Remerciements

Je remercie en premier lieu Éric Rivals, pour tout le temps qu'il m'a consacré, et pour son aide précieuse tout au long du stage et aussi dans la rédaction de ce rapport. Je remercie également toute l'équipe de BioInformatique du LIRMM, pour m'avoir accueillie, et tout particulièrement Vincent Ranwez et Stéphane Guindon, pour m'avoir hébergée dans leur bureau, pour leur soutien technique, et notamment leurs compétences en Latex.

Table des matières

Résumé

Nous traitons dans ce mémoire le problème de reconstruction d'histoires de répétitions en tandem. Ce problème naturel, nommé TRHIST, fût défini par Benson et Dong dans l'article [**BD99**]. Pour une séquence de répétition en tandem donnée, il consiste à trouver l'histoire de sa construction qui contient le moins de mutations ponctuelles (insertions, délétions, ou substitutions d'un caractère) et d'amplifications d'un motif. Par amplification on entend reproduction côte à côte d'un motif de la séquence (...T $\underline{\text{CGG}}$ AT... $\rightarrow$ ...T $\underline{\text{CGG}}$ $\underline{\text{CGG}}$ $\underline{\text{CGG}}$ AT...). L'opération inverse de l'amplification est appelée contraction.

Benson et Dong étudient des cas restreints : amplifications binaires, et/ou commençant à des positions particulières de la séquence. Ils proposent trois algorithmes heuristiques gloutons et donnent des bornes pour l'un d'eux. Après une présentation détaillée du problème, nous analysons dans le premier chapitre la fonction de coût et la règle de production du motif ancêtre de Benson et Dong, et nous montrons qu'elles ne sont pas adaptées au problème.

Nous étudions, dans la chapitre 2, une autre restriction du problème et le chapitre 3 est consacré au problème TRHIST avec une taille de motif fixe et les positions de début d'amplification restreintes .

Dans le chapitre 2, le problème avec « mutations milieux » admet une forte restriction concernant l'apparition des mutations ponctuelles. Cette hypothèse implique une structure « palindromique » de la séquence. Nous donnons, pour ce problème un algorithme linéaire optimal.

Dans le chapitre 3, le problème avec « mutations bords » admet des restrictions plus réalistes du point de vue biologique. Pour résoudre ce problème, nous nous basons sur la notion de portion. Les portions représentent des intervalles de la séquence. Nous avons développé pour ce problème, un algorithme qui minimise les mutations de motifs ($\neq$ mutations ponctuelles) et les contractions. Cet algorithme est aussi une heuristique pour la minimisation des mutations ponctuelles. Il utilise une procédure de résolution de portions qui est de complexité exponentielle mais dont on peut penser qu'elle est raisonnable en pratique. L'algorithme maximise le nombre de portions compatibles, nous montrons que c'est équivalent avec le problème du stable max dans un graphe de recouvrement. Nous donnons également une méthode alternative, que nous conjecturons optimale, pour maximiser le nombre de portions compatibles.

Il serait intéressant dans l'avenir de vérifier si cette méthode peut s'adapter au problème du stable max dans un graphe d'arcs circulaires. Une perspective autre de ce travail est de cerner la complexité du problème général.

CHAPITRE 1

Introduction

1.1. Description du problème

Aujourd'hui, on dispose de séquences génétiques représentant des portions de l'Acide DésoxyriboNucléique, plus connu sous le nom d'ADN. L'ADN est la molécule qui forme les chromosomes. Ces séquences sont écrites dans un alphabet à quatre lettres {A,C,G,T} qui correspondent aux quatre *bases* chimiques possibles de l'ADN.

On compare deux séquences en construisant un *alignement*, c.a.d. en mettant chacun des caractères en correspondance soit avec un caractère de l'autre séquence, soit avec un symbole “ - “. Un exemple d'alignement, les barres verticales signalent les caractères identiques :

$$\begin{array}{cccccc} \mathrm{A} & \mathrm{G} & \mathrm{G} & \mathrm{T} & \mathrm{C} & \mathrm{A} \\ | & & | & & | & | \\ \mathrm{A} & - & \mathrm{G} & \mathrm{C} & \mathrm{C} & \mathrm{A} \end{array}$$

La mise en correspondance d'une base et du symbole “ - “ est appelée une *insertion* du point de vue de la séquence du haut, ou symétriquement une *délétion* du point de vue de celle du bas. La mise en correspondance de deux bases différentes est une *substitution*.

Les biologistes considèrent ces différences comme des mutations ponctuelles qui sont survenues au cours de l'évolution des deux séquences (au départ elles étaient identiques, des mécanismes biologiques ont changé une base par une autre ou ont inséré une base supplémentaire dans l'une des séquences).

Un autre type de mutations que subit l'ADN est *l'amplification en tandem d'un motif* : un motif de la séquence est recopié côte à côte à l'identique. Voici un exemple :

...T $\underline{\text{CGG}}$ AT... $\rightarrow$ ...T $\underline{\text{CGG}}$ $\underline{\text{CGG}}$ $\underline{\text{CGG}}$ AT...

la flèche signale l'amplification, après celle-ci on observe trois fois le motif CGG là où il n'y en avait qu'un auparavant. Une telle portion de la séquence qui contient plusieurs occurrences d'un motif côte à côte s'appelle *répétition en tandem*.

Durant l'évolution une répétition en tandem peut subir, comme n'importe quelle autre partie de la séquence, des mutations ponctuelles et de nouvelles amplifications, et ce de manière entremêlée. Par exemple, le C du deuxième CGG peut être muté en T, puis le motif TGGCGG peut être amplifié quatre fois, et cela donne :

...T CGG $\underline{\text{TGGCGG}}$ $\underline{\text{TGGCGG}}$ $\underline{\text{TGGCGG}}$ $\underline{\text{TGGCGG}}$ AT...

Le processus peut se renouveler un nombre indéfini de fois. On peut donc parler d'une *histoire des répétitions en tandem*, c.a.d. de la suite ordonnée de mutations ponctuelles et d'amplifications. L'histoire d'une répétition en tandem correspond à la suite de transformations qu'il faut appliquer au motif ancêtre pour obtenir la séquence actuelle. Dans l'exemple ci-dessus, cela donne : "Le motif CGG s'est amplifié 3 fois, puis le C du deuxième CGG a été muté en T, et le motif TGGCGG a été amplifié 4 fois."

Des motifs amplifiés peuvent se trouver "à cheval" sur deux anciens motifs, comme GGC dans l'exemple ci-dessous :

CGA → CGA CGA CGA → CGA CG**G** CGA→ CGA CGG CGG CGG CGG CGA → ...

Ici l'histoire présentée est : " Le motif CGA, motif ancêtre, a été dupliqué trois fois, le A de la deuxième copie a été muté en G, et le motif GGC commençant à la position 5 a été dupliqué 4 fois".

Étant donné une répétition en tandem, le problème que l'on se pose est de retrouver son histoire. Comme la donnée de départ est la répétition, on tente de retracer son histoire à rebours dans le temps, on cherche une suite de transformations qui permet de remonter de la séquence jusqu'au motif ancêtre. Dans l'exemple ci-dessus cela donne : " Les quatre motifs GGC commençant à la position 5 sont contractés en un seul, le G de la sixième position est muté en A, et les trois motifs CGA sont contractés en un seul ", ou, écrit différemment :

CGA CGG CGG CGG CGG CGA → CGA CGG CGA→ CGA CG**A** CGA → CGA

L'opération de *contraction* est l'opposée de l'opération d'amplification. Pour changer le sens d'une histoire, il suffit d'inverser l'ordre des transformations, de remplacer contraction par amplification ou le contraire, et d'inverser le sens des mutations (G → A en A → G par exemple).

A partir d'une séquence, il existe un grand nombre d'histoires qui mènent à un motif ancêtre, ce qui nous intéresse est de trouver la meilleure de ces histoires. Pour comparer la qualité des histoires, on donne un coût à chacune des opérations que peut subir la séquence d'ADN, à savoir les mutations (insertions, délétions et substitutions) et les répétitions, ainsi, on pourra donner un coût total à l'histoire. Suivant le principe de parcimonie, l'histoire optimale à retrouver, est celle qui aura un coût minimum.

1.2. L'article de Benson et Dong

Benson et Dong dans un article de 1999, [**BD99**], proposent plusieurs formalisations plus ou moins restrictives de ce problème et étudient des approches heuristiques. Ils supposent que la séquence est constituée de n copies approximatives d'un motif de base de longueur k et ils alignent cette séquence dans une matrice de n lignes et k colonnes.

Présentation matricielle de l'exemple ci-dessus :

$$\begin{bmatrix} C & G & A \\ C & G & G \\ C & G & G \\ C & G & G \\ C & G & G \\ C & G & A \end{bmatrix}$$

Pour retrouver l'histoire de la séquence, ils appliquent à cette matrice une opération algorithmique de contraction qui remplace plusieurs copies de même longueur par une seule, la *copie consensus*, jusqu'à ce que la matrice n'ait plus qu'une seule ligne. Il y a des règles pour produire cette copie consensus. Chaque contraction admet un coût, déterminé par une fonction de coût. Une série de contractions qui réduit la matrice de départ à une seule ligne représente une histoire possible pour la séquence, et le motif restant dans la matrice est le motif ancêtre, le coût total d'une telle histoire est la somme des coûts des contractions qui la composent. Le coût proposé vise à minimiser seulement le nombre de mutations ponctuelles, mais pas le nombre de contractions.

1.2.1. Les hypothèses

. Dans leur article, les auteurs admettent certaines hypothèses concernant la séquence et le mécanisme d'amplification en tandem :
- La longueur du motif ancêtre est connue.
- On sait aligner la séquence dans une matrice.
- Les amplifications en tandem copient un nombre entier de motifs, c'est à dire qu'il n'y a pas de bout de motif incomplet à la fin d'une répétition, qui correspondrait à un nombre rationnel de motifs.
- La longueur du motif qui est répété est un multiple entier de la longueur du motif ancêtre.
- La donnée du problème est la séquence qui contient les motifs répétés approximativement, on n'a pas besoin de rechercher cette séquence.
- Les seules opérations appliquées à la séquence d'ADN sont les substitutions, délétions, insertions, et amplifications en tandem.

1.2.2. Les algorithmes proposés

. Les auteurs n'étudient pas la complexité, ni ne donnent un algorithme exact du problème général. Ils proposent deux algorithmes gloutons, GREEDY-TRHIST et GREEDY-MANY-TRHIST qui prennent en entrée la séquence présentant des répétitions en tandem, cette séquence étant alignée dans une matrice. Ces algorithmes essayent de trouver la série de contractions de plus faible coût, qui réduit la matrice à une seule ligne.

Ces deux algorithmes fonctionnent sur le même principe. A chaque étape, ils calculent toutes les contractions possibles, choisissent celle qui a le coût le plus faible et l'appliquent à la matrice, c.a.d. qu'ils remplacent un certain nombre de copies par leur copie consensus. Ils répètent ces opérations jusqu'à ce qu'il ne reste plus qu'une seule ligne dans la matrice, représentant le motif ancêtre. Lorsque des contractions ont le même coût, ils privilégient la contraction impliquant le plus de motifs. Les copies contractées ne sont pas nécessairement identiques. L'opération algorithmique de contraction de leurs algorithmes, correspond en fait à une contraction et à la résolution des mutations ponctuelles.

Les deux algorithmes diffèrent seulement par le fait que le premier ne considère que des contractions binaires, alors que le second traite des contractions de n'importe quelle arité. Ils utilisent tous deux, une fonction de coût qui détermine le coût d'une contraction et une règle pour produire la copie consensus issue d'une contraction.

1.2.3. Critiques de la fonction de coût et de la règle de production

. La règle pour produire la copie consensus et la fonction de coût présentées par les auteurs ne nous semblent pas adaptées à certaines instances du problème. Dans cette partie, nous présenterons chacune de ces méthodes, en expliquant les incohérences que nous avons remarquées, et enfin, dans un troisième paragraphe, nous proposerons des solutions pour améliorer ces méthodes.

1.2.3.1. *La règle pour produire la copie consensus*

. Si les copies à contracter ne sont pas identiques, la copie consensus aura, aux positions des différences, des caractères non déterminés. Cette ambiguïté sera représentée par des ensembles de caractères possibles (notés entre { }). Des contractions ultérieures pourront nous aider à résoudre ces indéterminations. La règle proposée par les auteurs, est que l'ensemble de caractères à la position i dans la copie consensus est l'intersection des ensembles de caractères se trouvant aux positions i dans les copies à contracter si cette intersection est non vide, et l'union de ces ensembles de caractères sinon. Quelques exemples :

$$\begin{bmatrix} A & C & T & T & A \\ A & C & T & T & C \\ A & C & T & T & A \\ A & C & T & T & A \end{bmatrix} \text{ ces 4 copies sont contractées en } \begin{bmatrix} A & C & T & T & \{A/C\} \end{bmatrix}$$

$$\begin{bmatrix} A & C & T & T & \{A/C\} \\ \{G/C\} & C & T & T & A \\ C & C & T & T & A \end{bmatrix} \text{ ces 3 copies sont contractées en } \begin{bmatrix} \{A/G/C\} & C & T & T & A \end{bmatrix}$$

Dans ce dernier exemple on peut constater qu'un des ensembles ambigus, {A / C}, a été résolu dans la contraction, tandis que l'autre {G / C}, a été agrandi par l'opération d'union.

Ce qui n'est pas cohérent dans cette règle c'est que lors de la formation d'un ensemble de caractères ambigu à la position i de la copie consensus, la règle de production ne tient pas compte de la proportion des caractères présents aux positions i dans les copies à contracter. Il en résulte que la contraction de 20 copies contenant à la position i, 10 A et 10 T par exemple, aura le même ensemble de caractères ambigu à la position i de la copie consensus, que le résultat de la contraction de ces mêmes 20 copies, où aux positions i il y aurait cette fois 19 A et 1 T. Cela nous empêche donc, dans le second cas, de déduire directement que le T provient d'une mutation.

1.2.3.2. *La fonction de coût*
. Le coût d'une contraction est égal aux nombres de mutations ponctuelles qu'il faut appliquer aux copies à contracter pour les rendre identiques. Le coût d'une contraction est donc le nombre d'ensembles ambigus formés ou agrandis dans la copie consensus.

EXEMPLE 1.2.1. Cette contraction a un coût de 3.

$$\begin{bmatrix} A & \{C/G\} & T & T & A \\ G & C & A & T & \{C/G\} \\ A & \{C/G\} & T & T & A \\ A & C & A & T & \{A/C\} \end{bmatrix} \longrightarrow \begin{bmatrix} \{A/G\} & C & \{T/A\} & T & \{A/C/G\} \end{bmatrix}$$

Leur fonction de coût donne un coût 0 aux contractions dont toutes les copies à contracter sont identiques, ce qui semble en contradiction avec le principe de parcimonie, en effet :

EXEMPLE 1.2.2. *Les motifs notés entre < >, sont les motifs à contracter.*

$$\begin{bmatrix} <C & G & G> \\ <C & G & G> \\ <C & G & G> \\ C & A & G \\ C & G & G \\ C & G & G \\ C & G & G \end{bmatrix} \longrightarrow \begin{bmatrix} C & G & G \\ C & A & G \\ <C & G & G> \\ <C & G & G> \\ <C & G & G> \end{bmatrix} \longrightarrow \begin{bmatrix} <C & G & G> \\ <C & A & G> \\ <C & G & G> \end{bmatrix} \longrightarrow \begin{bmatrix} C & \{A/G\} & G \end{bmatrix}$$

a un coût de 1 de même que

$$\begin{bmatrix} <C & G & G> \\ <C & G & G> \\ <C & G & G> \\ <C & A & G> \\ <C & G & G> \\ <C & G & G> \\ <C & G & G> \end{bmatrix} \longrightarrow \begin{bmatrix} C & \{A/G\} & G \end{bmatrix}$$

Ceci est dû au fait que les deux premières contractions de la série d'en haut ont un coût nul.

Le second problème de la fonction de coût donnée par les auteurs, vient du fait que l'on donne un coût unitaire aux ensembles de caractères ambigus que l'on forme sans tenir compte de ce qui les composent.

EXEMPLE 1.2.3. Les deux contractions suivantes ont le même coût 1.

$$\begin{bmatrix} G & C & T & A \\ G & C & A & A \\ G & C & T & A \end{bmatrix} \longrightarrow \begin{bmatrix} G & C & \{T/A\} & A \end{bmatrix}$$
$$\begin{bmatrix} G & C & T & A \\ G & C & A & A \\ G & C & C & A \end{bmatrix} \longrightarrow \begin{bmatrix} G & C & \{T/A/C\} & A \end{bmatrix}$$

Cependant, pour arriver à un motif consensus dans le premier exemple, on aura besoin d'une seule mutation (A $\rightarrow$ T) sur le deuxième motif, alors que dans le deuxième exemple, on aura besoin, quel que soit le consensus, de faire deux mutations. Ces deux contractions ne devraient pas avoir le même coût.

Pour des exemples contenant beaucoup de motifs, ou la disproportion entre les différents caractères d'un ensemble ambigu est grande, on peut arriver à une copie ancêtre aberrante.

Considérons la séquence suivante, et appliquons lui la méthode des auteurs, à savoir toujours choisir la contraction de coût minimum :

(AT**T**)*AT**A** (AT**T**)*(AG**C**)*AG**A** (AG**C**)*

Contraction de la première série de ATT : coût = 0
ATT ATA (ATT)*(AGC)*AGA (AGC)*

Contraction de la deuxième série de ATT : coût = 0
ATT ATA ATT (AGC)*AGA (AGC)*

Contraction de la première série de AGC : coût = 0
ATT ATA ATT AGC AGA (AGC)*

Contraction de la deuxième série de AGC : coût = 0
ATT ATA ATT AGC AGA AGC

Contraction des 3 premiers motifs : coût = 1
AT{T/A} AGC AGA AGC

Contraction des 3 derniers motifs : coût = 1
AT{T/A} AG{C/A}

Contraction finale : coût = 1, pour la création de l'ensemble {T/G}, en outre, la résolution de {T/A} et {C/A} en A coûte 0.

Cela donne : A{T/G}**A** qui est censé être le motif de départ.

Si l'on considère le troisième caractère des motifs au départ, on remarque qu'il y a beaucoup de T, de C, et seulement 2 A. Or, la méthode des auteurs, donne une copie ancêtre, où à la position 3, on retrouve le caractère A, ce qui est incohérent. En effet, en mettant un T ou un C à cette position, on aurait besoin de moins de mutations pour engendrer la séquence.

Le choix de la fonction de coût s'oppose à l'objectif de calculer l'histoire la plus parcimonieuse.

1.2.3.3. *Notre solution*

. **Nouvelle règle de production de la copie consensus :**

Supposons que l'on veuille contracter m occurrences d'un motif de taille k. Pour déterminer quels seront les caractères de la copie consensus, on répète le processus suivant pour les k positions de la copie :

1. Obtention d'un score pour chaque lettre : On compte le nombre d'apparitions pour chaque caractère de l'alphabet { A, C, G, T, - } à la position i dans les m occurrences, on prend même en compte leurs apparitions dans un ensemble ambigu. Chaque lettre obtient ainsi un score.
2. Détermination du caractère à la position i de la copie consensus : A partir des scores, on détermine quel(s) est (sont) le(s) caractère(s) majoritaire(s) :

- Si il n'y en a qu'un, c'est lui qui apparaîtra à la position i de la copie consensus.
- Sinon, on crée un ensemble de caractères ambigus contenant les différents caractères majoritaires.

Nouvelle fonction de coût :

Dans un premier temps, on donne un coût α aux contractions, pour éviter qu'une contraction ait un coût nul, ce qui est, comme on l'a vu précédemment, générateur d'aberration. Les mutations ont un coût β . Pour trouver le coût d'une contraction de m motifs, on commence par calculer un coût annexe c_i, pour chaque position i $\in$ {1, . . ., k} de la copie consensus.

Ce coût annexe correspond au nombre de mutations qu'il faudrait faire sur les positions i des m copies pour arriver à un même caractère, celui de la position i dans la copie consensus.

A chaque position i de la copie consensus, il y a soit un caractère, soit un ensemble de caractères. Si il y a un caractère, il faudra compter une mutation par copie où ce caractère n'apparaît pas à la position i. Si il y a un ensemble, c'est que les caractères le composant apparaissaient un même nombre de fois dans les m copies, mais quel que soit le caractère Υ, choisi ultérieurement, il faudra compter une mutation par caractère différent de Υ, présent à la position i. Pour faire le compte, on choisit un caractère quelconque de cet ensemble, (cela n'a pas d'importance car ils donneront tous le même résultat), et on dit que c'est lui le consensus (juste pour le calcul du coût).

Il suffit ensuite de compter le nombre d'occurrences, parmi les m copies à contracter, où le caractère consensus, choisi ou légitime, n'apparaît pas à la position i.

D'où $c_i = \beta$ * (nombre d'occurrences où le consensus n'apparaît pas).

Pour calculer le coût total, C, de la contraction, la formule est la suivante :

$$C = \sum_{i=1}^{k} c_i + \alpha$$

Retour à l'exemple 1.2.1 :

$$\begin{bmatrix} A & \{C/G\} & T & T & A \\ G & C & A & T & \{C/G\} \\ A & \{C/G\} & T & T & A \\ A & C & A & T & \{A/C\} \end{bmatrix} \longrightarrow \begin{bmatrix} A & C & \{T/A\} & T & A \end{bmatrix}$$

Dans la première colonne, le caractère A est majoritaire (3 A contre un seul G), c'est donc A qui apparaîtra à la première position de la copie consensus. De même dans la deuxième colonne, c'est le caractère C qui est majoritaire (4 C contre 2 G). Dans la troisième colonne, il y a deux caractères majoritaires, T et A, ayant chacun un score de 2, donc on va créer un ensemble de caractères ambigu {T / A} à la position 3 de la copie consensus. Pas de problème pour la quatrième colonne dans laquelle il n'y a que des T. Et en ce qui concerne la cinquième colonne, A est majoritaire (3 A, 2 C et 1 G), donc c'est lui qui sera à la position 5 de la copie consensus.

Calculons maintenant les coûts intermédiaires :

1. Le caractère consensus A apparaît dans toutes les copies à contracter à la position 1, sauf une, d'où $\underline{c_1 = \beta}$.
2. C apparaît dans toutes les copies à la position 2, d'où $\underline{c_2 = 0}$.
3. Que l'on prenne A ou T comme consensus, il y aura 2 copies dans lequel il n'apparaîtra pas à la position 3, d'où $\underline{c_3 = 2\beta}$.
4. Il n'y a aucune copie où T n'apparaît pas, donc $c\underline{_4 = 0}$.
5. Il y a une copie où A n'apparaît pas à la position 5, donc $\underline{c_5 = \beta}$.

d'où le coût total de cette contraction est de $\underline{\alpha + 4\beta}$.

Retour à l'exemple 1.2.2 :

$$\begin{bmatrix} <C & G & G> \\ <C & G & G> \\ <C & G & G> \\ C & A & G \\ C & G & G \\ C & G & G \\ C & G & G \end{bmatrix} \longrightarrow \begin{bmatrix} C & G & G \\ C & A & G \\ <C & G & G> \\ <C & G & G> \\ <C & G & G> \end{bmatrix} \longrightarrow \begin{bmatrix} <C & G & G> \\ <C & A & G> \\ <C & G & G> \end{bmatrix} \longrightarrow \begin{bmatrix} C & \{A/G\} & G \end{bmatrix}$$

Cette suite de contractions a maintenant un coût de $\alpha + \alpha + (\alpha + \beta)$ soit $3\alpha + \beta$, alors que

$$\begin{bmatrix} <C & G & G> \\ <C & G & G> \\ <C & G & G> \\ <C & A & G> \\ <C & G & G> \\ <C & G & G> \\ <C & G & G> \end{bmatrix} \longrightarrow \begin{bmatrix} C & \{A/G\} & G \end{bmatrix}$$

a un coût de $\alpha + \beta$, ce qui la rend moins chère et paraît correct au sens de la parcimonie.

Retour à l'exemple 1.2.3 :

De même, les deux contractions suivantes n'ont plus le même coût, et dans la première, il n'y a plus formation d'ensemble ambigu puisque T est majoritaire :

$$\begin{bmatrix} G & C & T & A \\ G & C & A & A \\ G & C & T & A \end{bmatrix} \longrightarrow \begin{bmatrix} G & C & T & A \end{bmatrix}$$

La contraction ci-dessus a un coût de $\alpha + \beta$, alors que celle d'en dessous a un coût de $\alpha + 2\beta$.

$$\begin{bmatrix} G & C & T & A \\ G & C & A & A \\ G & C & C & A \end{bmatrix} \longrightarrow \begin{bmatrix} G & C & \{T/A/C\} & A \end{bmatrix}$$

1.3. Les différentes restrictions du problème

Il y a plusieurs manières de restreindre le problème, les restrictions abordées dans l'article de Benson et Dong portent principalement sur l'arité des contractions (binaires ou n-aires), le début de la première copie à contracter (qui est soit dans la colonne 1 de la matrice, soit n'importe où) et la taille des copies à contracter (qui est soit de même longueur que le motif ancêtre, soit un multiple de cette longueur).

Les auteurs donnent des bornes de coût en nombre de mutation pour le problème restreint GREEDY-TRHIST-RESTRICTED qui traite des contractions binaires, dont la taille des copies à contracter est une fois la taille du motif de base, et dont la limite gauche de ces copies se situe dans la colonne 1 de la matrice.

Pour simplifier la compréhension, on va désormais représenter chaque motif par une lettre, les motifs identiques sont représentés par la même lettre. Soit u, v, w des motifs de taille k, on peut alors écrire une séquence comportant uniquement ces motifs sous la forme (u*v*w*)*.

Nous avons pensé à un autre type de restriction, qui est d'imposer que les mutations de caractères n'aient lieu que sur le motif du milieu d'une suite de trois motifs identiques. Si l'on a une séquence S = v u $\underline{\text{u u u u}}$ u v $\underline{\text{v}}$ v w $\underline{\text{w w}}$ w v $\underline{\text{v}}$ v u u v, on considère que seuls les caractères des motifs soulignés pourront subir une mutation, il ne peut pas y avoir de mutations qui se produisent dans des motifs situés "au bord" d'une suite de motifs identiques. Nous appellerons cette restriction "*mutations milieux*" en opposition à "*mutations bords*" qui permet aux mutations de se produire n'importe où, y compris au bord d'une suite de motifs identiques .

1.4. Analyse et propriétés du problème général

Dans cette section, nous donnons une définition, et une propriété de base concernant le problème général, et introduisons des notations.

Nous cherchons à reconstruire l'histoire de la formation d'une séquence, sectionnée en motifs, par mutations et expansions. Cela revient à obtenir l'histoire des mutations entre les motifs, c.a.d. savoir quel motif à engendrer quel autre, suite à une mutation. Cette relation de parenté entre motifs définit un temps de première apparition d'un motif dans la séquence. Le temps d'apparition du motif ancêtre est 1.

PROPOSITION 1.4.1. *Si l'on note le temps d'apparition d'un motif fils t, et le temps d'apparition de son père t', alors on a : t = t' + 1.*

La séquence permet de constater, par la similitude de motifs, que plusieurs motifs descendent d'un même père. Cependant, elle ne permet pas d'ordonner dans le temps, les apparitions des différents fils. On a donc une relation d'ordre partiel. Ainsi il peut y avoir plusieurs histoires équivalentes, différant par l'ordre d'apparition des fils d'un même père.

NOTATION 1.4.2.

$$\text{u} \rightarrow \text{v, } o\grave{u} \text{ u et v sont des motifs, signifie que } \begin{cases} \text{u est le père de v,} \\ \text{et que u se transforme en v.} \end{cases}$$

Nous allons maintenant définir la distance que nous emploierons pour comparer deux motifs.

DÉFINITION 1.4.3. Distance entre motifs.
La distance entre deux chaînes de caractères, définies sur l'alphabet {A, C, G, T, -}, est le nombre de positions auxquelles elles diffèrent.

Ainsi le coût, noté c, de chaque mutation est donné par la distance entre les motifs impliqués, c (A $\rightarrow$ B) est égal à la distance entre A et B.

On peut définir sur les séquences une opération θ de factorisation. Cela nous servira, par la suite, à les traiter de manière plus simple.

DÉFINITION 1.4.4. Séquence factorisée.
Si S est une séquence où chaque motif est représenté par une lettre, alors θ (S) est la séquence S où toutes les lettres identiques adjacentes ont été contractées en une seule.

EXEMPLE 1.4.5. Factorisation de séquence
Si S = $uuuuttuuxxxxwwwwxxxxxu$, alors θ (S) = $u^4t^2u^2x^4w^4x^5u^1$

Voici l'algorithme θ de factorisation, la variable c compte le nombre d'occurrences côte à côte d'un même motif, p indique la prochaine position à remplir dans le tableau F.

Algorithme 1: θ

Données : S contenant n motifs
Résultat : Un tableau F de 2 ligne, la première représentant la séquence sous forme factorisée, la seconde, les facteurs associés à chaque motif.

```
c := 0 ;
p := 1 ;
i := 1 ;
tant que i < n faire
    si S[i]=S[i+1] alors
        c := c+1 ;
    fin
    sinon
        F[p, 1] := S[i] ;
        F[p, 2] := c ;
        c := 0 ;
        p := p+1 ;
    fin
fin
F[p, 1] := S[n] ;
F[p, 2] := c +1;
```

Dans la suite du mémoire, nous allons aborder deux problèmes restreints, dans le but de pouvoir trouver des solutions exactes. Dans les deux problèmes, nous considérerons que les contractions ne sont pas forcément binaires, que la taille des motifs à contracter est une fois la taille du motif de base, et que la limite gauche de ces copies se situe dans la colonne 1 de la matrice. Dans le premier cas, on autorisera seulement les mutations milieux, tandis que dans le second, on pourra faire des mutations bords.

CHAPITRE 2

Mutations milieux

Dans ce chapitre, nous faisons l'hypothèse des mutations milieux : une mutation ne peut se produire que sur le motif central d'une suite de trois motifs identiques côte à côte. Cette hypothèse, qui restreint le problème général, n'est pas réaliste au point de vue biologique. Cependant, les amplifications produisant un grand nombre de copies du même motif, une mutation ponctuelle a une grande probabilité de modifier un motif entouré de deux motifs identiques. (A A A → A B A, où A et B sont des motifs). Comme nous le verrons plus bas, cette hypothèse conserve une structure "palindromique" des séquences.

Dans on premier temps, nous décrirons le problème, nous donnerons ensuite quelques propriétés de base, puis nous exposerons l'algorithme de résolution optimal pour ce problème, suivi de deux représentations graphiques de la solution, nous ferons enfin une remarque concernant l'indécidabilité de certaines séquences.

2.1. Description du problème

Nous rappelons dans ce paragraphe les caractéristiques du problème. Nous allons considérer que les répétitions ne sont pas forcément binaires, que la taille des motifs répétés est une fois la taille du motif de base et que la limite gauche de ces motifs se situe dans la colonne 1 de la matrice. De plus, les mutations ne peuvent se produire qu'entre deux motifs identiques.

EXEMPLE 2.1.1. Voici une suite de transformations possible :
CGG → CGG CGG CGG → CGG **AGA** CGG → CGG AGA AGA CGG

Un autre exemple.

EXEMPLE 2.1.2. Voici une suite de transformations impossible :
CGG → CGG CGG CGG → CGG **AGA** CGG → CGG AGG AGA CGG
Cette suite de transformation est impossible car si on aligne cette séquence dans une matrice, la position gauche du deuxième motif amplifié, GAG, ne se situe pas dans la colonne 1. De plus, les premier et troisième motifs CGG, ne peuvent pas subir de mutations, seul celui du milieu le peut.

On va maintenant représenter chaque motif par une lettre, on peut alors écrire la de transformations de l'exemple 2.1.1 de la manière suivante :

EXEMPLE 2.1.3. u → u u u → u v u → u v v u   où u = CGG et v = AGA

On peut donc parler de mutations de motifs, ces mutations ne correspondront pas forcément à des mutations de caractères. Dans l'exemple 2.1.3, on a la mutation u → v, cette mutation de motif correspond à deux mutations de nucléotides : C → A et G → A. Dans la suite, lorsque l'on parlera de mutations, ce seront des mutations de motifs, sauf précisions contraires.

2.1.1. Grammaire

. On peut définir une grammaire qui génère des séquences conformes aux restrictions données.

Soit $u_i : 1 \leq i \leq |\sum|^k$ les différents mots de longueur k. Les u_i sont les terminaux de la grammaire.

A chaque u_i est associé un non-terminal U_i.

S est l'unique non-terminal initial et il se dérive en n'importe quel U_i et en ε :

$$(1)\ S \rightarrow U_i \mid \varepsilon, \forall i$$

Chaque U_i se transforme en son non-terminal associé :

$$(2)\ U_i \rightarrow u_i, \forall i$$

ou bien crée une structure de palindrome avec un non-terminal différent :

$$(3)\ U_i \rightarrow U_i\ U_j\ U_i, \forall i, \forall j \neq i.$$

Si l'on définit simplement ces 3 règles génériques, la grammaire génère des séquences sous forme contractées. Pour que l'on puisse observer des amplifications à l'intérieur de la séquence, il faut rajouter la règle générique suivante :

$$(4)\ U_i \rightarrow U_i\ U_i\ \forall i$$

<u>Un exemple</u> :

On part d'un motif ancêtre u et on génère une séquence d'ADN suivant les règles de la grammaire, énoncées ci-dessus.

Le chiffre entre parenthèses correspond au numéro de la règle que l'on applique. Les lettres majuscules représentent les non-terminaux, alors que les lettres minuscules, les terminaux. Les motifs sont soulignés lorsqu'ils apparaissent pour la première fois. Les motifs en gras sont ceux sur lesquels on applique la règle.

1. (1) S → U
2. (3) **U** → U <u>V</u> U
3. (3) U **V** U → U V <u>W</u> V U
4. (3) U V W **V** U → U V W V <u>Y</u> V U
5. (3) U V **W** V Y V U → U V W <u>X</u> W V Y V U
6. (3) U V W X **W** V Y V U → U V W X W <u>U</u> W V Y V U
7. (2) U V W X W U W V Y V U → u v w x w u w v y v u

C'est la règle 3 de la grammaire qui génère les palindromes, à chaque application de cette règle, un nouveau motif apparaît entre deux motifs identiques, ce qui forme un palindrome de taille 3 au moins, comme U V U à la ligne 2 par exemple.

A la ligne 3, on a un palindrome de taille 5. A la ligne 4, il se forme le palindrome V Y V adjacent au palindrome V W V.

2.2. Propriétés de base

La restriction des mutations milieux induit quelques propriétés en ce qui concerne la parenté entre motifs.

PROPOSITION 2.2.1. *Si deux motifs u et v sont côte à côte dans la séquence, avec u ≠ v, alors, soit u → v, soit v → u.*

En effet, lorsqu'un nouveau motif v, apparaît dans la séquence, ses deux voisins directs sont des motifs représentant son père. Ces motifs voisins, que l'on notera u, ne peuvent pas être altérés par une mutation, par contre, si v s'amplifie, et que la suite de motifs identiques obtenue subit une mutation, v $\rightarrow$ w par exemple, alors on peut trouver dans la séquence, w v u qui se suivent. Quoi qu'il arrive, si v est entouré de motifs différents, alors au moins un des deux représente son père.

Nous allons maintenant discuter de la manière, de plus faible coût, de résoudre un palindrome. Lorsqu'en parcourant la séquence, on repère un palindrome de longueur 3, u v u, il peut y avoir plusieurs manières de le résoudre. Cela dépend du motif x qui suit ce palindrome. Si x $\neq$ v, la résolution possible la moins coûteuse est de dire que u est le père de v. Par contre, si x = v, il y a plusieurs résolutions possibles dont le coût dépend du nombre d'occurrences des motifs.

Soit la suite de transformations suivante :

u $\rightarrow$ $\underline{\text{u u u}}$ $\rightarrow$ u **v** u $\rightarrow$ u $\underline{\text{v v v v v v v v}}$ u $\rightarrow$ u v v v **u** v v v u

S = u v v v **u** v v v u
θ (S) u^1 v^3 u^1 v^3 u^1.

A partir de S, on peut donner deux histoires :

Histoire 1 :
Contraction des premiers v.
Mutation v $\rightarrow$ u.
Contraction des seconds v.
Mutation v $\rightarrow$ u.
Contraction des u.

Histoire 2 :
Mutation u $\rightarrow$ v.
Contraction des v.
Mutation v $\rightarrow$ u.
Contraction des u.

Dans le deuxième cas, on fait une contraction de moins, ceci est dû au fait que la mutation du u du milieu ne nécessite pas de contraction préalable.

Pour choisir la bonne histoire, on procède de la manière suivante :

Si on a un palindrome u^i v^j u^k, avec j > 1 et k = 1, il faut regarder le motif qui suit,
– Si c'est un v, on choisi la seconde histoire.
– Sinon, on choisi la première.

Les deux cas de figure se résument en la proposition suivante.

PROPOSITION 2.2.2. *Lorsque en parcourant une séquence factorisée, on repère un facteur de type u^i v^j u^k, la résolution la moins coûteuse (en nombre de mutations, ET en nombre de contractions) de ce facteur est donné par la règle suivante :*

Soit x le motif suivant ce facteur dans la séquence.

Si j > 1 et k = 1, et si x = v, alors contracter les k u, et muter le u en v.

Sinon contracter les j v, et muter v en u.

2.3. Algorithme de résolution

Calculer l'histoire est équivalent à calculer le temps d'apparition de chaque motif, i.e., un tableau T représentant une application qui à chaque motif donne sa date d'apparition.

L'algorithme travaille sur une séquence factorisée.

On utilise dans l'algorithme une pile P. Le sommet de la pile correspond à l'avant avant dernier motif lu, l'avant dernier motif est stocké dans la variable prec.

REMARQUE 2.3.1. Les deux premier motifs sont toujours numérotés 1 et 2.

Algorithme 2: Mutations milieux

Entrée : F = θ(S) de longueur m.

Sortie : Un vecteur T faisant correspondre à chaque motif, un temps d'apparition.

```
PUSH (1, P) ;
T[1] := 1 ;
T[2] := 2 ;
prec := 2 ;
i := 3 ;
i++ ;
tant que (i ≤ m) faire
    si (F[1, i] = F[1, TOP(P)]) alors
        si (i < n) et (F[1, i+1] = F[1, prec]) et (F[2, prec] > 1) et (F[2, i] = 1) alors
            T[i] := T[prec] + 1 ;
            T[i+1] := T[prec] ;
            prec := i + 1 ;
            i ++ ;
        fin
        sinon
            T[i] := T[TOP(P)] ;
            prec := POP(P) ;
        fin
    fin
    sinon
        T[i] := T[prec] + 1 ;
        PUSH(prec, P) ;
        prec := i ;
    fin
    i++ ;
fin
return T;
```

2.3.1. Génération d'une histoire optimale de la séquence à partir de T

. Dans un premier temps, l'histoire que l'on construit débute par la séquence et fini par le motif ancêtre.

U	V	W	W	X	X	W	U	W	V	Y	V	V	V	U
1	2	3	3	4	4	3	4	3	2	3	2	2	2	1

FIG. 2.4.1 – Temps d'apparition.

Lorsque l'on a les temps d'apparition de chaque motif on connaît l'histoire des mutations de la séquence. En effet, ce sont les centres de palindromes qui ont les temps les plus récents, ce sera eux que l'on mutera en premier. Un centre de palindrome, c, est entouré de motifs identiques, dont le temps d'apparition est inférieur. Ce sont donc eux les pères de c. Le résultat de la mutation de c est donc donné par la structure de la séquence.

PROPOSITION 2.3.2. *Une fois la mutation de c réalisée, sont père devient un centre de palindrome ou alors son temps d'apparition n'est pas le plus élevé de la séquence.*

On peut donc résoudre toutes les mutations, en les classant par ordre de temps décroissant. On ajoute les contractions aux endroits nécessaires, i.e., juste avant une mutation de motif dont plusieurs occurrences apparaissent côte à côte, et à la fin. On obtient ainsi une histoire de la séquence.

2.3.2. Complexité

. L'algorithme présenté ci-dessus a une complexité en θ(n), en temps et en espace.

On parcourt une fois la séquence, on empile et on dépile chaque position de motif au plus une fois. La taille maximale de la pile est au plus n/2, car on empile des positions tant qu'elles se trouvent sur une partie gauche d'un palindrome et on les dépile quand on atteint sa partie droite, le plus grand palindrome peut être de taille n, donc sa partie droite, de taille n/2.

D'où un traitement complet en θ(n).

2.3.3. Preuve

. Les propriétés 2.2.1 et 2.2.2 nous donnent directement la preuve de l'algorithme.

2.4. Représentation des solutions

Nous avons développer deux manières graphiques de visualiser les résultats de notre algorithme, la première à l'aide de lignes brisées, et la seconde avec des arcs.

2.4.1. Représentation avec une ligne brisée

. Soit la séquence S = u v v w w x w u w v y v v v u, on obtient, à la sortie de l'algorithme 2, un temps d'apparition pour chacun des motifs de cette séquence (cf. figure 2.4.1).

On peut représenter S par une ligne brisée. Cette représentation peut s'obtenir directement en sortie de l'algorithme 1, en rajoutant la procédure d'affichage décrite par l'algorithme 3, qui prend en entrée le vecteur T des temps d'apparition et la séquence sous forme factorisée F, F contient m motifs.

Le résultat graphique obtenu avec cette procédure d'affichage sur la séquence u v v w w x w u w v y v v v u, est représenté par la figure 2.4.2.

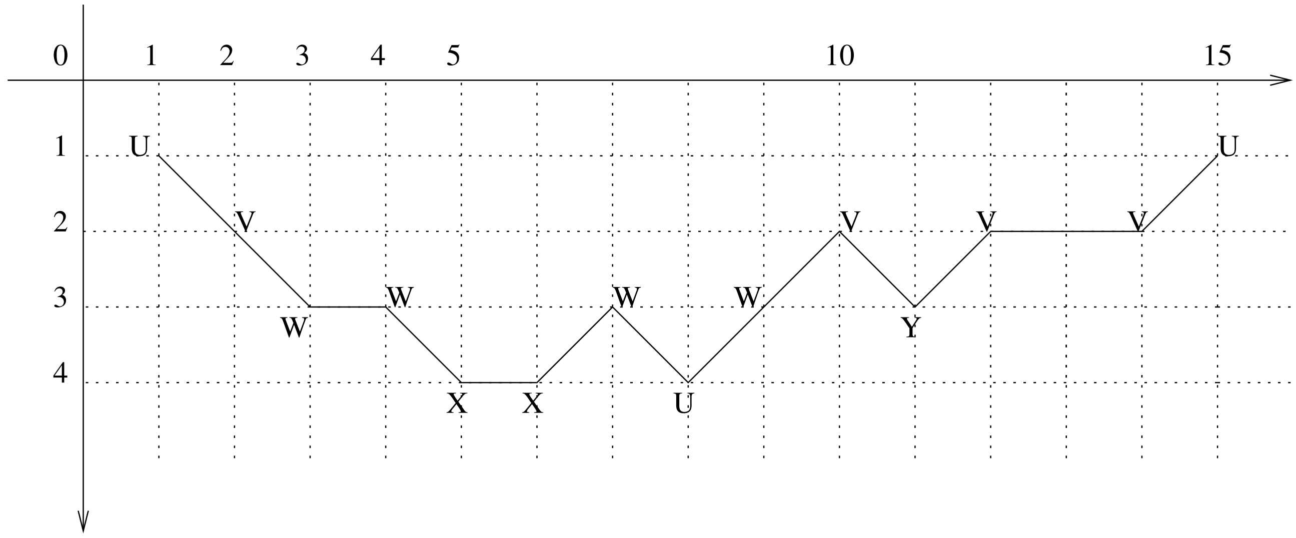


FIG. 2.4.2 – Représentation avec une ligne brisée.

Algorithme 3: Représentation par une ligne brisée

Données : F, T

Résultat : La représentation avec une ligne brisée.

Coord := (1, 1);
Abs := 1 ;
pour *i = 1 à m* **faire**
  X := (Abs, T[i]) ;
  AfficheSegment(Coord, X) ;
  AfficheLettre(F[1, i], X) ;
  si *F[2, i] > 1* **alors**
    Abs := Abs + F[2, i] ;
    Coord := (Abs, T[i]) ;
    AfficheSegment(X, Coord) ;
    AfficheLettre(F[1, i], Coord) ;
  fin
  sinon
    Coord := X ;
  fin
  Abs := Abs + 1
fin

---

où AFFICHESEGMENT(X, Y) affiche un segment dont les extrémités sont les points de coordonnées X et Y, et où AFFICHELETTRE(L, X), affiche la lettre L à la position X

2.4.2. Représentation avec des arcs

. La représentation avec des arcs ne peut se faire que sur des séquences ayant subies l'opération θ de factorisation. La séquence du paragraphe précédent devient u v w x w u w v y v u, et peut alors être représentée par le schéma de la figure 2.4.3.

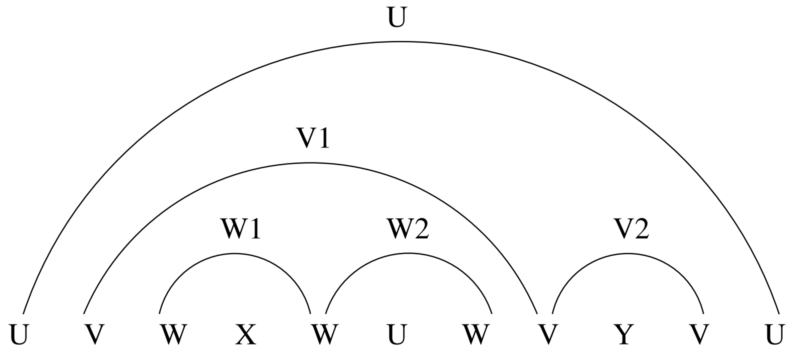


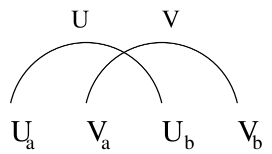
FIG. 2.4.3 – Représentation avec des arcs.

Les extrémités d'un arc délimitent une amplification de motifs dans laquelle au moins une mutation s'est produite. Un arc est assimilé à la première mutation qui a eu lieu entre ses extrémités. L'arc V1 représente la mutation v $\rightarrow$ w.

Un arc a toujours deux motifs identiques à ses extrémités, on étiquette l'arc par la lettre de ce motif. Si deux arcs ont le même nom, alors on leur rajoute simplement un numéro, correspondant à l'ordre dans lequel ils apparaissent quand on parcourt la séquence de gauche à droite.

Lorsque la séquence est sous forme factorisée, c.a.d. qu'il n'y a pas deux copies identiques à côté, la représentation avec des arcs est faite de manière à ce que tous les motifs sont support d'un arc sauf ceux qui apparaissent au milieu d'un palindrome exact (dans cet exemple : x, y et le u du milieu). En effet, on est sûr que ces motifs proviennent d'une mutation de leur voisin.

Il n'y a aucun arcs qui s'intersectent, ceci est dû au fait que les mutations se produisent uniquement entre deux motifs identiques. Une configuration où les arcs se croiseraient est impossible avec les hypothèses que l'on s'est donné.



L'histoire représentée par ce schéma est impossible, les deux arcs qui se croisent, U et V, sont incompatibles : d'après l'arc U, v_a est issu de la mutation d'un u qui se trouvait entre u_a et u_b, et l'arc V, nous dit que u_b est issu de la mutation d'un v qui était entre v_a et v_b, le premier arc nous dit que u_b est arrivé avant v_a, et le deuxième nous dit le contraire, donc les deux arcs U et V ne sont pas compatibles.

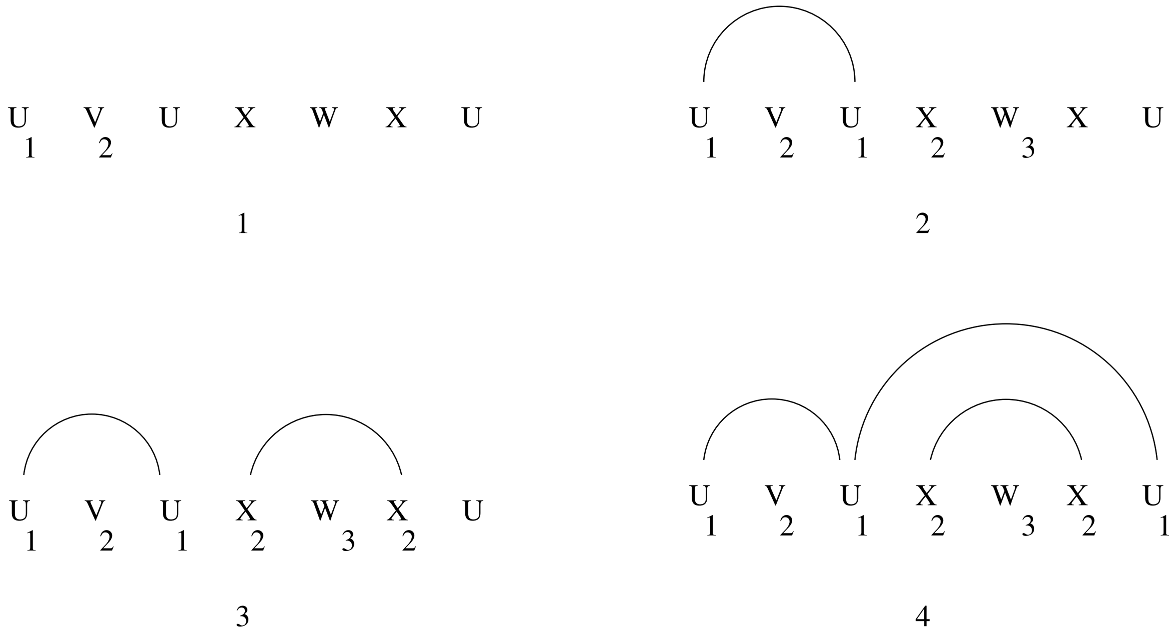


FIG. 2.4.4 – Application de l'algorithme 2 modifié.

2.4.2.1. *Construction des arcs*

. Pour construire la représentation, il suffit de rajouter dans l'algorithme 2, quelques lignes. Au fur et à mesure de la numérotation, on va construire un ensemble A, contenant des paires qui représentent les extrémités d'un arc. A la fin de l'algorithme, il suffit d'appeler une fonction dessine, qui a comme paramètre F, la séquence factorisée, et A, et qui donne la représentation de la séquence avec des arcs.

Un arc est construit chaque fois que le motif courant est identique au motif du sommet de pile.

Lignes à rajouter dans l'algorithme 2 :
entre la ligne 12 et la ligne 13 : A := A ∪ (prec, i+1) ;
entre la ligne 14 et la ligne 15 : A := A ∪ (TOP(P), i) ;
à la fin : dessine(F, A) ;

Un exemple d'application de l'algorithme 2, modifié, sur la séquence u v u x w x u, est donné dans la figure 2.4.4. Les numéros en indice des motifs, correspondent à leur temps d'apparition.

2.5. Histoires non décidables

Plusieurs histoires différentes peuvent mener d'un même motif ancêtre à la même séquence actuelle. Avec la restriction des mutations milieux, on sait simplement que les motifs qui apparaissent aux bords de cette séquence représentent le motif ancêtre.

PROPOSITION 2.5.1. *Le motif ancêtre d'une séquence est le premier motif.*

Lorsque l'on a une séquence, il n'est pas toujours possible de décider quelle histoire est la plus réaliste, même en se donnant un critère de coût à minimiser.

U X U X U   U X U X U

L'histoire de la première séquence serait : "Un motif de départ u a été amplifié 5 fois, puis les u des deuxième et quatrième positions ont été mutés en x". L'histoire de la deuxième séquence serait : "Un motif de départ u a été amplifié 3 fois, puis le u du milieu s'est muté en x, qui a été amplifié 3 fois, et le x du milieu s'est muté en u". Ces deux histoires ne mettent pas en jeu les même mutations.

De plus, on remarque, dans la première séquence, que l'on ne peut pas déterminer l'ordre dans lequel se sont produites les deux mutations u $\rightarrow$ x, ce sont des *événements indépendants*. Cela génère donc plusieurs histoires possibles : la mutation du premier x avant celle du deuxième x ou inversement. Ces histoires auront exactement le même coût et nous n'avons aucun moyen de savoir laquelle est la "meilleure", elles contiennent les mêmes mutations et les mêmes contractions, mais dans un ordre différent.

CHAPITRE 3

Mutations bords

3.1. Description du problème

Nous rappelons dans ce paragraphe les caractéristiques du problème. Nous allons considérer, comme dans le problème précédent, que les amplifications ne sont pas forcément binaires, que la taille des motifs répétés est une fois la taille du motif de base et que la limite gauche de ces motifs se situe dans la colonne 1 de la matrice. La différence vient du fait que désormais, les mutations peuvent se produire n'importe où.

Cela correspond au problème *TRHIST fixed boundary, fixed duplication size* de l'article de Benson et Dong.

EXEMPLE 3.1.1. Voici une suite de transformations possible :
u
→ u u u u u
→ u u **v** u u
→ u u v v v u u
→ u u v v **w** u u
→ u u v v w w w u u
→ u u v v **x** w w u u

La première mutation, u → v, à la ligne 3, est une mutation milieu, alors que les deux autres, lignes 5 et 7, sont des mutations bords. On remarque tout de suite, que la séquence ne présente plus de structure palindromique, ceci est clairement dû au fait que certaines mutations ne se produisent pas entre deux motifs identiques.

Si on considère le x de la séquence finale, on ne peut pas savoir de quelle mutation il est issu, il pourrait venir d'une mutation v → x, ou w → x, car v et w sont ses voisins directs, mais il pourrait très bien provenir d'une mutation u → x.

EXEMPLE 3.1.2. L'histoire de la séquence aurait pu être
u
→ u u u u u u
→ u u **x** u u u
→ u u x x u u u
→ u u **v** x u u u
→ u u v v x u u u
→ u u v v x **w** u u
→ u u v v x w w u u

Cette histoire à le même nombre de mutations que la précédente, mais ce coup-ci, le x et le w proviennent du u, et le v provient du x.

Pour savoir quelle histoire est la moins chère au niveau du coût des mutations, il faudrait calculer leur coût réel en nombre de mutations de caractères. En effet, en dépit du fait que les deux histoires ont le même nombre de mutations de motifs, elles n'ont pas les mêmes : dans la première on a, u $\rightarrow$ v, v $\rightarrow$ w et w $\rightarrow$ x, alors que dans la seconde, on a u $\rightarrow$ x, u $\rightarrow$ w et x $\rightarrow$ v, et à priori on ne peut pas savoir quel ensemble de mutations a le coût le plus faible.

L'avantage de la restriction des mutations milieux était qu'elle conférait à la séquence une structure palindromique qui nous indiquait quel ensemble de mutations de motifs était valide, on n'avait pas à prendre en considération le coût réel de chacune des mutations.

Dans ce chapitre on se préoccupera seulement de minimiser le nombre de mutations de motifs, car même si on ne minimise pas à coup sûr le nombre de mutations de caractères, on remarque qu'en pratique, on est très souvent à l'optimum. Cela nous permet de baisser la complexité du problème, et d'explorer seulement une sous-partie de l'ensemble des solutions. On ne tiendra pas compte dans un premier temps des amplifications. Pour l'instant notre but est de minimiser le nombre de mutations de motifs.

Dans le paragraphe suivant, on va définir la notion de portion, qui va nous aider à donner une certaine structure aux séquences, et nous permettre de trouver un moyen de construire leur histoire. Nous expliquerons ensuite comment résoudre ces portions, et nous exposerons enfin des méthodes pour résoudre le problème entier.

3.2. Les portions

Dans un premier temps, nous donnons une définition précise de la notion de portion, puis nous expliquerons pourquoi nous nous y sommes intéressés et comment les portions donnent un moyen de retrouver une histoire de la séquence. Nous parlerons, dans la section suivante, de la manière de résoudre ces portions.

REMARQUE 3.2.1. Dans cette section, nous travaillons exclusivement sur des séquences factorisées. Nous rappelons qu'une séquence est considérée comme une suite de motifs, et que chaque motif est représenté par une lettre. S[i] est le ième motif.

3.2.1. Qu'est ce qu'une portion ?

Nous allons tout d'abord définir formellement les portions.

DÉFINITION 3.2.2. Portion et portion simple.
Soit S une séquence de n motifs. Soit i et j deux entiers tels que $1 \leq i, j \leq n$. Alors (i, j) est une portion $\Leftrightarrow S[i] = S[j]$.

$$(i, j) \text{ est une portion simple} \Leftrightarrow \begin{cases} & S[i] = S[j] \\ \text{et} & \forall k,\ i < k < j,\ S[k] \neq S[i] \text{ et } S[k] \neq S[j], \\ \text{et} & \forall k, k',\ i < k, k' < j,\ S[k] = S[k'] \Leftrightarrow k = k'. \end{cases}$$

Une portion simple est donc une facteur de S délimité par deux motifs identiques, et tel que tout motif interne n'apparaît qu'une fois dans ce facteur. Une portion non simple peut inclure d'autres portions.

DÉFINITION 3.2.3. Motifs internes et externes d'une portion.
Les motifs internes à la portions (i, j) sont tous les motifs dont les positions sont strictement comprises en i et j. Les motifs externes à la portions (i, j) sont les motifs correspondant aux positions i et j.

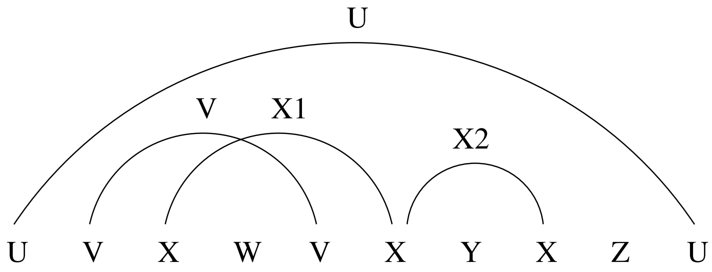


FIG. 3.2.1 – Exemple de portions.

On peut supposer que les motifs externes d'une portion sont les traces d'une amplification de ce motif et que tous les motifs internes proviennent du motif externe ancêtre par mutations. Si cette amplification appartient à la véritable histoire, alors on peut chercher à reconstruire l'histoire de la portion sans tenir compte du reste de la séquence.

Reconstruire l'histoire "indépendante" d'une portion, ou "résoudre une portion"

$$uv_1v_2\dots v_ju,$$

c'est trouver une suite de mutations ponctuelles qui la transforme en

$$u(u)^ju$$

qui sera ultérieurement contractée en u.

Si on ne considère que des mutations de motifs, alors l'histoire la plus parcimonieuse est :

$$v_1 \to u\ et\ v_2 \to u\ et\ \dots\ et\ v_j \to u$$

Du point de vue des mutations de caractères, cela revient à trouver la topologie d'arbre "phylogénétique" la plus parcimonieuse pour les "taxons" u, v_1, v_2, ..., v_j, u dans l'ordre. Seule la topologie est à trouver, car l'ordre des feuilles de l'arbre est fixé.

Une amplification d'un motif interne à la portion est possible dans l'histoire de la portion, mais elle forme alors une sous-portion qui peut être résolue indépendamment. Elle peut donc être résolue au préalable. D'où le fait, que dans la définition d'une portion simple, les motifs internes sont uniques.

REMARQUE 3.2.4. Si une portion P inclut des portions simples, après résolution de celles-ci, la portion P devient elle-même simple et peut-être résolue à son tour.

Exemple de portions :

La séquence uvxwvx**y**x**z**u contient plusieurs portions qui peuvent être représentées avec des arcs (cf figure 3.2.1). Il y a trois portions simples, V, X1 et X2, et une portion englobante, U.

Les extrémités des arcs délimitent une portion, on étiquette chaque portion comme dans le chapitre précédent (cf. section 2.4.2). La différence vient du fait que les arcs peuvent se croiser. Des portions peuvent se chevaucher, quand cela est le cas, les arcs qui les représentent se croisent.

DÉFINITION 3.2.5. Portions incompatibles.
Des portions sont incompatibles si elles se chevauchent.

Si des portions sont incompatibles, seulement l'une d'elles pourra être résolue. En effet si on en résout une, les autres disparaissent du fait de la perte d'une de leur extrémité qui se trouvait à l'intérieur de la portion résolue. Dans l'exemple ci-dessus, V et X1 sont incompatibles, si l'on résout X1, on ne peut plus résoudre V, et vice versa. Lorsque deux portions se chevauchent, au moins une des deux a ses extrémités qui ne proviennent pas de la même origine.

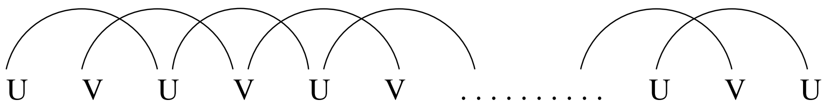
DÉFINITION 3.2.6. Portions compatibles.
Deux portions sont compatibles si elles ne se chevauchent pas.

Des portions compatibles pourront faire partie de la même histoire. Tous les couples suivant contiennent des portions compatibles entre elles : (V, X2), (V, U), (X1, X2), (X1, U) et (X2, U). On remarque que si une portion "englobe" une autre portion, elles sont compatibles, comme X2 et U par exemple.

On peut parler d'ensembles de portions 2 à 2 incompatibles, ou d'ensemble de portions 2 à 2 compatibles.
L'ensemble {U, X1, X2} est un ensemble de portion 2 à 2 compatibles.

Dénombrement des portions simples d'une séquence

Dans le pire des cas, on peut avoir n-2 portions, où n est le nombre de motifs de la séquence :



Ici, tous les ensembles de portions deux à deux incompatibles sont de cardinal 2. Il existe un ensemble de portions deux à deux compatibles de cardinal $\lceil \frac{n}{2} \rceil - 1$.

La résolution d'une portion évite la mutation d'un de ses motifs externes en un motif ancêtre, à cause de la contraction de la portion.

REMARQUE 3.2.7. La résolution de portion est le seul moyen d'éviter une mutation de motif. Donc, l'histoire qui minimise les mutations de motifs, est une histoire qui maximise le nombre de portions compatibles.

3.3. Résolution de portion simple

Nous rappelons qu'une portion simple est une suite de motifs tous deux à deux différents, sauf le premier et le dernier, qui sont identiques. Comme on l'a dit précédemment, résoudre une portion revient à trouver une suite d'opérations (mutations de caractères et contractions), de coût minimum, qui transforme les motifs internes de la portion, en motifs identiques à ceux des extrémités. Pour trouver une résolution optimale au niveau du coût en mutations réelles, on est obligé de considérer les caractères qui composent les motifs.

Pour résoudre les portions, on peut utiliser un algorithme de parcimonie classique. Nous proposons une méthode optimale qui tient compte de l'ordre des motifs.

3.3.1. Approche exhaustive : recherche du maximum de parcimonie

. Une approche exhaustive pour résoudre une portion consiste à énumérer toutes les topologies d'arbres binaires, et à calculer le nombre de mutations de caractères de l'histoire induite par l'arbre. Le résultat, est l'histoire de parcimonie maximale. La complexité est exponentielle comme le nombre d'arbre à n feuilles. [**DEKM98**]

3.3.2. Méthode basée sur un algorithme de type A* et le calcul de la borne inférieure de Benson et Dong

. Benson et Dong étudient un cas restreint du problème général, donnent un algorithme heuristique, GREEDY-TRHIST-RESTRICTED et exhibent des bornes inférieures et supérieures pour ce cas. Après avoir montré que ce cas correspond, par ses hypothèses, à celui de la résolution de portions simples, nous en déduirons la possibilité d'utiliser un algorithme A* avec leur borne inférieure.

Étant donné que tous les motifs internes sont différents, et que l'on cherche à minimiser le nombre de mutations de caractères, pour résoudre une portion, on s'intéressera seulement aux contractions binaires. En effet toute contraction d'arité > 2, peut se décomposer en contractions binaires, sans changer le coût en nombre de mutations de caractères.

En outre, nos motifs sont de taille k, et la limite gauche des motifs impliqués dans une contraction doit se trouver dans la colonne 1 de la matrice. Ces caractéristiques (taille des motifs, arité des contractions, et limite gauche des motifs impliqués) correspondent à un problème restreint de l'article de Benson et Dong : celui traité par l'algorithme GREEDY-TRHIST-RESTRICTED. Dans ce cas particulier, leur fonction de coût et leur règle pour produire la copie consensus sont équivalentes aux nôtres (si $\alpha = 0$), car on ne considère que des contractions binaires. Donc, les bornes inférieures et supérieures de coût, données dans la section 4 de [**BD99**] pour le cas restreint aux contractions binaires, sont aussi des bornes pour le coût de la résolution d'une portion simple.

La section 3.3.2.1 illustre le fait que l'histoire d'une portion simple est équivalente à un arbre binaire dont les feuilles sont les motifs. La section 3.3.2.2 reprend la démonstration des bornes inférieures et supérieures du coût d'une histoire, tandis que la dernière section explique comment réaliser l'algorithme A* et l'illustre par un exemple.

3.3.2.1. *Représentation de la solution*

. La solution de ce problème peut-être représentée par un arbre binaire, que l'on appellera *arbre de duplication*. Cet arbre est enraciné, ses feuilles et ses arêtes sont ordonnées.

<u>Un exemple :</u>

Soit la portion ACTTA, AC_TA, GGTTA, GATTA, GCTTA, ACTTA, contenant 6 motifs. Une résolution de cette portion peut s'exprimer sous la forme de l'arbre de duplication de la figure 3.3.1.

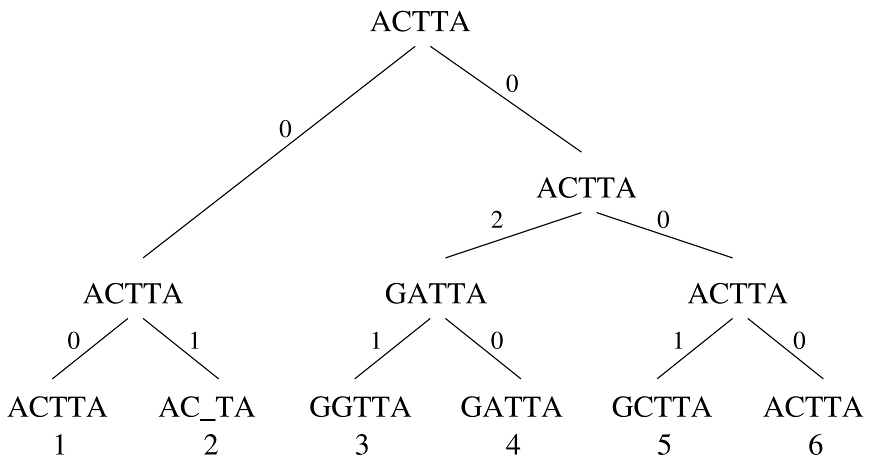


FIG. 3.3.1 – Arbre de duplication.

Chaque arête porte le nombre de mutations de caractères qui est nécessaire pour passer du motif fils au motif père. Le coût total de l'arbre est la somme des coûts des arêtes. L'arbre de la figure 3.3.1 a un coût de 5.

L'arbre de duplication de la figure 3.3.1 représente l'histoire suivante, et toutes les histoires issues de celle-ci en permutant les évènements indépendants :

1. <u>ACTTA</u>
2. → <u>ACTTA ACTTA ACTTA ACTTA ACTTA</u>
3. → ACTTA ACTTA <u>**G**ATTA</u> ACTTA ACTTA
4. → ACTTA ACTTA <u>GATTA GATTA</u> ACTTA ACTTA
5. → ACTTA AC**_**TA G**G**TTA GATTA **G**CTTA ACTTA

3.3.2.2. *Bornes pour le coût (d'après la section 4 de* [**BD99**]*)*
. Pour ce problème les auteurs ont trouvé des bornes pour le coût en nombre de mutations de caractères. Ils ont notamment, donné une borne supérieure qui utilise un *ordered spanning tree* (OST). Un OST est un spanning tree sur un ensemble ordonné de feuilles ayant la propriété suivante :

Si les noeuds sont dans l'ordre, alors, pour n'importe quels couples d'arêtes (i_1, j_1) et (i_2, j_2), où $i_1 < j_1$ et $i_2 < j_2$, on a

$$(i_2 - i_1)(i_1 - j_2)(j_1 - i_2)(j_1 - j_2) \geq 0.$$

Un OST peut-être dessiné sur un ensemble ordonné de noeuds arrangés de manière linéaire, où chaque arête occuperait le même demi plan sans se croiser (cf. figure 3.3.2). L'expression arithmétique ci-dessus vérifie simplement qu'il n'y a pas de croisements.

Le coût de l'OST de la figure 3.3.2 est 4. A chaque OST est associé un unique arbre de duplication de même coût (cf. figure 3.3.3).

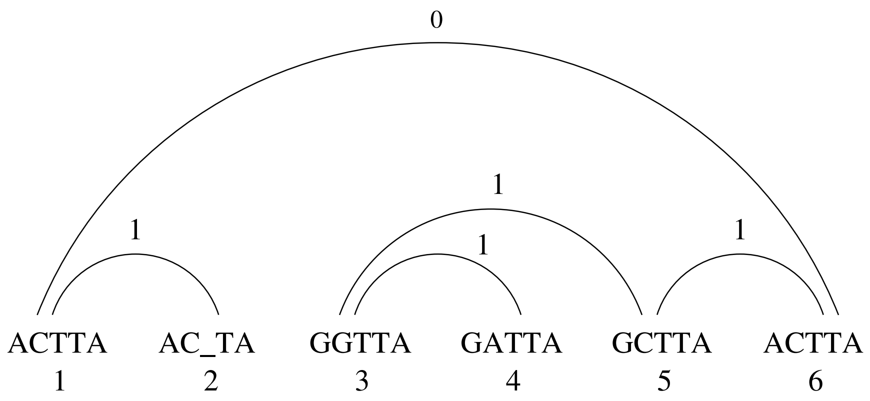


FIG. 3.3.2 – Ordered Spanning Tree.

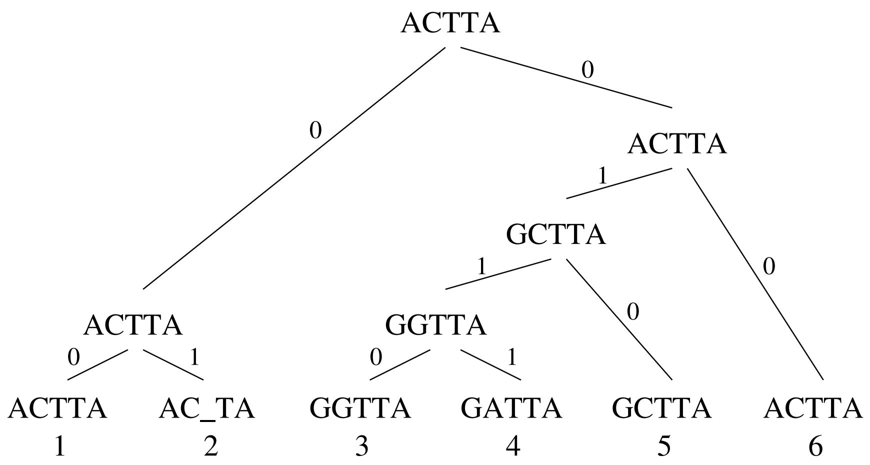


FIG. 3.3.3 – Arbre de duplication associé à l'OST de la figure 3.3.2.

Dans l'exemple (figures 3.3.2 et 3.3.3), l'arbre de duplication obtenu à partie de l'OST, a un coût optimal de 4, i.e., on ne pourra pas trouver d'arbre de coût inférieur.

Pour donner des bornes de coût, les auteurs utilisent le cycle SR. Une exploration en profondeur de l'arbre optimal produit un cycle R dans lequel tous les noeuds internes apparaissent deux fois, et les feuilles une fois. R possède 2 fois chaque arête de l'arbre optimal, donc coût (R) = $2 \times c^*$, où c^* est le coût de l'arbre optimal. Si on élimine tous les noeuds internes de R, on obtient un cycle simple SR qui contient seulement les feuilles. Quelle que soit la forme de l'arbre optimal, le cycle SR lui, sera toujours le même (cf. figure 3.3.4).
L'inégalité triangulaire nous donne : coût (SR) $\leq$ coût (R).

Si on enlève à SR, l'arête de poids maximum, on obtient un OST, noté TR. On a vu que l'on pouvait transformer un OST en arbre de duplication de même coût, soit AD l'arbre de duplication correspondant à TR.

On a alors : coût (AD) = coût (TR) $\leq$ coût (SR) $\leq$ coût (R) = $2 \times c^*$,

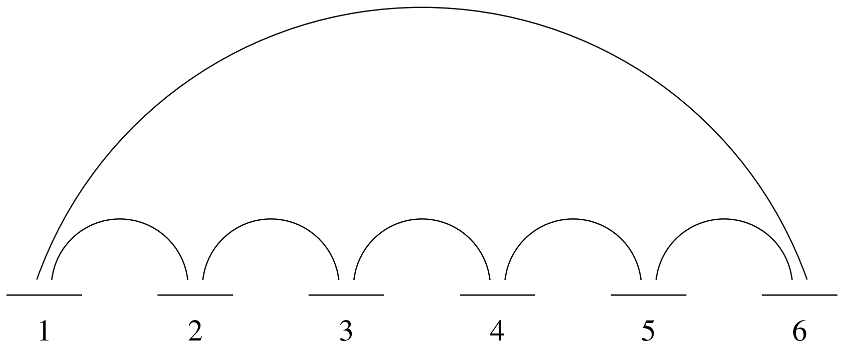


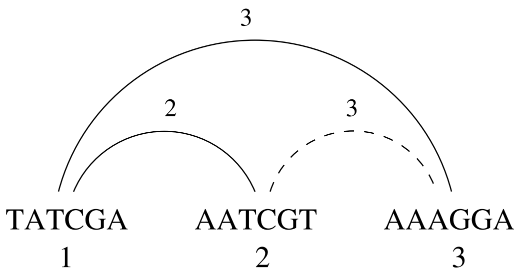
FIG. 3.3.4 – Cycle SR.

d'où coût (AD) $\leq 2 \times c^*$

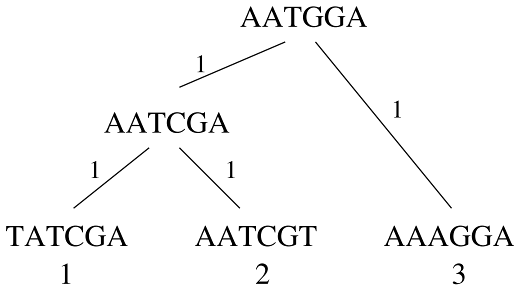
Ainsi, AD est une solution dont le coût ne dépasse pas deux fois l'optimal.

TR n'est pas nécessairement un *minimum* ordered spanning tree, i.e., un OST de poids minimum. Un minimum ordered spanning, noté MT, peut-être calculé en utilisant la programmation dynamique en $O(n^2k + n^3)$. MT peut lui aussi être transformé en arbre de duplication de même coût. Cependant, la méthode gloutonne des auteurs (cf section 1.2.2), donne des solutions de coût identique ou meilleur que le coût de MT. Ceci est dû au fait que les spanning trees ne peuvent pas introduire de motifs intermédiaires qui n'étaient pas présents dans la séquence de départ.

Donnons un exemple où le minimum ordered spanning tree a un coût supérieur à l'histoire fournie par la méthode de Benson et Dong. Soit la séquence TATCGA AATCGT AAAGGA. Le minimum ordered spanning tree a un coût de 5 :



La méthode des auteurs choisit de faire la contraction entre les motifs 1 et 2, qui est la moins chère. La copie consensus contient 2 ensembles de caractères ambigus : {A/T}ATCG{A/T}, en faisant la contraction de cette copie et du troisième motif, on résout ces 2 ensembles, et on en crée 2 autres, la copie ancêtre est : AA{A/T}{C/G}GA. On peut décider de lever ces ambiguïtés. Les deux contractions coûtent chacune 2, l'histoire trouvée par la méthode des auteurs a donc un coût de 4. Elle est représentée par l'arbre suivant :



Cet histoire est moins chère que celle que donne le minimum ordered spanning tree, car elle introduit le motif AATCGA, qui n'apparaissait pas dans les motifs de départ.

En ce qui concerne la borne inférieure :

Le coût de SR est facile à calculer, il correspond au nombre de mutations nécessaires pour passer du motif 1 au motif 2, puis du motif 2 au motif 3, ..., puis du motif n au motif 1.

De plus on a directement une borne inférieure simple : c* $\geq \frac{co\hat{u}t(SR)}{2}$.

3.3.2.3. *Algorithme de type A* pour la résolution d'une portion simple*
. Notre méthode consiste à développer un arbre d'exploration de l'espace des solutions, i.e., de toutes les combinaisons possibles de contractions binaires qui réduisent la matrice à une seule ligne. Chaque branche de cet arbre représente une telle combinaison. La racine de l'arbre est étiquetée par la séquence que l'on va traiter, alignée dans une matrice. Les feuilles sont étiquetées par des matrices d'une ligne, représentant chacune un motif ancêtre. Chaque noeud interne est étiqueté par une matrice, représentant un état intermédiaire dans une suite de contractions.

Les arêtes de cet arbre représentent une contraction binaire, effectuée sur la matrice étiquetant le noeud père, et dont le résultat est la matrice étiquetant le noeud fils. Les arêtes sont étiquetées par le coût des contractions qu'elles représentent. Un noeud, étiqueté par une matrice A, de m lignes, aura m-1 fils. Le fils 1 sera étiqueté par la matrice résultant de la contraction des lignes 1 et 2 de A, le fils 2 par la matrice résultant de la contraction des lignes 2 et 3, ..., le fils m-1 sera étiqueté par la matrice résultant de la contraction des lignes m-1 et m de A.

On n'explore pas entièrement cet arbre. On coupe des branches de cet arbre en se servant de borne inférieure de coût que l'on a donné plus haut : coût (SR) / 2 et du coût minimum (coût_min) calculé durant l'évaluation de l'arbre.

MT représente une résolution possible de la séquence, donc on peut initialiser la variable coût_min à $co\hat{u}t$ (MT) $- 1$. On coupe une branche dans l'arbre, lorsque la somme du coût déjà calculé de cette branche et du coût restant estimé, est supérieure au coût minimum. Lorsque l'on se trouve à un noeud η, avant de poursuivre l'exploration, on estime, le coût restant pour contracter entièrement la matrice étiquetant ce noeud. Pour cela, il faut calculer SR sur la matrice qui étiquette le noeud η.

Alors, si

$$\frac{co\hat{u}t\ (SR)}{2} + co\hat{u}t\ (\text{chemin reliant le noeud } \grave{a} \text{ la racine}) > co\hat{u}t\_min$$

on coupe la branche qui mène à η.

REMARQUE 3.3.1. Le coût de MT et le coût des arêtes sont forcément des nombres entiers, tandis que $\frac{\text{coût (SR)}}{2}$ est un multiple de 0,5. Si $\frac{\text{coût (SR)}}{2}$ n'est pas un nombre entier, le coût restant de la branche, qui lui est un nombre entier, ne pourra pas être égal à cette borne inférieure, il pourra atteindre au mieux la partie entière supérieure de cette borne. De plus, on coupe une branche lorsque l'estimation de son coût est strictement supérieure à coût_min. D'où le terme de -1 dans l'initialisation de la variable coût_min.

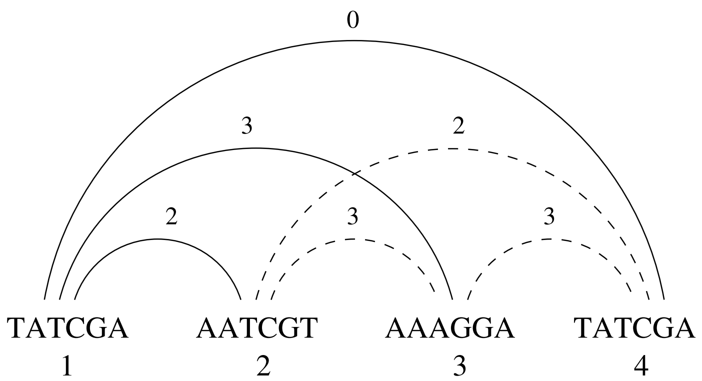
Si on a coupé toutes les branches avant d'obtenir un noeud où la matrice à une seule ligne, alors, on donne comme solution MT qui a, dans ce cas, un coût optimal. Sinon, on donne comme solution la suite de contractions représentée par une branche de longueur n-1, qui a le coût le plus faible parmi toutes les branches de même longueur.

La suite de contractions donnée par la méthode représente l'arbre de duplication optimal de la séquence.

REMARQUE 3.3.2. On peut montrer qu'il existe toujours un arbre de duplication de coût optimal Ψ, tel que le motif à la racine soit le motif externe de la portion. Ψ est l'arbre de duplication représentant la résolution de la portion.

Exemple :

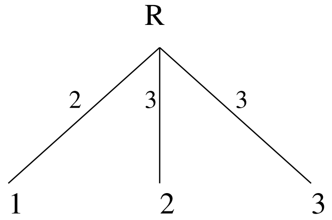
Soit la portion TATCGA AATCGT AAAGGA TATCGA. Voici un minimum ordered spanning tree (MT) calculé sur cette séquence, les arcs en pointillés ne font pas partie de MT :



Le coût de MT est 5.
Coût_min sera donc initialiser à 4.

Appliquons notre méthode à cette séquence. Commençons par l'aligner dans une matrice, et construisons le premier étage de l'arbre :

<u>Racine R</u> : $\begin{bmatrix} T & A & T & C & G & A \\ A & A & T & C & G & T \\ A & A & A & G & G & A \\ T & A & T & C & G & A \end{bmatrix}$



<u>Noeud 1</u> : $\begin{bmatrix} \{A/T\} & A & T & C & G & \{A/T\} \\ A & A & A & G & G & A \\ T & A & T & C & G & A \end{bmatrix}$ Résultat de la contraction des lignes 1 et 2 de la matrice initiale. Le coût de la contraction est 2 et le coût de SR est 5.

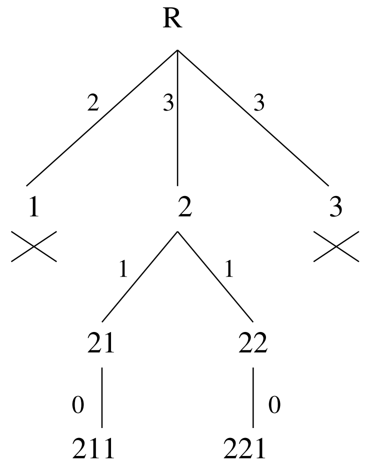
<u>Noeud 2</u> : $\begin{bmatrix} T & A & T & C & G & A \\ A & A & \{T/A\} & \{C/G\} & G & \{T/A\} \\ T & A & T & C & G & A \end{bmatrix}$ Résultat de la contraction des lignes 2 et 3. Coût de contraction = 3. Coût (SR) = 2.

<u>Noeud 3</u> : $\begin{bmatrix} T & A & T & C & G & A \\ A & A & T & C & G & T \\ \{T/A\} & A & \{T/A\} & \{C/G\} & G & A \end{bmatrix}$ Résultat de la contraction des lignes 3 et 4. Coût de contraction = 3. Coût (SR) = 3.

Pour les noeuds 1 et 3,
coût du chemin les reliant à la racine + Coût (SR)/2 = 4,5 > Coût_min, on n'arrête par conséquent leur exploration. Le coût total estimé pour la branche du noeud 2 est de 4 ce qui est égal à coût_min. Calculons maintenant les descendants du noeud 2 :

<u>Noeud 21</u> : $\begin{bmatrix} \{T/A\} & A & T & C & G & A \\ T & A & T & C & G & A \end{bmatrix}$ Résultat de la contraction des lignes 1 et 2 de la matrice du noeud 2. Coût de contraction = 1. Coût (SR) = 0. Le coût estimé de la branche est donc 4.

<u>Noeud 22</u> : $\begin{bmatrix} T & A & T & C & G & A \\ \{T/A\} & A & T & C & G & A \end{bmatrix}$ Résultat de la contraction des lignes 2 et 3 de la matrice du noeud 2. Coût de contraction = 1. Coût (SR) = 0. Le coût estimé de la branche est donc 4.

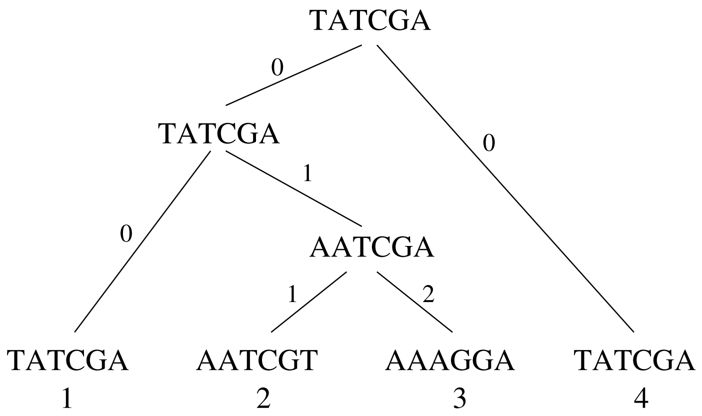


Pour les noeuds 21 et 22, le coût du chemin les reliant à la racine + Coût (SR)/2 $\leq$ coût_min , donc on continue l'exploration.

Noeud 211 : $\begin{bmatrix} T & A & T & C & G & A \end{bmatrix}$ Résultat de la contraction des deux lignes de la matrice du noeud 21. Coût de contraction = 0.

Noeud 221 : $\begin{bmatrix} T & A & T & C & G & A \end{bmatrix}$ Résultat de la contraction des deux lignes de la matrice du noeud 22. Coût de contraction = 0.

On obtient donc deux chemins de même coût représentant chacun une solution. Voici l'arbre de duplication induit par le chemin de R à 211 :



Cet arbre a un coût optimal de 4.

3.4. Résolution du problème

Nous allons montrer que chercher l'ensemble de portions deux à deux compatibles de cardinal maximum revient à résoudre le problème du stable max dans un graphe de recouvrement. Nous donnerons ensuite une méthode alternative, que nous conjecturons optimale. Nous finirons par quelques remarques sur le problème restreint étudié dans ce chapitre.

Pour trouver l'histoire de coût minimum en nombre de mutations de motifs, nous avons déjà vu qu'il suffisait de trouver un ensemble de portions deux à deux compatibles de cardinal maximum. Une fois que l'on a cet ensemble, pour obtenir l'histoire, il suffit d'ordonner les portions selon la relation d'inclusion.

Nous expliquons maintenant comment obtenir un ensemble de portions deux à deux compatibles de cardinal maximum.

3.4.1. Équivalence avec le problème du stable max dans un graphe de recouvrement

. Dans cette section, nous montrons comment convertir notre problème en un problème de graphe. Chaque portion, simple ou englobante, représente un intervalle de la séquence.

DÉFINITION 3.4.1. Relation d'incompatibilité.
Soit (i, j) et (k, l) deux portions. Notons R la relation d'incompatibilité. Alors

$$(i, j)\ R\ (k, l) \Leftrightarrow \begin{cases} & i < k \text{ et } j < l \text{ et } k < j \\ \text{ou} & k < i \text{ et } l < j \text{ et } j < k \\ \text{ou} & k = i \text{ et } j \neq l \\ \text{ou} & k \neq i \text{ et } l = j \end{cases}$$

Avec cette définition, la relation d'incompatibilité entre portions correspond à la relation de recouvrement entre intervalles. Par recouvrement, on entend chevauchement, inclusions non comprises.

On considère G, un graphe tel que les noeuds sont les portions de la séquence, et les arcs représentent la relation R, G = <P, R>, où P est l'ensemble de toutes les portions (simples et englobantes). Alors G est un graphe de recouvrement puisqu'il représente la relation de chevauchement entre un ensemble d'intervalles ordonnés linéairement. Il apparaît que trouver un ensemble de portions deux à deux compatibles de cardinal maximum revient à trouver le stable max dans G. Par définition, le stable max est le plus grand ensemble de sommets, tel qu'il n'existe aucune arête entre ces sommets.

Gavril dans [**Gav73**] propose un algorithme polynomial pour résoudre le stable max dans un graphe de recouvrement. Pour les détails de l'algorithme, nous renvoyons le lecteur à [**Gol80**]. Nous pouvons déterminer grâce à cet algorithme un ensemble de cardinal maximum de portions deux à deux compatibles en complexité polynomiale.

3.4.2. Méthode alternative

. Cette méthode se base sur les ensembles maximaux de portions deux à deux incompatibles, notés $D_1 \ldots D_m$. Ces ensembles sont maximaux dans le sens où toute portion qui n'appartient pas à D_i est compatible avec au moins une portion de cet ensemble. Ces ensembles forment une partition de l'ensemble des portions.

Un ensemble de portions deux à deux compatibles de cardinal maximum, que l'on notera S, ne peut pas contenir plusieurs portions appartenant à un même D_i. Donc card (S) $\leq$ m.

A chaque portion Γ, nous associons un *score de coupure*. Ce score représente le nombre minimum de portions que la résolution de Γ fera disparaître, c.a.d. le nombre minimum de portions que Γ chevauche, ou encore le nombre minimum de portions que l'arc de Γ *coupe*.

DÉFINITION 3.4.2. Score de coupure.

Le score de coupure d'une portion Γ, est le nombre d'ensembles de portions deux à deux incompatibles que Γ coupe *entièrement*. Γ coupe entièrement un ensemble D_i, si D_i - {X} n'est pas inclus dans D_j, $\forall$ i $\neq$ j.

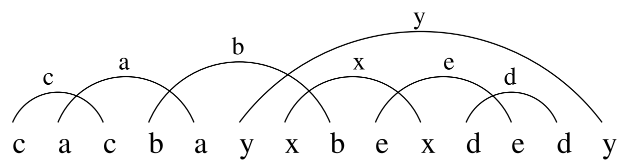
A chaque étape, notre méthode calcule un score de coupure pour chaque portion, puis élimine la portion ayant obtenue le plus grand score et recalcule les scores de coupure. En cas d'égalité de score, on élimine la portion qui a le plus de coupures réelles. Elle réitère ce processus jusqu'à ce qu'il reste au plus un élément dans chaque D_i, i $\in$ 1 $\ldots$ m, ou que tous les éléments aient un score nul. Si la méthode se termine parce que tous les scores sont à zéro, et qu'il reste plusieurs portions dans un même ensemble, on en choisit une arbitrairement.

L'ensemble de portions restantes est noté E. E est bien un ensemble de portions deux à deux compatibles. Sur l'exemple ci-dessous, et sur tous ceux que nous avons envisagés, l'algorithme donne une solution optimale. Nous conjecturons qu'il est optimal.

Il est intéressant d'examiner si notre algorithme peut s'adapter au problème du stable max pour les graphes de recouvrement et les circular-arc-graphs.

<u>Exemple :</u>

Soit la séquence c a c b a y x b e x d e d y. Donnons dans un premier temps la représentation graphique des portions contenues dans cette séquence :



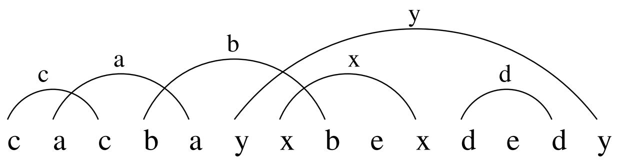
Donnons maintenant les ensembles de portions deux à deux incompatibles :

D_1 : {c, a}
D_2 : {a, b}
D_3 : {b, y}
D_4 : {b, x}
D_5 : {x, e}
D_6 : {e, d}

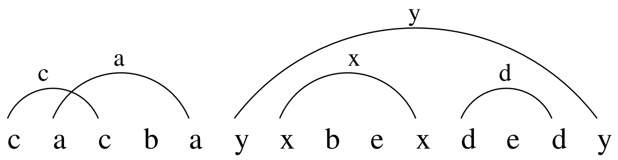
Calculons maintenant les scores de chacune des portions :

Portions	1er score	2ème score	3ème score
a	1	1	0
b	1	2	$\otimes$
c	0	0	0
d	0	0	0
e	1	$\otimes$	$\otimes$
x	0	0	0
y	0	0	0

Calcul du deuxième score :



Calcul du troisième score :



Toutes les portions ont un troisième score nul, donc on s'arrête. Les portions a et c appartiennent toutes deux à D_1, il faut en choisir une : prenons a. On a donc E = {a, d, x, y}, E est bien un ensemble de portions 2 à 2 compatibles de cardinal maximum.

3.4.3. Remarques

•

3.4.3.1. *Les contractions*

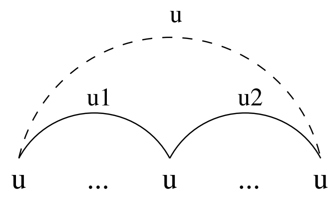
. Si on s'occupe seulement de minimiser le nombre de mutations de motifs dans l'histoire de la séquence, est-on optimal en nombre de contractions ?

On doit contracter toutes les occurrences multiples de motifs pour obtenir à la fin un seul motif ancêtre. Donc pour minimiser le nombre de contractions, il faudrait contracter tous les motifs identiques en un seul coup, mais pour cela, il faudrait qu'ils soient côte à côte. Or, lorsqu'on a un morceau de séquence du type : xxxxxyyyyyxxyyyy, on est obligé de contracter les x ou les y en 2 coups.

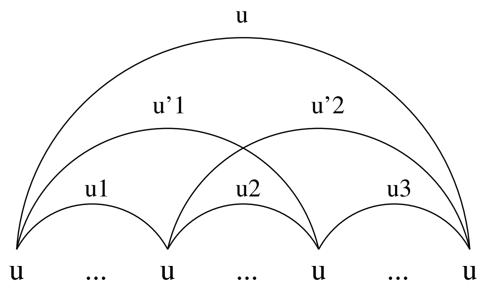
Le fait de maximiser le nombre de portions à résoudre, va également minimiser le nombre de contractions à faire pour les motifs identiques ne se trouvant pas côte à côte.

La résolution de portion minimise donc à la fois les mutations, mais permet également de contracter ensemble des motifs identiques (ceux des bords) n'apparaissant pas au même endroit.

3.4.3.2. *Cas particulier des portions englobant d'autres portions de même nom*
. Il y a des portions englobantes particulières. Ces portions partagent un motif externe avec une portion qu'elles englobent.



Comment gérer la portion u par rapport aux portions u1 et u2 ? En effet, on ne veut pas de solutions où l'on ait à la fois u et des portions de même bords qu'elle englobe. On a donc ajouter des "incompatibilités artificielles" (lignes 3 et 4 de la relation d'incompatibilité de la section 3.4.1), elles correspondent au fait que deux portions qui ont les mêmes motifs à leurs extrémités, l'une englobant l'autre et partageant un bord physique commun, sont considérées comme incompatibles :



Dans cet exemple, la définition d'incompatibilité (cf. 3.4.1) implique que :

u qui est incompatible avec u'1, u'2, u1 et u3.
u'1 qui est incompatible avec u, u'2, u1 et u2.
u'2 qui est incompatible avec u, u'1, u2 et u3.
u1 qui est incompatible avec u et u'1.
u2 qui est incompatible avec u'1 et u'2.
u3 qui est incompatible avec u et u'2.

Par contre, u et u2 sont compatibles, ainsi que u1 et u'2 par exemple.

Bibliographie

[BD99] Gary Benson and Len Dong. Reconstructing the Duplication History of a Tandem Repeat. In T. Lengaeur, R. Schneider, P. Bork, D. Brutlag, J. Glasgow, H.-W. Mewes, and R. Zimmer, editors, *Proceedings of the 7th International Conference on Intelligent Systems for Molecular Biology*, 1999.

[DEKM98] R. Durbin, S. Eddy, A. Krogh, and G. Mitchinson. *Biological Sequence Analysis*. Cambridge University Press, 1998.

[Gav73] Falinca Gavril. Algorithms for a maximum clique and a minimum independent set of a circle graph. *Networks*, 3 :261–273, 1973.

[Gol80] Martin C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, Inc., San Diego, 1980.