Université de Montpellier 2

Institut des Sciences de l'Évolution de Montpellier

Master 2 STIC pour la Santé
spécialité Bioinformatique, Connaissances et Données

Stage de Master 2 Recherche

Reconstruction de l'histoire évolutive des génomes : réconciliations et reconstructions de relations ancestrales

par Yoann Anselmetti

Encadrants de stage :
Mme Sèverine Bérard
M. Vincent Berry
Mme Annie Chateau

Tuteur pédagogique :
M. Konstantin Todorov

2013 - 2014

ANR Ancestrome

Table des matières

Table des figures

Table des algorithmes

Remerciements

Je tiens à remercier Vincent Berry pour m'avoir proposé ce projet de stage, en collaboration avec Sèverine Bérard et Annie Chateau, et pour tout l'expertise et le savoir-faire apportés dans l'élaboration d'algorithmes appliqués au domaine de la phylogénie.

Merci également à Sèverine Bérard de m'avoir fait découvrir l'algorithme DeCo et m'avoir permis d'entrevoir toutes les perspectives futures de cet algorithme.

Merci aussi à Annie Chateau pour son expertise dans la résolution de problèmes mathématiques qui nous a permis d'avancer dans l'élaboration de notre algorithme à un moment où nous étions bloqué.
Enfin, merci à tous les trois pour leur soutien, leur aide et leur appui lors de mes candidatures sur des projets de thèse et pour les corrections de ce mémoire.

Merci à Aude pour son soutien.

Introduction

Depuis tout temps, l'homme ne cesse de vouloir répondre à la question "D'où venons-nous et de qui descendons-nous ?".
La phylogénie est la branche de la science s'intéressant à l'étude de l'évolution des espèces, qui détermine l'enchaînement de processus évolutifs permettant d'expliquer les liens de parentés entre les espèces existantes.
La première théorie véritablement scientifique de l'évolution des espèces provient de Jean-Baptiste Lamarck, publiée en 1809 dans son livre *Philosophie zoologique*, qui classifie les espèces en fonction de leurs différences/similarités morphologiques.
Cette nouvelle théorie, à l'époque, s'oppose à la théorie du fixisme ou créationnisme qui estime que toute espèce présente sur Terre a été créée par Dieu et qu'elle n'a pas évolué depuis sa création (*réf. Wikipedia Créationnisme*).
La personne la plus connue ayant défendu et apporté de grands travaux sur la théorie de l'évolution est sans nul doute Charles Darwin qui publia en 1859 son très célèbre *On the origin of species*. Dans son livre, Darwin introduit le concept de sélection naturelle des espèces permettant d'expliquer l'évolution des espèces en fonction de l'environnement dans lequel elles ont évolué.
Au XIX^{e} siècle, la création d'arbre phylogénétique était basée sur les caractères morphologiques et non moléculaires, car nous n'avions pas de connaissances à ce sujet.
C'est à partir des années 1940 que les travaux de Theodosius Dobzhansky et Ernst Mayr vont permettre d'associer la théorie de l'évolution darwinienne à la génétique mendélienne. Cette association est la base de la phylogénie et est communément appelée "Théorie synthétique de l'évolution" (*réf. Wikipedia Théorie synthétique de l'évolution*).
Cette théorie est actuellement celle acceptée par la communauté scientifique. Depuis, plusieurs stratégies ont été développées pour modéliser des arbres phylogénétiques inférant l'histoire évolutive des génomes à différentes échelles à partir des séquences issues des molécules d'ADN : arbre de gènes, racontant l'histoire d'un gène en particulier ; arbre d'espèces, au niveau le plus global ; arbre de gènes réconcilié avec un arbre d'espèces, pour distinguer les événements de spéciation de ceux de duplications, pertes et transferts dans l'évolution de ce gène.
En 2012, Sèverine Bérard et ses collaborateurs ont décrit une méthode permettant de déterminer l'histoire évolutive de gènes adjacents dans les génomes [7]. Le but de mon stage a été d'effectuer dans un premier temps une analyse de la méthode DeCo (`mDeCo`), puis dans un second d'élaborer des protocoles de test pour déterminer la fiabilité et la robustesse de la méthode. Enfin, une dernière étape a été d'effectuer une réflexion algorithmique sur la méthode pour élaborer et intégrer un nouvel algorithme à intégrer dans la méthode afin de permettre d'améliorer les méthodes d'assemblage des génomes avec l'information supplémentaire apportée par les adjacences de gènes.
Dans une première partie de ce rapport, nous présentons le contexte dans lequel le stage s'est effectué. Nous enchaînons par la présentation des concepts de bases nécessaires à la compréhension de ce rapport. Nous présentons ensuite la méthode `mDeCo` et l'algorithme DeCo développés par l'équipe avant le stage. Enfin nous finirons sur la présentation de l'analyse et des améliorations apportées à `mDeCo` au cours de ce stage de Master 2.

Chapitre 1

Contexte du stage

Dans le cadre du Master 2 STIC pour la Santé parcours BCD, il est demandé aux étudiants d'effectuer un stage long de Master 2 en entreprise ou en laboratoire.
Ayant eu un intérêt particulier pour l'application de la bioinformatique au domaine de la phylogénie, j'ai effectué mon stage long de Master 2 au sein de l'ISE-M du 18 février au 18 août 2014 sous l'encadrement de Sèverine Bérard, Vincent Berry et Annie Chateau.

1.1 L'Institut des Sciences de l'Évolution de Montpellier (ISE-M)

L'institut regroupe des chercheurs provenant du CNRS, de l'IRD et de l'UM2, il est situé sur le campus de la faculté des sciences de l'UM2 dans les bâtiments 22 et 24. L'institut regroupe des chercheurs s'intéressant à l'évolution des espèces aussi bien à l'échelle morphologique que moléculaire. Mon stage s'est déroulé dans l'équipe Phylogénie et Évolution Moléculaire dirigé par le professeur Emmanuel Douzery. Site internet : `http://www.isem.univ-montp2.fr/`

1.2 Financement par le projet Ancestrome de l'ANR

Ce stage de Master 2 recherche s'effectue dans le cadre du projet Ancestrome financé par l'ANR suite à un appel à projet d'Investissements d'avenir pour la bioinformatique. Site internet : `http://ancestrome.univ-lyon1.fr/`
Ce projet implique quatre laboratoires :

- le LBBE de l'Université Lyon 1 (porteur du projet) ;
- l'équipe DYOGEN de l'ENS de Paris ;
- l'ISE-M de l'UM2 ;
- le laboratoire BMGE de l'Institut Pasteur à Paris.

Le but de ce projet est de développer de nouveaux outils bioinformatiques tels que des logiciels de modélisation phylogénétique (PhylDog, DeCo, ALE, ANGES, ...) et des bases de données phylogénomique (Genomicus, Hogenom et OrthoMam) permettant de mieux comprendre et modéliser l'histoire évolutive des espèces.

Chapitre 2

Notions de base en phylogénie moléculaire

Pour déterminer l'histoire évolutive de gènes, d'espèces et d'adjacences de gènes, plusieurs étapes sont nécessaires. Dans cette première partie, nous allons décrire les notions de bases permettant de comprendre comment reconstruire une histoire évolutive.

2.1 Notions de génomique

2.1.1 Génomes, chromosomes et gènes

L'évolution des espèces provenant de la modification du matériel génétique de celles-ci, il est donc important de comprendre les éléments structurels de l'information génétique. Le **génome** d'une espèce correspond à l'ensemble des chromosomes, chaque **chromosome** est l'association de deux chromatides assemblées par leurs centromères, une chromatide étant une séquence d'**ADN** compactée composée de gènes. Le **gène** est la première brique fonctionnelle du génome car il peut être transcrit en molécule fonctionnelle, l'**ARN**. Certains ARN sont ensuite traduits en **protéines**, d'autres ne sont pas traduits et sont appelés **ARNnc** et ont un rôle fonctionnel souvent impliqué dans les mécanismes de réplication de l'ADN, transcription de l'ADN en ARN ou traduction de l'ARN en protéine.

2.1.2 L'adjacence de gènes

L'adjacence de gènes est définie comme suit : un gène est adjacent à un autre, si ces deux gènes sont situés sur le même chromosome et qu'ils sont contigus (*cf.* Figure 2.1).
Comme nous l'avons vu précédemment les gènes sont contenus dans les génomes qui sont composés de chromosomes. Par cette définition de l'adjacence de gènes nous pouvons donc déterminer qu'un gène a soit une, soit deux adjacences dans les chromosomes (format linéaire). Au vu de la quantité de gènes contenus dans un chromosome, la proportion de gènes ayant une seule adjacence est minime (ex : chez Homo sapiens, il y a à peu près 20.000 gènes répartis sur 24 chromosomes (23 paires de chromosomes avec 24 chromosomes différents 1→22 + X + Y). Il y a donc $24 * 2 = 48$ gènes n'ayant qu'une seule adjacence ce qui représente $48/20000 * 100 = 0.24\%$ du nombre de gènes total). L'adjacence entre un gène v_1 et un gène v_2 est notée $v_1 \sim v_2$ ou $v_2 \sim v_1$.

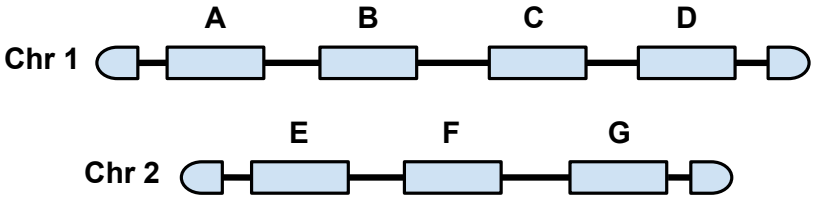


FIGURE 2.1 – Schéma représentant des adjacences de gènes sur des chromosomes. On a $B \sim A$ et $B \sim C$. A a une adjacence, tandis que B en a deux.

2.2 Les modifications chromosomiques

Pour déterminer l'histoire évolutive des espèces, les scientifiques ont d'abord dû comprendre les mécanismes moléculaires permettant le processus de l'évolution.
En 1944, les travaux de O. Avery *et al.* sur les pneumocoques [5] ont permis d'établir que l'ADN est le support de l'hérédité décrite par G. Mendel (1822-1884), rôle qui jusque-là était attribué aux protéines par la communauté scientifique.
L'évolution des espèces s'effectue donc par des modifications de l'ADN qui peuvent être classées principalement dans deux catégories, **les mutations génétiques** et **les réarrangements chromosomiques**.

2.2.1 Les mutations génétiques

Elles sont effectuées sur un **nucléotide** (A, T, C ou G) qui est l'unité de base de l'ADN. Ces mutations sont créées par des erreurs de la machinerie moléculaire lors de la réplication de l'ADN, ou par des facteurs externes (radiation, radicaux libres, agent intercalant de l'ADN, ...). Il existe trois types de mutations génétiques "ponctuelles" :

— la **substitution** qui remplace un nucléotide par l'un des trois autres ;
— l'**insertion** qui ajoute un nucléotide dans la séquence d'ADN ;
— la **délétion** qui enlève un nucléotide de la séquence d'ADN.

Dans les deux derniers cas, plusieurs nucléotides successifs peuvent être concernés. Ces mutations modifient la séquence nucléotidique et lorsque celles-ci offrent un avantage dans la survie ou l'adaptation de l'espèce dans son environnement, elles sont conservées par le processus de sélection naturelle. Cependant, ces événements ne suffisent pas pour expliquer l'histoire évolutive des gènes. D'autres mécanismes biologiques sont responsables de l'évolution des gènes, **les réarrangements chromosomiques**.

2.2.2 Les réarrangements chromosomiques

Ils s'effectuent sur de large portions chromosomiques contenant souvent plusieurs gènes. Il en existe plusieurs types :

— l'**inversion** est un phénomène découvert à la fin des années 30 grâce aux travaux de Sturtevant et Dobzhanski sur la drosophile [1]. Ils sont également les premiers à avoir défini l'histoire évolutive des drosophiles à partir de ce type de réarrangement ;

— la **translocation** chromosomique qui est un réarrangement entre deux chromosomes homologues qui entraîne un échange de portion chromosomique entre ces deux chromosomes ;

— la **fusion** chromosomique qui entraîne la jonction de deux chromosomes par leurs extrémités pour n'en former plus qu'un ;

— la **fission** chromosomique qui entraîne la rupture d'un chromosome et donne naissance à deux chromosomes ;

— la **duplication** de gène(s) qui résulte de la duplication et la transposition d'une portion chromosomique ;

— la **perte** de gène(s) provient de la suppression d'un gène par la perte d'une portion chromosomique ou par sa pseudogénisation, procédé par lequel un gène est inactivé suite à des modifications génétiques le rendant non-fonctionnel ;

— le **transfert** de gène(s) est un processus par lequel un organisme intègre dans son génome du matériel génétique provenant d'un autre organisme. Ce phénomène est très répandu chez les bactéries qui s'échangent par exemple les gènes de résistance aux antibiotiques via ce procédé.

2.3 Les événements évolutifs

Pour étudier l'évolution des espèces on fixe d'abord un modèle mathématique retenant un certain nombre d'événements évolutifs. Ce modèle sert ensuite de support aux développements informatiques. Suivant les équipes travaillant sur l'évolution des espèces, les modèles peuvent différer. Nous détaillons ci-dessous les événements évolutifs retenus par S. Bérard *et al.* [7] :

— la **spéciation** est le phénomène évolutif par lequel un groupe d'individus d'une espèce évolue significativement et distinctement des autres individus, c'est-à-dire subit un grand nombre de mutations et/ou de réarrangements chromosomiques conservés par les descendants. La différence entre ces individus et les autres est telle qu'elle donne naissance à une nouvelle espèce ;

Notation et représentation graphique : Spec 

— la **duplication de gène ou d'adjacence** est le processus de réarrangement chromosomique sus-mentionné qui duplique et transpose une portion de chromosome ;

Notation et représentation graphique : GDup  ADup 

— la **perte de gène ou d'adjacence** est également un type de réarrangement chromosomique et conduit à la fin de l'histoire évolutive d'un gène ou d'une adjacence ;

Notation et représentation graphique : GLos  ALos 

— la **création d'adjacence** correspond à la formation d'une nouvelle adjacence entre deux gènes. Les créations d'adjacences peuvent être expliquées par des réarrangements chromosomiques comme l'inversion ou la translocation d'une portion chromosomique ;

Notation et représentation graphique : Gain 

— la **cassure d'adjacence** correspond à la suppression d'une adjacence par un phénomène de réarrangement chromosomique.

Notation et représentation graphique : Break 

2.4 Les arbres phylogénétiques

Pour représenter les liens de parentés entre des gènes, des espèces ou des adjacences de gènes, la représentation la plus courante est celle d'un arbre.
Un arbre phylogénétique est un graphe non cyclique, connexe, considéré ici comme orienté de la racine aux feuilles. La **racine** correspondant à l'ancêtre commun le plus récent de toutes

les **feuilles** de l'arbre qui ; elles, correspondent aux **gènes/espèces/adjacences actuel(le)s**. Un **nœud** dans un arbre phylogénétique correspond à l'ancêtre commun le plus récent des gènes/espèces/adjacences descendants de ce nœud. Suivant les contextes, un nœud peut également être étiqueté avec un **événement évolutif**.
Les données pouvant être de différents types, il existe plusieurs sortes d'arbres phylogénétiques (*cf.* Figure 2.2) :

— l'**arbre de gènes** correspondant à l'histoire évolutive d'une famille de gènes homologues. C'est-à-dire un ensemble de gènes descendant d'un gène ancestral commun proche par rapport aux autres gènes et qui souvent ont des fonctions similaires ;
Événements évolutifs possibles : Spec, GDup, GLos.

— l'**arbre d'espèces** correspondant aux relations de parentés entre différentes espèces plus ou moins proches. De tels arbres ont d'abord été inférés à partir d'observations morphologiques et ont par la suite été affinés par inférence à partir d'arbres de gènes. Cependant les topologies des arbres de gènes pouvant être fortement divergents de celle de l'arbre des espèces associé, l'inférence d'un arbre des espèces nécessite une grande quantité d'arbres de gènes pour obtenir une topologie consensus relativement sûre de l'arbre des espèces [10][11] ;
Événement évolutif possible : Spec.

— l'**arbre d'adjacences de gènes** représente l'histoire évolutive d'adjacences de gènes, un tel arbre permet de visualiser la coévolution de gènes.
Événements évolutifs possibles : Spec, GDup, ADup, GLos, ALos, Gain, Break.

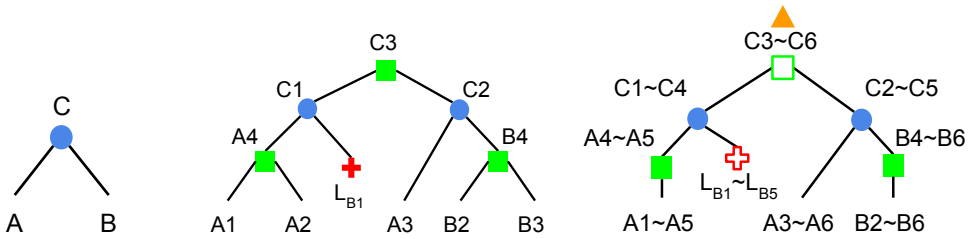


FIGURE 2.2 – Représentation des différents arbres phylogénétiques. Sur la gauche un arbre des espèces, au milieu un arbre de gènes et à droite un arbre d'adjacences de gènes (*cf.* 2.3 pour la signification des symboles).

Lorsque l'on considère un ensemble d'arbres de même type, on parle de forêt.

2.5 L'inférence d'arbres phylogénétiques

Pour inférer un arbre phylogénétique de gènes ou d'espèces, on se base sur l'alignement de séquences nucléotidiques ou protéiques des espèces étudiées. On utilise les ressemblances et dissemblances locales entre ces séquences, parfois résumées dans un score de similarité global. Plus les séquences sont similaires, plus leur ancêtre commun est supposé récent par rapport aux autres séquences étudiées. Il existe plusieurs méthodes pour inférer un arbre phylogénétique.

2.5.1 Les méthodes de distance

La première approche est basée sur une matrice de distances entre les séquences étudiées. Les distances peuvent être mesurées sur le nombre de caractères différents entre les séquences. Ainsi, les séquences ayant les distances les plus faibles sont associées et on compare de nouveaux les

séquences jusqu'à déterminer l'ancêtre commun à toutes les séquences. Cette méthode est celle utilisé notamment dans l'algorithme du *Neighbour Joining* [2].

2.5.2 Le maximum de parcimonie

Cette méthode est une approche qui infère les états ancestraux à partir de l'alignement des séquences en minimisant le coût des mutations génétiques. De cette façon, on peut comptezr le nombre de mutations impliquées par un arbre et choisir l'arbre le plus parcimonieux en nombre de mutations.
Deux approches ont été développées, la première par Fitch en 1971 [3] qui estime que toute substitution de nucléotide a un coût de 1 et la deuxième par Sankoff en 1975 [4] qui a modifié la matrice des coûts de substitution pour prendre en compte la diversité des fréquences de permutations des nucléotides.

2.5.3 Les méthodes probabilistes

Le maximum de vraisemblance

Cette approche prend en paramètres deux types de données :

— le jeu de séquences d'ADN que l'on étudie ;
— un modèle d'évolution des séquences (JC69, K2P, HKY85, GTR, ...) qui permet de prendre en compte les différents taux de permutations des nucléotides ;

Le maximum de vraisemblance est une méthode qui donne la topologie pour laquelle la probabilité de l'association du jeu de données suivant le modèle d'évolution et de cette topologie est maximale.

L'inférence bayésienne

Cette méthode est basée sur les probabilités postérieures en phylogénie en combinant la fonction de vraisemblance et des probabilités dites a priori sur les topologies et les paramètres du modèle d'évolution.
Cette approche est souvent utilisée en complément du maximum de vraisemblance, car l'inférence bayésienne détermine un ensemble de topologies de fortes probabilités généralement proches de la topologie trouvée par le maximum de vraisemblance.

Un arbre de gènes n'est pas toujours de la même topologie que l'arbre des espèces. Pour expliquer la différence entre ces topologies une approche a été développée, la **réconciliation**.

2.6 La réconciliation d'arbres phylogénétiques

La réconciliation permet d'expliquer les différences de topologie entre l'arbre des espèces et un arbre de gènes (*cf.* Figure 2.3). Ces différences s'expliquent par le fait que les gènes peuvent subir des événements évolutifs sans pour autant déclencher une spéciation. Cette méthode permet donc d'étiqueter les nœuds des arbres de gènes avec des événements évolutifs, pour permettre de réconcilier la topologie de l'arbre d'espèce avec celle de l'arbre de gènes, c'est-à-dire d'expliquer les différences de topologie entre ces arbres.
La réconciliation parcimonieuse est la première à avoir été décrite et consiste à minimiser le

nombre de pertes, duplications et/ou transferts de gènes pour réconcilier un arbre de gènes avec un arbre des espèces. Elle a été décrite par Goodman et al., qui s'intéressaient à inférer l'arbre de gènes de la famille des globines dans l'arbre des espèces des vertébrés [6]. Dans cet article, les auteurs ont défini un modèle de Duplication/Perte des gènes (*DL model*) qui a largement été réutilisé par la communauté scientifique [10][13][14][15].
Ce modèle est l'un des plus utilisés pour la réconciliation. Cependant, il ne prend pas en compte les événements de transfert de gène. Pour prendre en compte ce type de réarrangement, un nouveau modèle a été développé basé sur le *DL model* auquel on intègre les transferts de gènes. Ce modèle est appelé modèle de Duplication/Transfert/Perte (*DTL model*) [9][16][17].
D'autres méthodes de réconciliation utilisant des approches probabilistes ont été développées [18]. Cependant, ces méthodes ont un temps d'exécution très important et ne sont donc pas adaptées à de grands jeux de données.

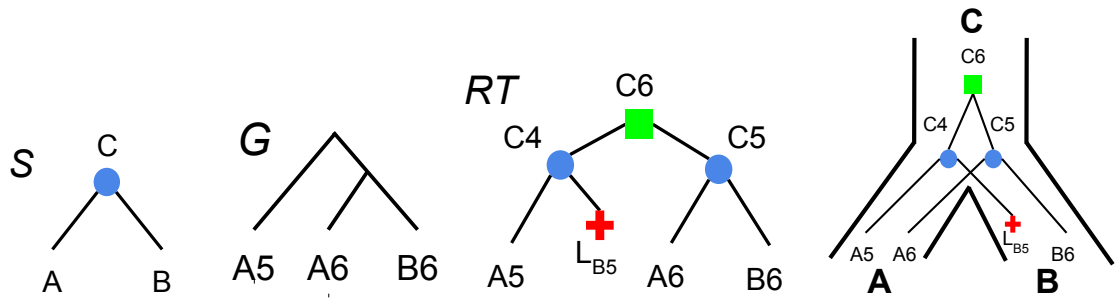


FIGURE 2.3 – Représentation de la réconciliation d'un arbre de gènes G avec un arbre des espèces S. Au milieu la représentation RT (*Reconciled Tree*) de la réconciliation de G avec la topologie de S. À droite une représentation de cette réconciliation avec l'arbre de gènes à l'intérieur de l'arbre des espèces. (*cf.* 2.3 pour la signification des symboles).

Chapitre 3

Algorithme DeCo

DeCo est un algorithme qui permet de reconstruire l'histoire évolutive ancestrale d'adjacences de gènes en minimisant le nombre de créations et de cassures d'adjacences. Cet algorithme est exact et s'exécute en temps polynomial, il est basé sur une **approche parcimonieuse** reprenant le principe de **programmation dynamique** de D. Sankoff [4].

3.1 Fonctionnement général de la méthode `mDeCo`

L'algorithme est inclus dans un programme de même nom qu'on appellera méthode `mDeCo` qui prend en entrée une **liste d'adjacences de gènes actuels**, des **arbres de gènes**, un **arbre des espèces** et un **fichier d'association gène-espèce**.
Elle s'effectue en deux phases :

- Une première **phase de pré-traitement des données** dans un premier temps **réconcilie parcimonieusement** les topologies des arbres de gènes avec celle de l'arbre des espèces et dans un deuxième temps **regroupe les adjacences actuelles** partageant potentiellement une adjacence ancestrale commune.
- La deuxième phase correspond à l'**algorithme DeCo** et permet d'inférer une forêt d'arbres d'adjacences à partir de deux arbres de gènes partageant des adjacences ayant une histoire évolutive commune. DeCo est divisé en deux phases : une première qui va **calculer une matrice de scores** entre les couples de nœuds/gènes provenant de la même espèce en optimisant un critère de coûts ; et une deuxième étape nommée ***backtracking*** qui va permettre de reconstruire une forêt d'arbres d'adjacences à partir de la matrice de scores.

3.2 Le pré-traitement des données

Comme dit précédemment, `mDeCo` prend en entrée un ensemble d'arbres de gènes noté E_G, une liste d'adjacences de gènes actuels notée L_{adj} et un arbre des espèces S, que nous définirons sous la forme $\Delta = (L_{adj}, E_G, S)$.
`mDeCo` effectue la réconciliation de l'ensemble des arbres de gènes contenus dans E_G avec S en utilisant la réconciliation parcimonieuse *LCA* élaborée par Goodman *et al.* [6]. L'ensemble des arbres de gènes réconciliés est noté E_{Grec}. Une fois l'attribution des événements de spéciations, duplications et pertes de gènes aux arbres de gènes, `mDeCo` crée des classes d'équivalence d'adjacences à partir de E_{Grec} et de L_{adj}.
Une classe d'équivalence d'adjacences permet de classer les adjacences qui ont potentiellement

une adjacence ancestrale commune. Dans la représentation gauche de la Figure 3.1, pour que les deux adjacences $A_1 \sim A_2$ et $B_1 \sim B_2$, respectivement chez les espèces A et B, appartiennent à la même classe, il faut que :

— A_1 et B_1 appartiennent à la même famille de gènes, c.-à-d. à un même arbre de gènes G_1 ;
— A_2 et B_2 appartiennent à la même famille de gènes, c.-à-d. à un même arbre de gènes G_2 ;
— l'espèce C soit l'ancêtre commun de A et B ;
— le gène C_1 soit le plus récent ancêtre de A_1 et B_1 et appartienne à l'espèce C ;
— le gène C_2 soit le plus récent ancêtre de A_2 et B_2 et appartienne à l'espèce C.

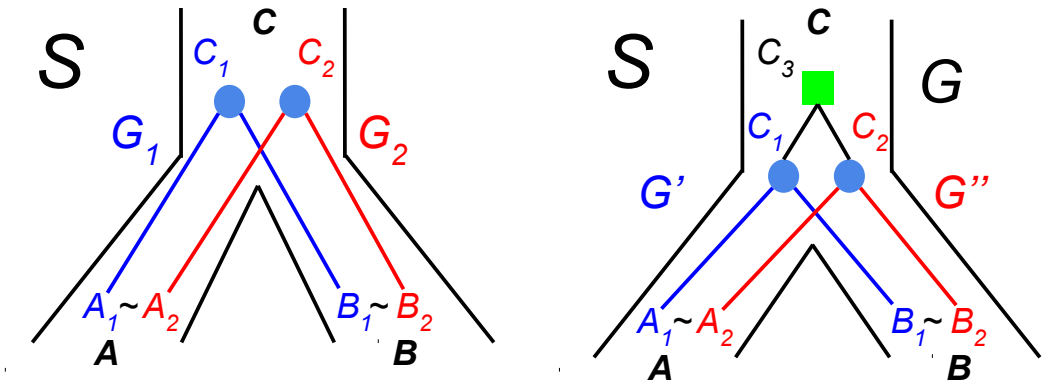


FIGURE 3.1 – Illustration des classes d'équivalence d'adjacences. À gauche, la représentation de deux arbres de gènes G_1 et G_2 d'une même classe d'adjacences dans un arbre d'espèces S et à droite la représentation d'un arbre de gène G, pouvant être décomposé en deux-sous arbres G' et G''appartenant à la même classe d'adjacences, dans un arbre des espèces S.

Par définition, les adjacences d'une même classe d'équivalence appartiennent au plus à deux arbres de gènes réconciliés. Si elles n'appartiennent qu'à un seul arbre de gènes noté G, une propriété définie dans l'article décrivant l'algorithme DeCo [7], établit que toutes les adjacences de cette classe sont contenues dans deux sous-arbres distincts de G (*cf.* Figure 3.1). Formellement, cela revient à dire que si A_1 et B_1 appartiennent à un même arbre de gènes G et que $A_1 \sim A_2$ et $B_1 \sim B_2$ sont dans la même classe d'adjacences, alors le plus petit ancêtre commun de A_1, A_2 et B_1, B_2 est le même et on le note $LCA(A_1, A_2) = LCA(B_1, B_2) = C_3$, dans l'exemple de droite de la Figure 3.1. Ainsi, lorsque que l'on retire C_3 de l'arbre de gène, on obtient deux sous-arbres de G notés G' et G'' appartenant à une même classe d'adjacences et qui seront traités simultanément par l'algorithme DeCo pour générer un arbre d'adjacences.

3.3 L'algorithme DeCo

Après avoir effectué l'étape de pré-traitement des données, pour chaque classe d'équivalence DeCo calcule une matrice de scores pour chaque couple de nœuds/gènes (v_1, v_2) des arbres de gènes G_1 et G_2 tels que $v_1 \in G_1$ et $v_2 \in G_2$, où l'espèce de v_1 est la même que celle de v_2 que l'on note $Esp(v_1) = Esp(v_2)$. Pour chaque couple de nœuds, deux coûts sont calculés :

— Le coût $c_1(v_1, v_2)$ qui correspond au coût minimum d'une histoire évolutive où v_1 et v_2 sont adjacents ;
— Le coût $c_0(v_1, v_2)$ qui correspond au coût minimum d'une histoire évolutive où v_1 et v_2 ne sont pas adjacents.

Après avoir calculé cette matrice de scores, l'algorithme effectue une étape de *backtracking* qui parcourt la matrice afin de construire la forêt d'arbres d'adjacence, correspondant à l'histoire des adjacences qui minimise le critère de coût.

3.3.1 Calcul d'une matrice de coût minimum entre les couples de nœud de deux arbres de gènes

Cette étape est basée sur une approche parcimonieuse de programmation dynamique reprenant le principe de l'algorithme de Sankoff-Rousseau [8]. La programmation dynamique consiste à résoudre un problème complexe en résolvant des sous-problèmes de même nature que celui-ci. Elle permet de calculer une matrice de coût minimum entre tous les couples de nœuds (v_1, v_2), afin de savoir si les gènes correspondant à ces nœuds ont été adjacents ou non dans l'histoire évolutive.

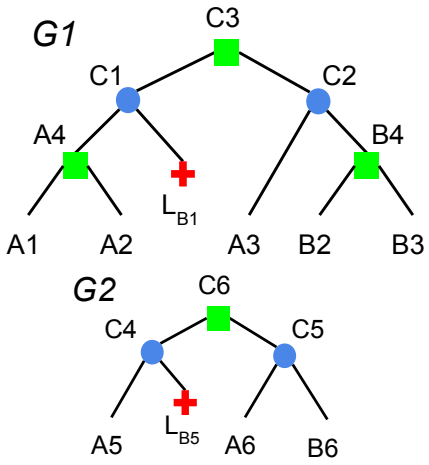
Chaque nœud v des arbres étant étiqueté d'un événement évolutif noté $E(v)$, le calcul du coût de chaque couple de nœuds ne sera pas le même selon les paires d'événements. Dans les arbres de gènes réconciliés, il existe quatre types d'événements évolutifs :

— Extant (Feuille) ;
— Spec (Spéciation) ;
— GDup (Duplication de gène) ;
— GLos (Perte de gène).

Pour calculer les coûts c_1 et c_0, les auteurs de l'article décrivant l'algorithme DeCo ont déterminé six cas de formules de récurrence correspondant aux différents couples d'événements évolutifs possibles qui sont détaillés dans l'article [7] (*cf.* Annexe p. 44.
Cette étape est une **méthode ascendante** qui calcule récursivement les scores des coûts c_1 et c_0 en partant des couples de feuilles jusqu'au couple de racines. La Figure 3.2 (p. 17) illustre le calcul de la matrice de coût entre les deux arbres G_1 et G_2 provenant de la même classe d'équivalence d'adjacences.

3.3.2 Étape de *Backtracking*

Une fois la matrice de calcul de coût des couples de nœuds établie, DeCo effectue une étape de ***backtracking*** permettant, à partir des résultats de cette matrice, de reconstruire un arbre d'adjacences modélisant l'histoire évolutive des adjacences contenues dans les arbres de gènes G_1 et G_2.
Pour recréer l'histoire évolutive des adjacences ancestrales depuis la matrice de scores entre couples de nœuds, l'étape de *backtracking* commence sur la case de la matrice correspondant au $min(c_1(R_1, R_2), c_0(R_1, R_2))$ où R_1 correspond à la racine de l'arbre de gènes G_1 et R_2 correspond à la racine de l'arbre de gènes G_2. La phase de *backtracking* est un **processus descendant** qui choisit les adjacences en suivant à rebours les combinaisons de scores ayant amené aux scores minimaux pour les couples de nœuds considérés, jusqu'à atteindre les adjacences actuelles entre feuilles des arbres de gènes. La Figure 3.3 (p. 18) illustre l'étape de *backtracking* correspondant à la reconstruction de l'arbre d'adjacences à partir des arbres de gènes G_1 et G_2 de la Figure 3.2 (p. 17).



c_1/c_0	A1	A2	A3	A4	L_{B1}	B2	B3	B4	C1	C2	C3
A5	0/∞	∞/0	∞/0	0/1							
A6	∞/0	∞/0	0/∞	1/0							
L_{B5}					0/0	0/0	0/0	0/0			
B6					0/0	0/∞	∞/0	0/1			
C4									0/1	1/0	0/1
C5									1/0	0/2	0/2
C6									0/1	0/2	0/2

FIGURE 3.2 – Représentation du calcul de la matrice de coût minimum à partir de deux arbres de gènes G_1 et G_2. Sur la gauche, représentation graphique de G_1 et G_2. Sur la droite, représentation de la matrice de coût minimum entre tous les couples de nœuds de G_1 et G_2. En colonne sont représentés les gènes contenus dans G_1 et en ligne sont représentés les gènes contenus dans G_2. Les cases vertes correspondent aux couples de nœuds provenant de la même espèce et pour lesquels on peut calculer un score minimum pour l'adjacence ou non entre les deux gènes. Les cases grises correspondent aux couples de nœuds ne provenant pas de la même espèce. L'algorithme DECO a une complexité quadratique, c.-à-d. en $O(n^2)$ où n représente le nombre de nœuds compris dans un arbre de gènes.

3.4 Limites de la méthode `mDeCo`

`mDeCo` est une méthode efficace car c'est une méthode exacte et de faible complexité. En pratique, elle traite un jeu de données de 11 espèces et 5039 arbres de gènes en environ 2 minutes. Cependant, plusieurs paramètres biologiques n'ont pas été pris en compte dans la modélisation de l'histoire évolutive des adjacences ancestrales.
L'algorithme DECO ne prend pas en compte la contrainte de linéarité des génomes, car il infère les gènes contenus dans les adjacences ancestrales indépendamment des uns des autres, ce qui peut inférer plus de deux adjacences à un gène donné. De plus, l'algorithme DECO ne permet pas de reconstruire des arbres d'adjacences à partir d'arbres de gènes contenant des transferts de gène car cet événement nécessite une modélisation plus élaborée, qui a été implémentée dans la méthode DeCoLT dérivée de la méthode `mDeCo` et développée par M. Patterson *et al.* [9].

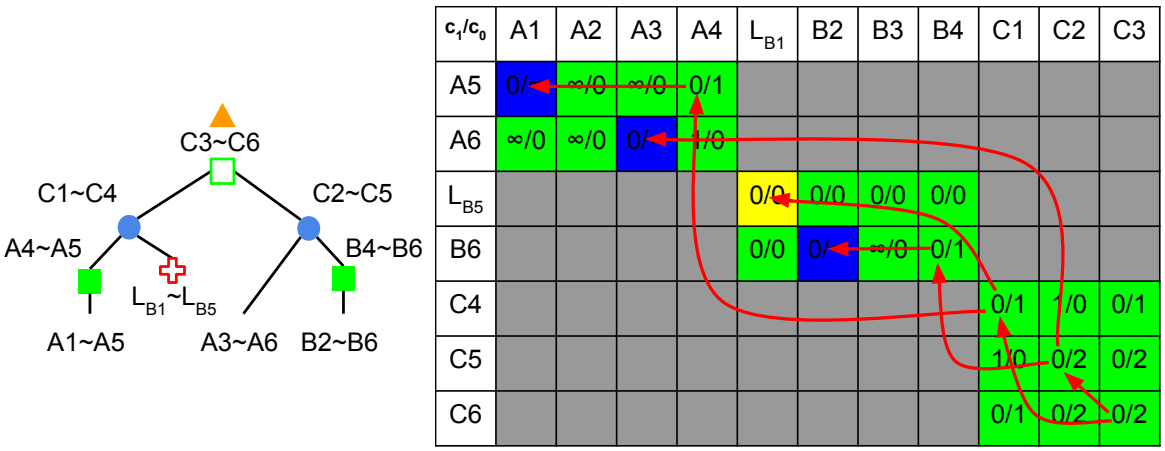


FIGURE 3.3 – Représentation de la reconstruction de l'arbre d'adjacences à partir de la matrice de calcul de coût minimum entre les couples de nœuds. Sur la gauche, représentation graphique de l'arbre d'adjacences généré à partir des arbres de gènes G_1 et G_2. Sur la droite, représentation de la matrice de coût minimum entre tous les couples de nœuds de G_1 et G_2. Les flèches représentent le fonctionnement de l'étape de *backtracking* sous la forme d'un parcours arboré permettant de reconstruire le ou les arbre d'adjacences. Les cases en bleu correspondent aux adjacences actuelles et la case en jaune correspond à une adjacence qui a été perdue au cours de l'histoire évolutive suite à la perte des deux gènes la composant.

Chapitre 4

Développements effectués au cours du stage

4.1 Rappels des objectifs du stage

L'objectif principal est d'effectuer une réflexion algorithmique sur la méthode afin de rendre la modélisation de l'histoire évolutive de la méthode plus proche de la réalité.
Dans un premier temps, il fallait faire une analyse de l'implémentation de `mDeCo` en vue des modifications qui seront apportées au cours de ce stage (*cf.* partie 4.2 p. 19). Un deuxième objectif consiste à déterminer la robustesse de l'algorithme DeCo (*cf.* partie 4.3 p. 27). Enfin, la dernière partie de ce stage a consisté à l'élaboration d'un algorithme basé sur le principe de l'algorithme DeCo et pouvant permettre d'améliorer la qualité d'assemblage des génomes.

4.2 Analyse et modularisation du code de `mDeCo`

Une première étape du stage a donc été d'analyser et comprendre l'implémentation de la méthode `mDeCo` écrit dans le langage C++, à l'aide de la librairie Bio++[12]. Au cours de cette analyse, nous avons observé que la méthode était composée de quatre phases :

- La phase de réconciliation ;
- La phase de création des classes d'équivalence d'adjacences ;
- La phase de création des arbres d'adjacences (Algorithme DeCo) ;
- La phase de synthèse statistique.

Pour rappel, la réconciliation consiste, à partir d'un arbre des espèces et d'un arbre de gènes, à étiqueter les nœuds de l'arbre de gènes avec des événements évolutifs afin de réconcilier les topologies de l'arbre de gènes avec celle de l'arbre des espèces (*cf.* partie 2.6). La phase de réconciliation doit réconcilier tous les arbres de gènes du jeu de données passé en entrée de `mDeCo`.
L'étape de création des classes d'adjacences consiste à associer les adjacences qui ont potentiellement une adjacence ancestrale commune (*cf.* partie 3.2).

Après analyse de l'implémentation, mon premier objectif a été de restructurer le code. Ceci permettra une modification de l'algorithme DeCo sans affecter le résultat rendu par les autres étapes de la méthode. La première étape a donc été de recenser l'ensemble des fonctions du code et de déterminer lesquelles étaient spécifiques aux différentes phases de `mDeCo` et celles utilisées par au moins deux phases afin de les factoriser.
Les modifications apportées à l'architecture du code sont représentées sur le diagramme de

dépendances fonctionnelles de la Figure 4.1. Un nouveau fichier nommé `General` a été créé pour stocker les fonctions utilisables par plusieurs exécutables de la méthode et qui provenaient des fichiers de fonctions `Reconcil` et `DECO`. Les fonctions spécifiques aux différentes parties ont été intégrées dans les fichiers contenant un `main()`. Pour plus de détails sur les fonctions qui ont été déplacées et la restructuration des fichiers voir les figures 3 (p. 47) et 4 (p. 48) en annexe.
Les modifications apportées sur les dépendances fonctionnelles rendent ainsi les quatre étapes de `mDeCo` indépendantes les unes des autres, ce qui n'était pas le cas avant le stage. Comme on peut le voir sur le schéma de la Figure 4.1, il y a également une cinquième étape appelée `Step0_dataset` qui va permettre de générer les fichiers d'entrée de `mDeCo` par extraction des informations depuis des banques de données en ligne.

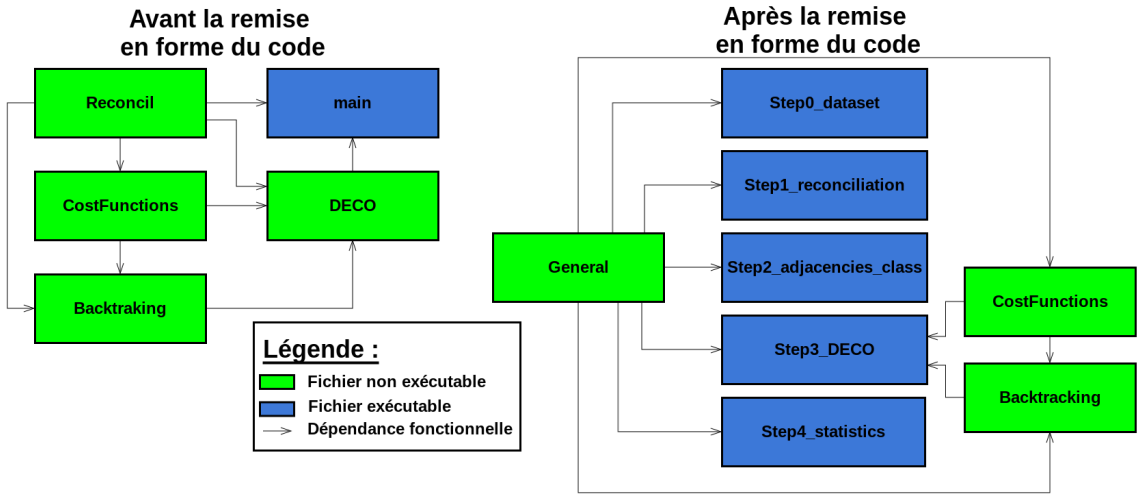


FIGURE 4.1 – Représentation des dépendances fonctionnelles des exécutables de `mDeCo` avant et après le stage.

Nous allons maintenant présenter les différentes améliorations apportées au code. Dans un premier temps, nous présenterons le module `Step0_dataset` permettant de générer les fichiers d'entrée de `mDeCo` puis nous présenterons le module permettant une analyse statistique semi-automatique des résultats en sortie de `mDeCo`.

4.2.1 Ajout d'un module pour générer les fichiers d'entrée de `mDeCo` à partir du système de données d'Ensembl

Pour l'élaboration de ce module, une première étape a consisté à effectuer une recherche des différentes banques de données pouvant fournir les données nécessaires à l'élaboration des fichiers, contenant les informations d'adjacence de gènes, pour la méthode `mDeCo`. Dans cette recherche plusieurs banques de données ont été analysées :

— la banque de données du NCBI qui est la banque de données génomiques de référence. Les fichiers d'annotations génomiques par espèce sont rentrés au format GenBank (*cf.* figures 6 p. 50 et 7 p. 51) qui est le format d'annotation génomique du NCBI. Cependant, lorsque l'on veut accéder aux annotations génomiques d'une espèce, les chromosomes du génome sont contenus dans d'autres fichiers et ces chromosomes sont fragmentés en contigs qui sont situés dans d'autres fichiers. La récupération des données est donc difficile et voulant mettre en place un moyen rapide, nous avons abandonné l'idée d'utiliser les données du NCBI.

— la banque de données Hogenom, développée par le LBBE impliqué dans le projet Ancestrome. Ici, les données ne contiennent pas toutes les informations d'annotations des génomes dont nous avons besoin.

— la banque de données du JGI, la récupération des données au JGI nécessite un compte ce qui complexifie la récupération des données pour les utilisateurs de `mDeCo`.

— la banque de données de Ensembl Genome Browser mise en place par la collaboration de l'EMBL, l'EBI et le WTSI. Les données d'annotations génomiques sont stockées au format EMBL (*cf.* figures 8 p. 52 et 9 p. 53) et peuvent être récupérées soit par le serveur FTP d'Ensembl, soit par l'**API!** Perl d'Ensembl.

Notre choix a été d'utiliser le serveur FTP d'Ensembl car, les données génomiques des espèces au format EMBL sont faciles d'accès et permettent la mise en place rapide d'un module de création des données d'entrée pour `mDeCo`. De plus, Ensembl est un système bioinformatique très connu dans le domaine de la génomique avec des données régulièrement mises à jour. Les espèces référencées dans Ensembl sont des représentantes des Opisthokontes (un embranchement des espèces eucaryotes) avec une majorité de mammifères (41 espèces parmi les 66 présentes).
Le module permettant de générer les fichiers d'entrée `mDeCo` est appelé `Step0_dataset` (*cf.* Figure 4.1 (p. 20) et algorithme 1). Cependant, pour comprendre l'algorithme général de ce module, il faut comprendre les différents algorithmes le composant.

Algorithme général de `Step0_dataset`

Algorithm 1: *Step*0*_dataset*

Données: L_{Esp}, $Arbre_{Esp\_entree}$, $Fichier_{EMF}$
Résultat: $Fichier_{Esp-Gene}$, $Fichier_{Adj}$, $Fichier_{Arbres\_genes\_reconcil}$, $Arbre_{Esp\_sortie}$, $Fichier_{Annot}$

vector<String> L_{Genes} ;
pour *Esp dans* L_{Esp} **faire**
  Création $FicAnnot_{Esp}(Esp)$; (*cf.* algorithme 2)
$restrict2commonsDeLuxe.pl(L_{Esp}, Arbre_{Esp\_entree})$;
  renvoyer $Arbre_{Esp\_sortie}$;
$Recup_{Arbres\_genes}(Fichier_{EMF})$; (*cf.* algorithme 3)
  renvoyer $Fichier_{Arbres\_genes}$;
pour *Arbre_genes dans* $Fichier_{Arbres\_genes}$ **faire**
  $reconcil(Arbre\_genes, Arbre_{Esp\_sortie})$;
  pour *Gene dans Arbres_genes_reconcil* **faire**
    Ajout *Gene* dans L_{Genes} ;
  renvoyer *Arbres_genes_reconcil* ;
Création $Fichier_{Annot}(L_{Genes}, L_{Esp})$; (*cf.* algorithme 4)
  renvoyer $Fichier_{Annot}$;
Création $Fichier_{Esp-Gene}$&$Fichier_{Adj}$; (*cf.* algorithme 5)
  renvoyer $Fichier_{Esp-Gene}$, $Fichier_{Adj}$;

Élaboration d'un fichier d'annotation génomique par espèce

Dans un premier temps, le module va permettre à partir d'une liste d'espèces, dont l'utilisateur souhaite reconstruire l'histoire évolutive des adjacences, de récupérer les fichiers d'annotations génomiques de ces espèces au format EMBL et de générer un fichier d'annotation génomique pour chaque espèce (*cf.* algorithme 2). Le format EMBL est, avec le format GenBank,

le format de référence pour stocker les informations sur la structure et le contenu génomique des espèces. Cette grande quantité d'informations a cependant l'inconvénient d'occuper une grande quantité d'espace mémoire, les fichiers EMBL du génome d'Homo sapiens représente 4,55 Go d'espace mémoire !
L'algorithme effectue une première étape de téléchargement et décompression des données assez longue vu le volume de données à télécharger. Elle est suivie par une étape de *parsing* des fichiers EMBL pour récupérer les informations d'intérêts et les stocker dans un fichier d'annotation génomique plus léger et contenant toutes les informations nécessaires pour générer les fichiers d'entrée de `mDeCo` où chaque ligne correspond aux informations d'un gène et se présente sous la forme suivante :

$$Nom_{Esp} \quad Nom_{Chr/Contig} \quad Pos_{Deb} \quad Pos_{Fin} \quad ID_{Gene} \quad ID_{ARN} \quad ID_{Prot}$$

L'algorithme prend en entrée le nom de l'espèce et le fichier EMBL lui correspondant à télécharger. Lors du parcours du fichier d'annotation génomique, on récupère le nom du chromosome/contig puis pour chaque gène, l'algorithme récupère les positions de début et de fin de ceux-ci ensuite l'algorithme récupère les identifiants de gène, d'ARN et de protéine, si l'ARN code pour une protéine, correspondants aux positions de gènes récupérés précédemment. Une fois toutes les informations d'un gène récupérées les informations sont retournées dans un fichier d'annotation génomique de l'espèce sous le format précédemment décrit.

Algorithm 2: Création $FicAnnot_{Esp}$

Données: Esp, $FicEMBL_{Esp}$
Résultat: $FicAnnot_{Esp}$

String N_{Contig}, P_{Deb}, P_{Fin}, ID_{Gene}, ID_{ARN}, ID_{Prot}, B_{Prot}, B_{Gene} ;
Télécharger et décompresser $FicEMBL_{Esp}$;
pour *Ligne* dans $FicEMBL_{Esp}$ **faire**
  si *Ligne* commence par un tag *ID* **alors**
    N_{Contig}=2e mot de la ligne ;
  si ligne contient les positions de début et fin d'un gène **alors**
    si P_{Deb} !="" **alors**
      Permet d'initialiser les variables ou d'écrire les gènes dans le fichier d'annotation d'espèce, car la position du gène est la première information génique contenu dans les fichiers EMBL (*cf.* Figure 9 (p. 53)) ;
      si ID_{Prot}==B_{Prot} **alors**
        Écrire "Esp N_{Contig} P_{Deb} P_{Fin} ID_{Gene} ID_{ARN}" dans $FicAnnot_{Esp}$
      sinon
        Écrire "Esp N_{Contig} P_{Deb} P_{Fin} ID_{Gene} ID_{ARN} ID_{Prot}" dans $FicAnnot_{Esp}$
    P_{Deb}=valeur de la position de début d'un gène ;
    P_{Fin}=valeur de la position de fin d'un gène ;
  si *Ligne* contient un identifiant de gène **alors**
    ID_{Gene}=identifiant de gène de cette ligne ;
  si *Ligne* contient un identifiant d'ARN **alors**
    si B_{Gene} !=ID_{Gene} **alors**
      ID_{ARN}=identifiant d'ARN de cette ligne ;
  si *Ligne* contient un identifiant de protéine **alors**
    si B_{Gene} !=ID_{Gene} **alors**
      ID_{Prot}=identifiant de protéine de cette ligne ;
      B_{Gene}=ID_{Gene} ;
si ID_{Prot}==B_{Prot} **alors**
  Écrire "Esp N_{Contig} P_{Deb} P_{Fin} ID_{Gene} ID_{ARN}" dans $FicAnnot_{Esp}$;
sinon
  Écrire "Esp N_{Contig} P_{Deb} P_{Fin} ID_{Gene} ID_{ARN} ID_{Prot}" dans $FicAnnot_{Esp}$;

Au cours de cette étape, nous avons observé plusieurs difficultés :

— Pour certaines espèces comme l'homme ou la souris, certains fichiers EMBL correspondent à des fragments chromosomiques qui ont récemment obtenu une mise à jour apportant des corrections sur l'annotation du génome et qui seront intégrées au génome lors de la prochaine version d'Ensembl. Or, lorsque ces fichiers sont analysés par notre programme, nous ne pouvons pas associer les adjacences des gènes situés aux extrémités du *PATCH* avec ceux localisés sur le chromosome et ainsi, nous générons un nouveau contig fictif. Deux solutions sont alors possibles :

  — soit nous effectuons un tri lors de la phase de récupération des fichiers au format EMBL (Les portions chromosomique contenu dans les *PATCHs* ont des identifiants particuliers en fonction des espèces. Par exemple pour Homo sapiens, les N_{Contig} ont cette forme là : HSCHR[0-9]*_[0-9][A-Z]*_[0-9][A-Z]* ou HG[0-9]*_PATCH. Pour Mus musculus, il est de cette forme là : MG[0-9]*_PATCH).

— soit on laisse l'étape de restriction de la liste des gènes aux arbres de gènes (*cf.* algorithme 4 (p. 25)) éliminer ces gènes. Ces corrections étant récentes, les nouveaux gènes contenus dans les *PATCHs* ne sont pas présents dans les arbres de gènes.

— Pour ce qui est de la récupération des ID_{Prot} et ID_{ARN} : l'algorithme récupère le premier ID_{ARN} et le premier ID_{Prot}. Or, pour un même gène il peut y avoir plusieurs ID_{Prot} et ID_{ARN}, et nous ne savons pas quel ID_{Prot} ou ID_{ARN} Ensembl sélectionne pour les arbres de gènes.

— Pour quelques génomes certains chromosomes sont parfois des ID difficiles à comprendre (Exemple : pour Pongo abelii, l'espèce contient 23 chromosomes et sont notés de 1 jusqu'à 22 (avec 2a et 2b) et dans les fichiers EMBL nous avons également les chromosomes 1_random, 2a_random, ..., 22_random en plus des chromosomes 1 à 22.

Restriction de l'arbre des espèces à la liste des espèces dont on veut reconstruire l'histoire évolutive des adjacences

Pour générer les données d'entrée de `mDeCo`, il faut restreindre l'arbre des espèces de référence, récupéré de la base de données d'Ensembl, à la liste des espèces dont nous avons les données génomiques. Pour cela nous utilisons un script ($restrict2commonsDeLuxe.pl(L_{Esp}$, $Arbre_{Esp\_entree})$) développé par V. Berry qui, à partir d'un arbre d'espèces de référence et la liste d'espèces dont nous voulons reconstruire l'histoire évolutive des adjacences, va générer l'arbre des espèces en entrée de `mDeCo`.

Récupération des arbres de gènes

L'étape suivante consiste à télécharger les arbres de gènes contenus dans la banque de données d'Ensembl qui sont au format EMF et les stocker dans un fichier au format NHX ou Newick (*cf.* figures 10 (p. 54) et 11 (p. 54) pour les formats et *cf.* algorithme 3).
Dans la base de données d'Ensembl, il y a le choix entre deux fichiers d'arbres de gènes, le fichier EMF contenant les arbres de gènes codant pour des protéines ou le fichier EMF contenant les arbres de gènes codant pour les ARN non codants. Le module développé ici permet de faire le choix entre l'un, l'autre ou bien les deux. Cependant, après avoir effectué des tests sur ces arbres de gènes, nous nous sommes aperçus que l'ensemble des arbres de gènes codants pour les ARNnc sont multifourchés, c'est à dire non binaires, et que `mDeCo` ne prend pas en compte ce type d'arbres. Nous nous sommes donc exclusivement basés sur les arbres de gènes au format NHX codants pour des protéines pour reconstruire l'histoire évolutive des adjacences de gènes.

Algorithm 3: $Recup_{Arbres\_genes}$

Données: $Fichier_{EMF}$
Résultat: $Fichier_{Arbres\_genes}$

Récupération fichier EMF des arbres de gènes protéiques au format NHX du site d'Ensembl (wget & gunzip) ;
pour *Ligne dans* $Fichier_{EMF}$ **faire**
  si *Ligne commence par "DATA"* **alors**
    Pour la ligne suivante :
    Échanger $ID_{Proteine}$ par ID_{gene} (sed) ;
    Stocker la ligne dans $Fichier_{Arbres\_genes}$;
Renvoyer $Fichier_{Arbres\_genes}$;

Réconciliation des arbres de gènes avec la topologie de l'arbre des espèces

Après téléchargement et transformation des arbres de gènes au format NHX, nous effectuons une étape de réconciliation de la topologie des arbres de gènes avec celle de l'arbre des espèces ($reconcil(Arbre\_genes, Arbre_{Esp\_sortie})$). Cette étape est la même que l'étape `Step1_reconciliation`, ainsi lorsque que l'on génère son jeu de données avec l'étape `Step0_dataset`, il est inutile de faire l'étape `Step1_reconciliation` lors du lancement de la méthode `mDeCo`.

Restriction de la liste de gènes aux gènes contenus dans les arbres de gènes

Une fois la réconciliation terminée, nous utilisons les gènes contenus dans les arbres de gènes pour épurer la liste des gènes contenus dans les fichiers d'annotations génomiques par espèce(*cf.* algorithme 4). L'algorithme prend donc en entrée la liste des gènes contenus dans les arbres de gènes et les fichiers d'annotations des espèces et génère en sortie un fichier d'annotations génomiques épuré des gènes qui ne sont pas contenus dans les arbres de gènes et contenant les informations génomiques de toutes les espèces dont nous voulons reconstruire l'histoire évolutive des adjacences de gènes. Cela permet lors de la création du fichier d'adjacences de gènes de ne pas générer des adjacences avec des gènes non présents dans les arbres de gènes.

Algorithm 4: Création $Fichier_{Annot}$

Données: L_{Genes}, L_{Esp}
Résultat: $Fichier_{Annot}$

pour *Esp* dans L_{Esp} **faire**
  Récupération fichier d'annotation de *Esp* ;
  pour *Ligne* dans $FicAnnot_{Esp}$ **faire**
    Récupérer ID_{Gene} ;
    pour *Gene* dans L_{Genes} **faire**
      si ID_{Gene}==*Gene* **alors**
        L_{Genes}.erase(Gene) ;
        Écrire *Ligne* dans $Fichier_{Annot}$;
        Break ;
Renvoyer $Fichier_{Annot}$;

Création des fichiers d'entrée de `mDeCo`

Après cet étape vient la dernière étape du module qui consiste à partir du fichier d'annotations génomiques épuré de généré simultanément le fichier d'association gène-espèce et le fichier d'adjacences de gènes (*cf.* algorithme 5).

Algorithm 5: Création $Fichier_{Esp-Gene}$&$Fichier_{Adj}$

Données: $FichierAnnot$
Résultat: $Fichier_{Esp-Gene}$, $Fichier_{Adj}$

String Esp, $Nom_{Chr/Contig}$, ID_{Gene}, $Buffer$;
Entier Pos_{Deb}=0 ;
Entier $Buffer_{Int}$;
pour *Ligne* dans $Fichier_{Annot}$ **faire**
  $Buffer$= 1er mot correspondant au Nom de l'espèce ;
  si $Buffer$==Esp **alors**
    $Buffer$= 2e mot correspondant au Nom de chromosome/contig ;
    si $Buffer$==$Nom_{Chr/Contig}$ **alors**
      Écrire ID_{Gene} dans $Fichier_{Adj}$; $Buffer_{Int}$= 3e mot correspondant à la position de début du gène ;
      si $Buffer_{Int}$<Pos_{Deb} **alors**
        Renvoyer "Erreur le fichier n'est pas trié dans l'ordre génomique !!!" ;
      sinon
        Pos_{Deb}=$Buffer_{Int}$;
        $Buffer$= 5e mot correspondant à l'identifiant du gène ;
        Écrire Esp ID_{Gene} dans $Fichier_{Esp-Gene}$;
        Écrire ID_{Gene} dans $Fichier_{Adj}$;
    sinon
      $Nom_{Chr/Contig}$=$Buffer$;
      Pos_D= 3e mot correspondant à la position de début du gène ;
      ID_{Gene}= 5e mot correspondant à l'identifiant du gène ;
      Écrire Esp ID_{Gene} dans $Fichier_{Esp-Gene}$;
  sinon
    Esp=$Buffer$;
    $Nom_{Chr/Contig}$= 2e mot correspondant au Nom de chromosome/contig ;
    Pos_D= 3e mot correspondant à la position de début du gène ;
    ID_{Gene}= 5e mot correspondant à l'identifiant du gène ;
    Écrire Esp ID_{Gene} dans $Fichier_{Esp-Gene}$;
Renvoyer $Fichier_{Esp-Gene}$ et $Fichier_{Adj}$;

4.2.2 Ajout d'un pipeline pour l'analyse statistique de `mDeCo`

Nous avons amélioré le module d'analyse statistiques `Step4_statistics` issu de la restructuration du code en ajoutant de nouvelles fonctions pour donner plus d'informations sur les données de statistiques. Certaines informations essentielles n'étaient jusque-là pas connues en sortie de la méthode (nombre d'espèces, distribution du nombre de voisins/gènes, ...).
Il a donc fallu compléter les informations en sortie de la méthode afin de juger de la fiabilité et de la robustesse de la méthode (ex : le nombre de voisins pour les gènes ancestraux).
La synthèse statistique de ces informations sur les données d'Ensembl est résumée dans un fichier où les informations sont classées dans différentes parties. Une première qui va synthétiser les informations liées aux espèces présentes dans l'arbre des espèces (Nombre d'espèces avec ou sans données génomiques, total et la liste des espèces sans données). Une deuxième qui donne les informations sur les gènes (nombre de gènes, ancestraux, actuels et totaux ; Nombre de gènes ancestraux, actuels et totaux avec au moins une adjacence ou sans adjacence, ...). Une troisième partie résume les informations sur les adjacences, la quatrième donne les statistiques sur les

classes d'équivalence d'adjacences, la cinquième sur les arbres d'adjacences et enfin la dernière résume le nombre des différents événements évolutifs.
Au cours de la mise en place de ces statistiques, nous avons observé que dans notre jeu de données, beaucoup d'espèces n'ont pas de données génomiques. Cette observation nous a fait réfléchir sur la qualité de notre jeu de données et sur le manque de méthodes pour visualiser nos données.
Nous avons donc mis en place un pipeline pour visualiser les différentes distributions du fichier de synthèse statistiques.
Dans un premier temps, un script permet de parser les fichiers de synthèse statistique sous la forme de fichiers au format CSV (*cf.* annexe Figure 5 (p. 49)) pour chaque distribution via un script python, nommé **parser_stats_machine.py**). Les fichiers CSV sont ensuite lus par un script R nommé **graphes_script.r** qui va permettre de générer les graphes des distributions pour visualiser nos données (exemples : *cf.* figures 4.3 (p. 28) et 4.4 (p. 30)). Le pipeline pour l'analyse statistique des données de `mDeCo` est représenté par le schéma de la Figure 4.2.

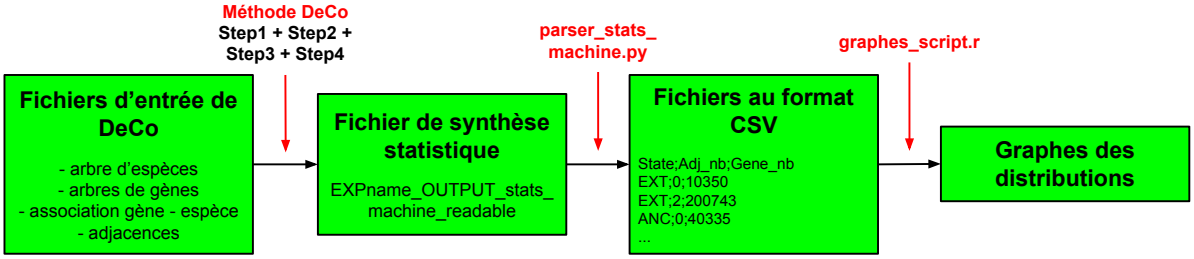


FIGURE 4.2 – Schéma représentant le fonctionnement du pipeline pour la visualisation des distributions des données statistiques de `mDeCo`.

La mise en place de ce pipeline d'analyse statistiques a permis de mettre à jour la mauvaise qualité de notre jeu de données et du besoin d'en générer un nouveau afin de tester la robustesse de l'algorithme DECO et d'apporter des modifications sur celui-ci.

Une fois la mise en place de ces deux modules en amont et en aval de `mDeCo` (pour la création de notre jeu de données et pour l'analyse statistique), nous avons ensuite effectué des tests pour évaluer la robustesse de l'algorithme DECO.

4.3 Tests effectués pour étudier la robustesse de l'algorithme DECO

4.3.1 Effet de l'absence de données génomiques

Des tests ont été effectués pour évaluer la fiabilité et la robustesse de DECO.
Une des premières observations que nous avons faites avec le jeu de données original était que pour un grand nombre d'espèces dans l'arbre d'espèces nous n'avions pas d'informations génomiques.

Nous nous sommes donc posés la question de savoir si l'absence de données génomiques pour certaines espèces pouvait affecter la reconstruction de l'histoire évolutive des adjacences de gènes des autres espèces ?

Nous avons effectué une expérience avec pour témoin un jeu de données avec un fichier d'arbres de gènes, un fichier d'adjacences de gènes et un arbre des espèces contenant 66 espèces

dont 11 seulement avec des données génomiques et un jeu de données test avec les données en entrée excepté l'arbre des espèces qui est restreint aux 11 espèces dont nous avons les données génomiques. La restriction de l'arbre des espèces va donc changer la réconciliation des arbres de gènes avec l'arbre des espèces, ce qui va donc potentiellement affecter les inférences d'adjacences des gènes.
Pour déterminer si la restriction de l'arbre améliore les prédictions faites par DeCo, nous nous sommes basés sur le critère du nombre d'adjacences pour les gènes ancestraux (*cf.* graphique de la Figure 4.3).

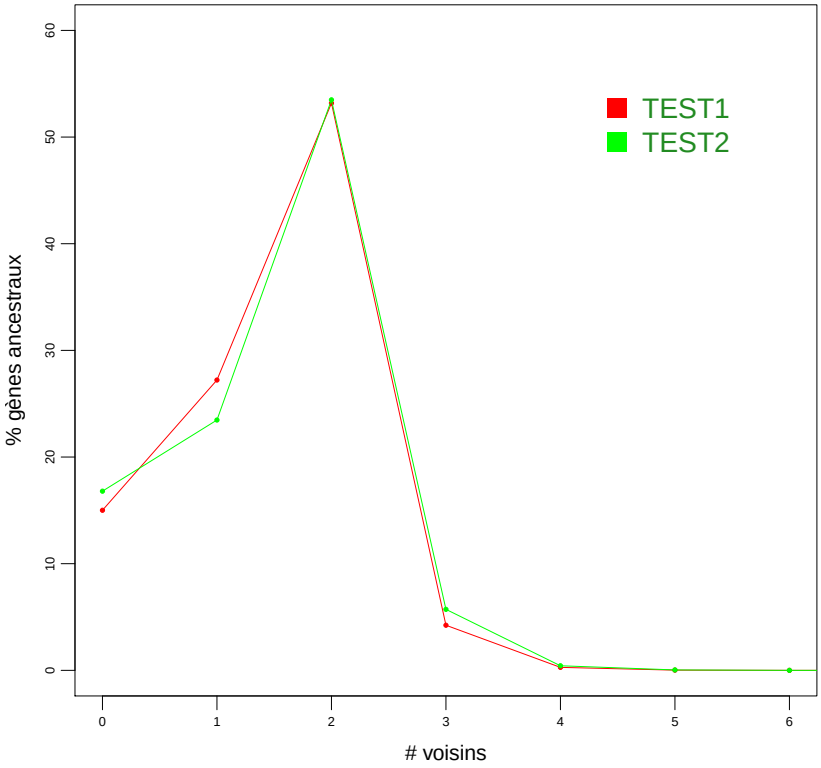


Figure 4.3 – Graphique présentant la distribution du pourcentage de gènes ancestraux en fonction du nombre de voisins, généré à partir du script R **parser_stats_machine.py**.

L'optimum recherché sur cette distribution est une courbe où plus de 99% des gènes ancestraux auraient 2 voisins et le reste des gènes auraient 1 voisin. Au vu du graphique, on observe qu'il y a très peu de différences entre l'arbre restreint aux espèces ayant des données génomiques (courbe verte TEST2) et l'arbre non restreint (courbe rouge TEST1). La méthode est donc peu sensible à la présence d'espèces sans donnée génomique, cependant lorsque l'on utilise l'arbre des espèces non restreint le programme effectue des calculs d'histoire évolutive inutiles et affecte des pertes de gènes et d'adjacences fictives dues à l'absence de données sur les gènes dans ces espèces. Cela entraîne donc une erreur lors de l'analyse statistique des événements évolutifs et raconte une histoire évolutive tronquée car, pour un bon nombre d'espèces aucune histoire évolutive n'est retranscrite dans les arbres d'adjacences.

4.3.2 Effet des génomes mal assemblés

Par la suite, nous avons effectué un autre test suite à la détection de certaines espèces ayant une qualité d'assemblage médiocre dans les banques de données, comme l'Ornithorynque (*Ornithorhynchus anatinus*), dont le génome est composé de 24 chromosomes. Les données génomiques d'assemblage du génome de l'Ornithorynque indiquent que son génome est fragmenté en 9844 morceaux chromosomiques et que sur les 19960 gènes qui le composent 8081 (40.5%) n'ont aucune adjacence, 3526 (17.7%) n'ont qu'une seule adjacence . Pour rappel, selon la définition d'une adjacence (*cf.* 4.2) seuls $24 * 2 = 48$ gènes (soit $50/19960 * 100 = 0.24\%$ des gènes totaux) devraient avoir une seule adjacence et le reste des gènes devraient avoir deux gènes adjacents.

La question qui était posée était de savoir si l'algorithme DeCo était robuste à la mauvaise qualité d'assemblage de certains des génomes considérés.

Nous avons tout d'abord créé une série de scripts permettant de générer des génomes mal assemblés : un premier script permettant à partir d'un fichier d'adjacences de supprimer un pourcentage voulu d'adjacences d'une espèce choisie ; un second script permettant de falsifier un pourcentage d'adjacences de gènes d'une espèce choisie et un dernier script permettant de lancer plusieurs exécutions de la méthode `mDeCo` de manière automatique.

Nous avons mis au point un protocole de test, pour lequel tous les arbres de gènes et l'arbre des espèces sont les mêmes mais d'autres facteurs changent, amenant aux 14 expériences suivantes :

- TEM1 : Arbre des espèces avec 11 espèces sans modification des données ;
- TEM2 : Arbre des espèces avec 10 espèces (suppression de l'Ornithorynque qui est l'espèce la plus mal assemblée) ;
- HOS1 : Arbre des espèces avec 10 espèces où l'on a retiré 50% des adjacences de l'espèce Homo sapiens ;
- HOS2 : Arbre des espèces avec 10 espèces où l'on a retiré 50% des adjacences et falsifié 10% des adjacences restantes de l'espèce Homo sapiens ;
- HOS3 : Arbre des espèces avec 10 espèces où l'on a retiré 50% des adjacences et falsifié 20% des adjacences restantes de l'espèce Homo sapiens ;
- HOS4 : Arbre des espèces avec 10 espèces où l'on a retiré 50% des adjacences et falsifié 30% des adjacences restantes de l'espèce Homo sapiens ;
- HOS5 : Arbre des espèces avec 10 espèces où l'on a retiré 50% des adjacences et falsifié 40% des adjacences restantes de l'espèce Homo sapiens ;
- HOS6 : Arbre des espèces avec 10 espèces où l'on a retiré 50% des adjacences et falsifié 50% des adjacences restantes de l'espèce Homo sapiens ;
- MOD1 à MOD6 : Idem qu'avec l'espèce Homo sapiens mais avec l'espèce Monodelphis domestica.

Nous avons choisis Homo sapiens et Monodelphis domestica pour la simulation de mauvaise qualité d'assemblage de génome car elles sont situées à deux extrêmes de la topologie de l'arbre des espèces (*cf.* Figure 12) et nous voulions savoir si la localisation d'une espèce mal assemblée pouvait affecter l'histoire évolutive d'adjacences reconstruite par la méthode `mDeCo` ?

Dans la Figure 4.4, la courbe en magenta représente la courbe optimale ou plus de 99% des gènes ont deux adjacences (voisins) et moins de 1% ont un voisin. On observe que la courbe

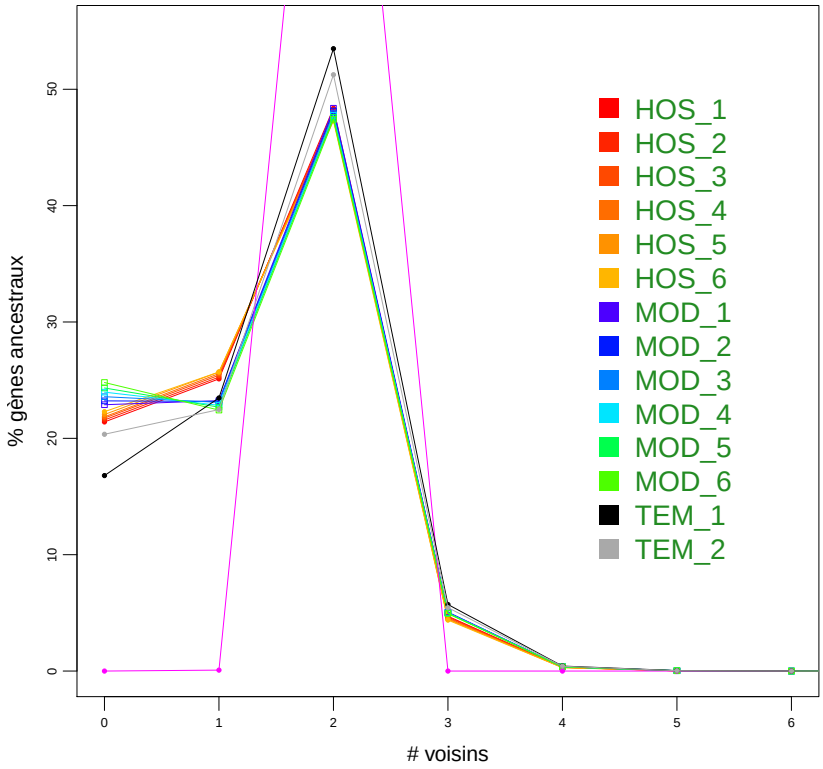


FIGURE 4.4 – Graphique présentant la distribution du pourcentage de gènes ancestraux en fonction du nombre de voisins, généré à partir du script R **parser_stats_machine.py**.

s'approchant le plus de cet optimum est celle du témoin TEM1 où on a conservé les 11 espèces dont l'ornithorynque qui possède un génome très mal assemblé. Or, on s'attendait en enlevant l'Ornithorynque de l'arbre des espèces, à améliorer le critère du nombre de voisins par gènes ancestraux ce qui n'est pas le cas bien qu'il y ait une faible différence entre les deux courbes.
Pour ce qui est de la simulation de génomes mal assemblés, on observe que malgré de très fortes perturbations dans les génomes (50% d'adjacences supprimées et 50% de gènes falsifiés entre les adjacences), il y a de très faibles différences par rapport au témoin TEM2. Nous en déduisons donc globalement que l'algorithme DECO est robuste à l'effet de quelques génomes mal assemblés.

Au cours de cette série de tests et de l'observation d'assemblages incomplets de certains génomes dans les banques de données internationales, nous avons réfléchi à une adaptation de la méthode DeCo pour son application à la problématique d'assemblage des génomes.

Chapitre 5

Élaboration d'un algorithme pour compenser l'assemblage imparfait des génomes

Actuellement, en génomique, une des plus grandes problématiques est de pouvoir à partir du séquençage d'une espèce pour laquelle on ne possède pas de génome de référence, reconstruire le génome à partir de millions de fragments d'ADN séquencés (appelés *reads*). Les méthodes actuelles consistent dans un premier temps à associer les *reads*, après séquençage du génome, en chevauchant les fragments identiques entre ces fragments. On obtient ainsi ce qu'on appelle des **contigs** qui sont des fragments de chromosomes contenant un à plusieurs gènes. Arrivé à cette étape les méthodes ont bien souvent un manque d'informations pour pouvoir associer les contigs et former les chromosomes et l'assemblage du génome est alors incomplet. L'algorithme DeCo permettant de reconstruire l'histoire évolutive des adjacences de gènes des espèces ancestrales, nous avons voulu étendre l'algorithme afin d'associer des contigs d'une espèce actuelle aux adjacences des autres espèces actuelles et aux adjacences inférées par les espèces ancestrales voisines. Ceci permettrait d'améliorer l'assemblage des génomes.

5.1 Réflexion algorithmique et élaboration des formules de calcul

Nous avons réfléchi à un moyen d'adapter DeCo à la problématique d'assemblage des génomes. L'approche que nous avons mis au point consiste à vouloir associer des adjacences de gènes à des gènes dont nous ne connaissons pas le voisinage. Pour cela, nous devons calculer en amont de DeCo les probabilités d'adjacences de chaque couple de gènes en fonction des adjacences qui sont déjà connues et celles que nous ignorons suite au manque d'informations des méthodes d'assemblage des génomes. Ces probabilités sont ensuite prises en compte dans le calcul de la matrice de coûts minimum entre les couples de gènes de deux arbres de gènes partageant une adjacence ancestrale commune. Pour cela, nous avons dû également modifier la méthode de calcul de la matrice de programmation dynamique que nous présenterons dans la partie 5.2.

L'algorithme que nous avons développé au cours de ce stage consiste dans un premier temps à calculer les probabilités d'adjacences de tous les couples de gènes en fonction des données génomiques.
Nous cherchons donc à calculer pour une espèce donnée la probabilité que deux gènes v_1 et v_2 soient adjacents notée $P(v_1 \sim v_2)$. Si les données donnent l'information que les deux gènes sont adjacents alors, $P(v_1 \sim v_2) = 1$ sinon, on calcule les solutions où v_1 et v_2 sont adjacents suivant les combinaisons d'adjacences possibles entre les contigs, où l'on a : $P(v_1 \sim v_2) + P(v_1 \not\sim v_2) = 1$.

Pour cela, nous distinguons différentes notations permettant d'expliciter cet algorithme :

— n = nombre de contigs assemblés ($n \geq 1$) ;
— p = nombre de chromosomes attendus ($p \geq 1$) (Connus grâce aux observations des génomes au microscope) ;
— v_1 = gène 1 ;
— v_2 = gène 2.

La formule de la probabilité que deux gènes v_1 et v_2 soient adjacents est :

$$P(v_1 \sim v_2) = \frac{\#\text{ solutions où } v_1 \sim v_2 \text{ sachant } n \text{ contigs et } p \text{ chromosomes}}{\#\text{ solutions pour transformer } n \text{ contigs en } p \text{ chromosomes}} \tag{5.1}$$

Dans un premier temps, nous avons élaboré une formule pour calculer le dénominateur de la formule, soit le nombre $f(n,p)$ de solutions pour assembler n contigs en p chromosomes :

$$f(n,p) = \frac{1}{p} \times \sum_{x=1}^{n-(p-1)} ((2^{x-1})x! \times C_n^x \times f(n-x, p-1)) \tag{5.2}$$

Pour rappel, voici la formule du coefficient binomial :

$$C_n^x = \binom{n}{x} = \frac{n!}{x!(n-x)!} \tag{5.3}$$

Les cas d'arrêt de la formule sont les suivants :

— $f(n,n) = 1$
— $f(n,1) = 2^{n-1} \times n!$
— $f(n < p, p) = Impossible$

Nous allons décomposer la formule pour expliquer comment nous sommes arrivés à ce résultat.
La $\sum$ pour la formule de récurrence va de $x = 1$ à $x = n - (p-1)$, où x = nombre de contigs contenus dans un chromosome. Car chaque chromosome contient au minimum 1 contig et au maximum $n - (p-1)$ contigs, ce cas se produit si les $(p-1)$ autres chromosomes contiennent 1 seul contig. Si $x = n - (p-1)$ alors $f(n-x, p-1) = f(a,a)$ où $a = (n-x) = (p-1)$ et $f(a,a) = 1$ selon le premier cas d'arrêt de la formule, avec x = nombre de contigs contenus dans un chromosome.
La formule $(2^{x-1})x!$ correspond au nombre de possibilités d'ordonner x contigs dans un chromosome. $x!$ étant le nombre de possibilités de permuter x éléments et 2^{x-1} permettant de prendre en compte les deux orientations possibles pour chaque contig.
C_n^x correspond au nombre de possibilités de prendre x contigs parmi n.
$f(n-x, p-1)$ permet de remplir les $(p-1)$ chromosomes restants avec les $(n-x)$ contigs restants.
$1/p$ en début de formule correspond au fait que des solutions vont apparaître plusieurs fois. Sur l'exemple de la Figure 5.1, la combinaison ABCD|EF (première ligne de la première colonne) est la même que DCBA|EF, il faut donc diviser par p pour éviter de compter une même solution plusieurs fois.

Une fois la formule (5.2) établie pour le calcul du nombre de possibilités de transformer n contigs en p chromosomes, nous avons élaboré une formule pour calculer le nombre de solutions où $v_1 \sim v_2$ sachant n contigs et p chromosomes. Une première étape pour déterminer le numérateur de la formule (5.1) est de définir un paramètre $\rho(v_1, v_2)$ permettant de déterminer le nombre d'associations possibles entre les gènes v_1 et v_2. Si un gène est seul sur un contig alors celui-ci est assigné comme ayant 0 gène voisin par-contre si un contig contient plusieurs gènes alors les deux gènes situés à ses extrémités ont déjà un gène adjacent chacun. La probabilité que le gène

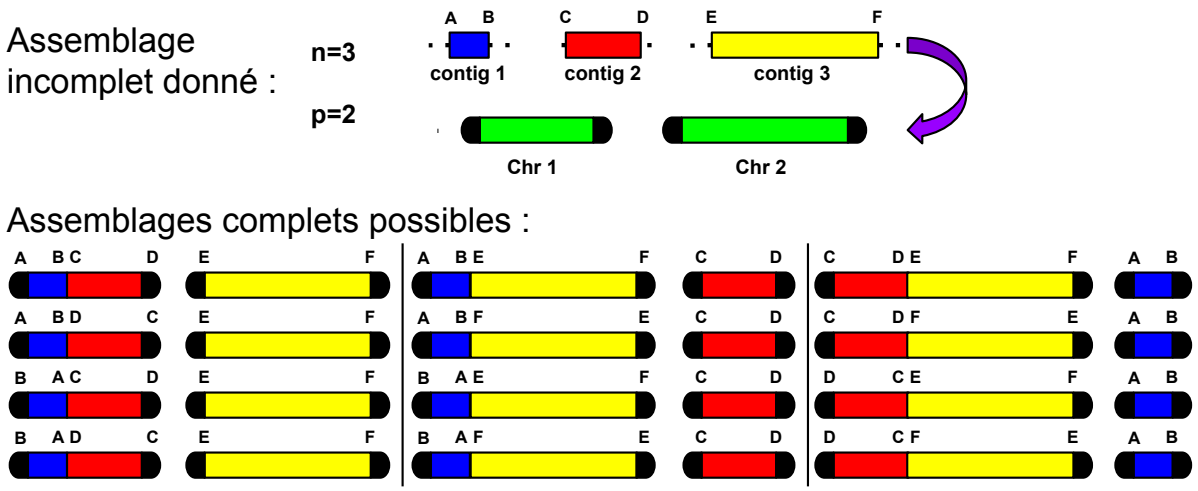


FIGURE 5.1 – Représentation des combinaisons possibles lorsque l'on a 3 contigs ($n = 3$) et que l'on veut obtenir 2 chromosomes ($p = 2$), soit $f(3, 2) = 12$ combinaisons possibles.

ayant 0 voisin soit adjacent à un autre gène est deux fois plus grande qu'un gène ayant déjà 1 voisin puisqu'il peut être adjacent soit par son côté gauche soit par son côté droit. Pour prendre en compte cela, on fixe le paramètre $\rho(v_1, v_2)$ comme suit :

$$\rho(v_1 \sim v_2) = \begin{cases} 4 & \text{si } v_1 \text{ et } v_2 \text{ ont tous deux 0 voisin} \\ 2 & \text{si } v_1 \text{ a 0 voisin et } v_2 \text{ a 1 voisin (et réciproquement)} \\ 1 & \text{si } v_1 \text{ et } v_2 \text{ ont tous deux 1 voisin et ne sont pas initialement adjacents} \\ 0 & \text{si } v_1 \text{ et/ou } v_2 \text{ a/ont 2 voisins et ne sont pas initialement adjacents} \\ \frac{f(n,p)}{\sum_{x=2}^{n-(p-1)}(g_x(n,p))} & \text{si } v_1 \sim v_2 \end{cases}$$

Ci-dessus, $g_x(n, p)$ correspond au nombre de solutions (c.-à-d. d'assemblabes complets) où $v_1 \sim v_2$ dans un chromosome contenant x contigs sachant initialement n contigs et p chromosomes. On le détermine ainsi :

$$g_x(n, p) = f(x - 1, 1) \times f(n - x, p - 1) \tag{5.4}$$

Cas d'arrêt de la formule :

$$g_x(n, 1) = \begin{cases} f(x - 1, 1) & \text{si } n = x \\ 0 & \text{sinon} \end{cases}$$

En détaillant la formule $g_x(n, p)$, on observe une première formule $f(x - 1, 1)$ correspondant au nombre de solutions où un gène adjacent sur un contig est adjacent à un gène situé sur un autre contig dans un chromosome contenant x contigs. Cette formule est ensuite multiplié par $f(n - x, p - 1)$ correspondant au nombre d'arrangements possibles en fonctions du nombre de chromosomes et de contigs restants.

Nous avons donc déterminé le numérateur de la formule (5.1) :

$$\rho(v_1 \sim v_2) \times \sum_{x=2}^{n-(p-1)} (g_x(n, p)) \tag{5.5}$$

En effet, le chromosome pouvant contenir l'adjacence entre v_1 et v_2 peut être composés de 2 à $n - (p - 1)$ contigs. De plus, il faut prendre en compte pour v_1 et v_2 leur nombre de voisins initialement, c'est pourquoi nous multiplions le tout par le facteur $\rho(v_1 \sim v_2)$.

Au final, nous en déduisons que $P(v_1 \sim v_2)$:

$$P(v_1 \sim v_2) = \frac{\rho(v_1 \sim v_2) \times \sum_{x=2}^{n-(p-1)} (g_x(n,p))}{f(n,p)} \tag{5.6}$$

Le calcul de cette probabilité reposant sur la formule $f(n,p)$, nous avons implémenté un algorithme permettant de calculer la matrice $f(n,p)$ avec le paramètre p en colonne et le paramètre n en ligne (*cf.* algorithme 6). L'algorithme remplit la matrice colonne par colonne.

Algorithm 6: $Matrice\_f(n,p)$

Données: n, p
Résultat: $Matrice\_f(n,p)$

$Entier\ nb\_chr = 0$;
$Entier\ nb\_contig = 0$;
$TabFnp[n+1][p+1]$;

Initialisation
pour *nb_chr de 1 à p* **faire**
  $TabFnp[nb\_chr][nb\_chr] = 1$;
Initialisation 1re *colonne*
pour *nb_contig de 1 à n* **faire**
  $TabFnp[nb\_contig][1] = 2^{nb\_contig-1} * (nb\_contig)!$;
Remplissage tableau
pour *nb_chr de 2 à p* **faire**
  pour *nb_contig de nb_chr + 1 à n* **faire**
    $S = 0$;
    Calcul de S
    pour *x de 1 à nb_contig − (nb_chr − 1)* **faire**
      $\Sigma+ = 2^{x-1}x! * C^x_{nb\_contig} * TabFnp[nb\_contig - x][nb\_chr - 1]$;
    $TabFnp[nb\_contig][nb\_chr] = \Sigma / nb\_chr$;
Renvoyer $TabFnp$;

Une fois la matrice de $f(n,p)$ calculée, celle-ci peut être utilisée pour calculer $P(v_1 \sim v_2)$ pour les couples de gènes pouvant partager une adjacence, sachant que :

$$P(v_1 \sim v_2) = \frac{\rho(v_1 \sim v_2) \times \sum_{x=2}^{n-(p-1)} (f(x-1,1) \times f(n-x,p-1))}{f(n,p)} \tag{5.7}$$

Une fois les valeurs de $f(n,p)$ pré-calculées. La formule nécessite donc uniquement de connaître le nombre d'adjacences initiales de chaque gène, le nombre de chromosomes et le nombre de contigs. Les probabilités seront donc calculées au moment où on en aura besoin, c.-à-d. à la volée dans les équations de programmation dynamique. Ces probabilités sont utilisées pour modifier le schéma de score de l'algorithme DeCo de la manière expliquée dans la section suivante.

5.2 Modification du schéma de scores de l'algorithme DeCo

L'idée principale est de considérer que deux gènes v_1 et v_2 non constatés adjacents mais appartenant à une espèce incomplètement assemblée ont une $P(v_1 \sim v_2)$ d'être en réalité adjacents. Pour intégrer les probabilités d'adjacences dans le calcul de matrice de coût minimum entre les

couples de gènes (*cf.* p. 15 pour rappel de l'algorithme DeCo), nous avons dû modifier les formules de récurrence de l'algorithme DeCo(*cf.* annexe p. 44 pour les cas de la version originale de DeCo) afin qu'elles puissent prendre en compte le nouveau schéma de scores. On note que $ca(v)$ et $cb(v)$ sont respectivement les fils gauche et droit du gène v, et que $E(v)$ est l'événement évolutif du gène v pouvant prendre pour valeur $\{$*Gène actuel*, *Spec*, *GDup*, *GLos*, *ADup*, *ALos*$\}$ qui ont tous été présentés dans la partie 2.3, excepté *Gène actuel* qui n'est pas un événement évolutif mais permet d'annoter les gènes actuels. Nous présentons ci-dessous les modifications apportés aux formules de récurrence :

Cas 1. $E(v_1)$ = Gène actuel et $E(v_2)$ = Gène actuel.
Le changement majeur du schéma de scores s'effectue au niveau du **Cas 1**, car les probabilités d'adjacences sont calculées uniquement entre deux gènes actuels. Nous avons intégré les probabilités d'adjacences des gènes v_1 et v_2 pour le calcul des coûts $c_1(v_1, v_2)$ et $c_0(v_1, v_2)$, et sommes arrivés au résultat suivant :
$c_1(v_1, v_2) = (1 - P(v_1 \sim v_2)) * C(Break)$ et $c_0(v_1, v_2) = P(v_1 \sim v_2) * C(Gain)$

Si les gènes v_1 et v_2 sont adjacents, $P(v_1 \sim v_2) = 1$ et on obtient $c_1(v_1, v_2) = (1 - 1) * C(Break) = 0$ et $c_0(v_1, v_2) = 1 * C(Gain) = C(Gain)$, le score minimal est alors $c_1(v_1, v_2)$ et on va donc reconstruire une histoire évolutive où v_1 et v_2 sont adjacents. Dans le cas contraire où v_1 et v_2 ne sont pas adjacents, le calcul de la $P(v_1 \sim v_2)$ va permettre de potentiellement créer des adjacences entre des gènes dont on ignorait qu'ils étaient adjacents entre eux.
Pour les cas où nous ne cherchons pas à déterminer les coûts c_0 et c_1 entre deux gènes actuels, le calcul des formules de récurrence est le même que pour la version originale de l'algorithme DeCo. Ainsi, pour les cas 2 et 3, le calcul est inchangé tandis que pour les cas 4, 5 et 6 de la version originale de DeCo, les formules ont été modifiées car, les formules peuvent nécessiter le calcul de coûts c_0 et c_1 entre deux gènes actuels.

Cas 2. $E(v_1) = GLos$ et $E(v_2) \neq GLos$.
→ Pas de modification par rapport à la version originale de l'algorithme DeCo (*cf.* annexe p. 44).

Cas 3. $E(v_1) = GLos$ et $E(v_2) = GLos$.
→ Pas de modification par rapport à la version originale de l'algorithme DeCo (*cf.* annexe p. 44).

Cas 4.1. $E(v_1) = Spec$ et $E(v_2) = GDup$.
→ Dans ce cas, reprendre les formules de récurrence du **Cas 4.** de la version originale de DeCo (*cf.* annexe p. 44).

Cas 4.2. $E(v_1)$ = Gène actuel et $E(v_2) = GDup$.
Dans le **Cas 4.2**, l'événement évolutif des fils $ca(v_2)$ et $cb(v_2)$ provenant de la duplication de v_2 peut être soit une spéciation, soit une duplication, soit une perte de gène. Ils peuvent également ne pas avoir subis d'événements évolutifs et sont alors des gènes actuels. La modification des formules de récurrence n'affectant que le cas où l'on cherche à déterminer l'adjacence entre deux gènes actuels, pour chaque coût c_0 et c_1 de chaque formule de la version originale de DeCo, il faut définir deux calculs :

- un calcul où les deux gènes n'ont pas subi d'événement évolutif et sont donc des gènes actuels où on va utiliser les calculs des coûts c_0 et c_1 du **Cas 1**.
- un autre calcul où au moins l'un des deux gènes a subi un événement évolutif et où l'on utilisera le calcul de la version originale de DeCo.

Dans le second cas, il y a cependant une contrainte qui est de savoir si à un coût c_0 est associé un coût de cassure d'adjacence ($C(Break)$) ou si à un coût c_1 est associé un coût de gain d'adjacence ($C(Gain)$). Ceci entraîne également une seconde contrainte car, dans certains calculs des formules de récurrence, un coût $C(Gain)$ peut être associé à un coût c_1 ou à un autre et un coût $C(Break)$ peut être associé à un coût c_0 ou à un autre, cela dépend du scénario initial. Cela nécessite donc de doubler ces cas pour prendre en compte tous les scénarios possibles.
Prenons l'exemple de la formule du **Cas 4** de la version originale de DeCo suivant (*cf.* 44) :

$$c_1(v_1, ca(v_2)) + c_1(v_1, cb(v_2)) + C(Gain)$$

Cette formule traduit le fait qu'au cours de l'évolution si v_1 était adjacent à v_2, après duplication de v_2, v_1 soit adjacent au fils gauche de v_2 ($c_1(v_1, ca(v_2))$) et à son fils droit ($c_1(v_1, cb(v_2))$). On a dans ce cas la création d'une adjacence et on ajoute $C(Gain)$, ce cas doit cependant être doublé car, le coût $C(Gain)$ peut être affecté soit au coût $c_1(v_1, ca(v_2))$ soit coût $c_1(v_1, cb(v_2))$. Cela dépend si le scénario propose que v_1 soit initialement adjacent à $ca(v_2)$ ou à $cb(v_2)$, après modification du calcul de la formule de récurrence, on obtient le résultat suivant :

On établit deux conditions pour déterminer si l'on doit prendre en compte le calcul de c_0 ou c_1 avec le nouveau schéma de score ou avec l'ancien :
$A : E(ca(v_2)) =$ Gène actuel
$B : E(cb(v_2)) =$ Gène actuel

Scénario où v_1 est initialement adjacent à $cb(v_2)$:

$$\begin{pmatrix} \text{Si } A : \\ (1 - P(v_1 \sim ca(v_2)) * C(Break)) \\ \text{Sinon :} \\ c_1(v_1, ca(v_2)) + C(Gain) \end{pmatrix} + \begin{pmatrix} \text{Si } B : \\ (1 - P(v_1 \sim cb(v_2)) * C(Break)) \\ \text{Sinon :} \\ c_1(v_1, cb(v_2)) \end{pmatrix}$$

Scénario où v_1 est initialement adjacent à $ca(v_2)$

$$\begin{pmatrix} \text{Si } A : \\ (1 - P(v_1 \sim ca(v_2)) * C(Break)) \\ \text{Sinon :} \\ c_1(v_1, ca(v_2)) \end{pmatrix} + \begin{pmatrix} \text{Si } B : \\ (1 - P(v_1 \sim cb(v_2)) * C(Break)) \\ \text{Sinon :} \\ c_1(v_1, cb(v_2)) + C(Gain) \end{pmatrix}$$

Les méthodes de calcul présentée ci-dessus ne sont pas spécifiques au **Cas 4.2** et sont également utilisées dans les cas 5 et 6 du schéma de scores.

Schéma complet des formules de récurrence du **Cas 4.2** :

On établit deux conditions pour déterminer si l'on doit prendre en compte le calcul de c_0 ou c_1 avec le nouveau schéma de score ou avec l'ancien :
$A : E(ca(v_2)) =$ Gène actuel
$B : E(cb(v_2)) =$ Gène actuel

$$c_1(v_1, v_2) = min \left\{ \begin{array}{l}
\left(\begin{array}{l} \text{Si } A : \\ (1 - P(v_1 \sim ca(v_2)) * C(Break)) \\ \text{Sinon :} \\ c_1(v_1, ca(v_2)) \end{array} \right) + \left(\begin{array}{l} \text{Si } B : \\ (P(v_1 \sim cb(v_2)) * C(Gain)) \\ \text{Sinon :} \\ c_0(v_1, cb(v_2)) \end{array} \right) \\
\left(\begin{array}{l} \text{Si } A : \\ (P(v_1 \sim ca(v_2)) * C(Gain)) \\ \text{Sinon :} \\ c_0(v_1, ca(v_2)) \end{array} \right) + \left(\begin{array}{l} \text{Si } B : \\ (1 - P(v_1 \sim cb(v_2)) * C(Break)) \\ \text{Sinon :} \\ c_1(v_1, cb(v_2)) \end{array} \right) \\
\left(\begin{array}{l} \text{Si } A : \\ (1 - P(v_1 \sim ca(v_2)) * C(Break)) \\ \text{Sinon :} \\ c_1(v_1, ca(v_2)) + C(Gain) \end{array} \right) + \left(\begin{array}{l} \text{Si } B : \\ (1 - P(v_1 \sim cb(v_2)) * C(Break)) \\ \text{Sinon :} \\ c_1(v_1, cb(v_2)) \end{array} \right) \\
\left(\begin{array}{l} \text{Si } A : \\ (1 - P(v_1 \sim ca(v_2)) * C(Break)) \\ \text{Sinon :} \\ c_1(v_1, ca(v_2)) \end{array} \right) + \left(\begin{array}{l} \text{Si } B : \\ (1 - P(v_1 \sim cb(v_2)) * C(Break)) \\ \text{Sinon :} \\ c_1(v_1, cb(v_2)) + C(Gain) \end{array} \right) \\
\left(\begin{array}{l} \text{Si } A : \\ (P(v_1 \sim ca(v_2)) * C(Gain)) \\ \text{Sinon :} \\ c_0(v_1, ca(v_2)) + C(Break) \end{array} \right) + \left(\begin{array}{l} \text{Si } B : \\ (P(v_1 \sim cb(v_2)) * C(Gain)) \\ \text{Sinon :} \\ c_0(v_1, cb(v_2)) \end{array} \right) \\
\left(\begin{array}{l} \text{Si } A : \\ (P(v_1 \sim ca(v_2)) * C(Gain)) \\ \text{Sinon :} \\ c_0(v_1, ca(v_2)) \end{array} \right) + \left(\begin{array}{l} \text{Si } B : \\ (P(v_1 \sim cb(v_2)) * C(Gain)) \\ \text{Sinon :} \\ c_0(v_1, cb(v_2)) + C(Break) \end{array} \right)
\end{array} \right.$$

$$c_0(v_1, v_2) = min \left\{ \begin{array}{l}
\left(\begin{array}{l} \text{Si } A : \\ (P(v_1 \sim ca(v_2)) * C(Gain)) \\ \text{Sinon :} \\ c_0(v_1, ca(v_2)) \end{array} \right) + \left(\begin{array}{l} \text{Si } B : \\ (P(v_1 \sim cb(v_2)) * C(Gain)) \\ \text{Sinon :} \\ c_0(v_1, cb(v_2)) \end{array} \right) \\
\left(\begin{array}{l} \text{Si } A : \\ (P(v_1 \sim ca(v_2)) * C(Gain)) \\ \text{Sinon :} \\ c_0(v_1, ca(v_2)) \end{array} \right) + \left(\begin{array}{l} \text{Si } B : \\ (1 - P(v_1 \sim cb(v_2)) * C(Break)) \\ \text{Sinon :} \\ c_1(v_1, cb(v_2)) + C(Gain) \end{array} \right) \\
\left(\begin{array}{l} \text{Si } A : \\ (1 - P(v_1 \sim ca(v_2)) * C(Break)) \\ \text{Sinon :} \\ c_1(v_1, ca(v_2)) + C(Gain) \end{array} \right) + \left(\begin{array}{l} \text{Si } B : \\ (P(v_1 \sim cb(v_2)) * C(Gain)) \\ \text{Sinon :} \\ c_0(v_1, cb(v_2)) \end{array} \right) \\
\left(\begin{array}{l} \text{Si } A : \\ (1 - P(v_1 \sim ca(v_2)) * C(Break)) \\ \text{Sinon :} \\ c_1(v_1, ca(v_2)) + C(Gain) \end{array} \right) + \left(\begin{array}{l} \text{Si } B : \\ (1 - P(v_1 \sim cb(v_2)) * C(Break)) \\ \text{Sinon :} \\ c_1(v_1, cb(v_2)) + C(Gain) \end{array} \right)
\end{array} \right.$$

Cas 5. $E(v_1) = Spec$ et $E(v_2) = Spec$.

Soit $Esp(ca(v_1)) = Esp(ca(v_2)), \quad Esp(cb(v_1)) = Esp(cb(v_2)),$

On établit deux conditions pour déterminer si l'on doit prendre en compte le calcul de c_0 ou c_1 avec le nouveau schéma de score ou avec l'ancien :
$C : E(ca(v_1)) = E(ca(v_2))$ = Gène actuel
$D : E(cb(v_1)) = E(cb(v_2))$ = Gène actuel

$$c_1(v_1, v_2) = min \left\{ \begin{array}{l} \left(\begin{array}{l} \text{Si } C : \\ (1 - P(ca(v_1) \sim ca(v_2)) * C(Break)) \\ \text{Sinon :} \\ c_1(ca(v_1), ca(v_2)) \end{array} \right) + \left(\begin{array}{l} \text{Si } D : \\ (1 - P(cb(v_1) \sim cb(v_2)) * C(Break)) \\ \text{Sinon :} \\ c_1(cb(v_1), cb(v_2)) \end{array} \right) \\ \left(\begin{array}{l} \text{Si } C : \\ (1 - P(ca(v_1) \sim ca(v_2)) * C(Break)) \\ \text{Sinon :} \\ c_1(ca(v_1), ca(v_2)) \end{array} \right) + \left(\begin{array}{l} \text{Si } D : \\ (P(cb(v_1) \sim cb(v_2)) * C(Gain)) \\ \text{Sinon :} \\ c_0(cb(v_1), cb(v_2)) + C(Break) \end{array} \right) \\ \left(\begin{array}{l} \text{Si } C : \\ (P(ca(v_1) \sim ca(v_2)) * C(Gain)) \\ \text{Sinon :} \\ c_0(ca(v_1), ca(v_2)) + C(Break) \end{array} \right) + \left(\begin{array}{l} \text{Si } D : \\ (1 - P(cb(v_1) \sim cb(v_2)) * C(Break)) \\ \text{Sinon :} \\ c_1(cb(v_1), cb(v_2)) \end{array} \right) \\ \left(\begin{array}{l} \text{Si } C : \\ (P(ca(v_1) \sim ca(v_2)) * C(Gain)) \\ \text{Sinon :} \\ c_0(ca(v_1), ca(v_2)) + C(Break) \end{array} \right) + \left(\begin{array}{l} \text{Si } D : \\ (P(cb(v_1) \sim cb(v_2)) * C(Gain)) \\ \text{Sinon :} \\ c_0(cb(v_1), cb(v_2)) + C(Break) \end{array} \right) \end{array} \right.$$

$$c_0(v_1, v_2) = min \left\{ \begin{array}{l} \left(\begin{array}{l} \text{Si } C : \\ (P(ca(v_1) \sim ca(v_2)) * C(Gain)) \\ \text{Sinon :} \\ c_0(ca(v_1), ca(v_2)) \end{array} \right) + \left(\begin{array}{l} \text{Si } D : \\ (P(cb(v_1) \sim cb(v_2)) * C(Gain)) \\ \text{Sinon :} \\ c_0(cb(v_1), cb(v_2)) \end{array} \right) \\ \left(\begin{array}{l} \text{Si } C : \\ (1 - P(ca(v_1) \sim ca(v_2)) * C(Break)) \\ \text{Sinon :} \\ c_1(ca(v_1), ca(v_2)) + C(Gain) \end{array} \right) + \left(\begin{array}{l} \text{Si } D : \\ (P(cb(v_1) \sim cb(v_2)) * C(Gain)) \\ \text{Sinon :} \\ c_0(cb(v_1), cb(v_2)) \end{array} \right) \\ \left(\begin{array}{l} \text{Si } C : \\ (P(ca(v_1) \sim ca(v_2)) * C(Gain)) \\ \text{Sinon :} \\ c_0(ca(v_1), ca(v_2)) \end{array} \right) + \left(\begin{array}{l} \text{Si } D : \\ (1 - P(cb(v_1) \sim cb(v_2)) * C(Break)) \\ \text{Sinon :} \\ c_1(cb(v_1), cb(v_2)) + C(Gain) \end{array} \right) \\ \left(\begin{array}{l} \text{Si } C : \\ (1 - P(ca(v_1) \sim ca(v_2)) * C(Break)) \\ \text{Sinon :} \\ => c_1(ca(v_1), ca(v_2)) + C(Gain) \end{array} \right) + \left(\begin{array}{l} \text{Si } D : \\ (1 - P(cb(v_1) \sim cb(v_2)) * C(Break)) \\ \text{Sinon :} \\ c_1(cb(v_1), cb(v_2)) + C(Gain) \end{array} \right) \end{array} \right.$$

Cas 6. $E(v_1) = GDup$ et $E(v_2) = GDup$.
Dans ce cas, $c_1(v_1, v_2) = min\{D1, D2, D12\}$ où
$D1$ est le coût où la duplication de v_1 vient en premier,
$D2$ est le coût où la duplication de v_2 vient en premier,
$D12$ est le coût où les duplications de v_1 et v_2 sont simultanées.

→ Pour le calcul de $D1$ et $D2$, on effectue le même calcul que pour la version originale de DeCo(*cf.* annexe p. 44).
→ Pour le calcul de $c_0(v_1, v_2)$ du **Cas 6**, il faut effectuer le même calcul que pour la version originale de DeCo(*cf.* annexe p. 44).

Le calcul de la partie $D12$ étant particulièrement long (22 formules à calculer pour déterminer le coût minimum) et la logique étant la même que pour les cas précédents, nous allons ici expliquer le dédoublement des formules de manière plus précise sur le **Cas 6** qui est le cas le plus complexe. Pour cela, nous allons reprendre le calcul de la version originale de DeCo :

$$D12 = min \begin{cases} \left.\begin{array}{l} (1)\ c_1(ca(v_1), ca(v_2)) + c_1(cb(v_1), cb(v_2)) + \\ c_0(ca(v_1), cb(v_2)) + c_0(cb(v_1), ca(v_2)) \\ (2)\ c_1(ca(v_1), ca(v_2)) + c_1(cb(v_1), cb(v_2)) + \\ c_0(ca(v_1), cb(v_2)) + c_1(cb(v_1), ca(v_2)) + C(Gain) \\ (3)\ c_1(ca(v_1), ca(v_2)) + c_1(cb(v_1), cb(v_2)) + \\ c_1(ca(v_1), cb(v_2)) + c_0(cb(v_1), ca(v_2)) + C(Gain) \\ (4)\ c_1(ca(v_1), ca(v_2)) + c_0(cb(v_1), cb(v_2)) + \\ c_0(ca(v_1), cb(v_2)) + c_0(cb(v_1), ca(v_2)) + C(Break) \\ (5)\ c_0(ca(v_1), ca(v_2)) + c_1(cb(v_1), cb(v_2)) + \\ c_0(ca(v_1), cb(v_2)) + c_0(cb(v_1), ca(v_2)) + C(Break) \end{array}\right\} \text{Cas où les adjacences sont non croisées. (\textit{cf.} schéma de gauche de la Figure 5.2)} \\ \left.\begin{array}{l} (6)\ c_0(ca(v_1), ca(v_2)) + c_0(cb(v_1), cb(v_2)) + \\ c_1(ca(v_1), cb(v_2)) + c_1(cb(v_1), ca(v_2)) \\ (7)\ c_0(ca(v_1), ca(v_2)) + c_1(cb(v_1), cb(v_2)) + \\ c_1(ca(v_1), cb(v_2)) + c_1(cb(v_1), ca(v_2)) + C(Gain) \\ (8)\ c_1(ca(v_1), ca(v_2)) + c_0(cb(v_1), cb(v_2)) + \\ c_1(ca(v_1), cb(v_2)) + c_1(cb(v_1), ca(v_2)) + C(Gain) \\ (9)\ c_0(ca(v_1), ca(v_2)) + c_0(cb(v_1), cb(v_2)) + \\ c_0(ca(v_1), cb(v_2)) + c_1(cb(v_1), ca(v_2)) + C(Break) \\ (10)\ c_0(ca(v_1), ca(v_2)) + c_0(cb(v_1), cb(v_2)) + \\ c_1(ca(v_1), cb(v_2)) + c_0(cb(v_1), ca(v_2)) + C(Break) \end{array}\right\} \text{Cas où les adjacences sont croisées. (\textit{cf.} schéma de droite de la Figure 5.2)} \\ \left.\begin{array}{l} (11)\ c_0(ca(v_1), ca(v_2)) + c_1(cb(v_1), cb(v_2)) + \\ c_0(ca(v_1), cb(v_2)) + c_1(cb(v_1), ca(v_2)) + C(Gain) + C(Break) \\ (12)\ c_0(ca(v_1), ca(v_2)) + c_1(cb(v_1), cb(v_2)) + \\ c_1(ca(v_1), cb(v_2)) + c_0(cb(v_1), ca(v_2)) + C(Gain) + C(Break) \\ (13)\ c_1(ca(v_1), ca(v_2)) + c_0(cb(v_1), cb(v_2)) + \\ c_0(ca(v_1), cb(v_2)) + c_1(cb(v_1), ca(v_2)) + C(Gain) + C(Break) \\ (14)\ c_1(ca(v_1), ca(v_2)) + c_0(cb(v_1), cb(v_2)) + \\ c_1(ca(v_1), cb(v_2)) + c_0(cb(v_1), ca(v_2)) + C(Gain) + C(Break) \\ (15)\ c_1(ca(v_1), ca(v_2)) + c_1(cb(v_1), cb(v_2)) + \\ c_1(ca(v_1), cb(v_2)) + c_1(cb(v_1), ca(v_2)) + 2 \times C(Gain) \\ (16)\ c_0(ca(v_1), ca(v_2)) + c_0(cb(v_1), cb(v_2)) + \\ c_0(ca(v_1), cb(v_2)) + c_0(cb(v_1), ca(v_2)) + 2 \times C(Break) \end{array}\right\} \text{Doublement des formules.} \end{cases}$$

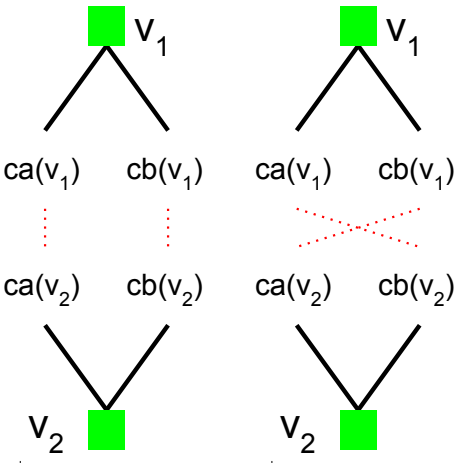


FIGURE 5.2 – Illustration es scénarios d'adjacences possibles provenant de l'évolution de l'adjacence de deux gènes v_1 et v_2 lors de la duplication simultanée des deux gènes. Le schéma de droite illustre le cas où $ca(v_1)$ est adjacent à $ca(v_2)$ et $cb(v_1)$ est adjacent à $cb(v_2)$, les adjacences représentées sous la forme de pointillés rouges. Le schéma de droite illustre le cas où $ca(v_1)$ est adjacent à $cb(v_2)$ et $cb(v_1)$ est adjacent à $ca(v_2)$, les adjacences représentées sous la forme de pointillés rouges.

Comme on peut le voir sur les formules de récurrence (*cf.* p. 39) et sur la Figure 5.2, les formules de $D12$ peuvent être classés en trois catégories :

- les formules (1) à (5) correspondent au cas où l'on suppose que les fils gauches sont adjacents entre eux, c.-à-d. $ca(v_1)$ adjacent à $ca(v_2)$) et les fils droits le sont également entre eux, c.-à-d. $cb(v_1)$ adjacent à $cb(v_2)$) ce qui revient à dire que les adjacences des fils de v_1 et v_2 sont non croisées. On a donc la formule : $c_1(ca(v_1), ca(v_2)) + c_1(cb(v_1), cb(v_2)) + c_0(ca(v_1), cb(v_2)) + c_0(cb(v_1), ca(v_2))$ étant le cas où l'on a ni gain, ni perte d'adjacences (*cf.* schéma de gauche de la Figure 5.2).
- les formules (6) à (10) correspondent au cas où l'on suppose que les adjacences sont croisées, c'est à dire que $ca(v_1)$ est adjacent à $cb(v_2)$ et $cb(v_1)$ est adjacent à $ca(v_2)$, ce qui revient à dire que les adjacences des fils de v_1 et v_2 sont croisées. On a donc la formule : $c_0(ca(v_1), ca(v_2)) + c_0(cb(v_1), cb(v_2)) + c_1(ca(v_1), cb(v_2)) + c_1(cb(v_1), ca(v_2))$ étant le cas où l'on a ni gain, ni perte d'adjacences (*cf.* schéma de droite de la Figure 5.2).
- les formules (11) à (16) correspondent au cas pour lesquels, les formules peuvent provenir soit du cas où les adjacences entre v_1 et v_2 sont croisées (*cf.* schéma de droite de la Figure 5.2), soit du cas où elles ne le sont pas (*cf.* schéma de gauche de la Figure 5.2).

Pour les formules (2) à (5), les coûts $C(Gain)$ et $C(Break)$ sont facilement interprétables car, par rapport à la formule (1), si un coût c_0 est transformé en c_1 on associe alors à ce coût c_1 un coût de $C(Gain)$ et dans le cas inverse, c_1 transformé en c_0, on associe au coût c_0 un coût $C(Break)$. Le même raisonnement est effectué pour les formules (7) à (10) par rapport à la formule (6).

Pour ce qui est des formules (11) à (16), comme dit précédemment les formules doivent être doublées car, elles peuvent tout aussi bien provenir du scénario illustré par le schéma de gauche, que du scénario illustré par le schéma de droite de la Figure 5.2. Prenons par exemple la formule (15), qui propose le scénario où $ca(v_1)$ est adjacent à $ca(v_2)$ et $cb(v_2)$, et $cb(v_1)$ est adjacent à $ca(v_2)$ et $cb(v_2)$. Deux cas sont alors possibles :

— soit l'on est dans le cas où les adjacences entre les fils de v_1 et v_2 ne sont pas croisées et on associe donc un coût $C(Gain)$ au coût $c_1(ca(v_1), cb(v_2))$ et un coût $C(Gain)$ au coût $c_1(cb(v_1), ca(v_2))$.

— soit l'on est dans le cas où les adjacences entre les fils de v_1 et v_2 sont croisées et on associe donc un coût $C(Gain)$ au coût $c_1(ca(v_1), ca(v_2))$ et un coût $C(Gain)$ au coût $c_1(cb(v_1), cb(v_2))$.

Ainsi, le calcul de $D12$ de la version initiale de DeCo passe de 16 formules à calculer à 22 formules à calculer.

Une fois l'élaboration de l'algorithme terminée, une dernière étape d'implémentation et test de l'algorithme est nécessaire pour valider le nouvel algorithme.

5.3 Test pour évaluer l'effet du nouvel algorithme DeCo sur la qualité d'assemblage des génomes et perspectives d'amélioration de l'algorithme

Cette partie n'a pas pu être effectuée au cours de ce stage, l'élaboration de cet algorithme et les différents problèmes rencontrés sur les jeux de données, ayant pris plus de temps que prévu. Cette étape nécessitera dans un premier temps l'implémentation de l'algorithme sous la forme d'un nouvel algorithme DeCo qui sera suivi de l'élaboration d'un protocole de tests pour déterminer l'effet de ce nouvel algorithme sur la reconstruction de possibles nouvelles adjacences ancestrales.
Le calcul des probabilités d'adjacences nécessitant l'utilisation de factorielles, il sera nécessaire d'optimiser ce calcul afin de pouvoir utiliser l'algorithme sur des génomes avec une très faible qualité d'assemblage.

Conclusion

Au cours de ce stage, nous avons dans un premier temps effectué une modularisation du code de la méthode `mDeCo` qui nous a permis de restructurer celui-ci afin de faciliter les modifications apportées à DeCo au cours de ce stage et dans le futur. Cette analyse a permis de mettre à jour des anomalies dans nos jeux de données et d'effectuer une recherche sur les banques de données internationales sur les données génomiques pour l'instauration d'un module (`Step0_dataset`) permettant de générer les fichiers d'entrée de la méthode `mDeCo`. Au cours de cette recherche nous avons observé que pour un certain nombre d'espèces les génomes ont une faible qualité d'assemblage, cela nous a permis de réfléchir sur l'utilisation de méthodes de reconstruction d'histoire évolutive pour améliorer la qualité des méthodes d'assemblage des génomes. Nous avons élaboré une adaptation de l'algorithme DeCo à cette problématique d'assemblage des génomes, pour cela nous avons mis au point un algorithme permettant d'associer des gènes et de reconstruire des adjacences ancestrales de ces gènes à partir des probabilités d'adjacences entre des gènes dont nous avions un manque d'informations sur les adjacences. Cependant, nous n'avons pas eu le temps de mettre au point un protocole de test afin de tester l'effet de l'adaptation de l'algorithme DeCo sur l'inférence de génomes ancestraux, qui permettrait de retrouver les probabilités d'adjacences ayant permis la reconstruction de ces adjacences ancestrales et potentiellement déterminer des adjacences de gènes candidates pour l'assemblage de contigs.
Lors de ces six mois en stage de master 2 recherche, j'ai acquis de l'expérience dans la réflexion et l'élaboration d'algorithme appliqué à la génomique et phylogénomique qui nécessite la formalisation du problème que l'on veut résoudre sous la forme d'un problème mathématique. Les recherches de données génomiques m'ont également permis de connaître l'état actuel des connaissances des génomes de l'ensemble des espèces qui semble faible au vu des technologies de séquençage de haut-débit actuelles mais qui s'explique par les difficultés d'assemblages des génomes. Le code de la méthode `mDeCo` étant écrit dans le langage C++, j'ai amélioré mes compétences dans ce langage de programmation qui est idéal pour gérer les grands volumes de données bien que la recherche par expression régulière m'ait paru laborieuse.
Pour les perspectives de la méthode plusieurs pistes sont possibles et feront l'œuvre d'un projet de thèse. Un premier objectif sera d'étudier l'effet de l'algorithme développé au cours de ce stage sur l'inférence de génomes anciens et sur l'assemblage des contigs de génomes actuels prédit par l'inférence de nouvelles adjacences actuelles déduite de l'inférence des adjacences ancestrales. Des problématiques évoquées dans de précédents rapports de l'algorithme DeCo doivent également être résolues, telles que la contrainte de linéarité des génomes qui implique qu'un gène ne peut être adjacent à plus de deux gènes et que l'algorithme DeCo ne prend pas en compte (*cf.* figure 4.4) et la mise en place de deux nouveaux événements évolutifs, la duplication et perte de gènes en bloc qui permet d'associer plusieurs duplications/pertes sous la forme d'une seule duplication/perte en bloc, qui a fait l'œuvre d'un stage de master 2 recherche [19] et doit être intégré dans la nouvelle version de l'algorithme DeCo. D'autres possibilités d'extensions de la méthode `mDeCo` sont envisagés et permettraient de reconstruire l'histoire évolutive de systèmes plus complexe tels que les interactions géniques et permettraient ainsi de comprendre l'évolution de mécanismes perdus au cours de l'évolution telle que la capacité de régénérescence tissulaire, omniprésente chez la salamandre et dont il ne reste que des vestiges chez l'humain (cicatrisation

de la peau et régénération du foie). Ces perspectives laissent à penser que l'algorithme DeCo à encore de beaux jours devant lui et permettra sûrement la compréhension de l'histoire évolutive de mécanismes complexes.

Annexes

Formules de récurrence de la version originale de DeCo. On note que $ca(v)$ et $cb(v)$ sont respectivement les fils gauche et droit du gène v.

Cas 1. $E(v_1) = Extant$ et $E(v_2) = Extant$.
Si $v_1 \sim v_2$ alors $c_1(v_1, v_2) = 0$ et $c_0(v_1, v_2) = \infty$;
Sinon $c_1(v_1, v_2) = \infty$ et $c_0(v_1, v_2) = 0$.

Cas 2. $E(v_1) = GLos$ et $E(v_2) \neq GLos$.
Dans ce cas $c_1(v_1, v_2) = 0$ et $c_0(v_1, v_2) = 0$.

Cas 3. $E(v_1) = Extant$ et $E(v_2) = GLos$
Dans ce cas $c_1(v_1, v_2) = 0$ et $c_0(v_1, v_2) = 0$.
Ce cas diverge du cas précédent par la procédure de *backtracking*.

Cas 4. $E(v_1) \in \{Extant, Spec\}$ et $E(v_2) = GDup$.

$$c_1(v_1, v_2) = min \begin{cases} c_1(v_1, ca(v_2)) + c_0(v_1, cb(v_2)) \\ c_0(v_1, ca(v_2)) + c_1(v_1, cb(v_2)) \\ c_1(v_1, ca(v_2)) + c_1(v_1, cb(v_2)) + C(Gain) \\ c_0(v_1, ca(v_2)) + c_0(v_1, cb(v_2)) + C(Break) \end{cases}$$

$$c_0(v_1, v_2) = min \begin{cases} c_0(v_1, ca(v_2)) + c_0(v_1, cb(v_2)) \\ c_0(v_1, ca(v_2)) + c_1(v_1, cb(v_2)) + C(Gain) \\ c_1(v_1, ca(v_2)) + c_0(v_1, cb(v_2)) + C(Gain) \\ c_1(v_1, ca(v_2)) + c_1(v_1, cb(v_2)) + 2 \times C(Gain) \end{cases}$$

Cas 5. $E(v_1) = Spec$ et $E(v_2) = Spec$.
On suppose que $S(ca(v_1)) = S(ca(v_2))$ et $S(cb(v_1)) = S(cb(v_2))$.

$$c_1(v_1, v_2) = min \begin{cases} c_1(ca(v_1), ca(v_2)) + c_1(cb(v_1), cb(v_2)) \\ c_1(ca(v_1), ca(v_2)) + c_0(cb(v_1), cb(v_2)) + C(Break) \\ c_0(ca(v_1), ca(v_2)) + c_1(cb(v_1), cb(v_2)) + C(Break) \\ c_0(ca(v_1), ca(v_2)) + c_0(cb(v_1), cb(v_2)) + 2 \times C(Break) \end{cases}$$

$$c_0(v_1, v_2) = min \begin{cases} c_0(ca(v_1), ca(v_2)) + c_0(cb(v_1), cb(v_2)) \\ c_1(ca(v_1), ca(v_2)) + c_0(cb(v_1), cb(v_2)) + C(Gain) \\ c_0(ca(v_1), ca(v_2)) + c_1(cb(v_1), cb(v_2)) + C(Gain) \\ c_1(ca(v_1), ca(v_2)) + c_1(cb(v_1), cb(v_2)) + 2 \times C(Gain) \end{cases}$$

Cas 6. $E(v_1) = GDup$ et $E(v_2) = GDup$.
Dans ce cas, $c_1(v_1, v_2) = min\{D1, D2, D12\}$ où
$D1$ est le coût où la duplication de v_1 vient en premier,
$D2$ est le coût où la duplication de v_2 vient en premier,
$D12$ est le coût où les duplications de v_1 et v_2 sont simultanées.

$$D1 = min \begin{cases} c_1(ca(v_1), v_2) + c_0(cb(v_1), v_2) \\ c_0(ca(v_1), v_2) + c_1(cb(v_1), v_2) \\ c_1(ca(v_1), v_2) + c_1(cb(v_1), v_2) + C(Gain) \\ c_0(ca(v_1), v_2) + c_0(cb(v_1), v_2) + C(Break) \end{cases}$$

$$D2 = min \begin{cases} c_1(v_1, ca(v_2)) + c_0(v_1, cb(v_2)) \\ c_0(v_1, ca(v_2)) + c_1(v_1, cb(v_2)) \\ c_1(v_1, ca(v_2)) + c_1(v_1, cb(v_2)) + C(Gain) \\ c_0(v_1, ca(v_2)) + c_0(v_1, cb(v_2)) + C(Break) \end{cases}$$

$$D12 = min \begin{cases} (1)\ c_1(ca(v_1), ca(v_2)) + c_1(cb(v_1), cb(v_2)) + \\ c_0(ca(v_1), cb(v_2)) + c_0(cb(v_1), ca(v_2)) \\ (2)\ c_1(ca(v_1), ca(v_2)) + c_1(cb(v_1), cb(v_2)) + \\ c_0(ca(v_1), cb(v_2)) + c_1(cb(v_1), ca(v_2)) + C(Gain) \\ (3)\ c_1(ca(v_1), ca(v_2)) + c_1(cb(v_1), cb(v_2)) + \\ c_1(ca(v_1), cb(v_2)) + c_0(cb(v_1), ca(v_2)) + C(Gain) \\ (4)\ c_1(ca(v_1), ca(v_2)) + c_0(cb(v_1), cb(v_2)) + \\ c_0(ca(v_1), cb(v_2)) + c_0(cb(v_1), ca(v_2)) + C(Break) \\ (5)\ c_0(ca(v_1), ca(v_2)) + c_1(cb(v_1), cb(v_2)) + \\ c_0(ca(v_1), cb(v_2)) + c_0(cb(v_1), ca(v_2)) + C(Break) \\ (6)\ c_0(ca(v_1), ca(v_2)) + c_0(cb(v_1), cb(v_2)) + \\ c_1(ca(v_1), cb(v_2)) + c_1(cb(v_1), ca(v_2)) \\ (7)\ c_0(ca(v_1), ca(v_2)) + c_1(cb(v_1), cb(v_2)) + \\ c_1(ca(v_1), cb(v_2)) + c_1(cb(v_1), ca(v_2)) + C(Gain) \\ (8)\ c_1(ca(v_1), ca(v_2)) + c_0(cb(v_1), cb(v_2)) + \\ c_1(ca(v_1), cb(v_2)) + c_1(cb(v_1), ca(v_2)) + C(Gain) \\ (9)\ c_0(ca(v_1), ca(v_2)) + c_0(cb(v_1), cb(v_2)) + \\ c_0(ca(v_1), cb(v_2)) + c_1(cb(v_1), ca(v_2)) + C(Break) \\ (10)\ c_0(ca(v_1), ca(v_2)) + c_0(cb(v_1), cb(v_2)) + \\ c_1(ca(v_1), cb(v_2)) + c_0(cb(v_1), ca(v_2)) + C(Break) \\ (11)\ c_0(ca(v_1), ca(v_2)) + c_1(cb(v_1), cb(v_2)) + \\ c_0(ca(v_1), cb(v_2)) + c_1(cb(v_1), ca(v_2)) + C(Gain) + C(Break) \\ (12)\ c_0(ca(v_1), ca(v_2)) + c_1(cb(v_1), cb(v_2)) + \\ c_1(ca(v_1), cb(v_2)) + c_0(cb(v_1), ca(v_2)) + C(Gain) + C(Break) \\ (13)\ c_1(ca(v_1), ca(v_2)) + c_0(cb(v_1), cb(v_2)) + \\ c_0(ca(v_1), cb(v_2)) + c_1(cb(v_1), ca(v_2)) + C(Gain) + C(Break) \\ (14)\ c_1(ca(v_1), ca(v_2)) + c_0(cb(v_1), cb(v_2)) + \\ c_1(ca(v_1), cb(v_2)) + c_0(cb(v_1), ca(v_2)) + C(Gain) + C(Break) \\ (15)\ c_1(ca(v_1), ca(v_2)) + c_1(cb(v_1), cb(v_2)) + \\ c_1(ca(v_1), cb(v_2)) + c_1(cb(v_1), ca(v_2)) + 2 \times C(Gain) \\ (16)\ c_0(ca(v_1), ca(v_2)) + c_0(cb(v_1), cb(v_2)) + \\ c_0(ca(v_1), cb(v_2)) + c_0(cb(v_1), ca(v_2)) + 2 \times C(Break) \end{cases}$$

$$c_0(v_1, v_2) = min \begin{cases} \text{Cas où la duplcation de } v_1 \text{ vient en premier} \\ c_0(ca(v_1), v_2) + c_0(cb(v_1), v_2) \\ c_0(ca(v_1), v_2) + c_1(cb(v_1), v_2) + C(Gain) \\ c_1(ca(v_1), v_2) + c_0(cb(v_1), v_2) + C(Gain) \\ c_1(ca(v_1), v_2) + c_1(cb(v_1), v_2) + 2 \times C(Gain) \\ \text{Cas où la duplcation de } v_2 \text{ vient en premier} \\ c_0(v_1, ca(v_2)) + c_0(v_1, cb(v_2)) \\ c_0(v_1, ca(v_2)) + c_1(v_1, cb(v_2)) + C(Gain) \\ c_1(v_1, ca(v_2)) + c_0(v_1, cb(v_2)) + C(Gain) \\ c_1(v_1, ca(v_2)) + c_1(v_1, cb(v_2)) + 2 \times C(Gain) \end{cases}$$

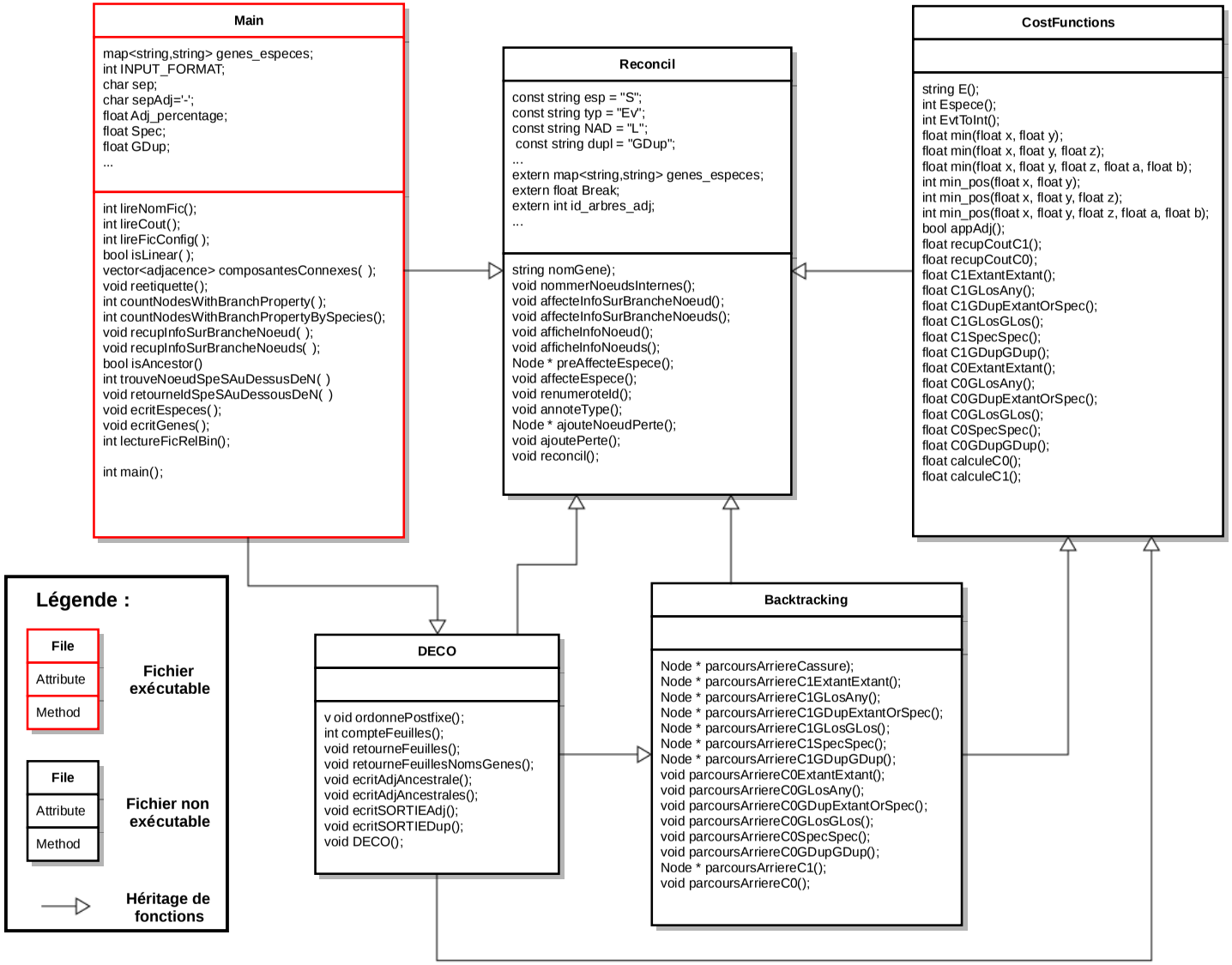


FIGURE 3 – Diagramme de dépendances fonctionnelles de DeCo avant modularisation du code

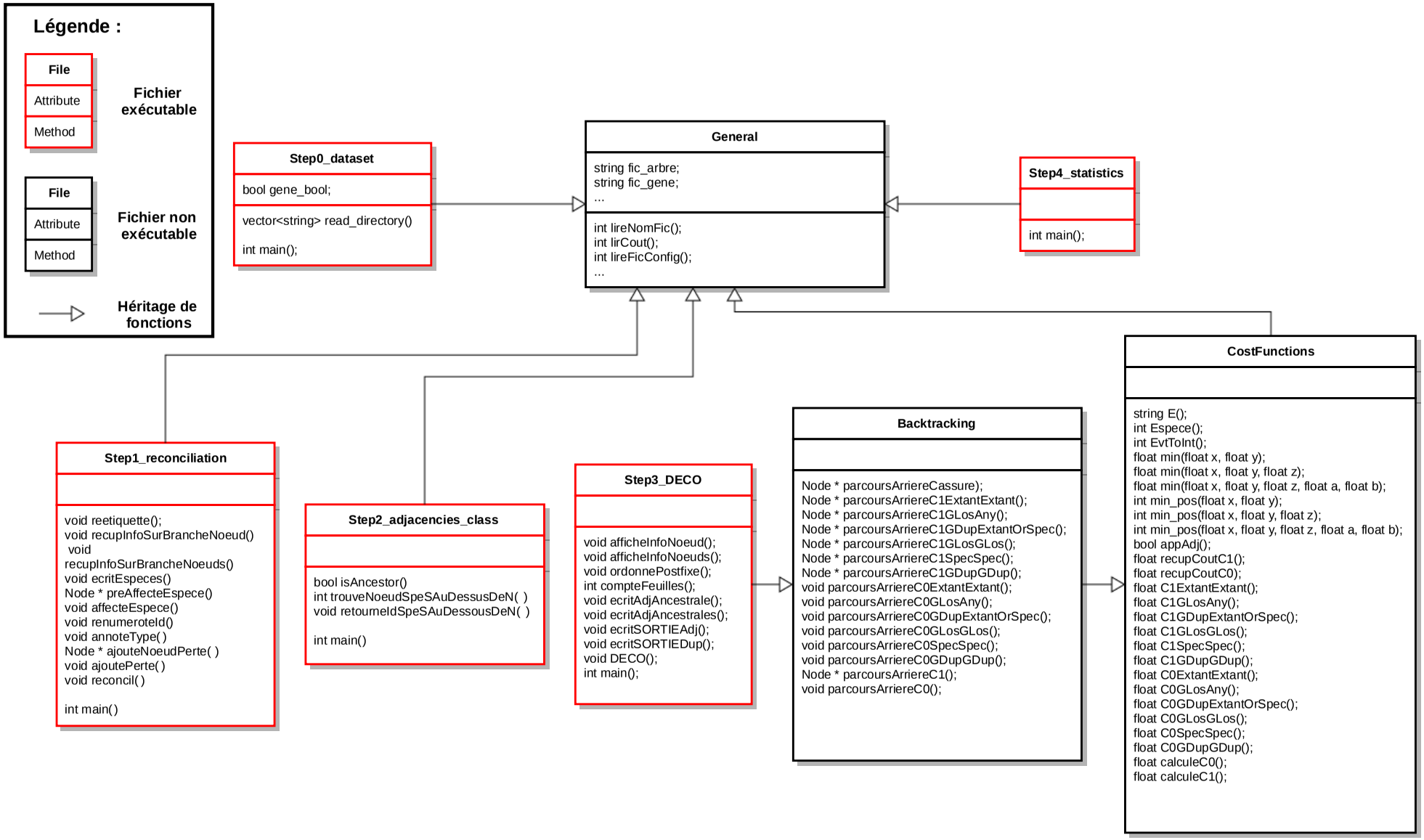


FIGURE 4 – Diagramme de dépendances fonctionnelles de DeCo après modularisation du code

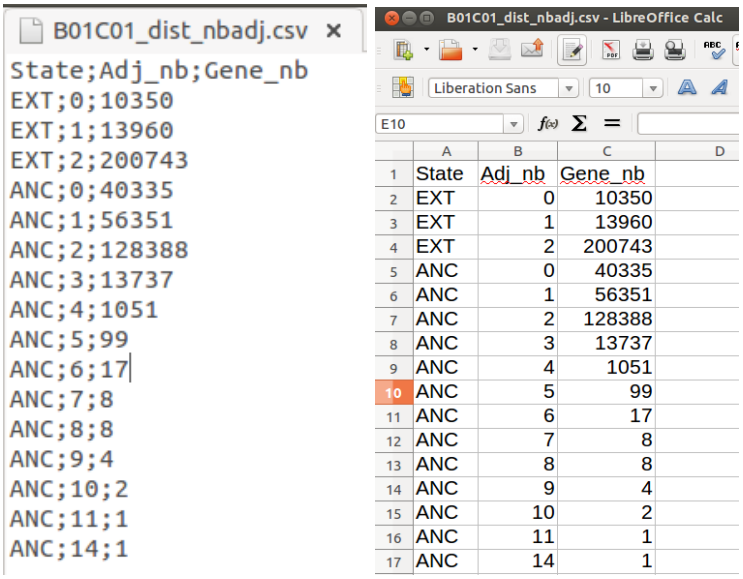


FIGURE 5 – Exemple d'un fichier au format CSV. Sur la gauche, un fichier CSV en texte brut et à droite la visualisation du même fichier dans un tableur. Le fichier CSV est un fichier tableur où les données des colonnes sont délimitées par un caractère de séparation (ici, le point-virgule) et les lignes sont délimitées par un saut de ligne.

```
LOCUS       SCU49845     5028 bp    DNA             PLN       21-JUN-1999
DEFINITION  Saccharomyces cerevisiae TCP1-beta gene, partial cds, and Axl2p
            (AXL2) and Rev7p (REV7) genes, complete cds.
ACCESSION   U49845
VERSION     U49845.1  GI:1293613
KEYWORDS    .
SOURCE      Saccharomyces cerevisiae (baker's yeast)
  ORGANISM  Saccharomyces cerevisiae
            Eukaryota; Fungi; Ascomycota; Saccharomycotina; Saccharomycetes;
            Saccharomycetales; Saccharomycetaceae; Saccharomyces.
REFERENCE   1  (bases 1 to 5028)
  AUTHORS   Torpey,L.E., Gibbs,P.E., Nelson,J. and Lawrence,C.W.
  TITLE     Cloning and sequence of REV7, a gene whose function is required for
            DNA damage-induced mutagenesis in Saccharomyces cerevisiae
  JOURNAL   Yeast 10 (11), 1503-1509 (1994)
  PUBMED    7871890
REFERENCE   2  (bases 1 to 5028)
  AUTHORS   Roemer,T., Madden,K., Chang,J. and Snyder,M.
  TITLE     Selection of axial growth sites in yeast requires Axl2p, a novel
            plasma membrane glycoprotein
  JOURNAL   Genes Dev. 10 (7), 777-793 (1996)
  PUBMED    8846915
REFERENCE   3  (bases 1 to 5028)
  AUTHORS   Roemer,T.
  TITLE     Direct Submission
  JOURNAL   Submitted (22-FEB-1996) Terry Roemer, Biology, Yale University, New
            Haven, CT, USA
```

FIGURE 6 – Entête d'un fichier au format GenBank où l'identifiant du locus se situe sur la ligne commençant par le *tag* LOCUS. Sur cet exemple l'identifiant du locus est SCU49845. L'entête d'un fichier au format GenBank est ensuite suivis de différentes informations telles le nom de l'espèce (Saccharomyces cerevisiae), le nombre de paires de bases (5028), les articles dans lesquelles ses données ont été prélevées, ...

```
FEATURES             Location/Qualifiers
     source          1..5028
                     /organism="Saccharomyces cerevisiae"
                     /db_xref="taxon:4932"
                     /chromosome="IX"
                     /map="9"
     CDS             <1..206
                     /codon_start=3
                     /product="TCP1-beta"
                     /protein_id="AAA98665.1"
                     /db_xref="GI:1293614"
                     /translation="SSIYNGISTSGLDLNNGTIADMRQLGIVESYKLKRAVVSSASEA
                     AEVLLRVDNIIRARPRTANRQHM"
     gene            687..3158
                     /gene="AXL2"
     CDS             687..3158
                     /gene="AXL2"
                     /note="plasma membrane glycoprotein"
                     /codon_start=1
                     /function="required for axial budding pattern of S.
                     cerevisiae"
                     /product="Axl2p"
                     /protein_id="AAA98666.1"
                     /db_xref="GI:1293615"
                     /translation="MTQLQISLLLTATISLLHLVVATPYEAYPIGKQYPPVARVNESF
                     TFQISNDTYKSSVDKTAQITYNCFDLPSWLSFDSSSRTFSGEPSSDLLSDANTTLYFN
                     VILEGTDSADSTSLNNTYQFVVTNRPSISLSSDFNLLALLKNYGYTNGKNALKLDPNE
                     VFNVTFDRSMFTNEESIVSYYGRSQLYNAPLPNWLFFDSGELKFTGTAPVINSAIAPE
                     TSYSFVIIATDIEGFSAVEVEFELVIGAHQLTTSIQNSLIINVTDTGNVSYDLPLNYV
                     YLDDDPISSDKLGSINLLDAPDWVALDNATISGSVPDELLGKNSNPANFSVSIYDTYG
                     DVIYFNFEVVSTTDLFAISSLPNINATRGEWFSYYFLPSQFTDYVNTNVSLEFTNSSQ
                     DHDWVKFQSSNLTLAGEVPKNFDKLSLGLKANQGSQSQELYFNIIGMDSKITHSNHSA
                     NATSTRSSHHSTSTSSYTSSTYTAKISSTSAAATSSAPAALPAANKTSSHNKKAVAIA
                     CGVAIPLGVILVALICFLIFWRRRRENPDDENLPHAISGPDLNNPANKPNQENATPLN
```

FIGURE 7 – Extrait d'un fichier au format GenBank montrant où l'on trouve les informations sur les gènes qui sont précédés par un *tag* FEATURES. Une protéine est identifiée par un *tag* CDS dans lequel on retrouve les positions de début et de fin de protéine (<1..206) ainsi que l'identifiant protéique (/protein_id="AAA98665.1"). Un gène est identifié par un *tag* gene dans lequel on trouve également les positions de début et de fin (687..3158) ainsi que l'identifiant du gène (/gene="AXL2").

```
ID   15    standard; DNA; HTG; 101991189 BP.
XX
AC   chromosome:GRCh38:15:1:101991189:1
XX
SV   15.GRCh38
XX
DT   21-JUL-2014
XX
DE   Homo sapiens chromosome 15 GRCh38 full sequence 1..101991189 annotated by
DE   Ensembl
XX
KW   .
XX
OS   Homo sapiens (human)
OC   Eukaryota; Opisthokonta; Metazoa; Eumetazoa; Bilateria; Deuterostomia;
OC   Chordata; Craniata; Vertebrata; Gnathostomata; Teleostomi; Euteleostomi;
OC   Sarcopterygii; Dipnotetrapodomorpha; Tetrapoda; Amniota; Mammalia; Theria;
OC   Eutheria; Boreoeutheria; Euarchontoglires; Primates; Haplorrhini;
OC   Simiiformes; Catarrhini; Hominoidea; Hominidae; Homininae.
XX
CC   This sequence was annotated by the Ensembl system. Please visit the Ensembl
CC   web site, http://www.ensembl.org/ for more information.
XX
CC   All feature locations are relative to the first (5') base of the sequence
CC   in this file.  The sequence presented is always the forward strand of the
CC   assembly. Features that lie outside of the sequence contained in this file
CC   have clonal location coordinates in the format: <clone
CC   accession>.<version>:<start>..<end>
XX
CC   The /gene indicates a unique id for a gene, /note="transcript_id=..." a
CC   unique id for a transcript, /protein_id a unique id for a peptide and
CC   note="exon_id=..." a unique id for an exon. These ids are maintained
CC   wherever possible between versions.
XX
CC   All the exons and transcripts in Ensembl are confirmed by similarity to
CC   either protein or cDNA sequences.
```

FIGURE 8 – Entête d'un fichier au format EMBL où l'identifiant du chromosome/contig se situe sur la ligne commençant par le *tag* ID. Sur cet exemple l'identifiant du chromosome est 1. D'autres informations sont renseignées dans l'entête comme le nom de l'espèce (Homo sapiens (human)) précédé par un *tag* OS, l'embranchement phylogénétique détaillé précédé par le *tag* OC, ...

```
FT   gene            complement(20011153..20011169)
FT                   /gene=ENSG00000271336
FT                   /locus_tag="IGHD1OR15-1A"
FT                   /note="immunoglobulin heavy diversity 1/OR15-1A (non-
FT                   functional) [Source:HGNC Symbol;Acc:HGNC:5487]"
FT   mRNA            complement(20011153..20011169)
FT                   /gene="ENSG00000271336"
FT                   /note="transcript_id=ENST00000605284"
FT   CDS             complement(20011153..20011169)
FT                   /gene="ENSG00000271336"
FT                   /protein_id="ENSP00000473787"
FT                   /note="transcript_id=ENST00000605284"
FT                   /db_xref="Vega_transcript:IGHD1OR15-1A-001"
FT                   /db_xref="Vega_transcript:OTTHUMT00000468908"
FT                   /db_xref="HGNC_trans_name:IGHD1OR15-1A-001"
FT                   /db_xref="Vega_translation:194111"
FT                   /db_xref="Vega_translation:OTTHUMP00000272787"
FT                   /db_xref="OTTP:OTTHUMP00000272787"
FT                   /db_xref="OTTT:OTTHUMT00000468908"
FT                   /db_xref="UniParc:UPI000011E7DE"
FT                   /translation="GITGTT"
FT   gene            20012741..20014208
FT                   /gene=ENSG00000277988
FT                   /locus_tag="RP11-1360M22.11"
FT   misc_RNA        join(20012741..20012896,20013768..20013909,
FT                   20014110..20014208)
FT                   /gene="ENSG00000277988"
FT                   /db_xref="Vega_transcript:OTTHUMT00000475616"
FT                   /db_xref="Vega_transcript:OTTHUMT00000475616"
FT                   /db_xref="Clone_based_vega_transcript:RP11-1360M22.11-001"
FT                   /db_xref="OTTT:OTTHUMT00000475616"
FT                   /note="unprocessed_pseudogene"
FT                   /note="transcript_id=ENST00000611556"
```

FIGURE 9 – Extrait d'un fichier au format EMBL, on remarque que toutes les lignes sont précédées d'un *tag* FT indiquant que l'on est situé dans les annotations génomiques du fichier EMBL renfermant toutes les informations sur les gènes. Un gène est identifié par un *tag* gene dans lequel on trouve les positions de début et de fin (complement(20011153..20011169))) ainsi que l'identifiant du gène (/gene=ENSG00000271336). Un ARN est identifié par un *tag* mRNA dans le cas d'un ARNm ou *tag* misc_RNA dans le cas d'un ARNnc dans lequel on trouve les positions de début et de fin (complenent(20011153..20011169)) ainsi que l'identifiant de l'ARN (/note="transcript_id=ENST00000605284"). Une protéine est identifiée par un *tag* CDS dans lequel on trouve les positions de début et de fin (complement(20011153..20011169))) ainsi que l'identifiant du gène (/protein_id="ENSP00000473787").

```
[&&NHX:D=N:G=ENSBTAT00000060307:T=9913])Eutheria:0.0398804[&&NHX:D=N:B=0:T=9347])Eutheria:0.0170659
[&&NHX:D=Y:B=0:T=9347],ENSTTRG00000024777:0.0841474[&&NHX:D=N:G=ENSTTRT00000024822:T=9739])Eutheria:0
[&&NHX:D=N:B=0:T=9347],ENSSSCG00000030477:0.0449181[&&NHX:D=N:G=ENSSSCT00000032335:T=9823])Eutheria:0
[&&NHX:D=Y:B=0:T=9347],ENSVPAG00000016531:0.0611196
[&&NHX:D=N:G=ENSVPAT00000016762:T=30538])Eutheria:0.0167502[&&NHX:D=N:B=0:T=9347])Eutheria:0.00233565
[&&NHX:D=N:B=0:T=9347])Eutheria:0.000353949[&&NHX:D=N:B=0:T=9347],((ENSLAFG00000027757:0.0247455
[&&NHX:D=N:G=ENSLAFT00000036672:T=9785],ENSPCAG00000019967:0.0436305
[&&NHX:D=N:G=ENSPCAT00000019620:T=9813])Eutheria:0.017086
[&&NHX:D=N:B=65:T=9347],ENSDNOG00000044634:0.0704651
[&&NHX:D=N:G=ENSDNOT00000051562:T=9361])Eutheria:0.0215157[&&NHX:D=N:B=8:T=9347])Eutheria:0
[&&NHX:D=N:B=0:T=9347],ENSOGAG00000024914:0.0542376[&&NHX:D=N:G=ENSOGAT00000029694:T=30611])Eutheria:0
[&&NHX:D=N:B=0:T=9347],(ENSOCUG00000019359:0.0727906
[&&NHX:D=N:G=ENSOCUT00000019359:T=9986],ENSCPOG00000022741:0.089811
[&&NHX:D=N:G=ENSCPOT00000023513:T=10141])Eutheria:0.00174715[&&NHX:D=N:B=0:T=9347])Eutheria:0.0110257
[&&NHX:D=N:B=0:T=9347],ENSSARG00000016851:0.0809184[&&NHX:D=N:G=ENSSART00000016856:T=42254])Eutheria:0
[&&NHX:D=N:B=0:T=9347],ENSETEG00000022661:0.10126
[&&NHX:D=N:G=ENSETET00000022661:T=9371])Eutheria:0.0156762
[&&NHX:D=N:B=0:T=9347],ENSEEUG00000016498:0.129423[&&NHX:D=N:G=ENSEEUT00000016502:T=9365])Eutheria:0
[&&NHX:D=N:B=0:T=9347],((((((ENSCJAG00000027160:0.135091
[&&NHX:D=N:G=ENSCJAT00000048145:T=9483],ENSMUSG00000065706:0.176385
[&&NHX:D=N:G=ENSMUST00000083772:T=10090])Eutheria:0.000744174
[&&NHX:D=N:B=0:T=9347],ENSECAG00000025643:0.0575994[&&NHX:D=N:G=ENSECAT00000027655:T=9796])Eutheria:0
[&&NHX:D=N:B=0:T=9347],ENSCHOG00000015311:0.153579[&&NHX:D=N:G=ENSCHOT00000015264:T=9358])Eutheria:0
[&&NHX:D=N:B=0:T=9347],ENSTBEG00000018242:0.129581
[&&NHX:D=N:G=ENSTBET00000018246:T=37347])Eutheria:0.0103551[&&NHX:D=N:B=0:T=9347],
(((((ENSBTAG00000043122:0.0285174[&&NHX:D=N:G=ENSBTAT00000060114:T=9913],ENSOARG00000022834:0.0293339
[&&NHX:D=N:G=ENSOART00000024736:T=9940])Eutheria:0.0880837
[&&NHX:D=N:B=100:T=9347],ENSCJAG00000027262:0.12984
[&&NHX:D=N:G=ENSCJAT00000048247:T=9483])Eutheria:0.00880711
[&&NHX:D=N:B=0:T=9347],ENSBTAG00000042499:0.090642[&&NHX:D=N:G=ENSBTAT00000059491:T=9913])Eutheria:0
[&&NHX:D=Y:B=0:T=9347],((ENSOARG00000024015:0.0358863
[&&NHX:D=N:G=ENSOART00000025917:T=9940],ENSBTAG00000042141:0.0361137
[&&NHX:D=N:G=ENSBTAT00000059133:T=9913])Eutheria:0.0308274
[&&NHX:D=N:B=89:T=9347],ENSTTRG00000021683:0.0411726
[&&NHX:D=N:G=ENSTTRT00000023647:T=9739])Eutheria:0.0129801[&&NHX:D=N:B=66:T=9347])Eutheria:0
```

FIGURE 10 – Extrait d'un fichier d'arbres de gènes au format NHX.

```
(((((((Pan_troglodytes:0.47416183,Homo_sapiens:0.4741824):0.37455067,
Pongo_abelii:0.48138735):0.37655756,Macaca_mulatta:0.49315298):1.00273293,
(Mus_musculus:0.5214741,Rattus_norvegicus:0.5256683):1.1254762):0.19933681,
(Equus_caballus:0.90970256,(Bos_taurus:0.94350074,Sus_scrofa:0.5348985):
0.39797628):0.39185649):0.56282494,Monodelphis_domestica:0.8928249):0.24032629,
Ornithorhynchus_anatinus:0.72105026);
```

FIGURE 11 – Fichier d'arbres d'espèces au format Newick.

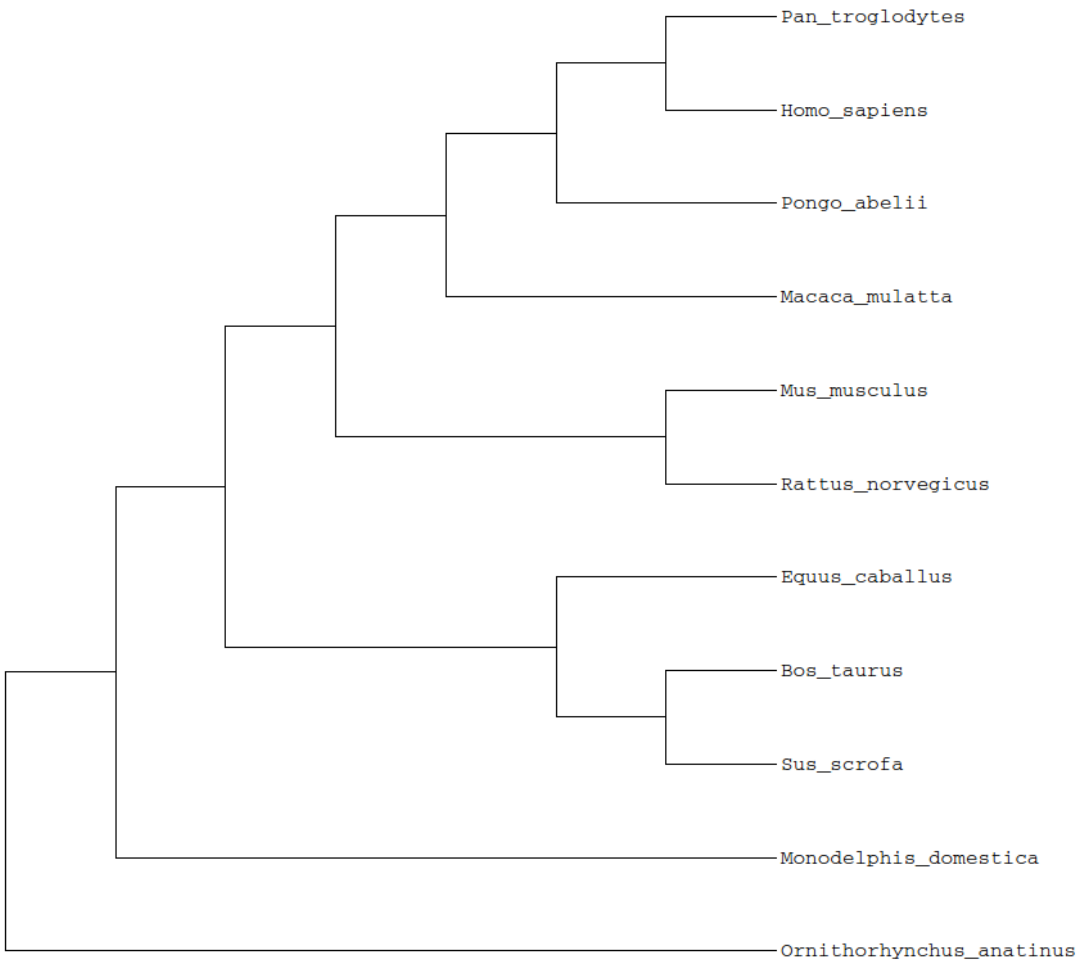


FIGURE 12 – Représentation graphique d'un arbre de 11 espèces correspondant à l'arbre au format Newick de la figure 11.

Acronymes

DeCo Detection of Coevolution
STIC Sciences pour les Technologies de l'Information et de la Communication
BCD Bioinformatique, Connaissances et Données
CNRS Centre National de la Recherche Scientifique
IRD Institut de Recherche pour le Développement
ISE-M Institut des Sciences de l'Évolution de Montpellier
UM2 Université de Montpellier 2
ANR Agence Nationale de Recherche
DYOGEN DYnamique et Organisation des GÉNomes
ENS École Normale Supérieure
LBBE Laboratoire de Biométrique et de Biologie Évolutive
BMGE Biologie Moléculaire du Gène chez les Extrémophiles
ADN Acide DésoxyriboNucléique
ARN Acide RiboNucléique
ARNnc ARN non codant
LCA Lowest Common Ancestor
DeCoLT Detection of Coevolution with Lateral Transfers
FTP File Transfer Protocol
EMBL European Molecular Biology Laboratory
EMF Ensembl Multi Format
NHX New Hampshire X
NCBI National Center for Biotechnology Information
JGI Joint Genome Institute
EBI European Bioinformatics Institut
WTSI Wellcome Trust Sanger Institute

Bibliographie

[1] Dobzhansky Th., Sturtevant A.H. *Inversions in the chromosomes of Drosophila pseudoobscura*. Genetics, 1937, Vol. 23, Issue 1, p. 28-64.

[2] Saitou N., Nei M. *The Neighbor-joining method : a new method for reconstructing phylogenetic trees*. Mol. Biol. Evol., 1987, Vol .4, Issue 4, p. 406-425.

[3] Fitch W.M. *Toward defining the course of evolution : minimum change for a specific tree topology*. Syst. Zoo., 1971, Vol. 20, Issue 4, p. 406-416.

[4] Sankoff D. *Minimal mutation trees of sequences*. SIAM J. App. Math., 1975, Vol. 28, Issue 1, p. 35-42.

[5] Avery O.T., McLeod C.M., McCarty M. *Studies on the chemical nature of the substance inducing transformation of pneumococcal types*. J. Exp. Med., 1944, Vol. 79, Issue 2, p. 137-158.

[6] Goodman M., Czelusniak J., Moore G.W., *et al. Fitting the gene lineage into its species lineage, a parsimony strategy illustrated by cladograms constructed from globin sequences*. Sys. Zoo., 1979, Vol. 28, Issue 2, p. 132-163.

[7] Bérard S., Gallien C., Boussau B., *et al. Evolution of gene neighborhoods within reconciled phylogenies*. Bioinformatics, 2012, Vol. 28, Issue 18, p. i382-i388.

[8] Sankoff D., Rousseau P. *Locating the vertices of a Steiner tree in an arbitrary metric space*. Math. Prog., 1975, Vol. 9, Issue , p. 240-246.

[9] Patterson M., Szöllősi G.J., Daubin V., *et al. Lateral Gene Transfer, Rearrangement and Reconciliation*. BMC Bioinformatics, 2013, Vol. 14.

[10] Maddison W.P. *Gene trees in species trees*. Syst. Bio., 1997, Vol. 46, Issue 3, p. 523-536.

[11] Ma B., Li M., Zhang L. *From gene trees to species trees*. SIAM J. Comput., 2000, Vol. 30, Issue 3, p. 729-752.

[12] Dutheil J., Gaillard S., Glémin S., *et al. Bio++ : a set of C++ libraries for sequence analysis, phylogenetics, molecular evolution and population genetics*. BMC Bioinformatics, 2006, Vol. 7, N°. 188.

[13] Doyon JP. *Algorithmes pour la réconciliation d'un arbre de gènes avec un arbre d'espèces*. Manuscrit de thèse, 2010.

[14] Sankoff D., El-Mabrouk N. *Duplication, rearrangement and reconciliation*. Comparative Genomics, 2000, Vol. 1, p. 537-550.

[15] Guigo R., Muchnik I., Smith TF. *Reconstruction of ancient molecular phylogeny*. Molecular Phylogenetics and Evolution, 1996, Vol. 6, Issue 2, p. 189-213.

[16] Doyon JP., Scornavacca C., Gorbunov K.Yu., *et al. An Efficient Algorithm for Gene-Species Trees Parsimonious Reconciliation with Losses, Duplications and Transfers*. Comparative Genomics, 2010, Vol. 6398, p. 93-108.

[17] Szöllősi G.J., Tannier É., Lartillot N., *et al. Lateral Gene Transfer, Rearrangement and Reconciliation*. Syst. Bio., 2013, Vol. 63, Issue 3, p. 386-397.

[18] Doyon JP., Ranwez V., Daubin V., *et al. Models, algorithms and programs for phylogeny reconciliation.* Brief. Bioinform., 2011, Vol. 12, Issue 5, p. 392-400.

[19] Jean PA. *Algorithmique pour l'évolution des interactions géniques.* Rapport de stage de Master 2, Université Montpellier 2, 2013.