maintenant la construire étape par étape.

En premier lieu nous allons écrire une première esquisse de la récurrence basée sur des techniques générales de programmation dynamique utilisant des décompositions arborescentes [11, 56] et pour laquelle nous n'autoriserons pas les suppressions (tous les nucléotides de la structure sont alignés). Nous améliorerons ensuite cette approche en l'adaptant à l'alignement (pas nécessairement structure-séquence d'ARN), sur les mêmes principes que les travaux de Xu *et al.* [65] concernant l'alignement de structures de protéines et en considérant les substitutions (d'arcs ou de bases, mais pas de suppressions). Enfin nous montrerons que pour pouvoir gérer les suppressions (c'est-à-dire comptabiliser leurs coûts), nous devrons enrichir la définition de l'alignement.

Nous oublions donc momentanément les suppressions en tous genres pour nous concentrer sur les substitutions. De fait, si μ est un alignement nous supposerons que toute position i possède un alignement $\mu(i) \neq \perp$ et les insertions (positions de la séquence sans antécédent par μ) ne seront pas comptabilisées dans le coût. Cette simplification permet alors d'utiliser les approches présentées dans [11] et plus particulièrement celle décrite dans [56] permettant une autre forme d'alignement, c'est-à-dire à plus gros grain, entre les éléments d'une structure secondaire (hélices pour l'ARN) d'ARN ou de protéines et une séquence.

Avant d'écrire l'équation de récurrence, nous allons en illustrer le concept par un exemple. Nous voulons aligner la structure de la Figure 4.2.A avec la séquence se situant juste en dessous (de la même figure). Pour cela, nous allons disposons d'une décomposition arborescente de la structure (Figure 4.2.A). Pour chaque sac

X_l de la décomposition, nous construisons une table contenant tous les alignements possibles du sac (Figure 4.2.B). Pour chacun des alignements *mu* du sac X_l, nous associons un coût, correspondant au coût de l'alignement optimal ω de la sous-structure descendante de X_l compatible avec μ (Figure 4.2.C). Pour calculer ce coût, nous calculons la contribution de μ, puis pour chaque fils $X_{l'}$ de X_l, nous recherchons dans sa table (d'alignements, construite précédemment de façon similaire) le meilleur alignement μ', c'est-à-dire celui associé à un coût minimum et compatible avec μ (Figure 4.2.D et 4.2.E). Ce coût, correspond par construction au coût de l'alignement optimal ω' de la sous-structure descendante de $X_{l'}$ compatible avec μ'. Une fois déterminé, nous ajoutons ces coûts (un pour chaque fils) à la contribution de μ pour calculer le coût associé à μ, c'est-à-dire le coût de l'alignement ω.

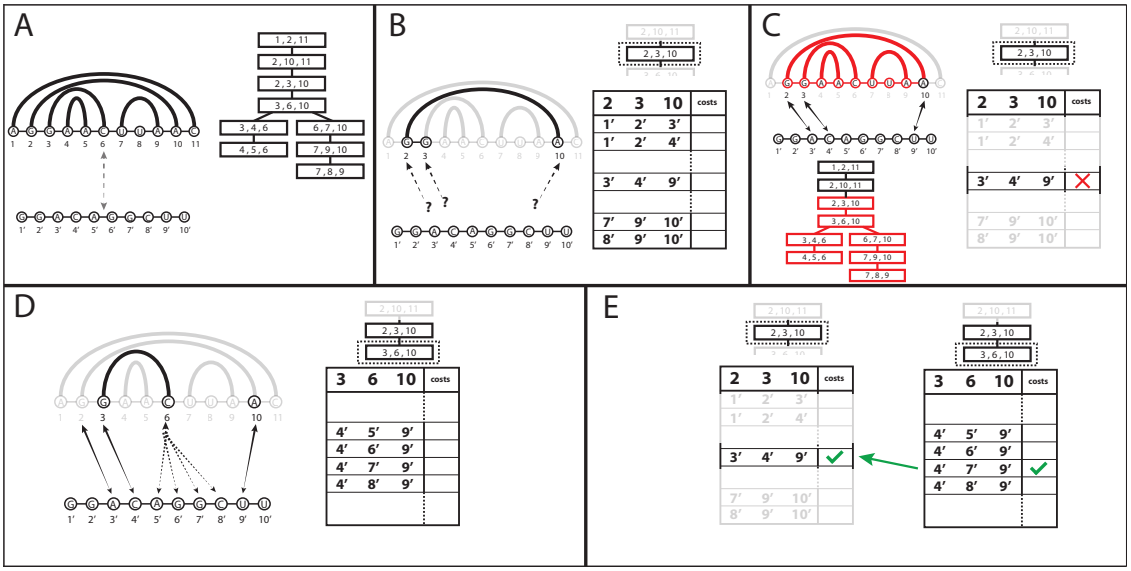


FIGURE 4.2 – *A : Une structure avec une de ses décomposition arborescente, avec laquelle nous voulons une séquence. B : Illustration de la table des alignements d'un sac de la décomposition. C : Sous-structure descendante (non-grisée, en rouge) du sac considéré. Le coût à calculer (indiqué par la croix) correspond au coût de l'alignement optimal de la structure non-grisée. D : Illustration de la table des alignements du fils du sac précédent. Les alignements représentés sont les alignements compatibles avec l'alignement précédent. E : Illustration de l'appel de récurrence (recherche du meilleur alignement compatible du fils).*

Le coût minimum dans la table des alignements de la racine correspond alors au coût de l'alignement optimal. En effet, par construction, l'alignement obtenu a un coût inférieur au égal à l'alignement optimum. De plus, nous pouvons prouver que l'alignement est correct, c'est-à-dire qu'il ne contient pas de conflits (deux positions de la structure alignées sur un même position de la séquence, deux alignements croisant...). Lors du calcul du coût de ω (en reprenant l'exemple précédent), nous nous assurerons de la compatibilité entre μ et μ'. Donc de fait, nous nous assurons

de la compatibilité entre μ et ω' pour les positions appartenant $X_l \cup X'_l$. Même si a priori rien n'assure la compatibilité entre μ et ω' (et par conséquent que ω n'ai pas de conflit) sur les autres positions de la descendance de X_l, les propriétés de la décomposition arborescente à prouver automatiquement que si μ et μ' sont compatibles alors μ et ω' le sont aussi (et par conséquent que ω n'a pas de conflit).

Cette preuve, que nous allons maintenant établir, nous permettra de mieux comprendre pourquoi il va être indispensable d'enrichir la définition d'un alignement, car une condition nécessaire à cette démonstration est que nous imposons un alignement pour chaque position ($\mu(i) \neq \perp$).

Lemme 1.
Soient X_l et $X_{l'}$ deux sacs d'une décomposition arborescente (X, D) d'une séquence arc-annotée S_A tels que $X_{l'}$ soit le fils de X_l et soit S_B une séquence. Soient μ un alignement de $\mathcal{F}|_{X_l}$, μ' un alignement de $\mathcal{F}|_{X_{l'}}$ compatible avec μ et ω' l'alignement optimal de $\mathcal{F}|_{\mathcal{D}(\mathcal{X}_{\updownarrow'})}$ compatible avec μ'. Alors le plongement ω de $\mathcal{F}|_{\mathcal{D}(\mathcal{X}_{\updownarrow})}$ égale à l'alignement μ sur X_l et à ω' sinon est un alignement.

Preuve: Lemme 1

Nous savons (par transitivité) que étant donné que ω est compatible avec μ, que μ est compatible avec μ' et que μ' est compatible avec ω' alors ω et ω' sont compatibles sur $X_{l,l'}$ pour tous fils $X_{l'}$ de X_l. Reste à montrer que les choix effectués par μ sur $X_l - \cup X_{l'}$ sont compatibles avec ω', c'est-à-dire montré que pour tout $i \in X_l - \cup X_{l'}$ et tout $j \in \mathcal{D}(X_l) - X_l$, si $i < j$ alors $\omega(i) < \omega(j)$ (et inversement).

Nous devons montrer que pour deux positions $i < j$ de $\mathcal{D}(X_l)$, alors $\omega(i) < \omega(j)$. Si i et j appartiennent ou n'appartiennent pas simultanément à $X_l \cup X_{l'}$ alors par construction de ω on a bien $\omega(i) < \omega(j)$. Soient $i < j$ deux positions de $\mathcal{D}(X_l)$ telles que seul i appartienne à $X_l \cup X_{l'}$ (le cas inverse se résout symétriquement). Comme j n'appartient pas à $X_{l'}$, il n'apparaît que dans (les sacs de) la descendance stricte de $X_{l'}$. Par conséquent $j - 1 \in \mathcal{D}(X_{l'})$ (autrement la paire $(j-1, j)$ n'apparaitrait pas dans la décomposition). Par construction de ω on a alors $\omega(j-1) < \omega(j)$ (car ω et égal à ω' sur $\mathcal{D}(X_{l'})$). Si $j - 1$ appartient à $X_{l'}$, alors $\omega(i) < \omega(j-1)$ et donc $\omega(i) < \omega(j)$. Sinon, par induction, en remplaçant j par $j - 1$ dans le raisonnement ci-dessus, et jusqu'à ce que $j-1$ appartienne à $X_{l'}$, nous arriverons à la conclusion : $\omega(i) < \omega(i+1) < \ldots < \omega(j-1) < \omega(j)$ prouverons ainsi le lemme. □

L'équation suivante permet alors de calculer le coût $\mathsf{Coût}(l, \mu)$ de l'alignement optimal ω de la structure descendante d'un sac donné X_l compatible avec un ali-

gnement donné de ce même sac $\mu \in \mathcal{F}|_{X_l}$:

$$\mathsf{Coût}(l,\mu) = \phi(X_l,\mu) + \sum_{l' \text{ fils de } l} \left(\min_{\substack{\mu' \in \mathcal{F}|_{X_{l'}} \\ \mu' \text{compatible avec } \mu}} \mathsf{Coût}(l',\mu') \right)$$

et où ϕ est la contribution local du coût de l'alignement, c'est à dire le coût de l'alignement μ.

Dans la mesure où une position ou une interaction peut apparaitre plusieurs fois dans la décomposition arborescente, il faut prendre garde à ne comptabiliser qu'une seule fois sa contribution dans les coût des alignements. Ainsi la fonction ϕ attribuera un coût à l'alignement d'une position (ou interaction) si et seulement si cette position (ou interaction) apparaît pour la première fois dans la décomposition arborescente (en partant de la racine). Nous avons alors l'assurance de ne pas compter plusieurs fois la contribution de l'alignement d'une position (ou interaction) dans le calcul du coût d'un alignement. Nous laissons en l'état la définition de ϕ mais nous la développerons en détail lorsque nous établirons l'équation de récurrence finale. Étant donné que nous nous assurons l'optimalité à chaque étape, et en considérant le Lemme 1, l'équation ci-dessus calcul bien le coût de l'alignement optimal.

Analysons rapidement la complexité d'un algorithme basé sur l'équation de récurrence précédente. Pour chacun des N sacs X_l, nous devons calculer et mémoriser autant de coûts que d'alignements possibles de X_l avec S_B, soit $O(m^{k+1})$ coûts où $k+1$ est la taille du sac. La complexité au pire en mémoire d'un tel algorithme est donc de $O(N \cdot m^{k+1})$ ou k est la largeur arborescente de la décomposition. Nous pouvons améliorer cette complexité en modifiant quelque peu l'approche comme présenté dans les travaux de Xu *et al.* [65] pour l'alignement de sous-structures de protéines (problème très proche de celui celui présenté dans [56]).

Lors du calcul de $\mathsf{Coût}(l,\mu)$, pour un l et un μ fixé, nous recherchons des coûts minimaux ($\mathsf{Coût}(l',\mu')$) pour chaque fils X_l' de X_l et pour tout alignement μ' compatible avec μ. À l'étape précédente des appels récursifs (c'est-à-dire lors du traitement des fils de X_l), il nous suffirait donc de retenir, pour chaque fils, uniquement le coût du meilleur alignement compatible avec μ. La compatibilité s'effectuant sur $X_{l,l'}$, nous n'avons donc besoin de ne retenir que $O(m^{|X_{l,l'}|})$ coûts minimaux. Nous gagnons alors au minimum un degré de puissance étant donné que $|X_{l,l'}| < |X_{l'}|$ pour $X_{l'}$ fils de X_l.

L'équation suivante permet alors de calculer le coût $\mathsf{Coût}(l,\mu)$ de l'alignement optimal ω de la structure descendante d'un sac donné X_l (fils de X_r) compatible avec un alignement partiel donné de ce même sac $\mu \in \mathcal{F}|_{X_l,r}$:

$$\mathsf{Coût}(l,\mu) = \min_{\substack{\mu' \in \mathcal{F}|_{X_l} \\ \mu' \text{compatible avec } \mu}} \left\{ \phi(X_l,\mu') + \sum_{s \text{ fils de } l} \mathsf{Coût}(s, \mu'|_{X_{s,l}}) \right\} \tag{4.1}$$

et où ϕ est la contribution local du coût de l'alignement.

Comme nous l'avions annoncé, cette équation ne permet pas de gérer les suppressions.

Lemme 2.
En considérant les suppressions, les plongements dont l'équation 4.1 calcule les coûts ne sont pas nécessairement des alignements.

Montrons le par un contre exemple.

Preuve: Lemme 2

Considérons la structure et la décomposition associée de la Figure 4.3.A et reprenons nos schémas de coûts dans lesquels nous autorisons les insertions et suppressions. Nous voulons calculer le coût de l'alignement optimal μ_0 de $\mathcal{D}(X_0)$ tel que $\mu_0(1) = 1'$, $\mu_0(2) = \perp$ et $\mu_0(6) = \perp$. Pour cela, nous cherchons parmi tous les alignements de $\mathcal{D}(X_1)$ compatibles, celui qui minimise le coût selon l'équation 4.1. Appelons le μ_1. La compatibilité impose alors que $\mu_0(2) = \mu_1(2) = \perp$ et $\mu_0(6) = \mu_1(6) = \perp$ et la minimisation se fait alors uniquement sur l'alignement de la position 5. Supposons que $\mu_1(5) = \perp$ (Figure 4.3.B). Par récurrence, pour calculer le coût de μ_1, nous devons chercher l'alignement de $\mathcal{D}(X_2)$ compatible qui minimise le coût. Appelons le μ_2. La compatibilité impose seulement $\mu_1(2) = \mu_2(2) = \perp$ et $\mu_1(5) = \mu_2(5) = \perp$ et l'alignement de la position 3 est laissé libre. Si nous supposons que $\mu_2(3) = 1'$ (Figure 4.3.B), alors bien que μ_0 soit compatible avec μ_1 et que lui-même soit compatible avec μ_2, μ_0 n'est pas compatible avec $\mu 2$. L'alignement ainsi formé serait donc incorrect (deux positions, 1 et 3, alignées sur la même autre position $1'$).

Ainsi nous montrons qu'en autorisant les suppressions, nous pouvons former des alignements incorrects. Et quand bien même l'alignement serait correct, par exemple $\mu_2(3) = 5'$ (Figure 4.3.C), nous ne pourrions pas prendre en compte le coût de l'insertion entre les positions 1 et 3 (insertion des positions $2'$ à $4'$). En effet, puisque nous ajoutons localement les coûts (c'est-à-dire, en quelque sorte les coûts associés à l'alignement d'un sac) et que les positions 1 et 3 ne se retrouvent jamais simultanément dans même sac, nous n'avons pas la possibilité de prendre en compte cette insertion dans le calcul du coût. □

Si dans le cas d'un alignement sans aucun gap la démarche précédente fonctionne, dès lors que nous voulons considérer des suppressions ou insertions la compatibilité entre les alignements peut être interrompue. Nous allons alors modifier la définition des alignements de façon à ce que la compatibilité se transmette de sac en sac, et ce même si nous n'alignons pas certaines positions.

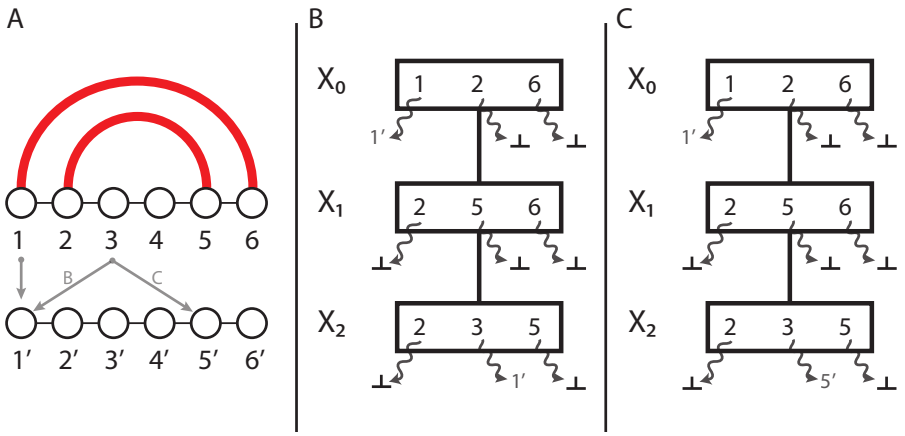


FIGURE 4.3 – *Structure et décomposition arborescente (à gauche) illustrant le problème lié à la formulation des alignements (à droite).*

4.2.3 Nouvelle définition et équation de récurrence

Le problème que nous venons d'évoquer est donc lié aux positions qui ne sont pas alignées. Ainsi, alors que les positions alignées peuvent transmettre de proche en proche la compatibilité, les positions non-alignées interrompent cette transmission. Voilà pourquoi nous devons définir un moyen d'effectuer un alignement même pour ces positions là.

Pour ce faire, à partir d'un alignement μ défini comme précédemment, toute position non-alignée i de A se verra alignée sur la même position i' de S_B que sa plus proche position voisine de droite alignée j ($\mu(i) = \mu(j) = i'$). Pour différencier les positions qui sont réellement alignées des autres, nous définissons une fonction $\boldsymbol{\delta}$ égale à 1 pour une position alignée et 0 sinon (Figure 4.4).

Définition 15. *(Alignement (revisité))*
Un **alignement structure-séquence** entre une séquence arc-annotée $A = (S_A, P_A)$ de longueur n et une séquence S_B de longueur m est une application $f = (\mu, \delta)$ telle que :
- $f(i) = (\mu(i), \delta(i))$.
- δ est une application de $[1, n]$ à valeurs dans $\{0, 1\}$
- μ est une application de $[1, n]$ dans $[1, m+1]$ telle que :
  - μ préserve l'ordre (au sens large) : $i < j \Rightarrow \mu(i) \leq \mu(j)$.
  - Pour tout $i \in [1, n-1]$, si $\mu(i) = \mu(i+1)$ alors $\delta(i) = 0$, sinon $\delta(i) = 1$.
  - Si $\mu(n) = m + 1$ alors $\delta(n) = 0$.

Il y a équivalence entre les deux définitions. C'est à dire qu'à chaque "ancien" alignement correspond un unique nouvel alignement et de même coût. Nous étendons alors toutes les notations précédentes à cette nouvelle définition.

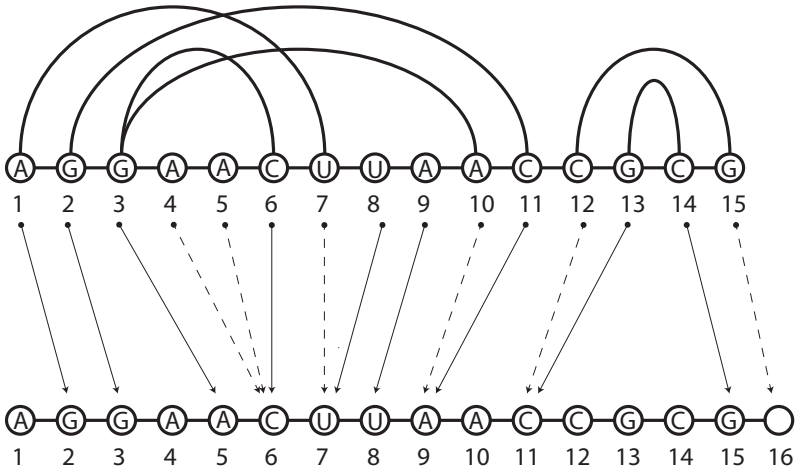


FIGURE 4.4 – *Représentation des nouveaux alignements. Les flèches pleines représentent un alignement de position* ($\delta(i) = 1$) *alors que les flèches pointillées représentent un non-alignement* ($\delta(i) = 0$).

Grâce à ce formalisme, nous pouvons prouver comme nous l'avons fait pour les alignements sans insertions ni suppressions que la compatibilité entre l'alignement f d'un sac X_l et l'alignement f' de son (ou ses) fils X_s implique la compatibilité entre f et l'alignement optimal de $\mathcal{D}(X_s)$ compatible avec f', et ce en considérant toutes les opérations possibles (la preuve est exactement la même, en remplaçant μ par $f = (\mu, \delta)$).

L'équation de récurrence devient alors (les notations $\gamma, \phi, \alpha_A, \alpha_B, \beta_A, \beta_B$, sont celles utilisées pour la définition du problème au chapitre précédent).

4.2.4 Étude de complexité

Nous allons maintenant nous intéresser à la complexité de l'algorithme de programmation dynamique découlant de l'équation de récurrence 1. Pour calculer la complexité en temps nous devons déterminer le nombre de coûts ($\mathsf{Coût}(l, f)$) à calculer, puis pour chacun d'entre eux, trouver le nombre d'opérations intervenant dans leur calcul. Pour un sac X_l donné, on pose $t = |X_l| - 1$ (la largeur du sac), $t' = |X_{l,r}|$ et $t'' = |X_l - X_r|$.

– **Dénombrement de l'ensemble des coûts ($\mathsf{Coût}(l, f)$) à calculer :**
La décomposition arborescente possède N sacs, donc $l \in [1, N]$.
L'application f représentant un alignement de $X_{l,r}$ avec S_B, nous avons t' positions à aligner avec S_B. Chaque position de $X_{l,r}$ peut donc être alignée ($\delta = 1$) et non-alignée ($\delta = 0$) à m différentes positions de S_B, ce qui laisse donc $2m$ choix différents par position. L'alignement f devant respecter la contrainte d'ordre, le nombre d'alignements possibles est au pire une combinaison (sans remise, sans ordre) de t' éléments parmi $2m$. Donc le nombre de $\mathsf{Coût}(l, f)$ que

Équation de récurrence 1.
Soient X_l un sac et X_r son père et soit $f = (\mu, \delta)$ un alignement des positions de $X_{l,r}$ avec S_B ($f \in \mathcal{F}|_{X_{l,r}}$). Alors le coût $\mathsf{Coût}(l, f)$ de l'alignement optimal entre la sous-structure descendante de X_l et S_B compatible avec f peut être calculé à partir de l'équation de récurrence :

$$\mathsf{Coût}(l, f) = \min_{\substack{f'=(\mu',\delta')\in\mathcal{F}|_{X_l} \\ f' \text{compatible avec } f}} \left\{ \phi(X_l, f') + \sum_{s \text{ fils de } l} \mathsf{Coût}(s, f'|_{X_{s,l}}) \right\},$$

où ϕ calcule la contribution locale du coût :

$$\phi(X_l, f') = \sum_{\substack{i\in X_l - X_r \\ \delta'(i)=1}} \gamma(i, \mu'(i)) + \sum_{\substack{i,j\in X_l \text{ t.q.} \\ i \text{ ou } j\in X_l - X_r \\ \text{et } (i,j)\in P_A}} \varphi(i, j, f'(i), f'(j))$$

// Substitution de bases et opérations sur les interactions

$$+ \sum_{\substack{i,i+1\in X_l \text{ t.q.} \\ i \text{ ou } i+1\in X_l - X_r \\ \text{et } \mu'(i+1)>\mu'(i)+1}} \alpha_B \cdot (\mu'(i+1) - \mu'(i) - 1) + \beta_B \qquad \text{// Insertions}$$

$$+ \sum_{\substack{i,i+1\in X_l \text{ t.q.} \\ i \text{ ou } i+1\in X_l - X_r \\ \delta'(i)=1 \text{ et } \delta'(i+1)=0}} \beta_A + \sum_{\substack{i\in X_l - X_r \\ \text{t.q. } \delta'(i)=0}} \alpha_A \qquad \text{// Suppressions}$$

nous devons calculer est en $O\left(N \cdot \binom{2m}{t'}\right)$.

– **Nombre d'opérations pour le calcul d'un coût ($\mathsf{Coût}(l, f)$) :**
Le calcul se base sur la détermination d'un minimum. Ce minimum se recherche sur l'ensemble des alignements f' de X_l et compatibles avec f. Donc toutes les positions de $X_{l,r}$ sont imposées par f, et seules les positions de $X_l - X_r$ peuvent évoluer. Nous recherchons donc un alignement optimal de t'' positions ce qui représente au pire une recherche de minimum parmi $O\left(\binom{2m}{t''}\right)$ éléments. Pour chacun de ces alignements f' nous devons calculer la fonction ϕ et ajouter le coût des alignements optimaux des structures descendantes des fils de X_l. Des quatre sommes composant la fonction ϕ, trois se calculent en $O(t'')$ opérations (première, troisième et quatrième sommes). La deuxième somme demande au pire $O\left(\binom{t''}{2}\right)$, correspondant au nombre d'interactions que peuvent former t'' positions entre elles.
La décomposition étant supposée binaire, le reste du calcul de $\mathsf{Coût}(l, f)$ s'effectue en temps constant.

Donc la complexité en temps est égale à :

$$\begin{aligned}
& O\left(\binom{2m}{t'} \cdot \binom{2m}{t''} \cdot \binom{t''}{2}\right) \\
= \; & O\left(\frac{(2m)^{t'}}{t'!} \cdot \frac{(2m)^{t''}}{t''!} \cdot t''(t''-1)\right) \\
= \; & O\left(\frac{(2m)^{t'}}{t'!} \cdot \frac{(2m)^{t''}}{(t''-2)!}\right) \\
= \; & O\left(\frac{(2m)^{t+1}}{t'!(t''-1)!}\right) \qquad\qquad \text{car } t'+t''=t+1 \\
= \; & O\left(\frac{2^{t'}2^{t''}}{t'!(t''-1)!} \cdot m^{t+1}\right) \\
= \; & O\left(m^{t+1}\right)
\end{aligned}$$

Pour calculer la complexité en mémoire, nous devons uniquement prendre en compte le nombre de coût ($\mathsf{Coût}(l, f)$) à calculer et enregistrer. Par conséquent, la complexité en mémoire est en $O\left(N \cdot \binom{2m}{t'}\right) = O\left(N \cdot m^{t'}\right)$. Dans le pire des cas $t' = t$, car comme nous l'avons vu précédemment $|X_{l,l'}| < |X_{l'}|$ pour $X_{l'}$ fils de X_l. Donc la complexité est égale au pire à $O\left(N \cdot m^t\right)$.

Nous pouvons alors énoncer le théorème suivant :

Théorème 1.
Soit $A = (S_A, P_A)$ une séquence arc-annotée de longueur n et S_B une séquence de longueur m. Supposons qu'il existe $D = (X, T)$, une décomposition arborescente de A de N sacs et de largeur k. Alors l'alignement structure-séquence de A avec B peut être calculé en $O\left(N \cdot m^{k+1}\right)$ temps et en $O\left(N \cdot m^k\right)$ espace en utilisant l'équation 1.

4.3 Décompositions arborescentes lissées

L'équation de récurrence 1 et le théorème que nous venons d'énoncer ont été élaborés pour résoudre le problème d'alignement structure-séquence à partir de décompositions arborescentes de séquences arc-annotées. Le temps nécessaire et l'espace mémoire pour cette résolution sont paramétrés par la largeur de la décomposition arborescente. C'est-à-dire que pour une décomposition arborescente de largeur k et contenant N sacs et une séquence de longueur m, la complexité en temps (et au pire) de l'alignement entre la décomposition et la séquence par l'approche présentée est en $O(N \times m^{k+1})$. Comme nous l'avons vu, cette approche est basé sur une technique de programmation dynamique utilisant des décompositions arborescentes

que nous avons du adapter à notre problème. Nous allons maintenant voir comment nous pouvons tirer partie de la spécificité du problème.

Cette spécificité est entièrement liée à présence au caractère additif des coûts des gaps. Intuitivement, nous allons utilisé la même astuce que celle développée par Gotoh [26] pour l'alignement de séquences avec pénalité de gaps affine. Cette astuce, que nous nommons **technique de lissage** (pour l'alignement structure-séquence), va nous permettre d'améliorer notre approche. Concrètement, dans l'équation de récurrence 4.1, nous recherchons actuellement un alignement f' du sac courant minimisant le coût de l'alignement la sous-structure descendante associée. Cet alignement f' doit être compatible avec l'alignement f fourni, ce qui implique, comme nous l'avons déjà remarqué, que seuls les alignements des positions de $X_l - X_r$ sont à déterminer, chacune d'entre elles ayant au pire $O(m)$ choix d'alignements possibles. Or, si le sac possède une certaine propriété, il va nous être possible, pour une de ces positions, de connaitre son alignement optimal en temps constant, diminuant ainsi la complexité de l'algorithme.

Illustrons ceci avec un exemple. Reprenons la structure de l'exemple de la figure 4.3, et pour simplifier la description, nous supposerons que les suppressions sont interdites. Nous voulons calculer le coût de l'alignement optimal ω de $\mathcal{D}(X_1)$ avec la séquence, compatible avec $f \in \mathcal{F}_{|X_1}$ tel que :

- $f(2) = (2', 1)$
- $f(6) = (6', 1)$

Nous devons alors choisir un alignement pour la position de $X_1 - X_0$, c'est à dire la position 5. Nous devrions alors tester tous les alignements possibles : $5 \mapsto 3'$, $5 \mapsto 4'$ et $5 \mapsto 5'$, en calculant leur coût respectif et en choisissant le minimum, tel que le fait l'équation de récurrence 1 (voir Figure 4.5). C'est ici qu'intervient la technique de lissage. Soient ω_1, ω_2 et ω_3 les alignements optimaux de $\mathcal{D}(X_1)$ compatibles avec f tel que la position 5 soit alignée respectivement sur $5'$ (c'est-à-dire pas d'insertion entre la position 5 et la position 6), $4'$ et $3'$ (voir Figure 4.5). Dans un premier temps nous calculons le coût de ω_1. Nous devons ensuite remarquer que nous avons potentiellement accès au coût du meilleur alignement restant (entre ω_2 et ω_3) et qu'il est donc inutile de le calculer. Ceci est dû à une propriété spéciale du sac X_1. Dans ce sac, il existe une position, la position 6, possédant une particularité intéressante : elle n'apparait pas dans la descendance de X_1 et n'est en interaction avec aucune position apparaissant dans X_1 (en partant de la racine). De plus, une des positions venant d'apparaitre (la position 5) se trouve être voisine de la position 6. Par conséquent, la seule contribution pour le coût de position la 6 pour l'alignement de la sous-structure $\mathcal{D}(X_1)$ porte sur les insertions possibles entre elle sa voisine de gauche (la position 5).

Supposons alors que nous ayons déjà calculé les coûts associés aux alignements optimaux de $\mathcal{D}(X_1)$ compatible avec des alignements de $\mathcal{F}_{|X_1}$ pour lesquels la position 6 est alignée avant la position $6'$, et appelons ω' l'un de ces alignement compa-

tible avec $f' \in \mathcal{F}_{|X_1}$ tel que (voir Figure 4.5) :

- $f'(2) = (2', 1)$
- $f'(6) = (5', 1)$

Soit ω'_2 et ω'_3 les alignements optimaux de $\mathcal{D}(X_1)$ compatibles avec f' tel que la position 5 soit alignée respectivement sur $4'$ et $3'$ (voir Figure 4.5). Étant donné que l'alignement de la position 6 n'apporte pas de contribution au calcul du coût autre que sa distance à l'alignement de la position 5, nous pouvons établir que la différence entre le coût de ω'_2 et de ω_2 est d'un gap et que de même la différence entre le coût de ω'_3 et de ω_3 est également d'un gap. Donc quoiqu'il en soit, l'un des coûts ω_2 ou ω_3 est égale au coût de ω' plus le coût d'un gap (voir Figure 4.5).

Par conséquent, le coût de ω est égale au minium entre le coût de ω_1 et le coût de ω' plus un gap (voir Figure 4.5).

Nous procédons de même pour ω', c'est-à-dire que nous le calculons en prenant le minium entre le coût de ω'_2 (pas de gap entre 5 et 6) et l'alignement optimal ω'' de $\mathcal{D}(X_1)$ compatible avec $f'' \in \mathcal{F}_{|X_1}$ tel que (voir Figure 4.5) :

- $f''(2) = (2', 1)$
- $f''(6) = (4', 1)$

Comme il n'y a qu'un choix d'alignement pour la position 5 (compatible avec f''), ω'' se calcule en temps constant (voir Figure 4.5).

Ainsi lorsque la situation se présente, nous pouvons calculer le coût d'un alignement optimal d'un sac X_l compatible avec un alignement de $X_{l,r}$ (X_r parent de X_l) en temps constant, et donc gagner un degré de puissance dans la complexité en temps pour le calcul du coût de l'alignement optimal. Nous allons définir rigoureusement les cas pour lesquels nous pouvons appliquer cette technique puis nous adapterons l'équation de récurrence 4.1 afin de l'incorporer.

4.3.1 Définitions

Pour pouvoir utiliser la technique que nous venons d'illustrer, un sac doit posséder une position (position **lissante**) n'apparaissant pas dans la descendance du sac et n'ayant aucune interaction avec les positions venant d'apparaitre dans le sac (en partant de la racine), ceci pour que l'alignement de cette position (la position 6 dans l'exemple précédent) apporte une contribution non liée aux substitutions dans le calcul du coût. De plus, ce sac doit posséder une de ses positions voisines (position **lissée**) et apparaissant pour la première fois dans la décomposition (position 5 dans l'exemple précédent). Voici la définition formelle de cette propriété :

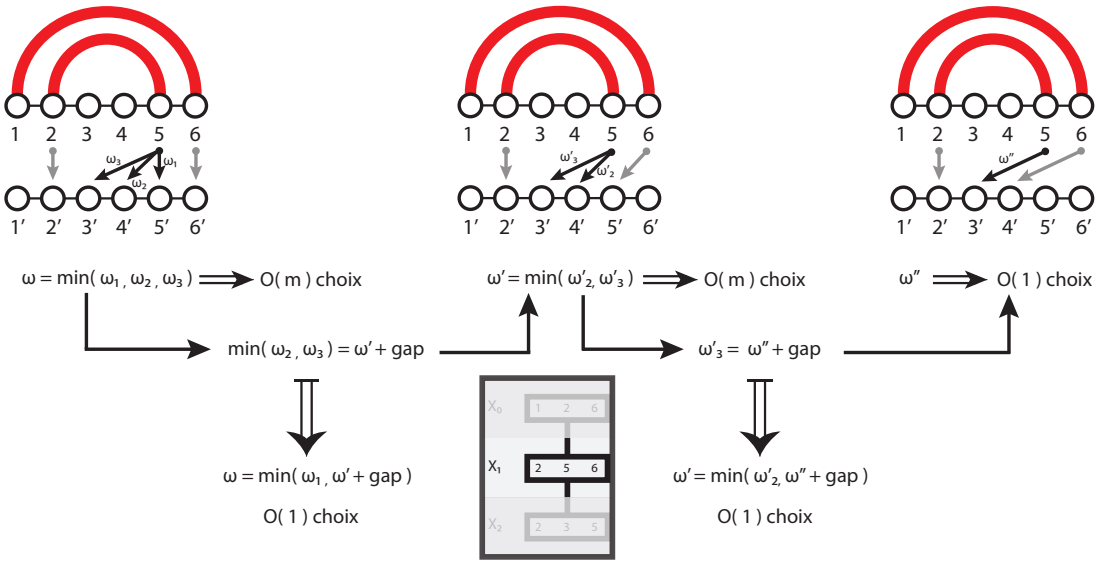


FIGURE 4.5 – *Illustration de la technique de lissage. Sur la figure, toutes les déclinaisons des ω font référence à des alignements optimaux (comme explicité dans le texte). En grisé apparaissent les alignements imposés par l'équation de récurrence, et en non-grisé l'alignement non imposé sur lequel nous cherchons à minimiser. Dans les formules (sous les figures), toutes les déclinaisons des ω font référence aux coûts associés aux alignements du même nom.*

Définition 16. *(Sac lissé d'une décomposition arborescente)*
Soit $D = (X, T)$ une décomposition arborescente d'une séquence arc-annotée $A = (S_A, P_A)$ et soit $X_l \in X$ un sac de D différent de la racine X_0. Soit X_r le père de X_l.
-X_l est **lissé** si et seulement si il existe deux positions consécutives i et j telles que $i \in X_l - X_r$, $j \in X_{l,r}$, j n'appartient à aucun fils de X_l et il n'existe pas de position de $X_l - X_r$ qui soit en interaction avec j ou consécutive à j (mis à part i). La position i est la position **lissée** et la position j est la position **lissante**.
-La racine X_0 est **lissée** si et seulement si une des deux conditions suivantes est vérifiée :

a. Il existe deux positions consécutives i et j appartenant à X_0 telles que j n'appartient à aucun fils de X_0 et $(i, j) \notin P$.

b. La largeur de X_0 est strictement inférieur à la largeur d'un de ses fils.

La racine est traitée différemment étant donné que la propriété définie ici fait intervenir une relation avec le parent. Nous ne pourrons donc pas lui appliquer la technique de lissage. Cependant, le but de cette méthode est de diminuer la complexité (au pire) en temps de notre approche. Les sacs qui devront être lissés sont donc en priorité les sacs de plus grande taille. Or si la racine est un de ces

grands sacs, mais qu'elle possède la propriété a, alors nous pouvons modifier la décomposition en transformant la racine en une chaîne de deux sacs, l'un identique à l'ancienne racine mais devenant lissé, et l'autre (la nouvelle racine) possédant la propriété b. Ainsi, même si la racine ne possède pas de position lissante (et lissée), son alignement pourra s'effectuer avec le même niveau de complexité que les sacs de plus grande taille.

Le nombre de sac lissés d'une décomposition arborescente va donc permettre de paramétrer plus efficacement la complexité de l'alignement structure-séquence.

Définition 17. *(Décomposition arborescente (faiblement-)lissée)*
Soit $D = (X, T)$ une décomposition arborescente d'une séquence arc-annotée $A = (S_A, P_A)$. D est **k-faiblement-lissée** si et seulement si au moins k de ses sacs sont lissés. D est **lissée** si et seulement si tous ses sacs sont lissés.

4.3.2 Équation de récurrence

Nous allons maintenant définir formellement comment effectuer la technique de lissage sur un sac lissé. Nous considérons deux cas distincts suivant que la position lissée i se trouve avant ou après la position lissante $j \in \{i-1, i+1\}$. Nous cherchons donc à calculer le coût de l'alignement optimal de la sous-structure descendante d'un sac X_l avec une séquence, compatible avec un alignement $f = (\mu, \delta)$ de $X_{l,r}$, où X_r est le parent de X_l. Quel que soit le cas que nous allons considérer, la position lissante j appartient par définition à $X_{l,r}$ et possède donc un alignement par f.

Le **cas 1** correspond au cas où la position lissée i se trouve après la position lissante $j = i - 1$. D'après la nouvelle définition des alignements, si la position $i-1$ n'est pas alignée par f ($\delta(i-1) = 0$) alors nécessairement le choix de l'alignement de la position lissée est imposée (voir Figure 4.6.A) : $\mu(i-1) = \mu(i)$ et $\delta(i) \in \{0, 1\}$ (**A**). Ce choix conditionné de l'alignement de la position lissée s'effectue donc en temps constant (deux choix possibles), et il reste ensuite à déterminer l'alignement optimal des autres positions de $X_l - X_r$ compatible avec l'un des deux alignements possibles de la position lissée.

Si la position $i-1$ est alignée par f ($\delta(i-1) = 1$), alors nous pouvons appliquer la technique de lissage. Nous commençons par calculer le coût des alignements sans insertion entre la position lissante $i-1$ et la position lissée i (voir Figure 4.6.B), c'est à dire les deux sous-cas possibles (**B**) : $\mu'(i) = \mu(i-1) + 1$, et $\delta'(i) \in \{0, 1\}$, puis trouver l'alignement optimal des autres positions de $X_l - X_r$ en compatibilité avec l'un de ces deux sous-cas. Il reste alors à comparer les coûts des alignements ci-dessus au coût des meilleurs alignements optimaux avec insertion(s) (**C**). Le coût de chacun des ces alignements est égal au coût de l'alignement optimal où la position lissante est alignée une position plus loin et où les autres positions du sac sont alignées de la même façon (voir Figure 4.6.C) et auquel nous ajoutons le coût d'un

gap. En commençant par calculer le cas limite, c'est à dire celui où l'alignement de la position lissante ne peut plus être décalé plus loin (voir Figure 4.6.C_3), nous obtenons en temps constant l'alignement optimal de la position lissée pour n'importe quel alignement du sac privé de cette position.

Pour pouvoir prendre en compte les pénalités de gaps affines, nous allons devoir calculer en double chacun des coûts. En effet lorsque nous effectuons l'appel récursif de la technique de lissage, nous devrions ajouter le coût d'un gap et le coût d'une ouverture de gap. Par conséquent, le coût d'une insertion calculé à partir d'une succession d'appels récursifs de la technique de lissage sera formé de plusieurs coûts d'ouverture de gap (autant que d'appels récursifs). Pour remédier à ce problème, nous calculons en amont les coûts des alignements avec pénalités de gaps linéaires et tels que la position lissante est systématiquement alignée (**D,E**). Puis, lorsque nous calculons les coûts des alignements avec pénalités de gaps affines, nous effectuons l'appel récursif de la technique de lissage sur les coûts précédemment calculés en prenant en compte l'ouverture de gap (**C**). De cette façon n'ajoutons qu'un coût d'ouverture de gap par insertion.

Équation de récurrence 2. *Cas 1*

$$\mathsf{Coût}(l,f) = \min \begin{cases} \boxed{\begin{array}{l}\textbf{A} \text{ Si } \delta(i-1)=0, \text{ agréger } i \text{ sur } i-1 : \ \boldsymbol{\mu'(i) = \mu(i-1)}. \\ \textbf{B} \text{ Si } \delta(i-1)=1, \text{ pas d'insertion entre } i \text{ et } i-1 : \boldsymbol{\mu'(i) = \mu(i-1)+1} \\ \ldots \end{array}} \\ \min\limits_{\substack{f'=(\mu',\delta')\in\mathcal{F}|_{X_l} \\ f' \text{ compatible avec } f}} \phi(X_l,f') + \sum\limits_{s \ fils \ de \ l} C(s, f'|_{X_{s,l}}) \\ \\ \boxed{\begin{array}{l}\textbf{C} \ldots \text{ou ouverture de gap et décalage de } i-1 \text{ pour éviter de } \textit{tester} \\ \text{tous les alignements possibles pour } i : \boldsymbol{\mu''(i-1) = \mu(i-1)+1, \text{ et}} \\ \boldsymbol{f'' = f} \textbf{ sinon}. \end{array}} \\ \alpha_B + \beta_B + D(l,f'') \end{cases}$$

$$D(l,f) = \min \begin{cases} \boxed{\textbf{D} \text{ Position déterminée pour } i : \boldsymbol{\mu'(i) = \mu(i-1)+1} \Rightarrow \text{ pas de gap supplémentaire.}} \\ \min\limits_{\substack{f'=(\mu',\delta')\in\mathcal{F}|_{X_l} \\ f' \text{ compatible avec } f}} \phi(X_l,f') + \sum\limits_{s\in sons(l)} C^s_{f'|_{X_{s,l}}} \\ \\ \boxed{\textbf{E} \text{ Poursuite de l'insertion : } \textbf{Si } \boldsymbol{\delta(i-1)=1, \mu''(i-1) = \mu(i-1)+1}} \\ \alpha_B + D(l,f'') \end{cases}$$

Pour le **cas 2** où la position lissée i se trouve avant la position lissante $i+1$, la situation n'est pas tout à fait symétrique. En effet les possibilités d'alignements où i n'est pas aligné ($\delta'(i)=0$) se résument à un seul choix (**A'**) : $\mu'(i) = \mu(i+1)$ (voir Figure 4.7.A'). Pour les autres alignements possibles f' ($\delta'(i)=1$), nous commençons

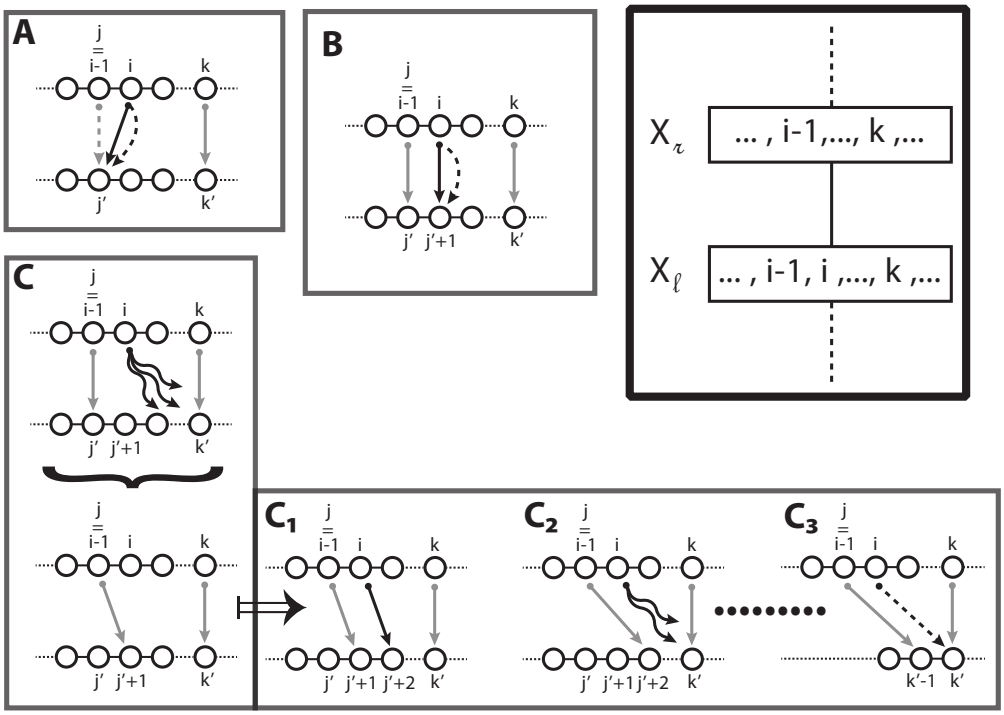


FIGURE 4.6 – *Illustration des différents sous-cas possibles du* ***cas 1****. Les flèches grisées représentent l'alignement de $X_{l,r}$ alors que les flèches noires représentent l'alignement de $X_l - X_r$ Sous-cas* **A** *et sous-cas* **B** *: la flèche pleine et la flèche en pointillés partant d'une même position doivent s'interpréter comme deux alignements différents. Sous-cas* **C** *: le coût du meilleur alignement avec insertions entre la position lissante et lissée est déterminé à partir du coût de l'alignement optimal où la position lissante est alignée une position plus loin (et l'alignement autres positions de $X_{l,r}$ reste le même), lui même déterminé de la même façon C_1, C_2 jusqu'au cas limite C_3.*

par calculer les coûts des alignements sans insertion entre la position lissante et la position lissée (**B'**), c'est-à-dire $\mu'(i) = \mu(i+1) - 1$ (voir Figure 4.7.B'). Puis nous comparons les coûts obtenus ci-dessus aux coûts des meilleurs alignements optimaux avec insertion(s) (**C'**) avec la même technique que pour le **cas 1** (voir Figure 4.7.C').

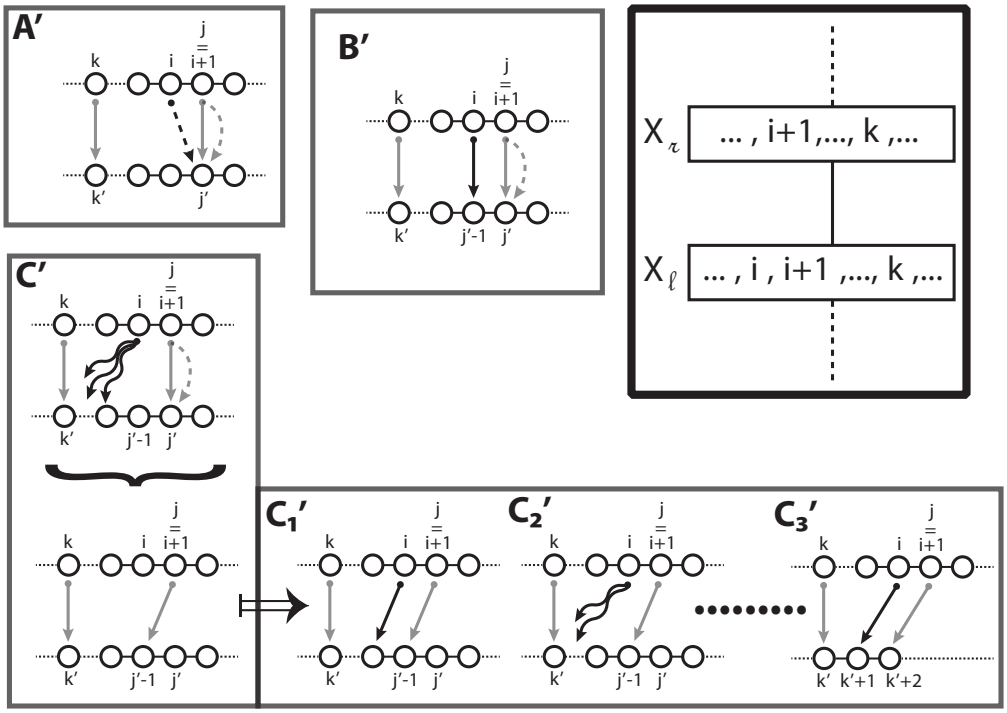


FIGURE 4.7 – *Illustration des différents sous-cas possibles du* ***cas* 2**. *Les flèches grisées représentent l'alignement de $X_{l,r}$ alors que les flèches noires représentent l'alignement de $X_l - X_r$ Sous-cas* **A'** *et sous-cas* **B'** *: la flèche pleine et la flèche en pointillés partant d'une même position doivent s'interpréter comme deux alignements différents. Sous-cas* **C'** *: le coût du meilleur alignement avec insertions entre la position lissante et lissée est déterminé à partir du coût de l'alignement optimal où la position lissante est alignée une position avant (et l'alignement autres positions de $X_{l,r}$ reste le même), lui même déterminé de la même façon C'_1, C'_2 jusqu'au cas limite C'_3..*

Équation de récurrence 3. *Cas 2*

$$\mathsf{Coût}(l,f) = \min \begin{cases} \boxed{\begin{array}{l}\textbf{A'} \text{ Si } \delta'(i)=0 : \text{Agréger } i \text{ sur } i+1 \ (\mu'(i)=\mu(i+1)).\\ \textbf{B'} \text{ Si } \delta'(i)=1 : \text{pas d'insertion entre } i \text{ et } i+1 : \boldsymbol{\mu'(i)=\mu(i+1)-1}\\ \ldots\end{array}} \\ \min\limits_{\substack{f'=(\mu',\delta')\in\mathcal{F}|_{X_l}\\ \text{s.t.} f'=f \text{ on } X_{l,r}}} \phi(X_l,f') + \sum\limits_{s\in sons(l)} C^s_{f'|_{X_{s,l}}} \\ \boxed{\textbf{C'} \ldots \text{ou ouverture de gap et décalage de } i+1 \text{ pour éviter de } \textit{tester} \text{ tous les alignements possibles pour } i. \ (\mu''(i+1)=\mu(i+1)-1, \text{ and } f''=f \text{ sinon})} \\ \alpha_B + \beta_B + D(l,f'') \end{cases}$$

$$D(l,f) = min \begin{cases} \boxed{\textbf{D'} \text{ If } \delta'(i)=1 : \text{Position déterminée pour } i \ (\mu'(i)=\mu(i+1)-1) \Rightarrow \text{pas de gap supplémentaire}} \\ \min\limits_{\substack{f'=(\mu',\delta')\in\mathcal{F}|_{X_l}\\ \text{s.t.} f'=f \text{ on } X_{l,r}}} \phi(X_l,f') + \sum\limits_{s\in sons(l)} C^s_{f'|_{X_{s,l}}} \\ \boxed{\textbf{E'} \text{ Poursuite de l'insertion : } \textbf{Si } \boldsymbol{\delta(i+1)=1, \mu''(i+1)=\mu(i+1)-1}} \\ \alpha_B + D(l,f'') \end{cases}$$

4.3.3 Étude de complexité

L'impact sur la complexité de la prise en compte de la propriété lissante des sacs est simple : les coûts associés à chaque sac lissé se calculent en temps $O(m^k)$ où k est la largeur du sac et non plus $O(m^{k+1})$. La preuve est immédiate puisque la technique de lissage permet de connaître en temps constant l'alignement optimal de la position lissée, associé (et compatible) à un alignement des autres positions du sac. Nous pouvons alors énoncer le théorème suivant :

Théorème 2.
Soit $A = (S_A, P_A)$ une séquence arc-annotée de longueur n et S_B une séquence de longueur m. Supposons qu'il existe $D = (X, T)$ une décomposition arborescente de A de N sacs et de largeur t. Si D est k-faiblement-lisse alors l'alignement structure-séquence de A avec B peut être calculé en $O(k \cdot m^{t+1} + N \cdot m^t)$ en utilisant les équations 2 et 3.

Nous avons tiré partie de la spécificité du problème d'alignement structure-séquence dans l'utilisation de décompositions arborescentes, mais pour ces efforts ne soient pas vains, nous allons devoir montrer qu'il nous est possible de construire efficacement des décompositions arborescentes de structures d'ARN, c'est-à-dire de les construire rapidement et surtout en minimisant autant que possible leur largeur.

4.4 Construire des décompositions arborescentes d'ARN

Dans la première partie de ce chapitre, nous venons de voir que pour résoudre le problème d'alignement structure-séquence d'ARN nous pouvons utiliser une décomposition arborescente de la structure. La complexité de l'alignement est alors paramétrée par la largeur arborescente de la décomposition et par le nombre de sacs lissés.

Jusqu'ici nous avons occulté la construction de la décomposition. Pourtant cette étape est loin d'être évidente, car comme nous l'avons déjà évoqué, construire une décomposition arborescente de largeur minimale d'une séquence arc-annotée est un problème NP-Difficile [5]. C'est pourtant ce que nous voudrions résoudre étant donné que moins la largeur de la décomposition est grande, moins la complexité est élevée.

En plus de devoir construire des décompositions de faibles largeurs, nous allons devoir en plus veiller à ce qu'elles contiennent le plus de sacs lissés possible si nous voulons pouvoir prétendre généraliser sans perte d'efficacité l'alignement structure-séquence.

Nous allons maintenant montrer comment répondre à cette problématique, et démontrer qu'il est possible d'utiliser pour cela une approche générale, c'est-à-dire ne pas considérer les classes de structures au cas par cas. Dans un premier temps nous diviserons la structure en sous-structures (que nous appellerons **structures primitives**) pour lesquels nous présenterons, dans un second temps un algorithme de décomposition arborescente. Nous expliquerons ensuite comment former une décomposition arborescente de la structure totale à partir des décompositions de ces sous-structures. Les décompositions obtenues seront alors optimales (du point de vue de la largeur et du nombre de sacs lissés) sur la majorité des structures présentes dans les bases de données ([59, 58, 4]) et nous décrirons dans une dernière section comment les améliorer dans les rares cas où cela est encore possible.

4.4.1 Division en structures primitives

Avant d'expliquer comment diviser une structure d'ARN nous allons en expliquer la raison. Imaginons un instant vouloir faire une décomposition arborescente d'une séquence arc-annotée qui possèderait deux parties ne partageant pas une seule interaction (voir Figure 4.8). Nous pourrions envisager de décomposer indépendamment chacune des parties et de les rassembler grâce à un simple sac supplémentaire (voir Figure 4.8).

Nous allons systématiser cette idée, c'est à dire diviser la structure en briques élémentaires que nous appellerons **structures primitives** que nous décomposerons ensuite indépendamment. Si nous voulons effectuer indépendamment les décompositions, nous devons faire en sorte de diviser la structure en primitives possédant elles aussi une notion d'indépendance. Nous baserons l'indépendance entre deux sous-structures sur le non-partage d'interaction. Ce qui implique l'indépendance de sous-structures qui seraient "consécutives" (comme sur la Figure 4.8), mais aussi l'indépendance de sous-structures imbriquées. Ainsi dans la structure de la Figure 4.8, les sous-structures induites par les séquences contenant les positions 1 à 13 d'une part et 14 à 17 d'autre part sont indépendantes entre elles. Les sous-structures induites par les séquences contenant les positions 1 et 13 d'une part et 9 et 12 d'autre part sont aussi indépendantes entre elles.

Nous définissons maintenant formellement les **structures primitives**. Pour cela, nous introduisons le graphe des conflits d'une séquence arc-annotée (voir Figure 4.9). Ce graphe va ensuite nous permettre de définir les structures primitives par l'intermédiaire de ses composantes connexes (voir Figures 4.9 et 4.10).

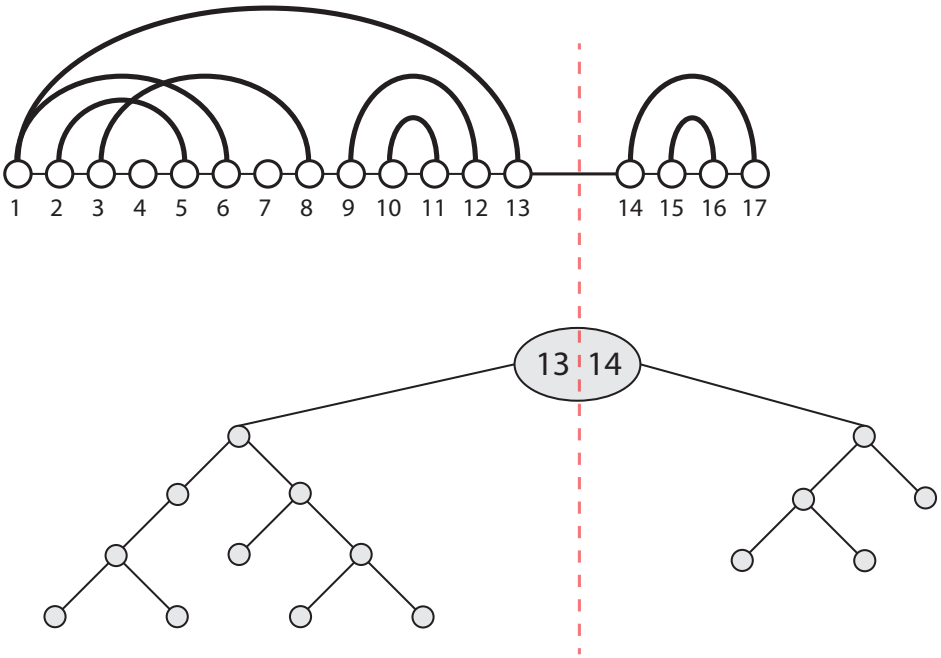


FIGURE 4.8 – *Illustration de l'intérêt de la division des structures en sous-structures pour la décomposition arborescente. Les deux sous-structures (à droite et à gauche de la ligne en pointillés) peuvent se décomposer indépendamment puis leurs décompositions sont ensuite assemblées ensemble par un unique sac contenant les positions* 13 *et* 14 *(c'est-à-dire contenant la dépendance entre les deux sous-structures).*

Définition 18. *(Graphe des conflits)*
Soit $A = (S_A, P_A)$ une séquence arc-annotée. Le **Graphe des conflits** de A est le graphe non-orienté $G = (V, E)$ tel que :
- $V = P$ (les sommets de G sont les interactions de A.
- E est l'ensemble des arêtes tel que : $(v_1, v_2) \in E$ avec $v_1 = (i_1, j_1)$ et $v_2 = (i_2, j_2)$ $(i_1 < i_2)$ si et seulement si $i_1 < i_2 < j_1 < j_2$ (les interactions se croisent).

Définition 19. *(Structures primitive)*
Soit $A = (S_A, P_A)$ une séquence arc-annotée. Les **structures primitives** de A sont les sous-structures de A induites par les composantes connexes du graphe des conflits de A.

Dans la suite, nous nous autoriserons à appeler **primitive** de A une structure primitive de A. Les primitives obtenues ne contiennent pas encore toutes les positions de la structure. En effet, nous remarquons que les positions libres ne sont contenues

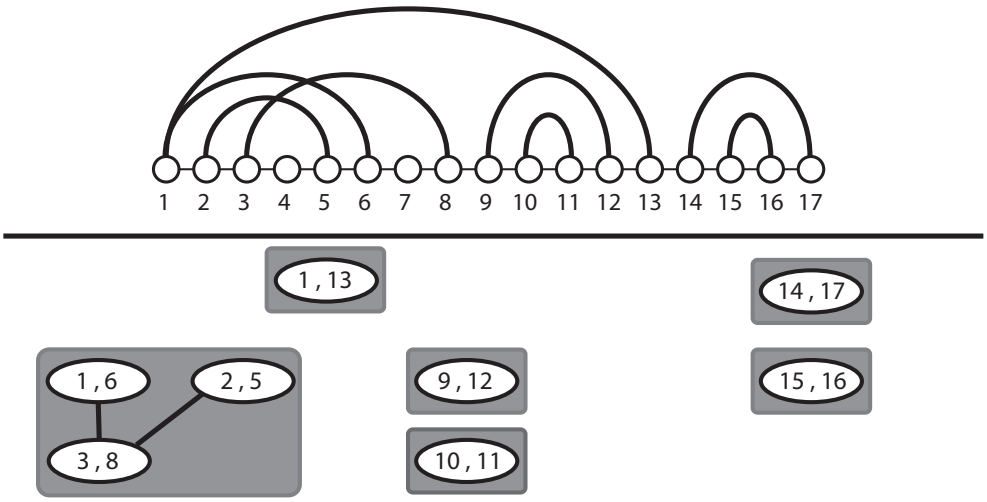


FIGURE 4.9 – *Une séquence arc-annotée et son graphe des conflits. Les sommet du graphe des conflits (formes elliptiques) sont regroupés en composantes connexes (cadres grisés).*

dans aucune primitive. Nous allons alors étendre les primitives, pour inclure toutes les positions de la structure dans au moins une primitive. Informellement, nous allons placer une position libre dans une primitive qui la contient visuellement au plus profond de la relation d'inclusion des primitives. Pour définir cela formellement, nous allons définir plusieurs notions :

Définition 20. *(Bords d'une structure primitive)*
Les **bords** d'une structure primitive sont les positions à l'extrémité gauche et droite de cette primitive.

Dans l'exemple de la Figure 4.10, la primitive P_2 à pour bord gauche la position 1 et pour bord droit la position 8.

Définition 21. *(Encapsulation)*
Soient A_1 et A_2 deux structures primitives d'une séquence arc-annotée et soient i_1 et j_1 les bords de A_1. Alors A_2 est **encapsulée** dans A_1 (ou A_1 **encapsule** A_2) si et seulement si pour toute position i de A_2, $i_1 \leq i \leq j_1$ et il existe au moins une position j de A_2 telle que $i_1 < j < j_1$.

Dans l'exemple de la Figure 4.10, la primitive P_1 encapsule les primitive P_2 et P_3. L'ensemble des encapsulation de la structure de la Figure 4.10 est résumé dans la Figure 4.11.

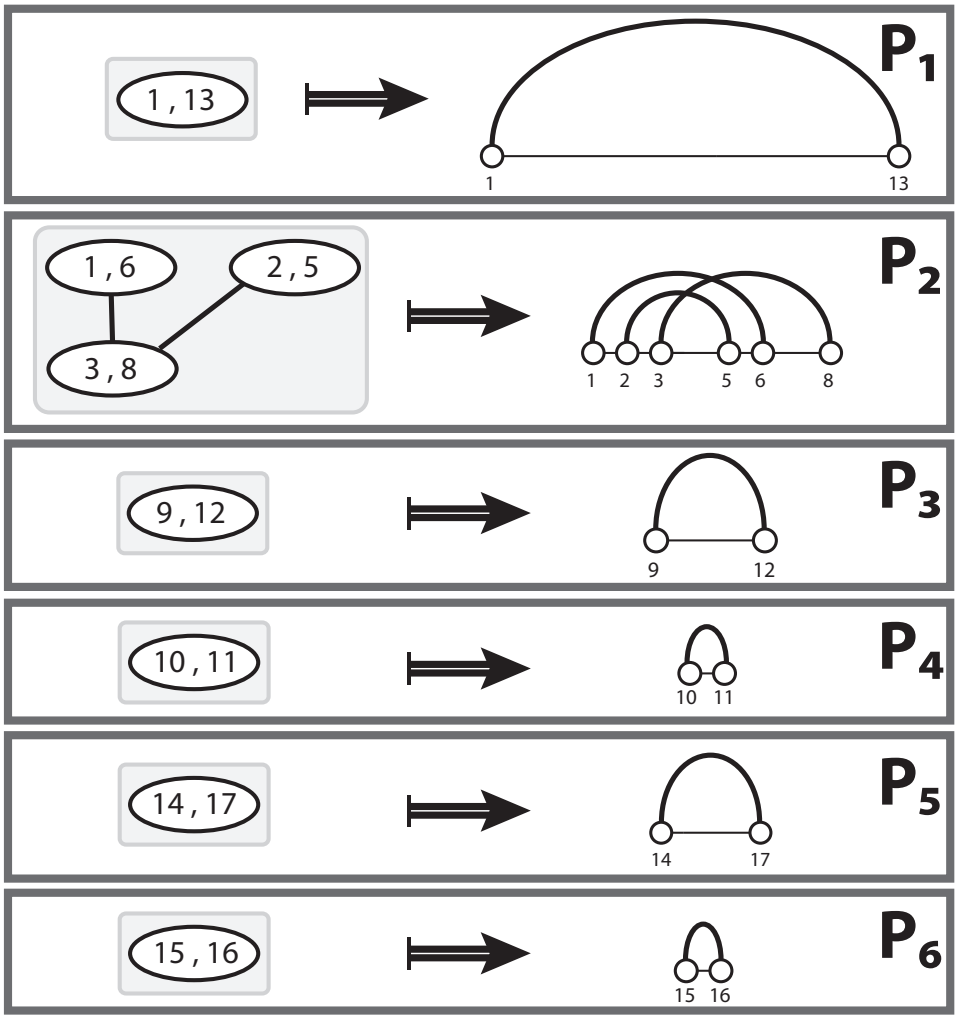


FIGURE 4.10 – *Une séquence arc-annotée et ses structures primitives obtenues à partir du graphe des conflits de la Figure 4.9.*

Définition 22. *(Profondeur d'une structure primitive)*
La **profondeur** d'une structure primitive A_p d'une séquence arc-annotée est le nombre de primitives qui l'encapsulent, elle se note $\mathsf{Profondeur}(A_1)$.

Nous allons alors étendre les primitives en y ajoutant les positions libres qu'elles encapsulent directement. En voici une définition formelle.

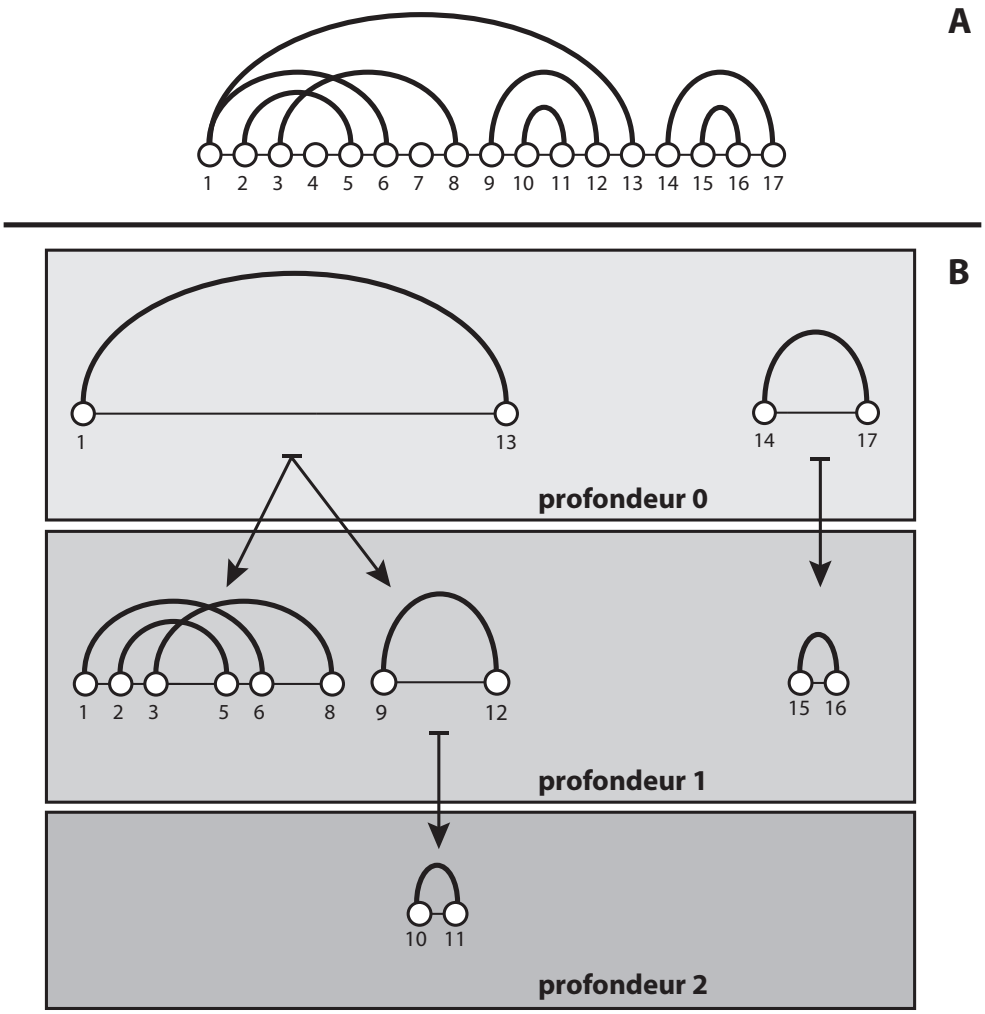


FIGURE 4.11 – *Schéma récapitulatif de l'encapsulation des primitives de la Figure 4.10. Les flèches représentent une encapsulation.*

Définition 23. *(Attribution d'une primitive pour une position libre)*
Soient i une position libre et A_p une primitive d'une même séquence arc-annotée et soient respectivement i_p et j_p le bord gauche et le bord droit de A_p. Alors i est **attribuée** à A_p si et seulement si le deux conditions suivantes sont respectées :et si
– $i_p < i < j_p$.
– il n'existe pas une autre primitive A'_p de bords $i_{p'}$ et $j_{p'}$ telle que $i_{p'} < i < j_{p'}$ et $\mathsf{Profondeur}(A_p) < \mathsf{Profondeur}(A_{p'})$.

Dans la Figure 4.12, les positions libres 4 et 7 sont attribuées à la primitive P_2.

Nous savons maintenant comment étendre nos primitives pour prendre en compte toutes les positions (libres ou non) d'une structure. Nous allons poursuivre notre

extension mais cette fois-ci à des fins purement pratiques, c'est-à-dire pour faciliter l'étape du de l'assemblage des décompositions des extensions des primitives (décompositions que nous allons effectuer indépendamment). Lorsque nous établirons l'algorithme d'assemblage des décompositions de primitives, nous allons devoir prendre en compte la façon dont les primitives sont encapsulées et agencées en terme de séquence. Pour faciliter l'écriture de l'algorithme d'assemblage nous allons simplement étendre les primitives en leur ajoutant les bords des primitives qu'elles encapsulent directement (voir Figure 4.12).

Définition 24. *(Encapsulation immédiate)*
Soit A_1 et A_2 deux structure primitives d'une séquence arc-annotée. A_2 est **immédiatement encapsulée** par A_1 (ou A_1 **encapsule immédiatement** A_2) si et seulement si A_1 encapsule A_2 et $\mathsf{Profondeur}(A_1) = \mathsf{Profondeur}(A_2) - 1$.

Les primitives de profondeur 0 n'étant encapsulées par aucune autre nous ne pouvons pas représenter leur dépendance par le fait qu'elles sont encapsulées dans une même primitive. Nous devons alors directement nous servir de leur séquentialité, un peu comme dans l'exemple de la Figure 4.8.B (voir Figure 4.12).

Définition 25. *(Primitive immédiatement à droite et immédiatement à gauche)*
Soit A_0 une primitive de profondeur 0 de bords i_0 et j_0 d'une séquence arc-annotée $A = (S_A, P_A)$. Alors la primitive **immédiatement à droite** (respectivement **immédiatement à gauche**) de A_0 est la primitive (si elle existe) A'_0 de bords i'_0 et j'_0 de A telle qu'il n'existe pas d'interaction $(i, j) \in P_A$ avec $j_0 < i < i'_0$ ou $j_0 < j < i'_0$ (respectivement $j'_0 < i < i_0$ ou $j'_0 < j < i_0$.

Définition 26. *(Extension d'une structure primitive)*
L'**extension** d'une structure primitive A_p de profondeur $k \neq 0$ d'une séquence arc-annotée est la sous-structure composée de :
- La primitive A_p.
- Les bords des structures primitives immédiatement encapsulées par A_p.
- Les positions libres attribuées à A_p.

L'**extension** d'une structure primitive A_p de profondeur 0 d'une séquence arc-annotée est la sous-structure composée de :
- La primitive A_p.
- Les bords des structures primitives immédiatement encapsulées par A_p.
- Les positions attribuées à A_p.
- Le bord de gauche de la structure immédiatement à droite de A_p.
- Les positions (libres) se trouvant entre le bord droit de A_p et le bord gauche de la structure immédiatement à droite.

Ainsi formées, les extensions des primitives vont faire chacune l'objet d'une décomposition arborescente constituant la réelle difficulté de la décomposition de la structure totale, leur assemblage étant plus beaucoup plus facile à réaliser. Nous allons maintenant expliquer comment effectuer une décomposition de faible largeur et lissée pour les primitives d'une séquence arc-annotée.

4.4.2 Les plongements en vagues

La largeur arborescente d'une séquence arc-annotée représente en quelque sorte sa distance à un arbre. L'agencement de ses primitives permet de dresser un schéma de l'arbre (la décomposition arborescente) à atteindre (voir Figure 4.11). Les primitives quant à elles ne contiennent à priori plus de propriété arborescente évidente. Nous allons alors les décomposer en chaînes, qui une fois assemblées formeront les branches de la décomposition. Comme nous allons le voir, les décompositions en chaînes des primitives que nous allons former seront lissées.

Pour décomposer les primitives en chaînes nous allons "détourner" l'algorithme de chaînage de Han [29] pour l'appliquer aux décompositions arborescentes tout en le généralisant à n'importe quel type de primitive. Pour cela, nous baserons la décomposition sur une partition des positions appelée **plongement en vagues ordonnant** de la primitive. Cette partition formera alors des **intervalles** de positions de la séquence. Nous l'utiliserons pour ordonner les positions de la primitive, ordre qui sera ensuite directement repris pour construire la décomposition arborescente.

Dans toute la suite de la section, $\mathcal{P}$ représente une extension de primitive d'une séquence arc-annotée. Les positions de $\mathcal{P}$ seront numérotées de 1 à n (en d'autres termes, nous réduisons la séquence arc-annotée à une extension de primitive).

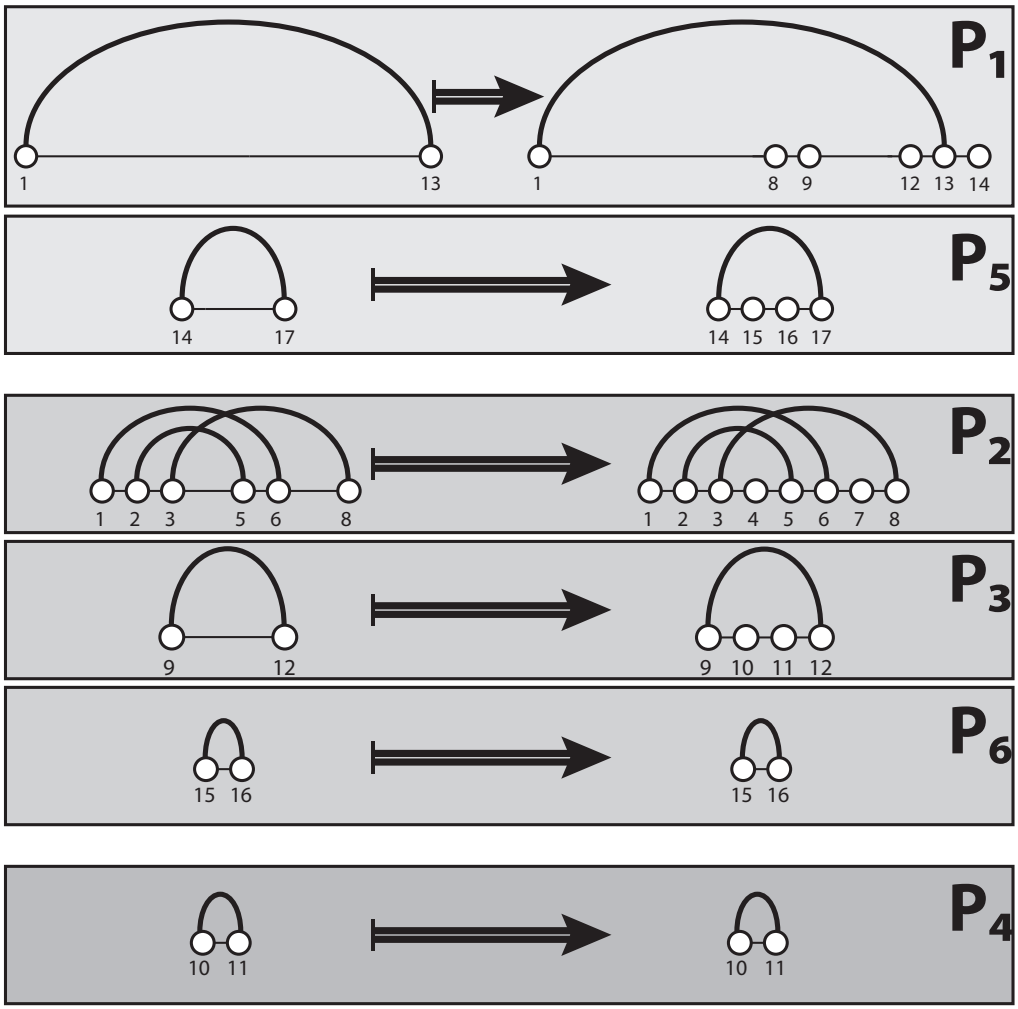


FIGURE 4.12 – *Extensions des primitives de la Figure 4.10.*

Nous commençons par définir une partition de $\mathcal{P}$ que nous appellerons un **plongement en vagues** (voir Figure 4.13.A). Dans cette définition, nous pouvons voir une similarité avec la définition des pseudo-nœuds standard, simples non standard et des triples hélices standard. En réalité, il s'agit là d'une généralisation du concept utilisé par toutes ces approches : pouvoir ordonner les positions.

Définition 27. *(Plongement en vague)*
Soit $\mathcal{P}$ une primitive d'une séquence arc-annotée. Un **plongement en vagues** W de $\mathcal{P}$ est défini par une séquence strictement croissante de positions de $\mathcal{P}$, $\mathbf{y} = \{y_i\}_{i=0}^{k}$ appelées **pivots** telle que :
– $y_0 = 1$.
– Les positions d'une interaction sont séparées par au moins un pivot : $\forall (i,j) \in \mathcal{P}$ telle que $i < j$, alors $i \in [1, y_k[$ et si $i \in [y_t, y_{t+1}[$ alors $y_{t+1} \leq j \ \forall t \in [0, k-1]$.
Le **degré** d'un plongement en vagues est le nombre de ses pivots.

Les différents ensembles associés à un plongement en vagues d'une primitive sont des **intervalles** de positions (voir Figure 4.13.A).

Définition 28. *(Intervalles d'un plongement en vagues)*
Soit W un plongement en vagues d'une primitive $\mathcal{P}$. Les **intervalles** de W sont les intervalles semi-ouverts $I_t = [y_t, y_{t+1}[$ pour $t \in [0, k-1]$ et l'intervalle $I_k = [y_k, n]$.

La raison d'un tel nom pour cette partition vient d'une représentation graphique très utile des plongements. Celle-ci facilite la compréhension de toutes les notions et de tous les résultats que nous allons établir (voir Figure 4.13.B), notamment l'ordre sur les positions que nous allons construire y apparait presque directement. Dans cette représentation, la séquence de $\mathcal{P}$ est dessinée en forme de succession de vagues, de telle sorte que chaque intervalle soit représenté verticalement, en direction du haut ou du bas alternativement (en commençant vers le bas). Aucune position n'est dessinée sur les boucles reliant les intervalles.

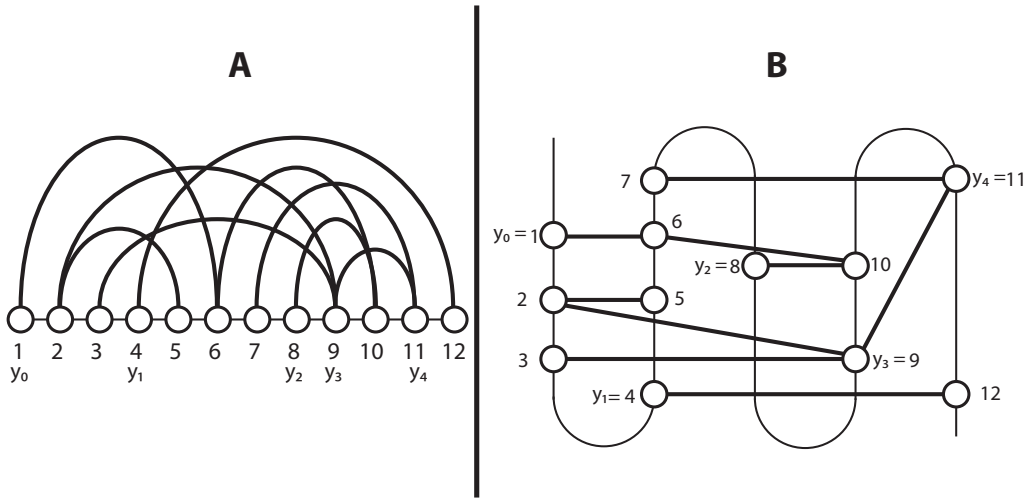


FIGURE 4.13 – ***A*** *: Une séquence arc-annotée avec un plongement en vagues associé, symbolisé par les pivots (en dessous de la séquence).* ***B*** *: Représentation en vagues de la séquence arc-annotée associée au plongement en vagues de la figure* ***A****.*

Nous définissons maintenant plusieurs notions qui vont nous être indispensables pour décider quels plongements en vagues pourront servir dans la construction d'une décomposition arborescente de la primitive. Nous voulons pouvoir décider formellement si une position est au-dessus ou au-dessous d'une autre dans la représentation en vagues, et ceci nécessite que le plongement possède une propriété que nous allons maintenant définir. Pour cela, nous allons en premier lieu définir un ordre partiel $\prec$ sur les positions. Cet ordre va ordonner les positions d'un même intervalle, en fonction de leur hauteur.

Définition 29. *($\prec$)*
Soit W un plongement en vagues d'une primitive $\mathcal{P}$, et soit I_t un intervalle de W. Pour tout $i, j \in I_t$, alors $\boldsymbol{i \prec j}$ si et seulement si $i < j$ et t est impair ou si et seulement si $i > j$ et t est pair.

Dans la Figure 4.13, nous avons par exemple, les relations : $6 \prec 7$ et $3 \prec 1$. En revanche nous ne pouvons pas comparer les positions 1 et 7 qui appartiennent à deux intervalles différents.

Définition 30. *(Position la plus haute d'un intervalle)*
Soit W un plongement en vagues d'une primitive $\mathcal{P}$, et soit I_t un intervalle de W. Alors i est la position la plus haute de I_t si et seulement si il n'existe pas de position $j \in I_t$ telle que $i \prec j$.

Dans la Figure 4.13, la position la plus haute du premier intervalle est 1 et la position la plus haute du deuxième intervalle est 7.

Définition 31. *(immédiatement en dessous)*
Soit W un plongement en vagues d'une primitive $\mathcal{P}$, et soit I_t un intervalle de W. Pour tout $i, j \in I_t$, alors i est **immédiatement en dessous** de j dans W si et seulement si $i = j - 1$ et t est impair ou si et seulement si $i = j + 1$ et t est pair, et on note $\boldsymbol{i^- = j}$. Si il n'existe pas de position immédiatement en dessous de i dans I_t alors on note $\boldsymbol{i^- = 0}$.

Dans la Figure 4.13, nous avons par exemple : $7^- = 6$, $1^- = 2$ et $9^- = 0$

Le but d'un plongement en vagues est d'ordonner (totalement) les positions d'une primitive. Nous allons étendre la relation $\prec$ pour pouvoir comparer toutes les positions entre elles. Pour cela utiliserons les interactions qui relient les intervalles entre eux. Le principe est alors de classer les interactions dans un premier temps puis d'utiliser ce classement pour ordonner les positions.

Informellement, une interaction est au-dessus d'une autre dans la représentation en vagues si l'une des positions de la première interaction est au-dessus de l'une des

positions de la deuxième interaction. Par construction des primitives, nous pouvons ainsi mettre en relation toutes les interactions. Par exemple dans la Figure 4.13, l'interaction $(6, 10)$ est au-dessus de l'interaction $(3, 9)$ car $9 \prec 10$. L'interaction $(7, 11)$ est au-dessus de l'interaction $(6, 10)$ car $6 \prec 7$ et par transitivité, l'interaction $(7, 11)$ est alors au-dessus de l'interaction $(3, 9)$. Si deux interactions se croisent nous allons avoir des contradictions (voir Figure 4.14) de telle sorte qu'un classement ne sera alors plus possible. Dans la figure 4.14, l'interaction $(3, 10)$ est au-dessus de $(2, 9)$ et aussi en dessous.

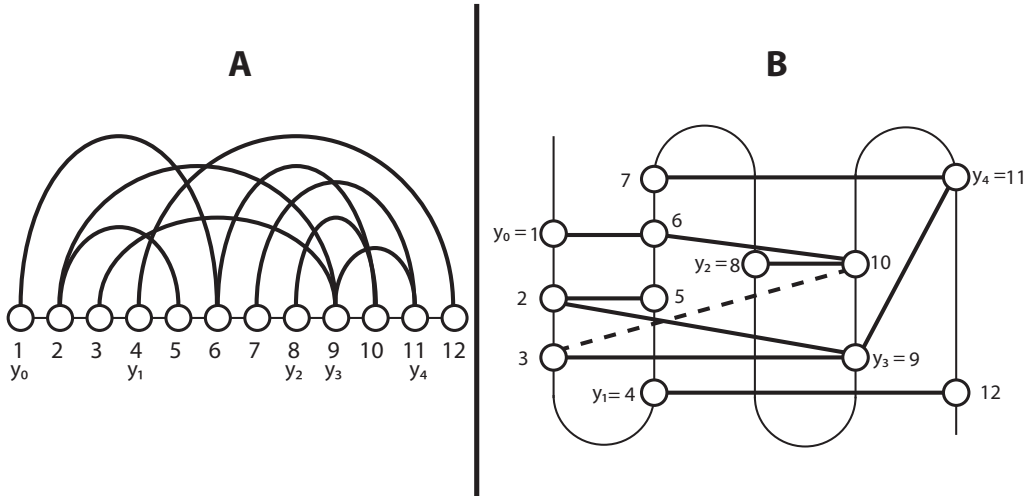


FIGURE 4.14 – ***A*** *: Une séquence arc-annotée avec un plongement en vagues associé, symbolisé par les pivots (en dessous de la séquence).* ***B*** *: Représentation en vagues de la séquence arc-annotée associée au plongement en vagues de la figure* ***A****. Le plongement contient un croisement, par exemple au niveau de l'interaction en pointillés.*

Pour pouvoir classer les interactions, il nous faudra donc impérativement construire des plongements en vagues pour lesquels la représentation en vagues ne contient pas de croisement. Pour définir formellement ceci, nous construisons le **graphe ascendant** d'un plongement en vaguess, que nous pouvons interpréter comme le graphe de la relation informelle donnée juste précédemment (voir Figure 4.15).

Définition 32. *(Graphe ascendant)*
Soit W un plongement en vaguess d'une primitive $\mathcal{P}$. Le **graphe ascendant** de $\mathcal{P}$ associé à W est le graphe orienté $G = (V_G, A_G)$ tel que $V_G = P$ (l'ensemble des interactions de $\mathcal{P}$), et A_G est l'ensemble des arcs $((i, j) \mapsto (i', j'))$ tel que i ou j est immédiatement en dessous de i' ou j' dans W.

Une contradiction dans la relation que nous tentons de définir (un croisement)

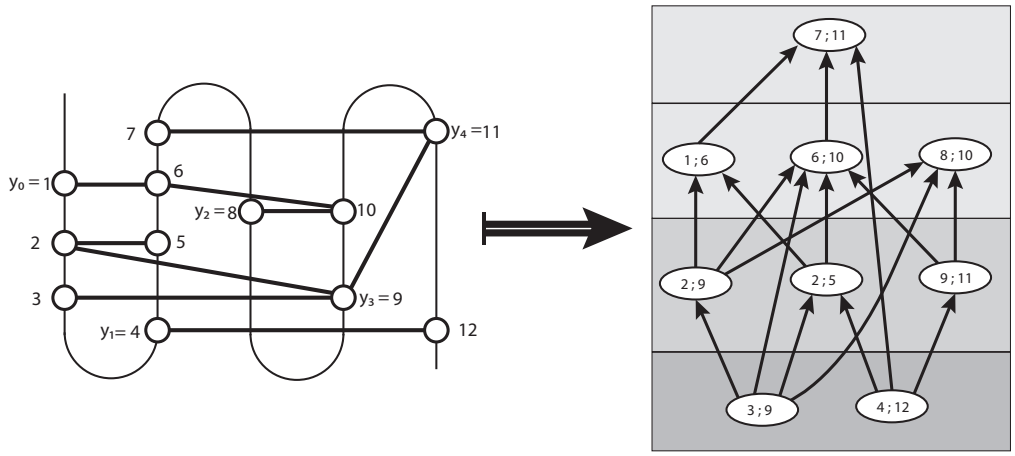


FIGURE 4.15 – *À gauche, représentation d'un plongement en vagues d'une séquence arc-annotée. À droite, le graphe ascendant correspondant.*

correspond à un cycle dans le graphe ascendant (voir Figure 4.16).

Si le graphe ascendant est acyclique nous parlerons de **plongement en vagues ordonnant**.

Définition 33. *(Plongement en vague ordonnant)*
Soit W un plongement en vaguess d'une d'une primitive $\mathcal{P}$. W est **ordonnant** si et seulement si son graphe ascendant associé est acyclique.

Nous pouvons alors attribuer un niveau pour chaque interaction (voir Figure 4.17) de telle sorte que les interactions de niveau 1 sont les interactions n'ayant pas d'autre interaction en dessous d'elles, les interactions de niveau 2 ayant seulement les interactions de niveau 1 en dessous d'elles...

Une simple adaptation de l'algorithme de Kahn [32] pour savoir si un graphe est acyclique ou non, permet alors de savoir si un plongement en vagues est ordonnant et dans ce cas, de classer ses interactions.

À partir des niveaux des interactions nous allons définir des niveaux pour les positions non-libres. Le niveau d'une position non-libre sera alors égal au plus petit niveau des interactions dans laquelle elle est impliquée.

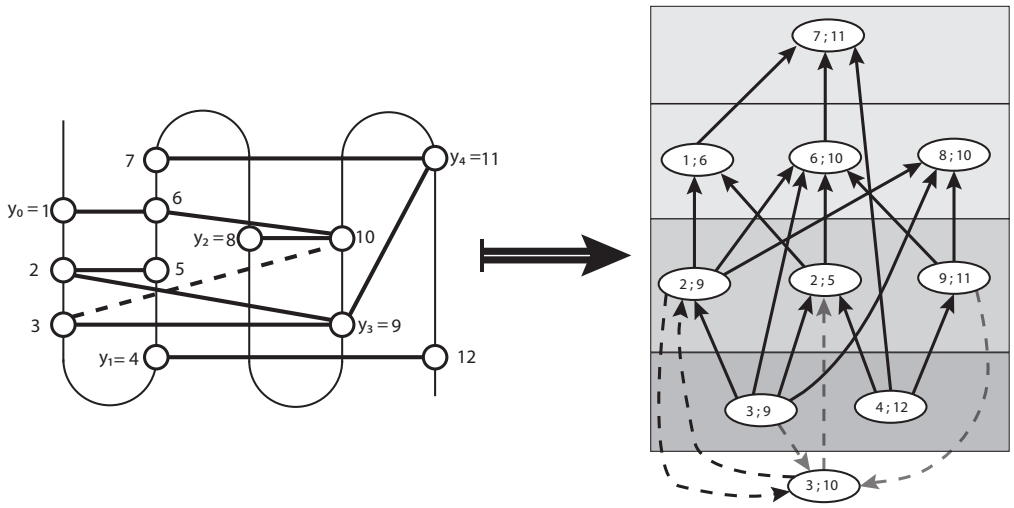


FIGURE 4.16 – *À gauche, une représentation en vagues d'une séquence arc-annotée. À droite, le graphe ascendant correspondant. Les flèches en pointillées correspondent à l'ajout de l'interaction en pointillées. Le graphe ascendant contient un cyclique (flèches pointillées non-grisées).*

Définition 34. *(Niveau d'une position)*
Soit W un plongement en vagues ordonnant d'une primitive $\mathcal{P} = (S, P)$. Le **niveau d'une position** i est défini par :
- Si i non-libre : $\mathsf{Niveau}(i) = \min_{(i,j)\in P}(\mathsf{Niveau}(i,j))$.
- Si i libre et appartient à l'intervalle I_t :
  - Si il existe au moins une position non-libre j telle que $i \prec j$, alors $\mathsf{Niveau}(i) = \min_{j,\ i\prec j}(\mathsf{Niveau}(j))$.
  - Sinon $\mathsf{Niveau}(i) = \max_{j\in\mathcal{P}}(\mathsf{Niveau}(j)) + 1$.

Enfin, à partir des niveaux des positions nous définissons un ordre total $\preccurlyeq$ sur l'ensemble des positions d'une primitive.

Définition 35. *($\preccurlyeq$)*
Soit W un plongement en vagues ordonnant d'une primitive $\mathcal{P}$, et soient i et j deux positions. Alors $i \preccurlyeq j$ si et seulement si l'une des deux conditions suivantes est vérifiée :

Si i et j appartiennent au même intervalle :
- $\mathsf{Niveau}(i) < \mathsf{Niveau}(j)$ ou $\mathsf{Niveau}(i) = \mathsf{Niveau}(j)$ et $i \prec j$.

Si i et j n'appartiennent pas au même intervalle :
- $\mathsf{Niveau}(i) < \mathsf{Niveau}(j)$ ou $\mathsf{Niveau}(i) = \mathsf{Niveau}(j)$ et $i < j$.

Algorithme 1 : Algorithme de **Niveau**

Entrée/Sortie : Un graphe orienté acyclique $G = (V_G, A_G)$
Spécification : Assigne un niveau à chaque interaction
- $L = \{v \in V_G,\ d^-(v) = 0\}$ une file $(FIFO)$.
- **Pour chaque** $v \in L$, $\mathsf{Niveau}(v) = 1$
- **Tant que** $L \neq \varnothing$:
  - Défiler v de L.
  - **Pour chaque** v' tel que $(v, v') \in A_G$:
    - supprimer (v, v') de A_G.
    - **Si** $d^-(v') = 0$:
      - Enfiler v' dans L.
      - $\mathsf{Niveau}(v') = \mathsf{Niveau}(v) + 1$.

L'exemple de la Figure 4.17 montre un classement des positions. En premier lieu nous attribuons un niveau à chaque interaction (chiffres encadrés dans la Figure 4.17.A). Puis nous en déduisons un niveau pour chaque position (Figure 4.17.B). Enfin nous classons les positions d'après la relation $\preccurlyeq$ (Figure 4.17.B).

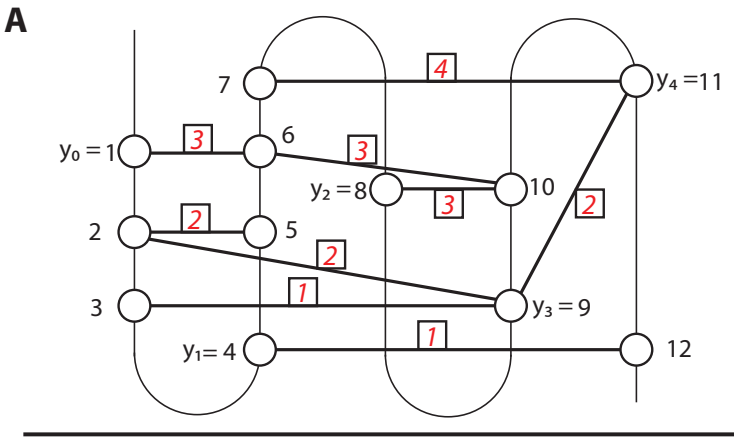
Grâce à cet ordre total, nous pouvons construire assez intuitivement une décomposition arborescente. Nous allons commencer par créer une racine, composée des positions les plus hautes de chaque intervalle. Parmi ces positions, nous recherchons alors la plus grande position selon l'ordre $\preccurlyeq$ et ajoutons à la racine la position immédiatement en dessous de la position recherchée. Pour former le sac suivant (nous rappelons que nous voulons construire une chaîne), nous reprenons les mêmes positions que le sac précédent en enlevant la position la plus grande (que nous avions recherchée à l'étape précédente). Parmi les positions restantes, nous recherchons de nouveau la position la plus grande et nous ajoutons au sac courant la position immédiatement en dessous la position recherchée. Nous continuons cette démarche jusqu'à avoir visité toutes les positions (voir Figure 4.18). L'Algorithme 2 reprend cette méthode :

Lemme 3.
L'Algorithme 2 forme une décomposition arborescente correcte.

Preuve: Lemme 3

Nous devons vérifier les 4 points de la définition d'une décomposition arborescente d'une séquence arc-annotée.

1. La racine X_0 contient au moins une position de chaque intervalle. L'algorithme s'arrête si est seulement s'il ne reste aucune position en dessous d'une des posi-



Niveau	4	3	3	3	3	2	2	2	1	1	1	1
Position	7	1	6	8	10	2	5	11	3	4	9	12

FIGURE 4.17 – ***A*** *: une représentation en vagues d'une séquence arc-annotée. Les chiffres encadrés associés aux interactions représentent leur niveau.* ***B*** *: niveau et classement des positions. La plus grande position (selon $\preccurlyeq$) est la position la plus à gauche du tableau.*

tions du sac courant. Nous pouvons donc conclure que si l'algorithme s'arrête, toutes les positions seront présentes dans la décomposition. L'algorithme ne pouvant effectuer plus de boucles (tant que) que de positions dans la primitive, il s'arrête nécessairement.

2. Supposons qu'il existe une interaction (i, j) telle que i et j ne se trouvent pas simultanément dans même sac. Comme toutes les positions sont présentes dans la décomposition, nous savons qu'une des deux positions, par exemple i, apparait avant j dans la décomposition (en partant de la racine), impliquant $j \preccurlyeq i$. Si i et j ne se trouvent pas simultanément dans même sac alors la position juste au-dessus de j, appelons la j^+, a un niveau inférieur à celui de i : $j^+ \preccurlyeq i$. Si j^+ est impliqué dans une interaction, alors le niveau de cette interaction est nécessairement strictement supérieur au niveau de l'interaction (i, j). Ceci étant vrai pour toute interaction dans laquelle j^+ peut être impliqué le niveau de j^+ est strictement supérieur à celui de i et donc on a $i \preccurlyeq j^+$. Par conséquent, le graphe ascendant du plongement en vagues W contient un cycle, ce qui est impossible. Si j^+ n'est pas impliqué dans une interaction, soit j^* la première position au-dessus de j^+ (du même intervalle) impliquée dans une interaction. Si un telle position existe, nous appliquons le même raisonnement et concluons que le niveau de j^* est est strictement supérieur à celui de i.

Algorithme 2 : Algorithme **générale de chainage**

Entrée : une séquence arc-annotée A et un plongement en vagues ordonnant W de degré k

Sortie : Une décomposition arborescente de A de largeur k

- Assignation d'un niveau pour chaque interaction en utilisant l'Algorithme 1, puis assignation d'un niveau pour chaque position.
- $X = \varnothing$ et T est un arbre vide.
- Création d'un sac X_0 composé des positions les plus hautes de chaque intervalle et faire de X_0 la racine de T.
- $l = 0$
- **Tant qu'** il existe une position $i \in X_l$ telle que $i^- \neq 0$
  - Chercher la position $p \in X_l$ de plus haut niveau et telle que $p^- \neq 0$
  - Ajouter p^- à X_l
  - Ajouter X_l à X
  - Si $l > 0$, faire de X_l le fils de X_{l-1}
  - $l = l + 1$ et $X_l = X_{l-1} - \{p\}$
- **Retourner** (X, T)

Puis, comme le niveau de j^+ est égale à celui de j^* on a par conséquent $i \preccurlyeq j^+$, impliquant une contradiction. Si une telle position j^* n'existe pas, alors le niveau de j^+ est par définition strictement supérieur à celui de i, et donc $i \preccurlyeq j^+$, impliquant encore la même une contradiction. Par conséquent il ne peut pas exister d'interaction ne se trouvant pas simultanément dans un même sac.

3. Puisque nous formons X_0 à partir des positions les plus hautes de chaque intervalle, tous les couples $(i, i+1)$, avec i appartenant à un intervalle d'indice impaire, se trouvent simultanément dans un même sac. Puis lorsque nous ajoutons une position, sa position juste au-dessus s'y trouve aussi par construction. Or comme toutes les positions sont ajoutées, tous les couples $(i, i+1)$ se trouvent simultanément dans un même sac.
4. Une position disparaissant de la décomposition ne réapparait jamais, par conséquent la dernière propriété des décompositions arborescentes de séquences arc-annotées est automatiquement vérifiée.

□

Nous pouvons alors énoncer le théorème suivant :

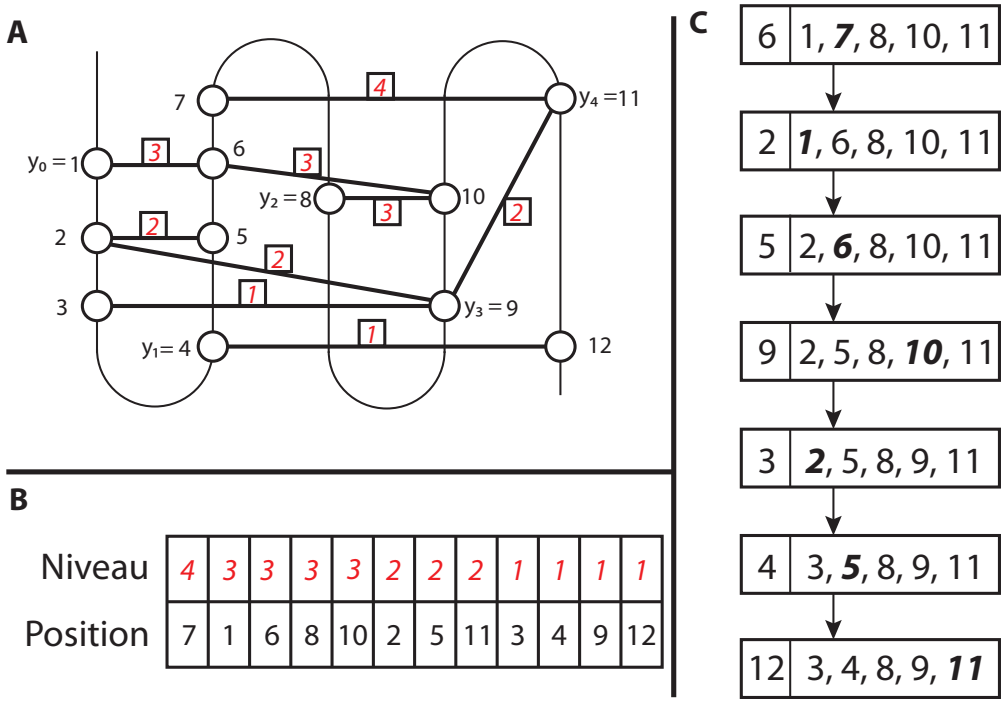


FIGURE 4.18 – ***A*** *: une séquence arc-annotée.* ***B*** *: un classement des positions de la séquence arc-annotée.* ***C*** *: une décomposition arborescente de la séquence arc-annotée obtenue via l'Algorithme 2. Chaque cadre représente un sac. La position à gauche de chaque sac est la position que nous ajoutons au sac courant par rapport à son sac parent (p^- dans l'Algorithme 2). La position en gras et italique est la position qui sera supprimée dans la construction du sac fils (p dans l'Algorithme 2).*

.

Théorème 3.
Étant donné un plongement en vagues ordonnant de degré k d'une primitive $\mathcal{P} = (S, P)$ de longueur n, il existe une décomposition arborescente lissée de $\mathcal{P}$ de largeur k qui peut être construite en temps $O(kn + |P|)$.

Preuve: Théorème 3

Nous devons montrer trois points : la largeur de la décomposition obtenue par l'Algorithme 2 est de k, elle peut être construite en temps $O(n + |P|)$ et elle est lissée.

Le premier point est évident. À un plongement en vagues de degré k nous associons k intervalles. La racine de la décomposition est donc formée de $k+1$ positions : une pour chaque intervalle (la plus haute de l'intervalle) plus la position en dessous

la plus grande position selon $\preccurlyeq$. Puis, comme nous formons un sac en ajoutant une position au sac parent après en avoir supprimée une, le nombre de positions dans chaque sac est donc le même, c'est-à-dire $k+1$.

Nous montrons maintenant que la complexité en temps (au pire) de l'Algorithme 2 est $O(n+|P|)$. La première étape de l'algorithme est d'assigner un niveau à chaque interaction via l'Algorithme 1. Cet algorithme a une complexité en temps linéaire en fonction du nombre de positions et d'interactions de la primitive, c'est-à-dire au pire $O(kn+|P|)$. Nous assignons ensuite un niveau à chaque position (peut être effectué en $O(n)$). La création de la racine coûte $k+1$ affectations plus une recherche parmi k éléments. Puis chaque création de sac coûte $k+1$ affectations et une recherche parmi k éléments. La primitive possédant n positions, nous allons créer un sac par position sauf pour les positions déjà dans la racine pour lesquels nous ne créons qu'un seul sac (la racine), nous créons donc $n-k$ sacs. Par conséquent la complexité en temps de l'algorithme est de $O(kn+|P|)$.

Pour montrer que la décomposition obtenue via l'algorithme est lissée, nous montrons que chacun de ses sacs est lissé. Soit X_l un sac de la décomposition différent de la racine X_0. En reprenant les notations de l'algorithme, nous allons montrer que X_l est lissé avec p comme position lissante et p^- comme position lissée. Nous savons par définition que p et p^- sont consécutives. Par construction p^- n'appartient pas au sac parent de X_l contrairement à p et p n'appartient pas au fils de X_l. Encore par construction, $X_l - X_r = \{p^-\}$ (avec X_r le parent de X_l). Or p et p^- appartiennent au même intervalle, donc p et p^- ne peuvent pas être en interaction. Par conséquent X_l est lissé. De la même façon nous montrons que X_0 est lissé selon le point a de la définition d'une racine lissée. □

4.4.3 Construire les plongements en vagues

Pour former des décompositions de faibles largeurs, nous devons alors construire des plongements en vagues ordonnant en tentant de minimiser leur degré. Pour cela, nous allons utiliser un algorithme glouton, qui même s'il ne garantit pas la construction d'un plongement ordonnant optimal dans l'absolu (c'est-à-dire de degré minimum), donne de très bons résultats expérimentalement une fois appliqué aux structures d'ARN.

L'algorithme que nous allons présenter construit un plongement en vagues en considérant les interactions les unes après les autres selon un ordre précis. Cet ordre trie les interactions en fonction de leur plus grande position.

Définition 36. *(Suite croissante des interactions)*
Soit $\mathcal{P} = (S, P)$ une séquence arc-annotée. Soient $(i_1, j_1), (i_2, j_2) \in P$ alors $(i_1, j_1) < (i_2, j_2)$ si et seulement si $j_1 < j_2$ ou $j_1 = j_2$ et $i_1 < i_2$. **La suite croissante des interactions** de $\mathcal{P}$ est la suite d'interactions $(U_l)_{1 \leq l}$ telle que $\forall 1 \leq l_1 < l_2 \leq |P|$, $U_{l_1} < U_{l_2}$.

Pour tout élément U_i de la suite $(U_l)_{1 \leq l}$, nous noterons $\boldsymbol{U_i^g}$ la plus petite position de l'interaction (celle de gauche) et $\boldsymbol{U_i^d}$ la plus grande (celle de droite).

Nous allons maintenant décrire le déroulement de l'algorithme que nous voulons établir sur l'exemple de la figure 4.19.

Une fois la suite croissante des interactions $(U_l)_{1 \leq l}$ construite (Figure 4.19, partie supérieure), nous initialisons le plongement en vaguess en construisant un plongement en vaguess pour la sous-structure induite uniquement par le premier élément de la suite ($U_1 = (1, 3)$). Le plongement contient alors deux pivots qui sont les positions de l'interaction $U_1 = (1, 3)$ (Figure 4.19 premier plongement). Puis nous continuons à construire le plongement en considérant l'interaction $U_2 = (1, 4)$. Le plongement construit précédemment est encore ordonnant pour la structure induite par U_1 et U_2 (Figure 4.19 deuxième plongement), nous continuons alors à construire le plongement en considérant l'interaction suivante ($U_3 = (2, 5)$). Comme le plongement construit n'est pas ordonnant pour la structure induite par U_1, U_2 et U_3 (Figure 4.19 troisième plongement) nous devons le modifier. Pour cela, nous ajoutons $U_3^d = 5$ au plongement (Figure 4.19 quatrième plongement). Dans notre exemple, la position U_3^g peut être placée dans le premier intervalle ou dans le deuxième. Nous choisissons alors systématiquement de placer U_3^g dans le deuxième intervalle de façon à remplacer le pivot (ici y_1 est remplacé par U_3^g, Figure 4.19 quatrième plongement). Nous continuons la construction du plongement en considérant l'interaction $U_4 = (3, 6)$. Comme le plongement construit n'est pas ordonnant pour la structure induite par U_1, U_2, U_3 et U_4, nous ajoutons $U_4^d = 6$ au plongement. La position $U_4^g = 3$ étant déjà placée, nous ne modifions rien d'autre. Nous continuons ainsi jusqu'à avoir considérer toutes les interactions de la suite U_l.

Les définitions suivantes reprenant les concepts de l'exemple vont permettre de simplifier l'écriture de l'algorithme.

Définition 37. *($D(U[1, i])$)*
Soit U_l la suite croissante des interactions d'une séquence arc-annotée $\mathcal{P}$. Alors $\boldsymbol{D(U[1, i])}$ est la sous-structure induite par les interactions U_1 à U_i.

Définition 38. *($\overline{\overline{\wedge}}$)*
Soit $\mathcal{P}'$ une sous-structure d'une séquence arc-annotée $\mathcal{P}$. Soit y_k un pivot d'un plongement en vagues W de $\mathcal{P}'$ et i une position de $\mathcal{P}'$. Alors $i \overline{\overline{\wedge}} y_k$ si et seulement si il n'existe pas de position de $\mathcal{P}'$ entre i et y_k (i et y_k sont consécutifs dans $\mathcal{P}'$).

Nous établissons maintenant l'algorithme.

Algorithme 3 : Algorithme de **plongement en vaguess glouton**

Entrée : une primitive $\mathcal{P} = (S, P)$ d'une séquence arc-annotée A
Sortie : W un plongement en vagues de $\mathcal{P}$
- Soit $W = \{\varnothing\}$ un plongement en vaguess vide.
- Construire $(U_l)_{1 \leq l}$ la suite croissante des interactions de $\mathcal{P}$.
- Initialisation du plongement en vaguess : ajouter $y_0 = U_1^g, y_1 = U_1^d$ à W.
- $k = 1$, le degré du plongement en vaguess courant.
- **Pour chaque** élément U_i de $(U_l)_{1 \leq l}$ (dans l'ordre) :
  - **Si** $U_i^g < y_0$
    - **Alors** $y_0 = U_i^g$
  - **Si** W n'est pas ordonnant pour $D(U[1, i])$ **alors** :
    - $y_{k+1} = U_i^d$
    - **Si** il existe un pivot y_j tel que $U_i^g \overline{\overline{\wedge}} y_j$ **alors** $y_j = U_i^g$
    - $k = k + 1$
- **Retourner** W

Lemme 4.
Soit $\mathcal{P}$ une séquence arc-annotée. L'Algorithme 3 construit un plongement en vaguess ordonnant de $\mathcal{P}$.

Preuve: lemme 4

Pour montrer ce lemme, il suffit de montrer qu'à chaque fin d'itération i de la boucle **Pour**, W est ordonnant pour $D(U[1 \dots i])$. Nous allons montrer cette propriété par récurrence.
Pour $i = 1$, W est ordonnant pour $D(U[1])$. Supposons que pour une itération i nous ayons construit un plongement en vaguess ordonnant W pour $D(U[1 \dots i])$ et soit G son graphe ascendant. Si W est ordonnant pour $D(U[1 \dots i+1])$ alors la propriété est montrée. Sinon, nous rajoutons alors l'élément U_{i+1}^d dans le plongement en vaguess W. Ceci suffit à rendre le plongement ordonnant. Pour le vérifier, nous vérifions simplement que le graphe ascendant G' du nouveau plongement W est acyclique. Pour construire G', nous partons du graphe G (acyclique par hypothèse)

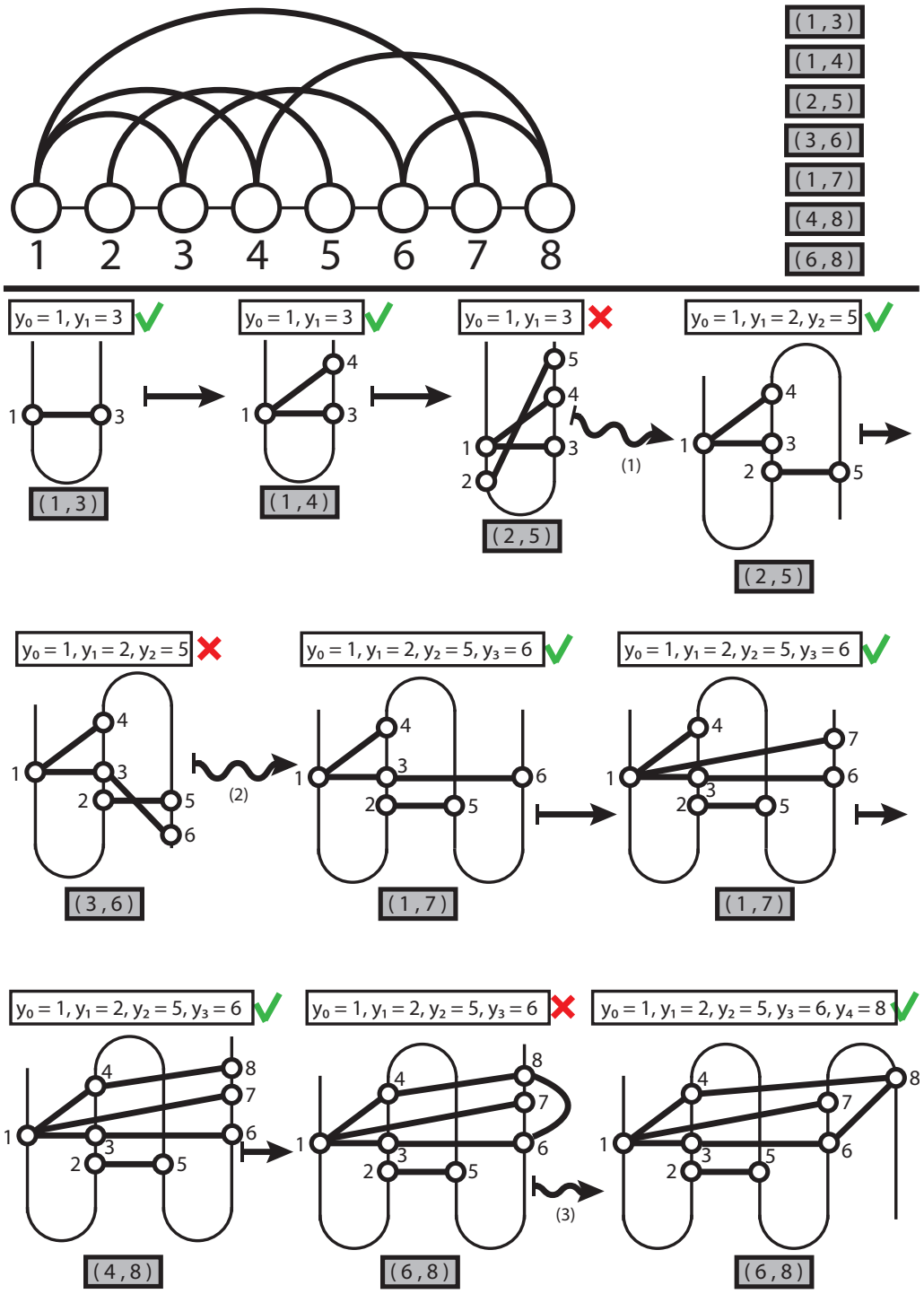


FIGURE 4.19 – *Illustration de l'Algorithme 3. Les interactions de la structure sont classé par ordre croissant de haut en bas (figure du haut). Les croix indiques que la prise en compte de l'interaction nécessite l'ajout d'un pivot dans l'Algorithme 3.*

et le modifions de façon à prendre en compte la nouvelle interaction U_{i+1}. Pour rappel, les arcs entrants et sortants d'un nœud du graphe ascendant sont définis par les interactions immédiatement au-dessus ou au-dessous des positions du nœud. Par conséquent, puisque la position U^d_{i+1} n'a pas de position immédiatement au-dessus ou au-dessous (puisqu'elle est seule dans son intervalle nouvellement définit), les changements à opérer sont uniquement provoqués par U^g_{i+1}. Supposons que U^g_{i+1} se positionne entre deux positions j_1 et j_2 de $D(U[1 \dots i])$ telles que $j_1 \prec U^g_{i+1} \prec j_2$. Chaque arc supprimé entre une interaction J_1 et J_2 (où $j_1 \in J_1$ et $j_2 \in J_2$) est remplacé par un arc entrant entre J_1 et U_{i+1} et un arc sortant entre U_{i+1} et J_2. On obtient alors le graphe G'. Si la transformation induit un cycle c alors il est nécessairement de la forme $c = \{J_1, U^g_{i+1}, J_2, \dots J_1\}$. Or dans ce cas le chemin $\{J_1, J_2, \dots, J_1\}$ de G (où $\dots$ représente le même sous-chemin) est par construction un cycle, ce qui n'est pas possible. Si U^g_{i+1} se positionne en dernière position d'un intervalle I_{j-1} (tout en haut des vagues impaires, et tout en bas des vagues paires dans la représentation graphique en vague) la preuve est immédiate (nous ne rajoutons alors qu'un nœud qui est soit une source soit un évier, et ne créons ainsi aucun cycle).

Dans ce dernier cas, nous pouvons placer U^g_{i+1} tout au début de I_j, c'est à dire remplacer la valeur du pivot y_j par la valeur de U^g_{i+1} et montrons de la même façon que la transformation ne crée pas de cycle.

En conclusion, G' est cyclique, et donc le nouveau plongement W est ordonnant pour $D(U[1 \dots i+1])$. □

Définition 39. *(Degré en vague glouton d'une extension de primitive)* Soit $\mathcal{P} = (S, P)$ une extension de primitive de longueur n d'une séquence arc-annotée. Alors le **degré en vague glouton** de $\mathcal{P}$, noté $\mathcal{G}(\mathcal{P})$ est le degré du plongement en vaguess de $\mathcal{P}$ obtenu par l'Algorithme 3.

Lemme 5.
Soit $\mathcal{P}$ une séquence arc-annotée. L'Algorithme 3 calcule le degré en vague glouton de $\mathcal{P}$ en $O(|P|^2)$ (au pire).

Preuve: lemme 5

Pour montrer ce lemme il suffit de montrer que la complexité de l'Algorithme 3 est en $O(|P|^2)$.

L'étape d'initialisation s'effectue en $O(|P|^2)$ au pire pour le trie des interactions.

La boucle effectue un nombre d'itérations en $O(|P|)$, et tester si le plongement est ordonnant peut s'effectuer en $O(|P|)$. Donc la complexité finale de l'algorithme est en $O(|P|^2)$. □

Nous avons maintenant un moyen de créer des plongements en vagues, et donc de décomposer les primitives.

4.4.4 Assemblage des décompositions des structures primitives

Après avoir décomposée chacune des extensions des primitives nous devons bien évidement assembler toutes ces décompositions arborescentes ensemble. Pour effectuer cette étape, nous allons simplement nous appuyer sur les relations d'encapsulation entre les primitives (voir Figure 4.20). Ainsi, nous allons connecter ensemble deux décompositions de deux extensions de primitives si et seulement si l'une des deux primitives est directement encapsulée dans l'autre (avec un cas particulier de connexion pour les primitives de profondeur 0).

Étant donné que l'union des extensions des primitives recouvre toutes les positions, toutes les interactions, et toutes les paires de positions consécutives, nous devons uniquement porter notre attention sur la dernière contrainte des décompositions arborescentes afin de former une décomposition finale correcte. Lors du découpage en primitive, nous avons définie les extensions afin de faciliter leur assemblage. Soient P_t et $P_{t'}$ deux extensions de primitives telles que la primitive associée à $P_{t'}$ est directement encapsulée dans celle associée à P_t. Par construction, l'intersection de P_t et $P_{t'}$ est égale au bords de $P_{t'}$, et toujours par construction ces deux bords sont consécutifs dans P_t. Si nous voulons connecter P_t et $P_{t'}$, c'est-à-dire connecter un sac de P_t à un sac de $P_{t'}$, il nous sera alors aisé de le faire en utilisant un des sacs de P_t contenant les bords de $P_{t'}$. Appelons ce sac χ. Nous avons ensuite deux types de situations.

La première, la plus facile, est la situation dans laquelle il existe un sac χ' de $P_{t'}$ contenant les deux bords de $P_{t'}$. Cette situation correspond aux cas où le degré du plongement en vague est pair, ou le cas particulier où le dernier intervalle du plongement en vague est réduit à une seule position, celle du bord droit de $P_{t'}$. Pour connecter P_t et $P_{t'}$ il nous suffira alors de connecter χ et χ' (voir Figure 4.20).

Dans la deuxième situation, il n'existe pas de sac de la décomposition de $P_{t'}$ contenant les deux bords de $P_{t'}$. Ce cas apparait lorsque le degré du plongement en vagues de $P_{t'}$ est impair et que le dernier intervalle n'est pas réduit à un seul élément (donc la dernière position de $P_{t'}$ n'est pas la plus haute position de son intervalle). Dans ces cas, nous allons simplement choisir un sac χ' de la décomposition de $P_{t'}$ contenant par exemple le bord gauche de $P_{t'}$. Le bord droit de $P_{t'}$ se trouvant alors dans d'autres sacs. Nous devons alors ajouter le bord droit de $P_{t'}$ dans tous les sacs

se trouvant sur le chemin entre χ' et le premier sac contenant le bord droit de $P_{t'}$ (voir Figure 4.20).

Nous devons aussi gérer en premier lieu l'assemblage des décompositions des extensions des primitives de profondeur 0. Soient P_t et $P_{t'}$ deux extensions de primitives de profondeur 0 et telles que P_t et $P_{t'}$ soient consécutives. Par construction P_t contient le bord gauche de $P_{t'}$, unique position de l'intersection entre P_t et $P_{t'}$. L'assemblage est alors évident, il nous suffit de sélectionner un sac de la décomposition de P_t contenant le bord gauche de $P_{t'}$ et le connecter à un sac de la décomposition de $P_{t'}$ contenant le bord gauche de $P_{t'}$ (voir Figure 4.20)). L'Algorithme 4 est basé sur ce raisonnement et la discussion ci-dessus prouve qu'il forme une décomposition arborescente correcte.

Algorithme 4 : Algorithme de **chainage arborescent**

Entrée : L'ensemble des extensions des primitives d'une structure $A = (S_A, P_A)$

Sortie : Une décomposition arborescente de A

- Construire une décomposition de toutes les extension des primitives en utilisant l'Algorithme 2.
- **Pour chaque** extension de primitive P_t de profondeur 0 :
  - Soit (X^t, T^t) la décomposition arborescente de P_t et (X'^t, T'^t) celle de l'extension de la primitive immédiatement à droite $P_{t'}$.
  - Soit χ le sac de (X^t, T^t) contenant le bord gauche de $P_{t'}$.
  - Connecter χ à la racine de (X'^t, T'^t).
- **Pour chaque** extension de primitive $P_{t'}$ de profondeur $i > 0$ en ordre croissant :
  - Soit (X'^t, T'^t) la décomposition arborescente $P_{t'}$ et (X^t, T^t) celle de l'extension de primitive P_t dans laquelle $P_{t'}$ est directement encapsulée.
  - Soit χ le sac premier sac de (X^t, T^t) (en partant de la racine) contenant les bords de $P_{t'}$.
  - Connecter la racine de (X'^t, T'^t) à χ.
  - Ajouter le bord droit de $P_{t'}$ à tous les sacs de (X'^t, T'^t) de la racine jusqu'au premier sac contenant de bord droit de $P_{t'}$.
- La racine de la décomposition de l'extension de la primitive de profondeur 0 contenant la position 1 devient la racine de la décomposition finale.

Si une décomposition d'extension de primitive ne possède qu'un sac χ', alors nous vérifions qu'il ne soit pas totalement inclut dans son père (dans la décomposition que nous formons). Sinon nous le supprimons de la décomposition (par contraction, voir Figure 4.20).

De cette façon, la largeur de la décomposition finale est donc nécessairement inférieure ou égale à la plus grande largeur des décompositions de ces primitives plus 1. Lorsque nous avons décri l'algorithme de décomposition arborescente des extensions des primitives, nous avons établi qu'il formait des décompositions arborescentes lissées. Ceci est vrai si l'on considère que dans une extension de primitive

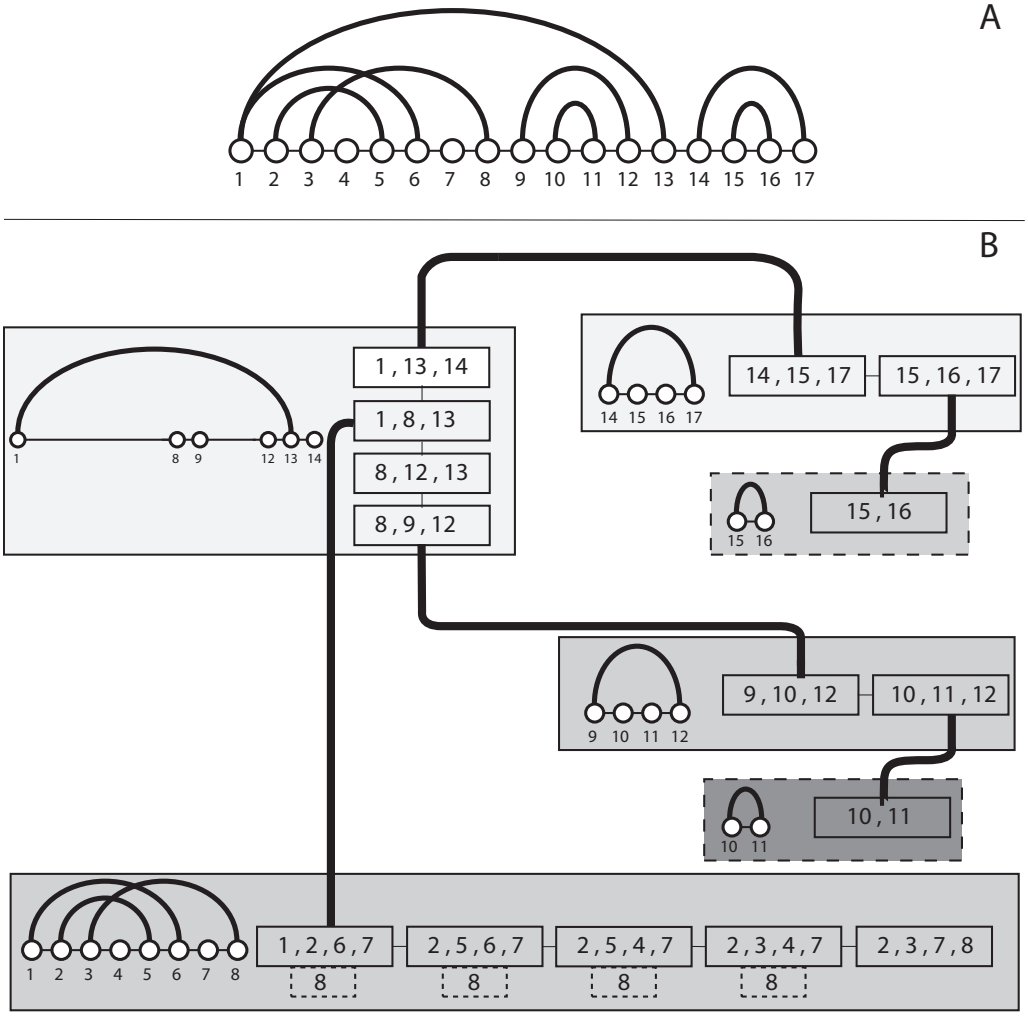


FIGURE 4.20 – *Illustration de l'Algorithme 4. Les sacs en pointillés sont des sacs inclus dans leur père que l'on doit supprimer. Les positions* 8 *encadrées en pointillés symbolisent l'ajout de la position dans les sacs correspondant, nécessaire pour former une décomposition correcte.*

deux positions sont consécutives si il n'existe pas d'autres (dans l' extension de primitive) entre les deux. Par exemple dans l'extension de primitive P_1 de la 4.12, les positions 1 et 8 sont considérées comme consécutives, alors qu'en réalité elles ne le sont pas (dans la structure totale). Donc utilisées dans la construction de la décomposition finale, les décompositions des extensions de primitives ne sont plus nécessairement lissées. Autrement dit, si une primitive encapsule immédiatement plusieurs (strictement supérieure à une) autres primitives, alors la décomposition de son extension par l'Algorithme 2 ne sera pas lissée. Plus précisément, à chaque primitive qu'elle encapsule immédiatement nous associerons un sac non-lissé (si elle encapsule immédiatement au moins deux primitives).

Définition 40. *(Structures primitives sœurs)*
Soient $\mathcal{P}_1$ et $\mathcal{P}_2$ deux primitives d'une séquence arc-annotée A. Alors $\mathcal{P}_1$ et $\mathcal{P}_2$ sont des **structures primitives sœurs** si et seulement si elles sont immédiatement encapsulées par une même autre structure primitive.

Définition 41. *(Degré d'embranchement)*
Soit $A = (S_A, P_A)$ une séquence arc-annotée. Alors le **degré d'embranchement** de A, noté $\kappa(A)$ est le nombre de structure primitives ayant une sœur.

si une primitive encapsule directement κ autres primitives alors la décomposition de son extension par l'Algorithme 2 est κ-faiblement-lissée.

Théorème 4.
Soit A une séquence arc-annotée de longueur n. Étant donné des plongements en vagues ordonnant de degrés au plus k pour chaque extensions des structures primitives de A, il existe une décomposition arborescente de largeur au plus $k+1$ et κ-faiblement lisse où κ est le nombre de primitives qui encapsulent au moins deux autres primitives. De plus cette décomposition arborescente se construit en temps $O(k \cdot n)$.

Preuve: Théorème 4

Il reste à montrer que la complexité de l'Algorithme 4 est en $O(k \cdot n)$. En considérant que la recherche d'éléments dans un sac de taille k coûte $O(k)$, alors puisque au pire nous effectuons une recherche dans tous les sacs, la recherche des bords des primitives se fait au pire en $O(k \cdot n)$ (le nombre de sacs étant supposé en $O(n)$). Les ajouts des bords droits des primitives s'effectue au pire en $O(n)$ pour la totalité d'entre elles. La complexité finales est donc en $O(k \cdot n)$. □

4.5 Résultats finals de complexité

Dans la section 2 et 3 de ce chapitre nous avons établis des formules de récurrence permettant de résoudre le problème d'alignement structure-séquence à partir d'une décomposition arborescente de la structure. Dans la section 4 nous avons construit des algorithmes permettant de diviser la structure en primitives, de déterminer des plongements en vagues pour chacune des (extension des) primitives, de construire une décomposition arborescente de primitives à partir d'un plongement en vagues

et enfin d'assembler toutes les décompositions des primitives pour former une décomposition finale de la structure.

En résumer voici l'algorithme final découlant de tous les algorithmes que nous avons établis :

Algorithme 5 : Algorithme général et paramétré d'**alignement structure-séquence**

Entrée : Une séquence arc-annotée $A = (S_A, P_A)$ et une séquence S_B
Sortie : Une alignement entre A et B.
(1) Division de A en structures primitives.
(2) Pour chaque extension de primitive $\mathcal{P}$ de A :
(2′) Construire le plongement en vague $W_{\mathcal{P}}$ selon l'Algorithme 3.
(2″) Construire la décomposition arborescente $(X^{\mathcal{P}}, T^{\mathcal{P}})$ de $\mathcal{P}$ selon l'Algorithme 2 en utilisant le plongement en vague $W_{\mathcal{P}}$.
(3) Construire la décomposition arborescente (X, T) de A par assemblage des décompositions $(X^{\mathcal{P}}, T^{\mathcal{P}})$ selon l'Algorithme 4.
(4) Alignement de (X, T) avec S_B, où κ est le nombre de primitives de A qui encapsulent immédiatement au moins deux autres primitives, selon les équations de récurrence 1,2,3.

Nous pouvons alors établir le résultat final suivant.

Définition 42. *(Degré en vague glouton d'une séquence arc-annotée)*
Soit $A = (S_A, P_A)$ une une séquence arc-annotée. Alors le **degré en vague glouton** de A, noté $\Gamma(A)$ est le plus grand degré glouton de ses extensions de primitives.

Théorème 5.
Soient une séquence arc-annotée $A = (S_A, P_A)$ de longueur n et S_B une séquence de longueur m. Alors le problème d'alignement structure-séquence peut être résolu en temps $O(|P_A|^2 + \Gamma(A) \cdot n + n \cdot m^{\Gamma(A)+1} + \kappa(A) \cdot m^{\Gamma(A)+2})$ par l'Algorithme 5.

Preuve: Théorème 5

Par application de tous les théorèmes du chapitre 4 nous pouvons déduire la complexité de chacune des étapes :

L'étape **(1)** s'effectue en $O(n + |P_A|)$ par recherche des composantes connexes d'un graphe.

L'ensemble des étapes **(2′)** s'effectue au pire en $O(|P_A|^2)$. L'ensemble des étapes **(2″)** s'effectue au pire en $O(\Gamma(A) \cdot n + |P_A|)$. Par conséquent l'étape **(2)** s'effectue au pire en $O(|P_A|^2 + \Gamma(A) \cdot n)$.

L'étape **(3)** s'effectue au pire en $O(\Gamma(A) \cdot n)$.

L'étape **(4)** s'effectue au pire en $O(n \cdot m^{\Gamma(A)+1} + \kappa(A) \cdot m^{\Gamma(A)+2})$.

Donc la complexité finale est de $O(|P_A|^2 + \Gamma(A) \cdot n + n \cdot m^{\Gamma(A)+1} + \kappa(A) \cdot m^{\Gamma(A)+2})$.
□

4.6 Comparatifs

Nous allons maintenant comparer la complexité de notre approche à celle des méthodes décrites dans le chapitre 3 pour la résolution de l'alignement structure séquence, d'une structure $A = (S_A, P_A)$ de longueur n et une séquence S_B de longueur m. Pour chaque comparatifs, nous analysons étape par étape la complexité de l'Algorithme 5.

4.6.1 Comparatifs avec les méthodes existantes

Structures secondaires

La complexité en temps de l'algorithme de Jiang est en $O(l \cdot m^3 + n \cdot m^2)$ où l est le nombre d'hélices impliquées dans des jonctions.

Regardons maintenant la complexité de l'Algorithme 5 étape par étape pour une structure secondaire.

Puisque les structures secondaires ne considèrent que les interactions canoniques, le nombre d'interactions maximum d'une structure secondaire est de $\lfloor \frac{n}{2} \rfloor$. Par conséquent si A est une structure secondaire alors $|P(A)| = O(n)$.

(1) : L'étape **(1)** s'effectue alors en $O(n)$.

(2) : Étant donné qu'aucune interaction se croisent, les primitives sont réduites à une seule interaction. Donc quelque soit l'extension de primitive $\mathcal{P}$ de A, $\mathcal{G}(\mathcal{P}) = 2$, et le plongement en vagues associé se construit en $O(1)$. De même, les décompositions des extensions de primitives se calculent chacune en $O(1)$ (donc $O(n)$ pour la totalité). L'étape **(2)** s'effectue en $O(n)$.

(3) : L'étape **(3)** s'effectue alors en $O(n)$.

(4) : Comme tous les plongements en vagues des extensions de primitives que nous créons sont de degré égale à 2 (et donc pair), alors nous n'avons pas besoin d'ajouter des positions dans les décompositions arborescentes associées lors de leur assemblage. En conséquence, la décomposition arborescente que nous construisons pour A est de largeur $\Gamma(A) = 3$. Dans une structure secondaire,

une primitive possède une sœur si elle est la première interaction d'une hélice impliquée dans une jonction. Donc $\kappa(A) = O(l)$. L'étape **(4)** s'effectue alors en $O(l \cdot m^3 + n \cdot m^2)$.

Nous pouvons en déduire la propriété suivante.

Propriété 1.
La complexité de l'Algorithme 5 pour les structures secondaires est en $O(l \cdot m^3 + n \cdot m^2)$.

Pseudo-nœuds standard

La complexité en temps de l'algorithme de Han est en $O(n \cdot m^k)$ où k est le degré du pseudo-nœud standard à aligner (ici A).

Regardons maintenant la complexité de l'Algorithme 5 étape par étape pour un pseudo-nœud standard de degré k.

Pour les même raison que pour les structures secondaires, si A est une structure secondaire avec pseudo-nœud alors $|P(A)| = O(n)$.

(1) : L'étape **(1)** s'effectue alors en $O(n)$.

(2) : Un pseudo-nœud standard de degré k ne possède qu'une seule structure primitive : lui même. L'étape **(2)** s'effectue alors en $O(n^2)$.

(3) : L'étape **(3)** n'existe pas étant donné qu'il n'y a qu'une primitive.

(4) : Par le même argument (une seule primitive) nous pouvons en déduire que la décomposition obtenue par l'étape précédente est lissée.

Nous montrons maintenant par récurrence sur k que $\Gamma(A) = k$, c'est-à-dire que le degré du plongement construit à l'étape **(2′)** est égale à k.

En reprenant les notations de la définition des pseudo-nœuds standard, la structure est partionnée en sous-structures $A_t, t \in [1, k-1]$ de type CHAIN (Figure 4.21). Par définition du tri de l'étape **(2′)**, pour deux indices $t < t'$, les interactions de A_t sont classées avant celles de $A_{t'}$. Donc l'Algorithme 3 considère les sous-structures les une après les autres par ordre croissant de leur indice.

Le plongement en vague de la sous-structure A_1 est de degré 2 car A_1 est de type CHAIN(sans croisement, voir Figure 4.21).

Soit t un indice strictement inférieur à $k-1$. Supposons que le plongement en vagues glouton W de la structure A^t formée des sous-structure de A_1 à A_t soit de degré $t+1$. Soient (i, j) la première interaction (selon le tri) de A_{t+1} et I_t est l'intervalle d'indice t du plongement en vagues W de A^t (avec $i < j$). Suivant la situation, la position i peut être placée dans l'intervalle I_t (Figure 4.22) ou dans celui d'avant (Figure 4.21). Si il existe $i' \in I_t$ telle que $i' < i$ alors nécessairement $i \in I_t$ (Figure 4.22). Par conséquence, W n'est pas ordonnant pour la structure A^t a laquelle nous ajoutons l'interaction (i, j). Sinon, quelque soit l'interaction (i', j') de A_t, nous avons $i' < i < j' < j$

(Figure 4.21). Par conséquence, W n'est pas ordonnant pour la structure A^t a laquelle nous ajoutons l'interaction (i, j). Nous en déduisons qu'il est nécessaire d'ajouter j au plongement en vague. La prise en compte des autres positions de A_{t+1} ne nécessite pas d'augmenter le degré de W puisque que A_{t+1} est de type CHAIN. Nous en déduisons que le plongement en vagues glouton de A^{t+1} est de $t + 2$.
Par récurrence, nous avons montré que le degré du plongement en vagues glouton de A est égal à k (car $A^{k-1} = A$).
L'étape **(4)** s'effectue alors en $O(n \cdot m^k)$.

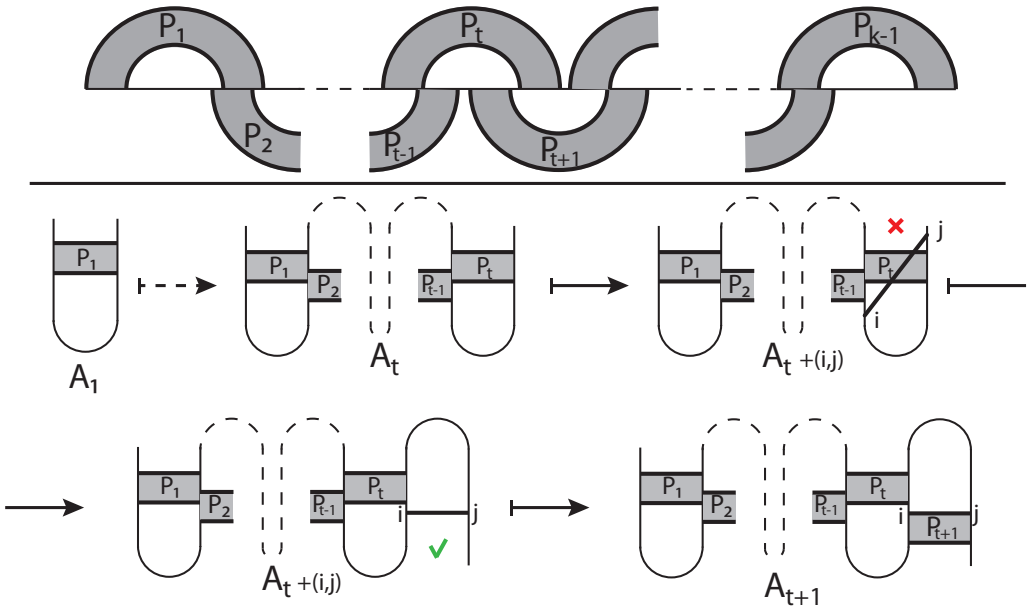


FIGURE 4.21 – *Illustration de l'Algorithme 3 sur les pseudo-nœuds standard. Figure du haut : Les interactions sont regroupées en sous-ensemble P_t, induisant chacun une sous-structure A_t de type* CHAIN. *Figure du bas : le plongement en vagues de A_1 est toujours de degré 2. Nous prenons en compte A_{t+1} dans la construction du plongement en ajoutant un degré au plongement.*

Nous pouvons en déduire la propriété suivante.

Propriété 2.
La complexité de l'Algorithme 5 pour les pseudo-nœuds standard est en $O(n^2 + n \cdot m^k)$.

Comparer à la complexité de l'algorithme de Han, nous avons un terme n^2 supplémentaire. Cependant, dans absolument tous les cas pratiques $n = O(m)$ (la structure et la séquence ont le même ordre de grandeur). Comme le degré des pseudo-nœuds standard est supérieure ou égale à 3 (pour l'algorithme de Han), la complexité de

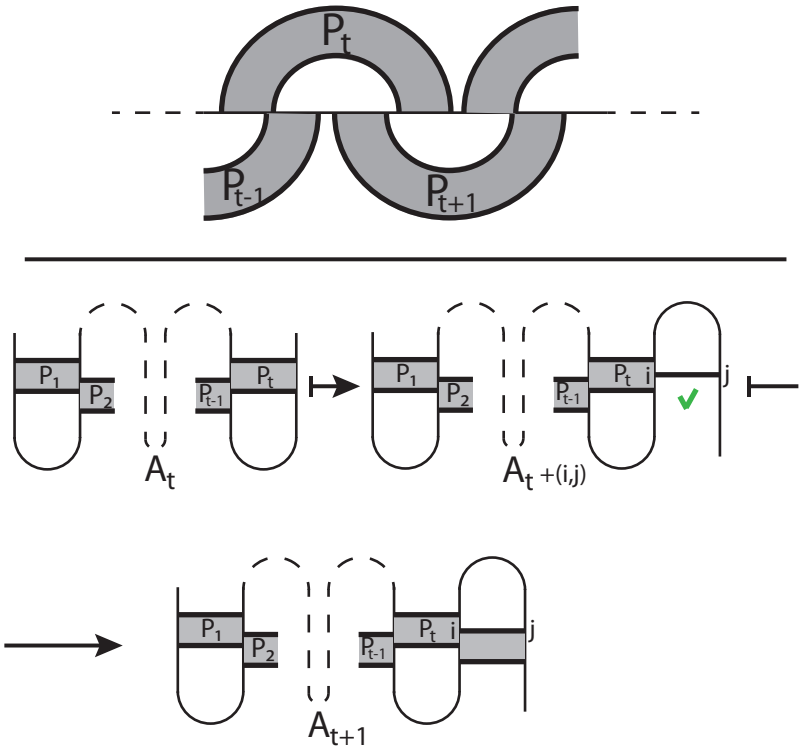


FIGURE 4.22 – *Autre illustration de l'Algorithme 3 sur les pseudo-nœuds standard. La position i de la première interaction de A_{t+1} se positionne dans l'intervalle I_t. Dans ce cas nous devons nécessairement faire un nouvel intervalle.*

l'Algorithme 5 n'est pas affectée par ce terme n^2. Notons aussi que pour l'algorithme de Han, les interactions sont supposées déjà triées ce qui permet de reconnaitre un pseudo-nœud standard en temps linéaire (et fait disparaitre le terme n^2 de la complexité de l'algorithme de Han).

Pseudo-nœuds simple non-standard

La complexité en temps de l'algorithme de Wong est en $O(n \cdot m^{k+1})$ où k est le degré du pseudo-nœud simple non-standard à aligner (ici A).

Regardons maintenant la complexité de l'Algorithme 5 étape par étape pour un pseudo-nœud simple non-standard de degré k.

A est une structure secondaire avec pseudo-nœuds donc $|P(A)| = O(n)$.

(1) : L'étape **(1)** s'effectue alors en $O(n)$.

(2) : Un pseudo-nœud simple non-standard de degré k ne possède qu'une seule structure primitive : lui même. L'étape **(2)** s'effectue alors en $O(n^2)$.

(3) : L'étape **(3)** n'existe pas étant donné qu'il n'y a qu'une primitive.

(4) : Par le même argument (une seule primitive) nous pouvons en déduire que la décomposition obtenue par l'étape précédente est lissée.

Nous montrons maintenant $\Gamma(A) = k + 1$ (pour le type I), c'est-à-dire que le degré du plongement construit à l'étape **(2′)** est de k.
En reprenant les notations de la définition des pseudo-nœuds simples non-standard, la structure est partionnée en sous-structures $A_t, t \in [1, k-2]$ et une dernière partition A', induites par les sous-ensembles d'interactions P_t et P' (Figure 4.23). Par définition du tri de l'étape **(2′)**, pour deux indices $t < t'$, les interactions de A_t sont classées avant celles de $A_{t'}$ et A' est considérée en dernière. Donc l'Algorithme 3 considère les sous-structures les une après les autres par ordre croissant de leur indice et termine par A'.
Soit A^t, $t < k-2$ la sous-structure formée des sous-structures de A_1 à A_t. Le degré glouton de A_t est de $t + 1$ car A^t est un pseudo-nœud standard.
Nous montrons que la prise en compte de A' par l'Algorithme 3 augmente le degré d'au plus 2. Deux cas sont à considérer en fonction de k (le degré) et τ, l'indice de l'intervalle des positions de gauche des interactions de P' (voir définition des pseudo-nœuds simple non-standard). Si $k - \tau$ est pair (voir Figure 4.23) alors l'ajout de la première interaction de A' peut engendrer un croisement. Dans ce cas nous devons ajouter un pivot et les autres interactions de A' ne nécessitent pas d'augmenter le degré car A' est de type CHAIN (Figure 4.23). Sinon, la deuxième interaction de A' engendrera un croisement (car A' est de type CHAIN), et nous devons alors augmenter le degré pour prendre la deuxième interaction et se retrouver alors dans la même configuration que précédemment. Si $k - \tau$ est impair (voir Figure 4.23), l'ajout de la première interaction peut engendrer un croisement. Dans ce cas nous devons ajouter un pivot. La deuxième interaction engendrera un croisement (car A' est de type CHAIN), et nous devons alors augmenter le degré pour prendre la deuxième interaction dans le plongement. Les autres interactions de A' ne nécessitent pas d'augmenter le degré car A' est de type CHAIN (Figure 4.23). Si la première interaction n'engendre pas de croisement, n'importe quelles autres interactions de A' pourra par la suite engendrer un croisement lors de sa prise en compte. Nous nous retrouverons alors dans le cas précédent. Il est néanmoins possible qu'aucune interaction engendre un croisement (voir Figure 4.23).
Dans tous les cas ajoutons au pire deux pivots au plongement de A^{k-2} pour prendre en compte A', donc au pire le degré du plongement en vagues glouton de A est égal à $k + 1$. Dans certains cas nous ajoutons un seul ou aucun pivot, le degré du plongement en vagues glouton de A est alors égal à $k - 1$ ou k.
L'étape **(4)** s'effectue alors en $O(n \cdot m^{k+1})$ au pire.

La démonstration pour les type II est très proche. Les même sous-cas sont en prendre en compte, et les conclusions sont identiques. Nous pouvons en déduire la propriété suivante.

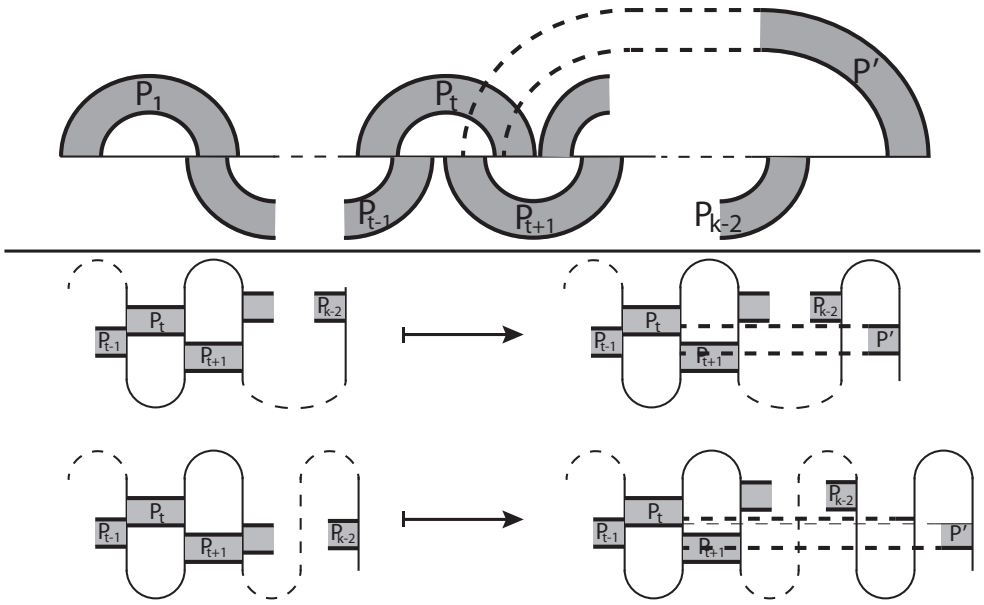


FIGURE 4.23 – *Illustration de l'Algorithme 3 sur les pseudo-nœuds simple non-standard. Figure du haut : Les interactions sont regroupées en sous-ensemble P_t et le sous ensemble P' enduisant des sous-structures A_t et A' (pour P') toutes de type* CHAIN. *Figures du bas : À partir d'un plongement en vagues d'un pseudo-nœud standard constitué des interactions de P_1 à P_{k-2}, nous construisons le plongement final. En fonction de la parité de $k-\tau$ nous avons plusieurs deux premiers sous-cas possibles. Le premier du haut cas correspond à $k-\tau$ paire et celui du bas $k-\tau$ impair. Les configurations des sous-cas présentés sont les configurations au pire.*

Propriété 3.
La complexité de l'Algorithme 5 pour les pseudo-nœuds simple non-standard est en $O(n^2 + n \cdot m^{k+1})$.

Ceci est la complexité au pire. Pour certains cas l'alignement à partir de la décomposition arborescente peut s'effectuer en $O(n \cdot m^k)$ (cas de l'ajout d'un seul pivot pour la prise en compte de A', voir ci-dessus). Ceci ne constitue pas une amélioration de l'algorithme de Wong qui possède la même complexité pour ces cas particuliers. En revanche, les cas où l'alignement à partir de la décomposition arborescente peut s'effectuer en $O(n \cdot m^{k-1})$ est une amélioration. Pour ces cas l'algorithme de Wong effectue l'alignement en $O(n \cdot m^{k+1})$. L'amélioration est non-négligeable puisque que dans la pratique le degré k est borné (dans la majorité des cas) par 5.

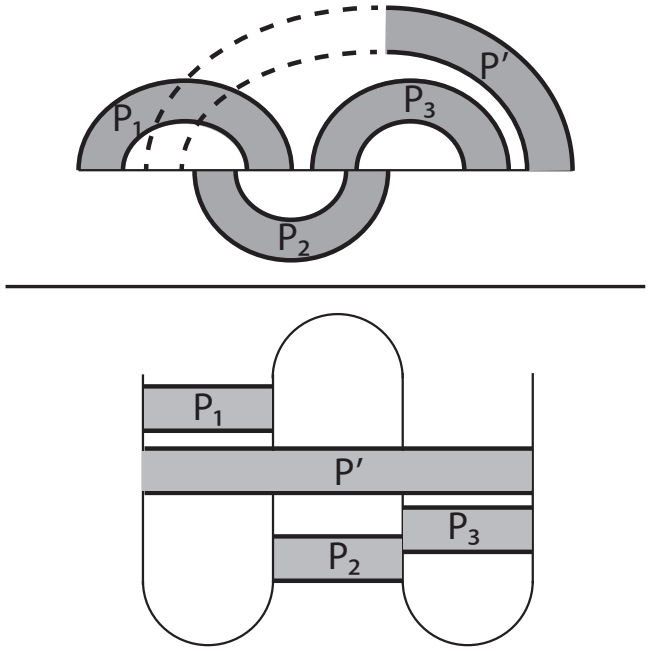


FIGURE 4.24 – *Autre illustration de l'Algorithme 3 sur les pseudo-nœuds simple non-standard. Dans ce cas particuliers, il n'est pas nécessaire d'ajouter un pivot pour prendre en compte P'.*

Triples hélices standard

La complexité en temps de l'algorithme de Wong pour l'alignement de triples hélices standard est en $O(n \cdot m^3)$. Les triples hélices standard sont une généralisation de pseudo-nœuds standard de degré 3 dans lesquels certaines interactions multiples sont considérées. En redéfinissant le type CHAIN de façon à pouvoir considérer les interactions multiples dans les pseudo-nœuds standard ("Pour toutes interactions (i, j) et (i', j') d'une structure de type alors $i < i' < j' < j$" devient "Pour toutes interactions (i, j) et (i', j') d'une structure de type alors $i \leq i' < j' \leq j$") alors les nouveaux pseudo-nœuds standard de degré 3 inclus les triples hélices standard. En reprenant les mêmes arguments que pour l'analyse de l'Algorithme 5 pour les pseudo-nœuds standard, nous pouvons directement montrer que les nouveaux pseudo-nœuds standard peuvent être aligné avec la même complexité. Par conséquent, l'alignement des triples hélices standard par l'Algorithme 5 se calcule en $O(n^2 + n \cdot m^3)$.

Propriété 4.
La complexité de l'Algorithme 5 pour les triple hélices standard est en $O(n^2 + n \cdot m^3)$.

4.6.2 Comparatifs avec les heuristiques classiques de décompositions arborescentes

L'heuristique que nous utilisons pour construire les décompositions arborescentes (Algorithme 3 et 2) est-elle efficace par rapport aux heuristiques classiques de décomposition arborescente de graphes ? La pertinence des décompositions issues des plongements en vagues gloutons ne vient pas de la taille des décompositions mais du fait qu'elles soient lissées (ou faiblement lissées lors de l'assemblage). Or la propriété lissée n'étant pas un objectif connu des autres heuristiques, elles ne garantissent pas un tel résultat.

Les heuristiques cherchant à construire des décompositions arborescentes de petites largeurs sont basées sur la recherche d'un ordonnancement d'élimination parfaite des sommets du graphe. Pour former l'ordonnancement elles utilisent des critères de sélection basés par exemple sur le degré des sommets. Parmi ces heuristiques, les plus connues et les plus efficaces sont l'heuristique de remplissage minium et l'heuristique de degré minimum. Appliquées aux ARN, ces heuristiques donnent une solution de largeur équivalente à notre approche (bien souvent optimale) mais non lissée. La Figure 4.25 montre les décompositions issues des heuristiques citées d'un pseudo-nœud standard de degré 3. La largeur des décompositions est de 3 comme le résultat de notre heuristique. Seulement aucune des décompositions n'est lissée, contrairement à notre approche.

4.7 Nouvelles classes de structures

Nous avons montré que l'Algorithme 5 remplit bien l'objectif que nous avions fixé : il permet de résoudre l'alignement structure-séquence pour n'importe quelle structure et possède la même complexité que les algorithmes présentés précédemment sur leur classes de structures respectives. Nous avons vu que la définition des pseudo-nœuds standards pouvait être enrichie pour prendre en compte les interactions multiples, simplement en modifiant la définition de la classe CHAIN. Nous appellerons cette nouvelle classe les **structures standards**.

Propriété 5.
La complexité de l'Algorithme 5 pour les structures standard est en $O(n^2+n\cdot m^k)$.

Exactement de la même façon nous définissons les **structures simples non-standard** comme la classe de structures des pseudo-nœuds simple non-standard enrichie des interactions multiples.

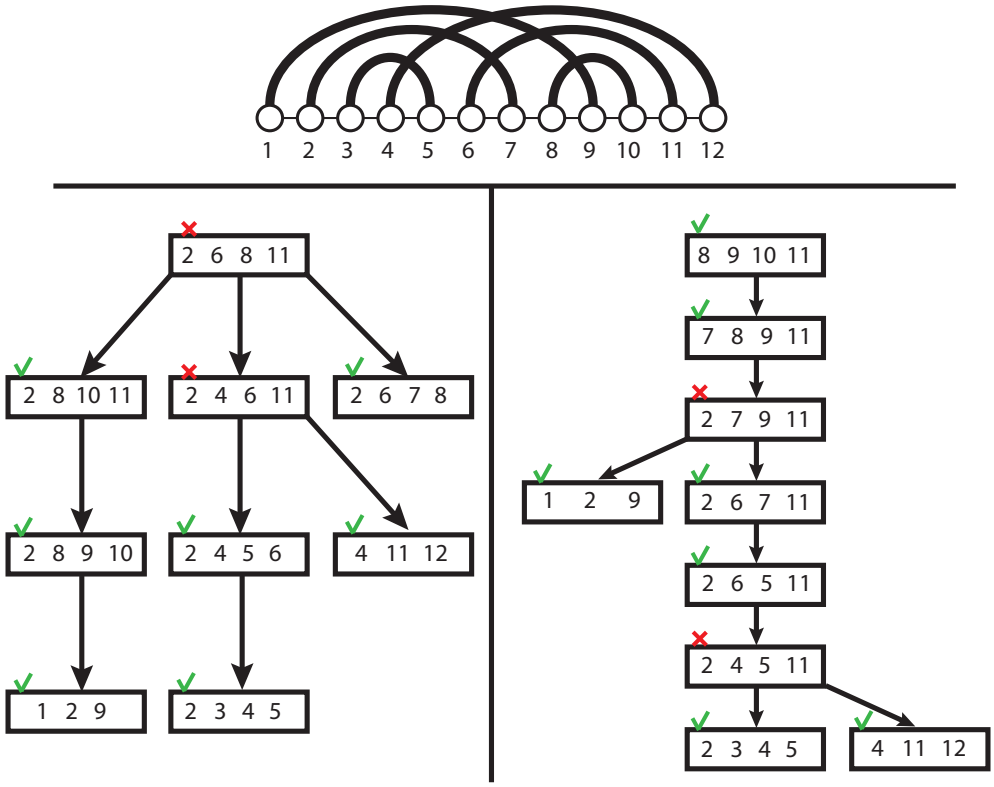


FIGURE 4.25 – *Décompositions arborescentes issues des heuristiques degré minimum (à gauche) et remplissage minium (à droite) d'un pseudo-nœud standard de degré* 3. *Les sacs non lissés sont marqués d'une croix (rouge)*

Propriété 6.
La complexité de l'Algorithme 5 pour les structures simples non-standard est en $O(n^2 + n \cdot m^{k+1})$.

L'intérêt de ces structures est de pouvoir prendre en compte certains motifs tertiaires très important. Comme nous l'avons plusieurs fois évoqué, il existe d'autres motifs récurrents dans les structures d'ARN mise à part les hélices, les boucles internes ou terminales, les renflements et les pseudo-nœuds. Les motifs tertiaires sont des motifs récurrents qui contiennent tous des interactions non-canoniques et presque tous des interactions multiples. Les plus connus de ces motifs sont les Kink-

Turn, C-Loop, Sarcin-Ricin, E-Loop, Hook-Turn et A-minor [37, 39, 38].

À l'origine ils ont été définis à partir de formes tridimensionnelles récurrentes dans les structures $3D$ d'ARN ou de complexe ARN-protéines [38, 30]. Il est difficile de montrer que ces motifs appartiennent à une classe de structures car leur définition formelle en terme de séquences arc-annotées ou de graphes n'existe pas. Á un motifs donné nous pouvons faire correspondre plusieurs séquences arc-annotées [39]. Néanmoins, un motif possède un schéma de base duquel ne s'éloigne que très peu ses différentes occurrences [39]. De nombreuses méthodes ont d'ailleurs été développées pour identifier automatiquement les motifs [37, 66] et même en prédire de nouveaux [17].

Même si nous ne pouvons pas affirmer que toutes les occurrences possibles d'un motifs appartiennent à une classe de structures, nous pouvons essayer de classer les occurrences des motifs présents dans les bases de données [55] et la littérature [37, 39, 40]. Étant donné que tous les cités motifs possèdent des interactions multiples (mise à part le Hook-Turn), seules ces nouvelles classes peuvent les inclure.

Nous observons que ces motifs appartiennent très fréquemment à l'une des trois classes de structures que nous avons défini (voir Figure 4.26).

Nous définissons une dernière classe de structure permettant des représenter plus fidèlement les triples hélices que la classe des triples hélices standard. Ainsi, nous dirons qu'une structure appartient à la classe des **triples hélices standard étendues** si et seulement si son plongement en vagues optimal est de degré 3.

Pour ces structures, nous pouvons montrer que leur degré de plongement en vagues glouton est aussi égal à 3. Les interactions de ces structures peuvent se partitionner en trois sous-ensembles P_1, P_2 et P_3 (voir Figure 4.27). La prise en compte de la structure induite par P_1 par l'Algorithme 3 formera un plongement en vagues de degré 2 sur le même principe que les démonstrations précédentes (voir Figure 4.27). Ensuite pour toute nouvelle première interaction qu'elle appartienne à P_2 ou P_3 engendrera un croisement et par conséquent un ajout de pivot (voir Figure 4.27). Par la suite, comme il existe un plongement en vague de degré 3 les autres interactions de P_2 et P_3 seront prises en compte sans créer de croisement (voir Figure 4.27).

Propriété 7.
La complexité de l'Algorithme 5 pour les triples hélices standard étendues est en $O(n^2 + n \cdot m^3)$.

Par définition les triples hélices standard sont inclues dans les triples hélices standard étendues car nous pouvons toujours leur associer un plongement en vague de degré 3 (la démonstration est immédiate d'après leur définition [63]). En revanche elles ne permettent pas d'inclure les interactions du sous-ensemble P_3 de la Figure 4.27. Or celles si peuvent être présentes dans les triples hélices [50].

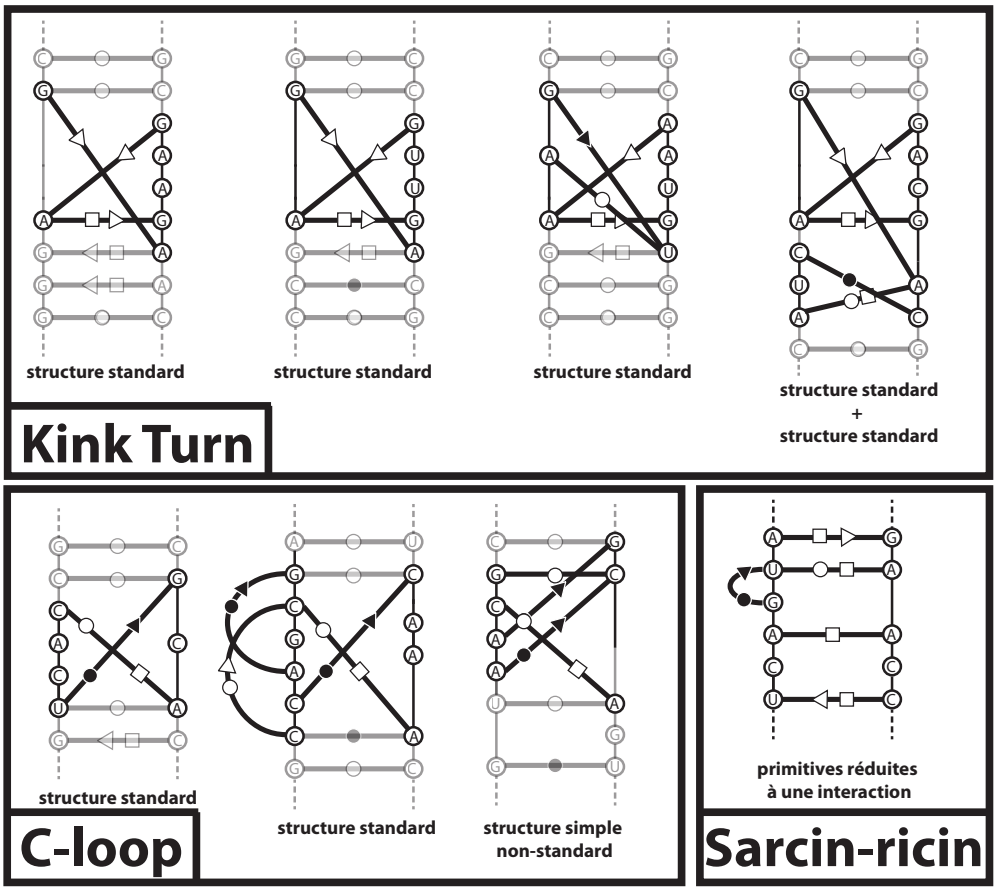


FIGURE 4.26 – *Exemple de motifs (Kink Turn, C-loop et sarcin-ricin). Les différentes occurrences représentées (structures grisées+non-grisées) sont toutes combinaisons de structures secondaires et de structures primitives non secondaires (structures non-grisées). Ces structures primitives appartiennent néanmoins à l'une des classes que nous avons définies.*

Enfin, nous définissons la classe des structures classiques récursives comme la classe de structures formée de n'importe quelle combinaison de structures du type structures secondaires, structures standard, structures simples non-standard et triple hélices standard étendues. Un récapitulatifs des classes est donné en Figure 4.28.

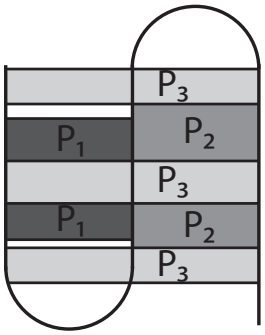


FIGURE 4.27 – *Illustration d'une triples hélice standard étendue et de la construction de son plongement en vagues glouton.*

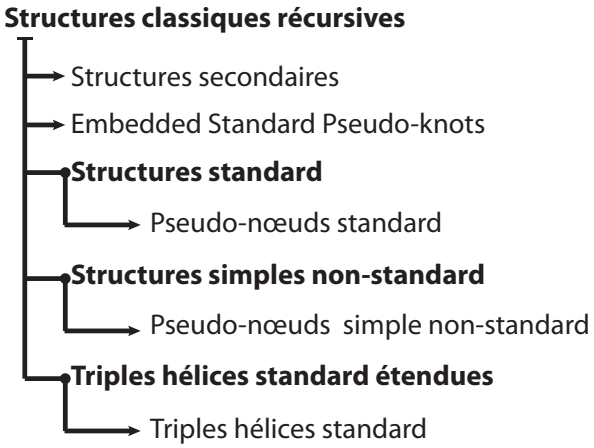


FIGURE 4.28 – *Récapitulatif des classes de structures d'ARN. La structure arborescente représente les relations d'inclusions entre classes.*

Chapitre 5

Perfectionnement pour l'application

Sommaire

Nous avons établi un algorithme permettant de résoudre le problème d'alignement structure-séquence dans son cadre général. Comme nous l'avons vu, ce problème tente de résoudre deux problèmes biologiques : la prédiction de structures et la recherche d'homologues. La résolution du problème bio-informatique (l'alignement structure-séquence) ne garanti pas à 100% celle du problème biologique.

Pour palier à ce constat, une approche classique consiste à s'intéresser aux solution sous-optimale du problème. Nous allons l'appliquer à notre problème.

Pour un application efficace nous devons aussi prendre en compte la constatation suivante : la recherche d'ARN homologues s'effectue sur de grandes bases données, le nombre d'alignements à effectuer pouvant alors être très grand (de l'ordre du million). Ainsi, même pour les structures secondaires, le temps de calcul d'un programme associé à notre algorithme (ou celui de Jian) serait bien trop grand pour des ordinateurs classiques. Nous allons donner des pistes pour résoudre ce problème.

Enfin, pour pouvoir calculer une solution optimale, il est nécessaire de posséder une fonction de coût. Nous allons exposer les fonctions existantes en expliquant rapidement leur fonctionnements puis nous expliquerons comment raffiner certaines d'entre elles.

5.1 Sous-optimaux

Pour déterminer des solutions sous-optimales nous ne voulons bien évidement pas avoir à recalculer un alignement mais réutiliser le calcul de la solution optimale. La programmation dynamique offre un cadre adapté à la recherche de sous-optimaux. Nous pouvons citer par exemple la recherche de solutions sous-optimales pour l'alignement séquence-séquence [60, 61].

La technique employée est standard. Elle consiste à rechercher toutes les solutions ayant un coût au maximum ε plus grand que le coût de la solution optimale (pour un ε donné).

Illustrons la recherche de sous-optimaux par un exemple utilisant l'approche développée dans cette thèse (Figure 5.1). Supposons que nous ayons déterminé la solution optimale (et son coût). Ce coût optimal C_0 est le coût minimum contenu dans la table des alignements de la racine. Dans cette table, nous recherchons les coûts C'_0 tels que $C'_0 - C_0 < \varepsilon$. Ces coûts correspondent à un alignement sous-optimal (à ε près).

Pour chacun de ces coûts (le coût optimal inclut), nous effectuons un retour sur trace pour reconstruire l'alignement. Soit C'_0 l'un de ces coûts et μ' l'alignement de la racine associé. La première étape du retour sur trace nous mène à la table des alignements du fils de la racine (ou des fils). Le coût C'_0 a été calculé à partir du coût C'_1 de l'un de ces alignements. Nous pouvons alors rechercher tous les autres alignements μ'' compatible avec μ et de coût C''_1 tel que $C''_1 - C'_1 < \varepsilon - \Delta$, où $\Delta = C'_0 - C_0$. Nous continuons le raisonnement ainsi de suite et pouvons alors reconstruire toutes les solutions sous-optimales (à ε près).

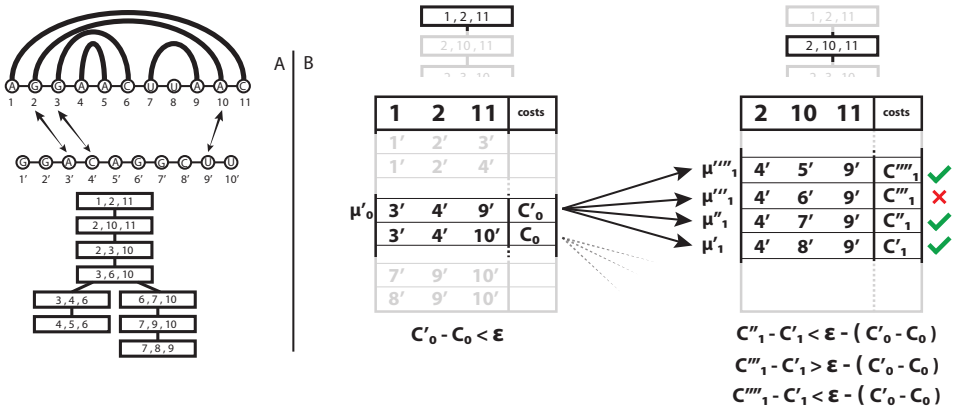


FIGURE 5.1 – *Illustration de la recherche de solution sous-optimale.*

La complexité associée à la recherche des solutions sous-optimales est exponentielle étant donné que le nombre de sous-optimaux possibles peut être exponentiel. Même s'il est nécessaire d'utiliser cette technique avec parcimonie, en choisissant un ε très petit (en le limitant à la somme de quelques coûts d'opérations élémentaires) cette recherche peut s'effectuer efficacement.

Un autre type d'approche consiste cette fois à déterminer les k premières solutions sous-optimales. Nous illustrons cette technique sur le même exemple que précédemment. Nous recherchons les k meilleurs coûts contenus dans la racine que nous plaçons dans une liste. Le coût optimal C_0 est nécessairement le meilleur coût, mais les autres coûts de la liste ne sont pas nécessairement les $k-1$ meilleurs autres coûts. Soit C_{max} l'élément k de la liste. Soit C'_0 un des $k-1$ premiers coûts de la liste correspondant à l'alignement μ de la racine. Nous effectuons alors un retour sur trace à partir duquel nous avons accès au coût C'_1 de l'alignement μ' qui nous a permis de calculer C'_0. Dans la table des alignements du fils de la racine nous recherchons les alignements μ'' compatible avec μ de coût C''_1 tel que $C''_1 - C'_1 + C'_0 < C_{max}$. Ces alignements ont un coût inférieur au plus petit élément de la liste. Nous continuons ainsi jusqu'à obtenir les k meilleures solutions.

Pour que ces techniques fonctionnent, il est essentiel que les alignements soit construit de façon non-ambigüe, c'est-à-dire qu'un alignement ne puisse s'obtenir qu'à partir d'un seul et unique retour sur trace. Ceci est garanti par la représentation

des alignements que nous avons défini (où l'on aligne toutes les positions). En effet, deux retours sur trace sont différents si et seulement si pour l'un des sacs de la décomposition arborescente l'alignement μ du sac utilisé pour le calcul du coût est différent. Or si l'alignement d'un sac est différent alors les alignements sont nécessairement différents.

Nous pouvons donc bien utiliser ces techniques pour calculer les solutions sous-optimales.

5.2 Heuristiques

Nous présentons maintenant comment utiliser l'algorithme précédent au sein d'heuristiques. Dans la pratique, nous espérons casser systématiquement la complexité exponentielle du problème (pour l'alignement des pseudo-nœuds et des structure secondaires étendues) grâce à notre paramétrisation. En effet nous avons tenter de construire la paramétrisation spécifiquement pour les structures d'ARN. Nous pouvons appliquer des couches heuristiques sur notre algorithme exacte et casser encore d'avantage la complexité, en considérant cette fois les spécificités des solutions optimales. Concrètement cela donne les deux heuristiques suivantes.

Les points d'ancrage

Le problème d'alignement structure-séquence vise à remplacer l'alignement classique de deux séquences lorsque celles-ci ont une faible similarité. Les mutations compensatoires impliquent que les mutations s'effectuent un peu partout dans la séquence. Néanmoins, il existe tout de même beaucoup de cas où des portions de séquences restent conservés. Pour ces cas là, nous pouvons appliquer un pré-traitement avant l'alignement structure-séquence en effectuant un premier alignement de séquence classique recherchant des segments commun (comme BLAST par exemple [3] ou plus récemment YASS [47] qui est plus adapté à l'alignement des nucléotides).

Le but étant de déterminer des zones contigües conservées entre la structure et la séquence (les ancres) que nous n'aurons plus à aligner par la suite et qui délimiteront les alignements possibles des autres positions (Figure 5.2).

La prise en compte de cette heuristique dans l'approche de ce mémoire s'effectue au niveau de l'alignement de la décomposition arborescente avec la séquence. L'alignement des positions appartenant à une ancre est imposé (par le pré-traitement) et l'alignement des autres positions est contraint de façon à être compatible avec les ancres. Nous pouvons ainsi réduire considérablement le nombre d'alignements possibles pour les sacs de la décomposition. Notons qu'au mieux nous alignons entièrement la structure ou la séquence et qu'au pire nous alignons rien du tout.

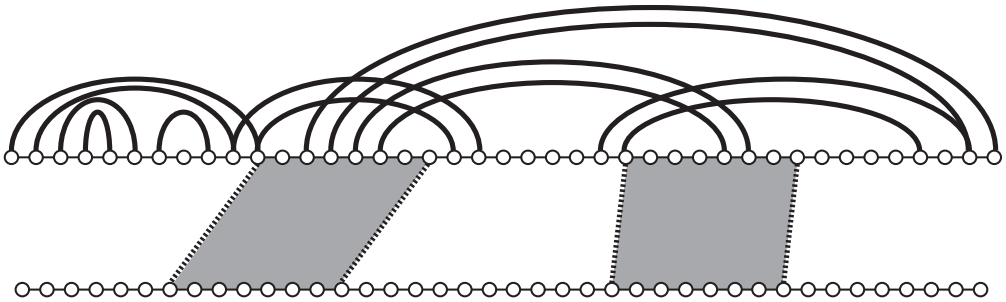


FIGURE 5.2 – *Illustration de l'heuristique des points d'ancrage (en gris). Il reste à aligner les zones non-grisées en respectant les ancres.*

Alignement par fenêtres

Pour résoudre le problème de l'alignement structure-séquence de façon exacte nous devons envisager tous les alignements possibles. Or, l'alignement mettant en correspondance la première position de la structure avec la dernière position de la séquence n'a pas besoin d'être évaluer pour être décrété comme étant un mauvais alignement. Sur ce principe, nous pouvons associé à chaque position de la structure une zone fixe (la fenêtre) de la séquence sur laquelle nous autorisons l'alignement (Figure 5.3).

Une première piste pour déterminer la taille des fenêtres est de comparer la différence de longueur entre la structure et la séquence. Si elles ont la même longueur à deux ou trois nucléotides près, nous pouvons espérer que si elles sont homologues alors leur le nombre de gaps est réduit. Nous pouvons alors définir des fenêtres de taille fixe centrées sur la position i de la séquence pour l'alignement de la position i de la structure. Si leur longueur est sensiblement la même (la différence des longueurs étant par exemple de l'ordre du logarithme des longueurs), nous pouvons envisager de tester une fenêtre de taille $log(m)$ centré sur la position i de la séquence pour l'alignement de la position i de la structure (Figure 5.3). Enfin si leur longueur est foncièrement différente, (la différence de taille étant par exemple égale au tiers de la plus grande) nous devons nécessairement envisager une fenêtre dont la taille est de l'ordre de la séquence (et nous ne gagnerons qu'une constante dans la complexité). Malheureusement, même s'ils sont rares, de tels cas existent [12] provenant de grandes insertions.

La prise en compte de cette heuristique s'effectue de la même manière que la précédente, c'est-à-dire en restreignant les alignements des sacs en fonction des contraintes imposées par les fenêtres. Notons que nous pouvons effectuer le prétraitement des points d'ancrages en premier lieu et ensuite utiliser le système de fenêtres. Les longueurs à comparer devenant alors les longueurs se situant entre deux points d'ancrages consécutifs (Figure 5.4).

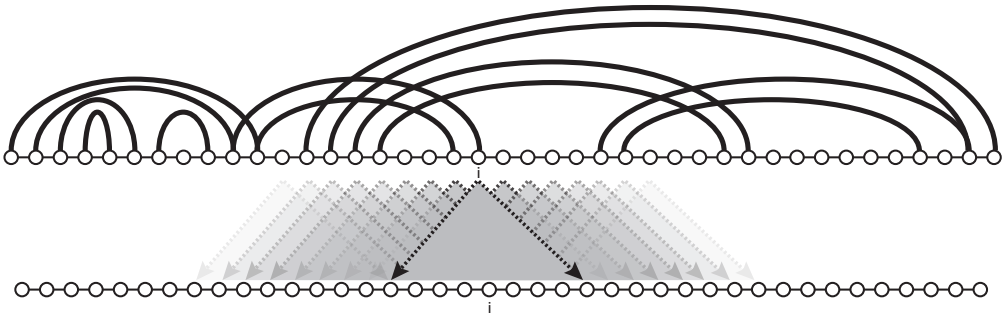


FIGURE 5.3 – *Illustration de l'heuristique d'alignement par fenêtres.*

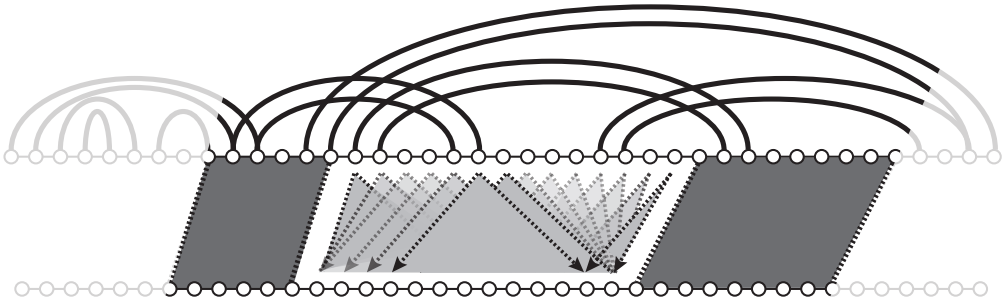


FIGURE 5.4 – *Illustration de l'utilisation combiné des deux heuristiques.*

5.3 Les fonctions de coûts

Pour terminer ce chapitre nous allons aborder un point central de l'alignement structure-séquence que nous avons un peu mis de coté. Le problème d'optimisation que représente l'alignement structure-séquence repose sur une fonction de coûts. Jusqu'à présent nous avons simplement donné la forme de cette fonction. Celle-ci est donc composée de coût élémentaires pour l'alignement des bases et des interactions ainsi qu'un coût élémentaire pour les gaps (que nous avons imposé affine). Nous allons maintenant voir comment déterminer chacun de ces coûts élémentaires.

5.3.1 Les matrices RIBOSUM

Les matrices RIBOSUM [33] sont des matrices dites de substitutions. Pour chaque substitution (de bases ou d'interactions) elles attribuent un coût. Elles remplissent donc parfaitement leur rôle de fonctions de coûts.

Chaque coûts est calculé à partir d'un alignement multiple de structure secon-

daire d'ARN (ribosomique) où l'on supprime les gaps (et les nucléotides alignés sur avec un gap). Nous considèrerons qu'un alignement multiple est un bloc de deux dimensions, l'une (nombre de lignes) correspondant au nombre de séquences, et l'autre (nombre de colonne) correspondant à la longueur des séquences. Chaque cellule du bloc contient évidemment un nucléotide. Les coûts élémentaires sont alors calculés par les formules suivantes :

– Substitution de bases : $s_{ij} = log_2(\frac{f_{ij}}{g_i g_j})$.

où i et j sont deux nucléotides alignés, f_{ij} est la fréquence empirique observée de l'alignement i avec j, et g_i et g_j sont les fréquences observées des nucléotides i et j.

– Substitution de d'interaction : $s_{ij,kl} = log_2(\frac{f_{ij,kl}}{g_i g_j g_k g_l})$.

où ij et kl sont deux interactions alignées, $f_{ij,kl}$ est la fréquence empirique observée de l'alignement ij avec kl, et g_i, g_j, g_k et g_l sont les fréquences observées des nucléotides i et j.

Un exemple de matrice est donné dans la Figure 5.5.

AA	**2.30**															
AC	**1.78**	**3.49**														
AG	-0.62	-0.10	**1.32**													
AU	**0.06**	**0.07**	-0.67	**1.79**												
CA	**0.40**	**0.30**	**0.54**	-1.92	**1.78**											
CC	-0.33	-0.61	**0.41**	-1.17	**0.76**	**1.41**										
CG	-1.80	-3.00	**0.90**	-1.40	**0.90**	**0.78**	**1.13**									
CU	**1.43**	**1.75**	-0.93	**0.75**	-0.70	**0.39**	-0.56	**3.38**								
GA	-0.11	**0.36**	-1.71	-0.53	-1.48	-1.32	-2.31	-0.94	**3.25**							
GC	-0.62	**0.36**	-0.90	**0.13**	-1.82	-0.37	-0.96	-0.74	-0.25	**1.03**						
GG	-1.20	-0.43	**0.88**	-1.38	**1.24**	**0.95**	**0.72**	-0.40	**1.33**	-1.03	**2.99**					
GU	**0.49**	**0.32**	-1.34	-0.33	-3.21	-1.64	-2.59	**1.65**	**2.63**	-0.62	**0.60**	**2.93**				
UA	**1.50**	-2.70	-1.16	**0.09**	-0.14	-0.63	-1.16	-1.61	-1.28	-0.91	-2.27	-2.86	**2.06**			
UC	-0.17	**0.57**	-1.47	-3.13	-1.78	-0.82	-3.14	-1.25	-0.53	**0.77**	-1.37	-2.61	-0.77	**3.83**		
UG	-0.41	**0.76**	**0.16**	-2.44	**0.75**	-0.10	-0.14	-1.02	-1.67	-1.51	-0.75	-2.80	**0.05**	**1.15**	**2.24**	
UU	-0.57	-0.82	**0.17**	-0.49	-1.58	-0.64	-0.74	**0.18**	-1.24	-0.70	-0.76	-0.36	-0.54	**1.86**	**2.09**	**2.66**
	AA	AC	AG	AU	CA	CC	CG	CU	GA	GC	GG	GU	UA	UC	UG	UU

A	**1,00**			
C	-1,70	**1,59**		
G	-0,86	-1,58	**1,60**	
U	-1,02	-0,40	-0,94	**1,28**
	A	C	G	U

FIGURE 5.5 – *Illustration d'une matrice RIBOSUM (extrait de [33]).*

Nous pouvons en déduire qu'une substitution et sa substitution inverse ont le même coût. Nous pouvons remarquer que les coûts des substitutions des interactions sont calculés à partir des fréquences des bases et non des fréquences des interactions (le dénominateur de la fraction). À priori rien ne justifie ce choix et même si ce point a déjà été soulevé aucun développement en conséquence n'a vu le jour concrètement.

Une force de cette approche est qu'elle peut s'adapter à une situation en fonction de la similarité de la structure et de la séquence. En effet un point passé sous silence est que la méthode sélectionne au préalable seulement les séquences de l'alignement ayant une similarité de séquence (globale) supérieure à une valeur seuil. Puis lors du calcul des fréquences, les substitutions associées à des paires de séquences n'ayant pas une similarité de séquence supérieur à un second seuil se sont pas prises en compte. Il existe donc plusieurs matrice RIBOSUM possibles en fonction des deux valeurs seuils. Malheureusement, comme nous ne pouvons à priori pas connaitre à l'avance la similarité de séquence entre la structure et la séquence, cela s'avère être plus un inconvénient qu'un avantage. Par exemple, le logiciel RSEARCH pour lequel

les matrices RIBOSUM ont été originellement développées possède 170 matrices de substitutions (dernière version). Il n'est pas envisageable de tester systématiquement les 170, et force est de constater que la matrice par défaut est bien souvent la seule utilisée (voir par exemple le benchmark [22]).

Enfin, nous devons citer un point important pour lequel il reste du travail à accomplir. Les matrices RIBOSUM ne concernent que les substitutions. Les coûts associés aux insertions, suppressions (de bases ou d'interactions) et altérations ne sont pas donnés. Dans la pratique, les coûts associés aux insertions et suppressions de bases sont repris des coûts classique d'alignement de séquences. Le coût d'une suppression d'interaction est souvent assimilé à une double suppression de bases, quant à l'altération d'interaction, elle n'est simplement jamais considérée et se résume à une suppression et une substitution de bases. La considération d'une interaction comme entité propre étant alors mise de coté (sauf pour la substitution).

5.3.2 Les matrices d'isostérie

Un autre type d'approche non basée sur le calcul de statistiques est possible. Pour cela il faut analyser la conformation $3D$ des interactions.

Dans [35], les auteurs dressent le constat que deux interactions d'une même famille géométrique (Watson-Crick/Watson-Crick ou Sugar/Watson-Crick ou Sugar/Hoogsteen...) ne sont pas nécessairement isostériques (ayant la même forme tridimensionnelle). Partant de là, ils définissent pour chaque famille géométrique une matrice d'isostérie indiquant pour chaque interaction de la famille sa classe d'isostérie (voir Figure 5.6). Ces classes d'isostérie ont ensuite été utilisées pour

		Hoogsteen			
Watson-Crick	*cis*	**A**	**C**	**G**	**U**
	A			I_3	(I_3)
	C		I_2	I_1	(I_1)
	G	I_3		I_4	
	U	I_1		I_1	I_2

FIGURE 5.6 – *Illustration de la matrice d'isostérie de la famille géométrique cis Watson-Crick/Hoogsteen (extraite de [35]). Les classes d'isostérie sont représentées par les I_t. Les cases vides indiquent que l'interaction correspondante est impossible. Les classes entre parenthèses sont les classes théoriquement possibles mais encore non observées.*

tenter de caractériser les motifs structuraux, et il a été montré entre autre qu'elles peuvent être utilisées pour l'alignement de motifs structuraux [39]. Plus récemment

encore dans [57], les auteurs définirent des scores “quantitatifs“, les indices d'écart d'isostérie (IDI), basés sur les mêmes critères. Ainsi pour chaque paires d'interactions possibles ils associent un score représentant la différence tridimensionnelle des deux interactions. À partir de là, nous pouvons construire des matrices de substitutions IDI [57] et les utiliser comme fonction de coûts (Figure 5.7).

cWW	Family	LSW 2002 isosteric groups	Updated isosteric groups	Count	CC	UU	CA	UG	UA	AU	GC	CG	GU	AC	UU	CC	UC	CU	AA	GA	AG	AA
CC	cWw	$I_{1.6}$	$I_{1.6}$	8	0.00	2.37	4.49	5.06	5.30	5.39	5.56	5.49	6.25	5.91	4.31	3.02	8.25	7.97	9.77	8.86	8.82	9.05
UU	cwW	$I_{1.6}$	$I_{1.7}$	96	2.37	0.00	2.39	2.89	3.63	3.80	3.94	3.80	5.27	5.21	4.36	4.31	6.46	5.97	8.18	6.96	6.91	6.91
CA	cWW	$I_{1.2}$	$I_{1.2b}$	16	4.49	2.39	0.00	0.80	2.47	2.75	2.78	2.55	4.76	4.93	5.21	5.91	5.30	4.58	6.70	5.14	5.05	4.80
UG	cWW	$I_{1.2}$	$I_{1.2b}$	772	5.06	2.89	0.80	0.00	2.11	2.40	2.39	2.14	4.48	4.76	5.27	6.25	4.59	3.80	6.07	4.44	4.33	4.10
UA	cWW	$I_{1.1}$	$I_{1.1}$	2410	5.30	3.63	2.47	2.11	0.00	0.31	0.34	0.21	2.40	2.75	3.80	5.39	3.57	3.50	4.66	3.67	3.67	4.52
AU	cWW	$I_{1.1}$	$I_{1.1}$	2410	5.39	3.80	2.75	2.40	0.31	0.00	0.21	0.34	2.11	2.47	3.63	5.30	3.50	3.57	4.52	3.67	3.67	4.66
GC	cWW	$I_{1.1}$	$I_{1.1}$	7222	5.56	3.94	2.78	2.39	0.34	0.21	0.00	0.26	2.14	2.55	3.80	5.49	3.39	3.44	4.38	3.49	3.50	4.50
CG	cWW	$I_{1.1}$	$I_{1.1}$	7222	5.49	3.80	2.55	2.14	0.21	0.34	0.26	0.00	2.39	2.78	3.94	5.56	3.44	3.39	4.50	3.50	3.49	4.38
GU	cWW	$i_{1.2}$	$I_{1.2a}$	772	6.25	5.27	4.76	4.48	2.40	2.11	2.14	2.39	0.00	0.80	2.89	5.06	3.80	4.59	4.10	4.33	4.44	6.07
AC	cWW	$i_{1.2}$	$I_{1.2a}$	16	5.91	5.21	4.93	4.76	2.75	2.47	2.55	2.78	0.80	0.00	2.39	4.49	4.58	5.30	4.80	5.05	5.14	6.70
UU	cWw	$I_{1.6}$	$I_{1.7}$	96	4.31	4.36	5.21	5.27	3.80	3.63	3.80	3.94	2.89	2.39	0.00	2.37	5.97	6.46	6.91	6.91	6.96	8.18
CC	cwW	$I_{1.6}$	$I_{1.6}$	8	3.02	4.31	5.91	6.25	5.39	5.30	5.49	5.56	5.06	4.49	2.37	0.00	7.97	8.25	9.05	8.82	8.86	9.77
UC	cWW	$I_{1.5}$	$I_{1.5}$	12	8.25	6.46	5.30	4.59	3.57	3.50	3.39	3.44	3.80	4.58	5.97	7.97	0.00	1.53	2.71	2.25	2.33	3.77
CU	cWW	$I_{1.5}$	$I_{1.5}$	12	7.97	5.97	4.58	3.80	3.50	3.57	3.44	3.39	4.59	5.30	6.46	8.25	1.53	0.00	3.77	2.33	2.25	2.71
AA	cWw	$I_{1.4}$	$I_{1.4}$	3	9.77	8.18	6.70	6.07	4.66	4.52	4.38	4.50	4.10	4.80	6.91	9.05	2.71	3.77	0.00	2.18	2.41	4.52
GA	cWW	$I_{1.3}$	$I_{1.3}$	121	8.86	6.96	5.14	4.44	3.67	3.67	3.49	3.50	4.33	5.05	6.91	8.82	2.25	2.33	2.18	0.00	0.33	2.41
AG	cWW	$I_{1.3}$	$I_{1.3}$	121	8.82	6.91	5.05	4.33	3.67	3.67	3.50	3.49	4.44	5.14	6.96	8.86	2.33	2.25	2.41	0.33	0.00	2.18
AA	cwW	$I_{1.4}$	$I_{1.4}$	3	9.05	6.91	4.80	4.10	4.52	4.66	4.50	4.38	6.07	6.70	8.18	9.77	3.77	2.71	4.52	2.41	2.18	0.00

FIGURE 5.7 – *Illustration de la matrice de substitutions IDI de la famille géométrique cis Watson-Crick/Watson-Crick substituée à elle même (extraite de [57]).*

Malheureusement comme pour les matrices RIBOSUM il reste à travailler sur les suppressions et altérations d'interactions pour le moment oubliés. En revanche, l'énorme avantage de ces coût est qu'ils ne nécessite pas une base de données de structures conséquente pour les structures secondaires étendues ou $3D$ d'ARN.

5.3.3 Raffinement par apprentissage

Parallèlement aux matrices d'isostérie, nous pouvons mettre en place une méthode que nous pouvons qualifier de raffinement. La méthode que j'essaie de mettre en place est basée sur une approche par apprentissage et plus exactement sur un algorithme génétique [25].

Dans un premier temps nous décrivons la méthode générale, puis j'expliquerai comment je comptes l'adapter au problèmes biologiques que tente de résoudre l'alignement structure séquence.

1. **Jeu d'entrainement** : nous disposons d'un jeu de données, appelé jeu d'entrainement, constitué d'alignements structure-séquence que nous considèrerons correct ou de familles d'homologues.
2. **Initialisation** : nous initialisons une **population** de n_p jeux de coûts élémentaires. Chaque jeu de coûts est appelé **individu** et contient tous les coûts élémentaires recherchés.

3. **Évaluation** : chaque individu w est évalué de la façon suivante. Pour tous les couples structure-séquence, nous effectuons l'alignement structure-séquence en utilisant w. Puis en comparant le résultat obtenu au résultat attendu nous calculons un score. Enfin nous ajoutons tous les scores obtenus avec les couples structure-séquence du jeu d'entrainement pour calculer l'**adaptation** de w.
4. **Sélection et croisement** : nous effectuons n_p fois les étapes suivantes pour construire une nouvelle population (population fils).
   - **Sélection** : nous sélectionnons aléatoirement deux individus de la population (avec remise), la probabilité de sélectionner un individu étant proportionnel à son adaptation.
   - **Croisement** : nous effectuons un croisement entre les deux individus sélectionnés, c'est-à-dire que nous formons un nouvel individu fils composé d'un mélange de ses deux individus parents. Nous plaçons ce nouvel individu dans la population fils.
5. **Mutation** : pour chaque individu nous effectuons une mutation aléatoire pour chacun de ses coûts élémentaires.
6. Nous réitérons ce procéder à partir de l'évaluation sur la population fils jusqu'à obtenir une adaptation en dessous d'une valeur seuil.

Nous décrivons maintenant plus en détails chacun des points ci-dessus en fonction du problème : la prédiction de structure et la recherche d'homologue.

Jeu d'entrainement

Prédiction de structure.

Nous avons besoin de couples structure/séquence d'ARN considérés comme homologues. Nous pouvons construire ces couples à partir d'alignements multiples précis comme les alignements de références (les graines) de la base Rfam [27] (parfois vérifiés par l'œil de l'expert). Ces alignements multiples possèdent une structure consensus à partir de laquelle nous pouvons directement inférer une structure pour chacune des séquences. Cette base regroupe les structures secondaires et les structures secondaires avec pseudo-nœuds de plus de 2000 familles d'ARN non-codants mais pas de structures secondaires étendues. La plus grande base de données de structures secondaires étendues est la base FR3D. Elles contient les annotations de plus de 2000 structures d'ARN mais pas d'alignements. Néanmoins tous ces ARN sont aussi catalogués dans Rfam. Nous pourront donc retrouver des alignements par cet intermédiaire.

Recherche d'homologue.

Nous utiliserons les mêmes bases de données que pour la prédiction de structures. Nous ne construisons pas des couples structure/séquence mais des groupes d'ARN homologues. Pour chaque groupes d'homologues nous nommons des représentants (qui peuvent changer pour chaque itération de l'algorithme génétique, c'est-à-dire à

chaque génération d'individus) que nous utiliserons pour la recherche d'homologue. Ces représentants seront donc les structures du problème d'alignement structure-séquence, et les autres ARN seront les séquences.

Initialisation

L'initialisation de la population peut se faire complètement aléatoirement. Cependant pour accélérer la convergence de l'algorithme génétique nous pouvons initialiser la population en définissant nos individus à partir des matrices RIBOSUM ou des matrices de substitutions IDI (de la famille Waston-Crick/Watson-Crick). Pour les deux problèmes (prédiction de structures et recherche d'homologues) l'initialisation est identique.

Évaluation des jeux de coûts

Prédiction de structure.

Nous calculons l'adaptation d'un individu w à partir de la précision et du rappel de chaque structures prédites. À partir d'un l'alignement structure-séquence obtenu en utilisant w, nous pouvons inférer une structure pour la séquence (structure prédite). Nous la comparons alors à la structure inférée à partir de l'alignement l'alignement attendu (structure référence). La précision est égale au nombre d'interactions correctement prédites (interactions de la structure prédite apparaissant aussi dans la structure référence) divisé par le nombre total d'interactions prédites. Le rappel est égal au nombre d'interactions correctement prédites divisé par le nombre total d'interactions de la structure de référence. La précision associée à w est alors la moyenne des précisions obtenues (il en va de même pour le rappel). Nous pouvons alors définir l'adaptation comme la précision, le rappel ou une combinaison des deux (F-mesure).

Recherche d'homologue.

Nous devons procéder différemment en utilisant le coût des alignements optimaux. Chaque représentant w de chaque groupe d'homologues est aligné avec le jeu d'entrainement pour chercher ses homologues. Le but étant que les alignements des ARN du même groupe que w doivent avoir un coût moins élevé que les alignements des autres ARN. Si nous classons les ARN en fonction du coût de l'alignement (du plus petit au plus grand) nous souhaitons avoir les homologues en début de classement et les non-homologues en fin de classement. Nous construisons alors la courbe ROC [41] à partir du classement obtenu (voir Figure 5.8). Pour chaque représentant nous calculons l'air sous la courbe ROC et associons à l'adaptation de w la moyenne de ces airs.

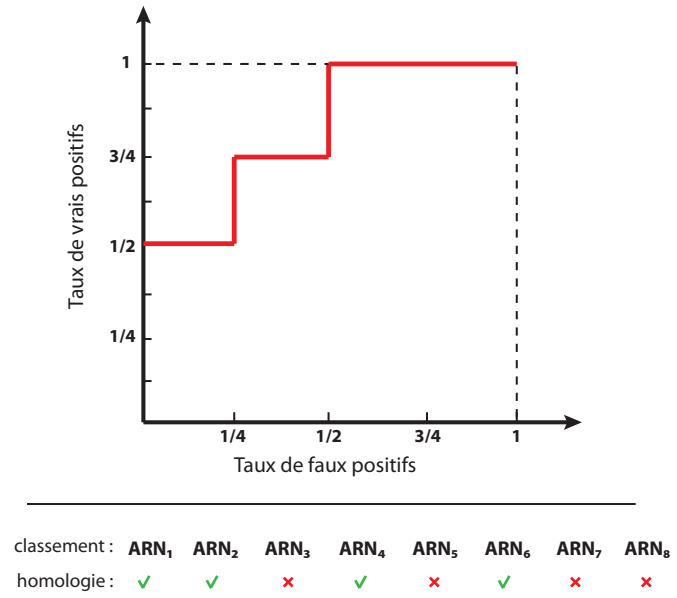


FIGURE 5.8 – *Exemple d'une courbe ROC. Les ARN sont classés par coûts croissants. Ceux marqués d'une croix ne sont pas homologues alors que les autres le sont. Pour construire la courbe, nous considérons les t premiers ARN comme homologues en faisant varier t de 0 à 8. Pour $t = 0$, nous prédisons aucun ARN comme homologue, donc le taux d'homologue correctement prédit (taux de vrai positifs) est de 0. De même le taux d'ARN non-homologues prédit comme homologues (le taux de faux positifs) est de 0. Pour $t = 1$, 1/4 des homologues est correctement prédit, et toujours 0 non-homologue ont été prédit homologues. Nous continuons ce raisonnement pour former la courbe ROC.*

Premiers tests

Nous avons effectué des premiers tests pour la prédiction de structures secondaires sur un jeu d'entrainement réduit (et avec un autre jeu de données différent pour l'évaluation finale). Nous améliorons sensiblement les prédictions de structures en utilisant un jeu de coûts obtenu par apprentissage par rapport à ceux de RIBOSUM. Ainsi nous arrivons à plus de 90% de précision sur nos prédictions (contre un peu plus de 85% pour RIBOSUM en choisissant systématiquement la matrice la plus avantageuse). Nous devons passer la méthode à plus grande échelle, c'est-à-dire prendre un plus grand jeu d'entrainement et tester la méthode pour tous les niveaux de structures. L'apprentissage étant très couteux (nous pouvons effectuer des millions d'alignement avant la convergence) nous envisageons maintenant de l'utiliser en combinaison avec les heuristiques présentées précédemment.

Chapitre 6

Conclusion

Nous avons amorcé ce mémoire par une description des ARN et de leurs principaux problèmes biologiques : **la recherche d'homologues et la prédiction de structures d'ARN**. La formalisation des structures ARN par des graphes ou des séquences arc-annotées offre une ouverture pour la résolution de ces deux problèmes.

Pour résoudre ces problématiques il existe différentes méthodes. Les plus précises sont les méthodes dites comparatives, c'est-à-dire celles basées sur l'homologie. Parmi ces approches, l'**alignement structure-séquence** est celle que nous avons retenu car elle demande peu d'information (une seule structure) et son efficacité est comparable au meilleures autres méthodes.

Les approches existantes d'alignement structure-séquence que nous avons décrites abordent le problème de façon sensiblement identique et où la **programmation dynamique** joue un rôle essentiel. Nous avons vu comment chacune de ces méthodes arrivait à tirer partie de la spécificité d'une classe de structures pour résoudre le problème en temps polynomial alors que celui-ci est NP-difficile dans sa généralité. La description détaillée des algorithmes d'alignements structure-séquence nous a laissé percevoir le cœur commun de leurs logiques que j'allais ensuite interpréter et formaliser par la construction de **décompositions arborescentes**.

Nous avons alors étudié l'ensemble de mon travail, c'est-à-dire l'**unification et la généralisation** des algorithmes d'alignement structure-séquence. Après avoir présenté le concept de la décomposition arborescente nous avons expliqué comment utiliser cette transformation des structures d'ARN pour l'alignement structure-séquence. Une fois la structure transformée en décomposition arborescente, l'alignement de la décomposition avec la séquence a demandé encore un peu de travail. En effet, nous avons mis en évidence qu'il était nécessaire de modifier la représentation des alignements pour pouvoir employer une approche utilisant les décompositions arborescentes pour l'alignement structure-séquence, les techniques standard ne permettant pas nativement de répondre au problème. Nous avons aussi défini une nouvelle propriété, les **décompositions arborescentes lissées**, pour tirer partie de la spécificité du problème et améliorer la complexité en temps de ma méthode.

Nous avions alors à ce stade un moyen de résoudre le problème d'alignement structure-séquence de façon générale en temps polynomial, le degré du polynôme étant égal à un de plus que la largeur de la décomposition arborescente utilisée. Nous avons donc naturellement poursuivi ce mémoire sur la construction d'un algorithme de décomposition arborescente. La détermination de la décomposition arborescente de largeur minimum étant NP-difficile, nous avons entrepris la description d'une heuristique. Celle-ci fonctionne en trois temps : division, décomposition, assemblage. Nous avons expliqué pourquoi et comment une structure d'ARN peut être **divisée** en plusieurs sous-structures : les **primitives**. Les primitives contiennent la réelle complexité de la structure, c'est-à-dire que sont elles qui sont difficiles à décomposer. Néanmoins leurs décompositions peuvent être effectuées indépendamment puis

ré-assemblées facilement, simplifiant alors le problème. Pour les décomposer, nous avons introduit les **plongements en vagues** qui permettent de définir un ordre total sur les positions d'une primitives. Cet ordre est ensuite directement utilisé pour la construction des décompositions arborescentes. C'est cette étape qui permet de généraliser les approches précédentes. La difficulté pour chacune d'entre elles était de définir cet ordre et d'effectuer simultanément l'alignement. En dissociant complètement ces deux aspects, la généralisation de l'alignement structure-séquence apparaît de façon claire. Il reste alors à déterminer les plongements en vagues de chacune des primitives. L'algorithme que nous avons introduit ne garantit pas une décomposition optimale des primitives mais étant élaboré spécifiquement pour les structures d'ARN, les résultats de décompositions arborescentes que nous obtenons vont permettre à la généralisation de se situer favorablement par rapport aux autres approches.

Après avoir étudié plus en détail la complexité de mon algorithme d'alignement structure-séquence nous avons entrepris de le comparer aux autres approches. nous avons montré qu'en considérant que la structure et la séquence ont une longueur équivalente, la complexité de mon approche était **égale sinon meilleure** que chacune des autres méthodes sur leur domaine de résolution respectif. L'unification est donc atteinte. Nous avons aussi démontré que c'est notamment grâce à la propriété de décompositions arborescentes lissées que ce résultat a pu être atteint. Par conséquent, les heuristiques de décompositions arborescentes classiques (n'offrant pas nécessairement des décompositions lissées) ne permettent pas d'atteindre un tel résultat.

Nous avons ensuite défini trois nouvelles classes de structures : **les structures standard, les structures simple non-standard, et les triples hélices standard étendues**. Chacune de ces classes de structures est une extension d'une classe de structure définie pour les approches précédentes que nous avons généralisé. L'intérêt des structures standard et des structures simple non-standard est de pouvoir prendre en compte les interactions multiples et par conséquent pouvoir représenter des motifs tertiaires. Quant aux triples hélices standard étendues elles permettent une représentation plus fidèle des triples hélices.

En guise de perspectives, nous pouvons envisager différentes améliorations à partir du travail effectué. Ainsi nous avons évoqué pourquoi et comment nous pouvons améliorer la qualité des résultats de l'alignement structure-séquence à partir de ma méthode en utilisant des recherches classiques de **solutions sous-optimales**. Nous pouvons ajouter différentes couches **heuristiques** sur mon algorithme pour améliorer sa vitesse. Enfin, nous devons aussi concentrer nos efforts sur les fonctions de coûts car nous l'avons vu les approches de références de ce point crucial ne sont pas tout à fait satisfaisantes. Nous pouvons par exemple envisager de les améliorer par une approche par apprentissage supervisé en utilisant un **algorithme génétique** donnant déjà les prémices d'applications efficaces.

Bibliographie

[1] Gautheret D.and Lambert A. Direct RNA motif definition and identification from multiple sequence alignments using secondary structure profiles. *J. Mol. Biol.*, 313 :1003–1011, 2001.

[2] T Akutsu. Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots. *Discrete Applied Mathematics*, 104(1–3) :45 – 62, 2000.

[3] S.F. Altschul, W. Gish, W. Miller, Myers E.W., and Lipman D.J. Basic local alignment search tool. *J. Mol. Biol.*, 215 :403–410, 1990.

[4] M. Andronescu, V. Bereg, H. Hoos, and A. Condon. RNA strand : The RNA secondary structure and statistical analysis database.

[5] S Arnborg, D Corneil, and A Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Meth.*, 8(2) :277–284, 1987.

[6] S Arnborg and A Proskurowski. Linear time algorithms for np-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, 23(1) :11–24, 1989.

[7] V. Bafna and S. Zhang. Fastr : fast database search tool for non-coding RNA.

[8] Markus Bauer, Gunnar Klau, and Knut Reinert. An exact mathematical programming approach to multiple RNA sequence-structure alignment. *Algorithmic Operations Research*, 3(2), 2008.

[9] Eckart Bindewald, Tanner Kluth, and Bruce A. Shapiro. Cylofold : secondary structure prediction including pseudoknots. *Nucleic Acids Res*, 38(suppl 2) :W368–W372, 2010.

[10] Guillaume Blin, Guillaume Fertin, Irena Rusu, and Christine Sinoquet. Extending the hardness of RNA secondary structure comparison. In *MFCS*, volume 4614 of *Lecture Notes in Computer Science*, pages 140–151, 2007.

[11] Hans L. Bodlaender. Treewidth : Algorithmic techniques and results. In *MFCS-97*, volume 1295 of *Lecture Notes in Computer Science*, pages 19–36, 1997.

[12] J.W. Brown. The ribonuclease p database. *Nucleic Acids Res*, 27(1) :314, 1999.

[13] R. Chenna, H. Sugawara, T. Koike, R. Lopez, T.J. Gibson, D.G. Higgins, and J.D. Thompson. Multiple sequence alignment with the clustal series of programs. *Nucleic Acids Res*, 31(13) :3497–3500, 2003.

[14] R. Das and D. Baker. Automated de novo prediction of native-like RNA tertiary structures. *Proceedings of the National Academy of Sciences*, 104(37) :14664–14669, 2007.

[15] Mathews D.H., M. Disney, Childs J.L., S.J. Schroeder, M.and Douglas H. Zuker, and Turner D.H. Incorporating chemical modification constraints into a dynamic programming algorithm for prediction of RNA secondary structure. *PNAS*, 101(19) :7287–7292, 2004.

[16] Y. Ding and C.E. Lawrence. A statistical sampling algorithm for RNA secondary structure prediction. *Nucleic Acids Res*, 31(24) :7280–7301, 2003.

[17] M Djelloul and A Denise. Automated motif extraction and classification in RNA tertiary structures. *RNA*, 14 :2489–2497, 2008.

[18] R. Downey, M. Fellows, and Langston M.

[19] S. R. Eddy. Profile hidden markov models. *Bioinformatics*, 14(9) :755–763, 1998.

[20] Sean R. Eddy and Richard Durbin. RNA sequence analysis using covariance models. *Nucleic Acids Res*, 22(11) :2079–2088, 1994.

[21] Patricia Anne Evans. *Algorithms and Complexity for Annotated Sequence Analysis*. PhD thesis, University of Victoria, 1999.

[22] Eva K. Freyhult, Jonathan P. Bollback, and Paul P. Gardner. Exploring genomic dark matter : A critical assessment of the performance of homology search methods on noncoding RNA. *Genome Research*, 17(1) :117–125, January 2007.

[23] P. Gendron, S. Lemieux, and F. Major. Quantitative analysis of nucleic acid three[U+2010]dimensional structures. *J. Mol. Biol.*, 308 :919–936, 2001.

[24] R.F. Gesteland, T.R. Cech, and J.F. Atkins. The RNA world, 3rd edn. *Cold Spring Harbor Laboratory Press*, 2006.

[25] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Kluwer Academic Publishers, Boston, MA, 1989.

[26] O. Gotoh. An improved algorithm for matching biological sequences. *J Mol Biol*, 162(3) :705–708, 1982.

[27] S. Griffiths-Jones, S. Moxon, M. Marshall, A. Khanna, S. R. Eddy, and A. Bateman. Rfam : annotating non-coding RNAs in complete genomes. *Nucleic Acids Research*, 33(suppl 1) :121–124, 2005.

[28] R Halin. S-functions for graphs. *Journal of Geometry*, 8(2) :171–186, 1976.

[29] Buhm Han, Banu Dost, Vineet Bafna, and Shaojie Zhang. Structural alignment of pseudoknotted RNA. *Journal of Computational Biology*, 15(5) :489–504, 2008.

[30] S.R. Holbrook. RNA structure : the long and the short of it. *Current Opinion in Structural Biology*, 15 :302–308, 2005.

[31] Tao Jiang, Guo-Hui Lin, Bin Ma, and Kaizhong Zhang. A general edit distance between RNA structures. *Journal of Computational Biology*, 9(2) :371–388, 2002.

[32] Arthur B Kahn. Topological sorting of large networks. *Communications of the ACM*, 5(11) :558–562, 1962.

[33] R. Klein and S. Eddy. Rsearch : Finding homologs of single structured RNA sequences.

[34] A. Lamiable, D. Barth, A. Denise, F. Quessette, S. Vial, and E. Westhof. Automated prediction of three-way junction topological families in RNA secondary structures. *Computational Biology and Chemistry*, 37 :1–5, 2012.

[35] N. Leontis, J. Stombaugh, and E. Westhof. Survey and summary : The non-watson–crick base pairs and their associated isostericity matrices. *Nucleic Acids Res*, 16(30) :3497–3531, 2002.

[36] N B Leontis and E Westhof. Geometric nomenclature and classification of RNA base pairs. *RNA*, 7 :499–512, 2001.

[37] N.B. Leontis, J. Stombaugh, and E. Westhof. Motif prediction in ribosomal RNAs - lessons and prospects for automated motif prediction in homologous RNA molecules. *Biochimie*, 84 :961–973, 2002.

[38] N.B. Leontis and E. Westhof. Analysis of RNA motifs. *Curr. Opin. Struct. Biol.*, (13) :300–308, 2003.

[39] A. Lescoute, N.B Leontis, C. Massire, and Westhof E. Recurrent structural RNA motifs, isostericity matrices and sequence alignments. *Nucleic Acids Resh*, 33 :2395–2409, 2005.

[40] A. Lescoute and E. Westhof. The a-minor motifs in the decoding recognition process. *Biochimie*, 88 :993–999, 2006.

[41] Charles Ling, Jin Huang, and Harry Zhang. Auc : A better measure than accuracy in comparing learning algorithms. In *Advances in Artificial Intelligence*, volume 2671, pages 991–991. Springer Berlin / Heidelberg, 2003.

[42] M. Mandal and R.R. Breaker. Gene regulation by riboswitches. *Nat Rev Mol Cell Biol*, 5 :451–463, 2004.

[43] D.H. Mathews, J. Sabina, M. Zuker, and D.H. Turner. Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure. *J. Mol. Biol*, 288 :911–940, 1999.

[44] J. S. McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29(6-7) :1105–1119, 1990.

[45] Eric P. Nawrocki, Diana L. Kolbe, and Sean R. Eddy. Infernal 1.0 : inference of RNA alignments. *Bioinformatics*, 25(10) :1335–1337, 2009.

[46] Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, (48) :443–453, 1970.

[47] L. Noé and G. Kucherov. Yass : enhancing the sensitivity of dna similarity search. *Nucleic Acids Res*, 33(suppl 2) :540–543, 2005.

[48] R. Nussinov, G. Pieczenik, J. R. Griggs, and Kleitman D. J. Algorithms for loop matchings. *SIAM J. Appl. Math.*, 31 :68–82, 1978.

[49] M. Parisien and F. Major. The MC-Fold and MC-Sym pipeline infers RNA structure from sequence data. *Nature*, 452(7183) :51–55, March 2008.

[50] F Qiao and T.R. Cech. Triple-helix structure in telomerase RNA contributes to catalysis. *Nat Struct Mol Biol*, 15(6) :634–640, 2008.

[51] E Rivas and S.R Eddy. A dynamic programming algorithm for RNA structure prediction including pseudoknots. *Journal of Molecular Biology*, 285(5) :2053 – 2068, 1999.

[52] Seymour P.D. Robertson, N. Graph minors. ii. algorithmic aspects of tree-width. *Journal of Algorithms*, 7 :309–322, 1986.

[53] Seymour P.D. Robertson, N. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6) :1305–1317, 1996.

[54] D. Sankoff. Simultaneous solution of the RNA folding, alignment and protosequence problems. *SIAM J. on Appl. Mathematics*, 45(5) :810–825, 1985.

[55] C.L Sarver, M. and. Zirbel, J. Stombaugh, A. Mokdad, and N.B. Leontis. Fr3d : Finding local and composite recurrent structural motifs in RNA 3d structures. *Journal of Mathematical Biology*, 56 :215–252, 2008.

[56] Yinglei Song, Chunmei Liu, Xiuzhen Huang, Russell L. Malmberg, Ying Xu, and Liming Cai. Efficient parameterized algorithms for biopolymer structure-sequence alignment. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3 :423–432, 2006.

[57] J. Stombaugh, C. Zirbel, E. Westhof, and N. Leontis. Frequency and isostericity of RNA base pairs. *Nucleic Acids Res*, 37(7) :2294–2312, 2009.

[58] M. Taufer, A. Licon, R. Araiza, D. Mireles, F. H. D. van Batenburg, A. P. Gultyaev, and M. Leung. Pseudobase++ : an extension of pseudobase for easy searching, formatting and visualization of pseudoknots. *Nucleic Acids Res*, 37(suppl 1) :D127–D135, 2009.

[59] F. H. D. van Batenburg, A. P. Gultyaev, C. W. A. Pleij, J. Ng, and J. Oliehoek. Pseudobase : a database with RNA pseudoknots. *Nucleic Acids Res*, 28(1) :201–204, 2000.

[60] M. S. Waterman. Sequence alignment in the neighborhood of the optimum. *Proc. Natl.*, 80 :3123–3124, 1983.

[61] M. S. Waterman and T. H. Byers. A dynamic programming algorithm to find all solutions in a neighborhood of the optimum. *Math. Biosciences*, 77 :179–188, 1985.

[62] Z. Weinberg and W.L. Ruzzo. Exploiting conserved structure for faster annotation of non-coding RNAs without loss of accuracy. *Bioinformatics*, 20(suppl 1) :334–341, 2004.

[63] T K Wong and S M Yiu. Structural alignment of RNA with triple helix structure. *Journal of Computational Biology*, 19(4) :365–78, 2012.

[64] T.K.F. Wong, T.W. Lam, W.K Sung, B.W.Y Cheung, and S.M. Yiu. Structural alignment of RNA with complex pseudoknot structure. *Journal of Computational Biology*, 18(1), 2011.

[65] Jinbo Xu, Feng Jiao, and Bonnie Berger. A tree-decomposition approach to protein structure prediction. *Computational Systems Bioinformatics Conference, 2005. Proceedings. 2005 IEEE*, (1) :1–2, 2000.

[66] H. Yang, F. Jossinet, N. Leontis, L. Chen, J. Westbrook, H.M. Berman, and E. Westhof. Tools for the automatic identification and classification of RNA base pairs. *Nucleic Acids Res*, 31(13) :3450–3460, 2003.

[67] Kaizhong Zhang, Lusheng Wang, and Bin Ma. Computing similarity between RNA structures. In *CPM-99*, volume 1645 of *Lecture Notes in Computer Science*, pages 281–293, 1999.

[68] M Zuker and P. Stiegler. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucleic Acids Res*, 9(1) :133–148, 1981.