UNIVERSITÉ DU QUÉBEC À MONTRÉAL

MÉTHODES BIOINFORMATIQUES POUR L'ÉVALUATION DE LA CLASSIFICATION DU VIRUS DU PAPILLOME HUMAIN

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

BRUNO DAIGLE

OCTOBRE 2013

REMERCIEMENTS

Je tiens d'abord à remercier sincèrement le Professeur Abdoulaye Baniré Diallo pour sa direction subtile et intelligente, ses conseils de grande valeur, sa sagesse, son aide, sa patience et sa confiance.

Ma famille et mes amis ont tous contribué à mon succès, et je tiens particulièrement à remercier mes enfants Félix-Antoine, Julien et Sophie pour leur enthousiasme et leur encouragement à la poursuite de mes études à ce moment-ci. Je souligne aussi l'appui constant de ma sœur Louise, quoiqu'elle eût préféré un autre domaine, celui de feu mes parents, Anselme et Pauline, qui ont toujours accordé une place importante à l'éducation et finalement, celui de mon amie Johanne Roy avec qui je ne discute pas assez souvent des hauts et des bas de la vie, mais toujours de façon enrichissante.

Merci également à mes collègues du laboratoire de bioinformatique de l'UQAM, et particulièrement à Mohamed Amine Remita, pour les échanges stimulants et enrichissants qui m'ont parfois permis d'avancer plus rapidement ou de façon différente.

Mes remerciements vont aussi au CRSNG (conseil de recherches en sciences naturelles et en génie du Canada) et à l'UQAM, qui, par le biais de la bourse d'études supérieures du Canada Alexander-Graham-Bell et de la bourse d'excellence pour les cycles supérieurs de la Faculté des sciences, ont grandement contribué à la poursuite de mes études.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

RÉSUMÉ

La classification des centaines de virus du Papillome Humain (VPH) est toujours un enjeu majeur en virologie et revêt une grande importance pour la prévention, le diagnostic et le traitement des maladies associées. Depuis 2003, les VPH sont officiellement répertoriés en trois classes hiérarchiques principales correspondant à des seuils de pourcentage de similarité de leur séquence d'ADN. Avec le nombre croissant de génomes séquencés depuis ce temps, les frontières entre les différentes classes pourraient se chevaucher n'offrant pas ainsi une classification unique et cohérente. Pour le vérifier, à partir de l'information taxonomique extraite des 560 séquences de VPH obtenues de GenBank du NCBI, une méthode a été développée pour reconstruire un arbre où les feuilles sont les séquences génomiques, et où les nœuds internes correspondent à la hiérarchie des différentes classes. L'arbre est analysé à l'aide de programmes informatiques développés pour vérifier le respect des seuils de similarité. Une nouvelle mesure a aussi été élaborée afin d'évaluer l'étanchéité des différentes classes. Les résultats montrent que les seuils de pourcentages de similarité ne sont pas strictement observés et que les différentes classes se chevauchent. Un autre résultat montre qu'il est possible de générer exactement la classification actuelle en utilisant uniquement un algorithme informatique, ce qui n'est pas l'opinion de certains spécialistes du domaine. Il pourrait régner une confusion entre le procédé de classification et son résultat. Il apparait que la similarité entre les séquences est bien une caractéristique a posteriori de la classification, et qu'elle ne puisse pas être considérée comme un critère unique de décision permettant de la générer.

INTRODUCTION

La classification actuelle du virus du Papillome Humain (VPH), qui a été officialisée en 2003 [de Villiers et al., 2004], est issue d'un processus s'étalant sur de nombreuses années [de Villiers, 2013]. Ce système de classification, reconnu internationalement, est basé principalement sur la *similarité* des génomes entre les différents variants du virus. Le génome du VPH est long d'environ 8Ko, et compte de six à huit gènes (E1, E2, E4, E5, E6, E7, L1, L2), mais la classification actuelle est basée uniquement sur le gène L1. Les VPH sont répertoriés en trois classes hiérarchiques principales selon des seuils de similarité de la séquence d'ADN du gène L1 : le genre, l'espèce et le type. Au niveau du *genre*, la similarité entre le gène L1 des virus est de plus de 60%. Sous les genres se retrouvent les *espèces*, dont la similarité du gène L1 est de plus de 70%. Finalement, sous les espèces, se trouvent les *types* dont la similarité est de 90% ou plus. Ces seuils de similarité ont été établis à l'époque à partir de l'analyse de 118 virus et sont toujours utilisés.

La problématique de départ est que depuis ce temps, de nombreux nouveaux génomes de virus ont été complètement séquencés et ajoutés à la classification, mais les seuils de pourcentage de similarité n'ont pas été revus. Il y a maintenant 241 types de virus dont 150 de type humain et plusieurs variants de chaque type [Van Doorslaer et al., 2013]. Le système basé sur la similarité classifie les séquences de manière relative, les unes par rapport aux autres. L'ajout de centaines de séquences peut avoir un effet en changeant les interrelations de l'ensemble initial. Le but de ce travail est de mesurer cet effet au niveau des virus humains uniquement - les VPH - et de recommander éventuellement des ajustements de la classification pour tenir compte de la nouvelle réalité.

La question de base à laquelle ce travail de recherche tente de répondre est donc : la classification actuelle des VPH est-elle toujours congruente avec celle établie en 2003 et en respecte-t-elle les critères ? Des sous-questions se posent également : est-il possible de reproduire cette classification ? Possède-t-elle les qualités inhérentes d'une bonne classification, à savoir une bonne séparabilité des différentes classes ? Les objectifs du travail sont en ligne avec les questions de recherches :

- énumérer précisément les caractéristiques (ou critères) de classification établis en 2003 (section 2.1) ;
- récupérer la classification actuelle de tous les VPH (sections 2.2, 2.3) ;
- vérifier que la classification actuelle respecte toujours les caractéristiques (chapitre 3) ;
- déterminer si la classification peut être reproduite automatiquement et de manière non ambigüe (sections 3.5, 3.6) ;
- mesurer la séparabilité de la classification actuelle (chapitres 4, 5).

Dans ce mémoire, la validité des pourcentages de similarité est évaluée par rapport à la classification actuelle de 560 séquences publiques de VPH, du point de vue algorithmique. Cette classification est aussi évaluée par rapport à la similarité de tous les gènes, et comparée aux arbres phylogéniques inférés à partir du gène L1 et du génome complet des VPH.

Pour ce faire, à partir de l'information taxonomique extraite des 560 séquences de VPH obtenues de GenBank du NCBI, une méthode a été développée pour reconstruire un arbre où les feuilles sont les séquences génomiques, et où les nœuds internes de même hauteur correspondent à la hiérarchie des différentes classes (genre, espèce, type). Cet arbre est donc une incarnation de la classification. L'arbre est analysé à l'aide de programmes informatiques développés pour vérifier la conformité par rapport à la similarité. Une nouvelle mesure a aussi été élaborée afin d'évaluer l'étanchéité des différentes classes. Un autre algorithme a été implémenté pour comparer l'arbre taxonomique reconstruit avec les arbres phylogéniques inférés du gène L1 et du génome complet. Le travail d'analyse porte uniquement sur les aspects algorithmiques, informatiques et bioinformatiques de la classification, et à partir des données publiques accessibles.

Ce mémoire de maîtrise se divise en cinq chapitres, une introduction, une conclusion et une série d'annexes listant le code qui implémente les principaux algorithmes développés. Plusieurs algorithmes sont présentés sous forme de pseudo-code dans un but documentaire, autant pour alléger le texte de descriptions fastidieuses que pour préciser la façon dont les calculs ont été effectués, et donc pas nécessairement pour rendre compte de leur complexité. Le chapitre 1 survole les différentes notions nécessaires à la compréhension des analyses subséquentes. Dans la section biologique, la relation entre la cellule et le virus est établie, la description de leurs principales molécules est faite, suivie d'une introduction au VPH. S'en suit un survol des sources de données bioinformatiques et des méthodes de comparaison de séquences. Le chapitre 2 décrit la classification actuelle et en énumère d'abord spécifiquement chacune de ses caractéristiques,

pour détailler ensuite les étapes de la reconstruction de l'arbre taxonomique, qui est l'incarnation de la classification, pour finalement détailler le calcul de la similarité entre les virus. Le chapitre 3 reprend une à une les caractéristiques de la classification, en décrivant pour chacune la façon dont elle a été vérifiée et les résultats obtenus. La façon de générer la classification à partir de l'ensemble des séquences est discutée. Le chapitre 4 aborde une autre façon d'analyser la classification en considérant plutôt celle-ci comme un regroupement. Enfin le chapitre 5 présente une nouvelle mesure de validation de regroupement et son application à la classification actuelle. L'ensemble des constatations et les perspectives sont finalement exposés en conclusion.

Les résultats partiels de ce travail ont été présentés oralement à la conférence «European Conference on Data Analysis» (ECDA) le 11 juillet 2013 au Luxembourg (`http://gfkl2013.lu/`) et feront l'objet d'une publication.

CHAPITRE I

NOTIONS PRÉLIMINAIRES

Ce chapitre présente quelques éléments de biologie ainsi que le type de données informatiques utilisées et plusieurs méthodes de comparaison de séquences génomiques.

1.1 Éléments de biologie

1.1.1 La cellule

La cellule est l'élément constituant de tous les êtres vivants. Elle a la capacité de se reproduire et de transmettre ses caractères à ses descendants. Lors de la reproduction, un certain nombre de changements du matériel génétique peut s'opérer, constituant l'évolution de la cellule au fur et à mesure que les nouvelles générations apparaissent.

En plus de l'eau qui constitue plus de 70% de sa masse, on retrouve dans la cellule plusieurs molécules dont les acides nucléiques et les protéines [Cooper et Hausman, 2004]. Les acides nucléiques se divisent en acide désoxyribonucléique (ADN) et en acide ribonucléique (ARN). L'ADN et l'ARN sont des chaînes de nucléotides (des séquences) consistant en des bases de purines et pyrimidines liées à des molécules de sucre (désoxyribose ou ribose) et à des groupes de phosphates. L'ADN contient 2 purines, l'adénine (A) et la guanine (G), et 2 pyrimidines, la cytosine (C) et la thymine (T). L'ARN contient l'adénine, la guanine, la cytosine, et l'uracile (U) au lieu de la thymine. L'ADN stocke l'information génétique de la cellule à travers l'ordre des nucléotides dans la chaîne. Un gène est une section d'ADN qui spécifie la synthèse d'une protéine ou d'un ARN. Le génome de la cellule est l'ensemble de l'information génétique, donc l'ensemble des chaînes d'ADN.

Une protéine est un polymère unique d'acides aminés. Les acides aminés (AA), au nombre de 20,

sont des molécules constituées d'un atome de carbone lié à un groupe acide et un groupe amine, et d'une chaîne latérale distinctive, par exemple un atome d'hydrogène dans le cas de la glycine. Les protéines remplissent les rôles fonctionnels de la cellule (structure, communication, défense, catalyse, etc.), par opposition à l'ADN qui a un rôle informationnel [Cooper et Hausman, 2004].

La synthèse d'une protéine s'effectue en deux phases selon le dogme central de la biologie moléculaire : la transcription et la traduction, schématisées à la figure 1.1. La transcription consiste à convertir l'ADN d'un gène en ARN messager. Ensuite, lors de la traduction, les acides aminés sont assemblés selon l'information contenue dans l'ARN messager. Chaque groupe de 3 nucléotides de l'ARN (un codon) est traduit en un acide aminé, selon le code génétique (la table de traduction) qui fait correspondre un acide aminé à chacune des 64 possibilités de combinaison des 4 nucléotides A, C, G et U. Il est important de mentionner que tous les gènes ne sont pas traduits en protéines. Ceux qui sont traduits sont appelés les gènes codants et les autres, les gènes non codants. Les gènes non codants ont un rôle important dans la régulation des autres gènes.

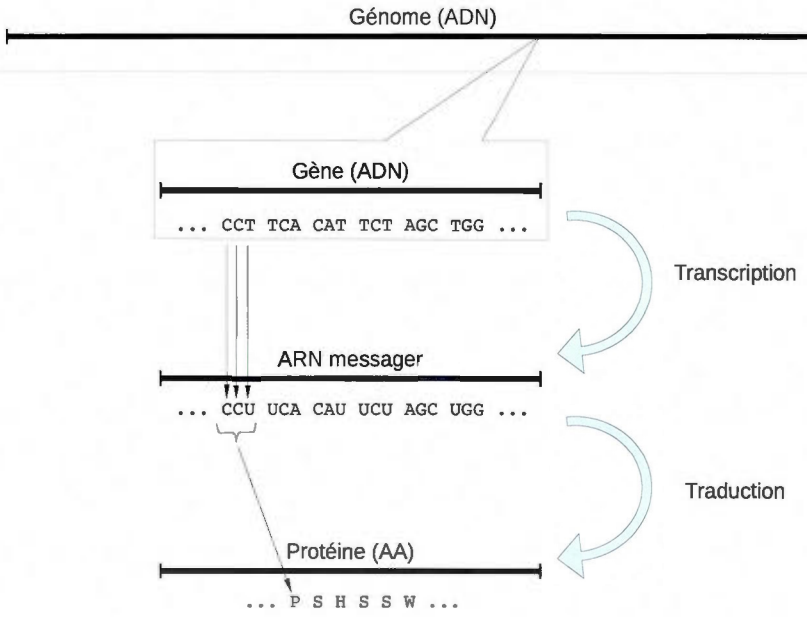


Figure 1.1: Dogme central de la biologie moléculaire.

1.1.2 Les virus

Les virus sont de petits agents infectueux qui exploitent la machinerie moléculaire de la cellule pour se répliquer. Ils sont également composés d'acides nucléiques, et entourés d'une couche protectrice de protéines (capside). De nouveaux virus peuvent apparaître suite à des mutations aléatoires «viables» de leur ADN, et qui sont perpétuées par la suite.

Malgré les spécificités propres à chaque type de virus, les principales étapes de la réplication sont les suivantes :

- Attachement. Les protéines à la surface du virus se lient à des récepteurs spécifiques à la surface de la cellule hôte.
- Pénétration. Le virus pénètre à l'intérieur de la cellule.
- Désassemblage. Le génome viral est relâché et disponible pour la transcription.
- Transcription et traduction. Le génome viral se réplique et les protéines virales sont produites.
- Assemblage. Les nouveaux virus sont ré-assemblés à partir des nouveaux génomes et protéines viraux.
- Libération. Les nouveaux virus quittent la cellule.

Les virus se distinguent selon le type d'acides nucléiques de leur génome [Dolja, Koonin et al., 2011] :

- Les virus à ARN, dont le génome est composé directement d'ARN.
- les virus à ADN, dont le génome est composé d'ADN et est transformé en ARN qui sera traduit par la cellule infectée.
- les retrovirus qui utilisent à la fois l'ADN et l'ARN pour leur réplication.

1.1.3 Le virus du Papillome Humain

La famille des *Papillomaviridae* (PV) regroupe des centaines de virus qui infectent les vertébrés [Antonsson et al., 2000]. Les PV qui ont été détectés chez les humains, au nombre actuel d'environ 150, sont identifiés comme *Virus du Papillome Humain* (VPH) et l'acronyme anglais HPV est souvent utilisé comme préfixe d'identification.

Bien que la plupart des infections au VPH ne provoquent aucun symptômes, certains types causent des tumeurs bénignes (verrues, papillomes) et d'autres encore, considérés à haut risque, sont directement impliqués dans le développement de tumeurs malignes comme le cancer du col

de l'utérus et de différentes muqueuses [Muñoz et al., 2006; Schiffman et Castle, 2003]. Parmi les 15 types considérés à haut risque, les HPV16 et HPV18 sont associés à 70% des cancers du col de l'utérus. Les autres types à risque sont les HPV31, 33, 35, 39, 45, 51, 52, 56, 58, 59, 68, 73 et 82 [Muñoz et al., 2003]. L'identification des types de VPH revêt donc une importance médicale, que ce soit pour le diagnostic ou le traitement [Bernard, 2005; Tota et al., 2011].

Le VPH est un virus composé d'ADN à double brin (2 chaînes d'ADN complémentaires reliées en spirale par des liaisons chimiques) d'environ 8000 nucléotides. Il comporte typiquement les gènes codant les protéines L1 et L2 qui forment la capside, les protéines E1 et E2 qui modulent la transcription et la traduction, et les protéines E5, E6 et E7 qui régulent le processus de transformation [Zheng et Baker, 2006]. La figure 1.2 montre une reconstruction de la capside du virus formée de protéines du gène L1, et la figure 1.3 représente sommairement l'organisation de son génome.

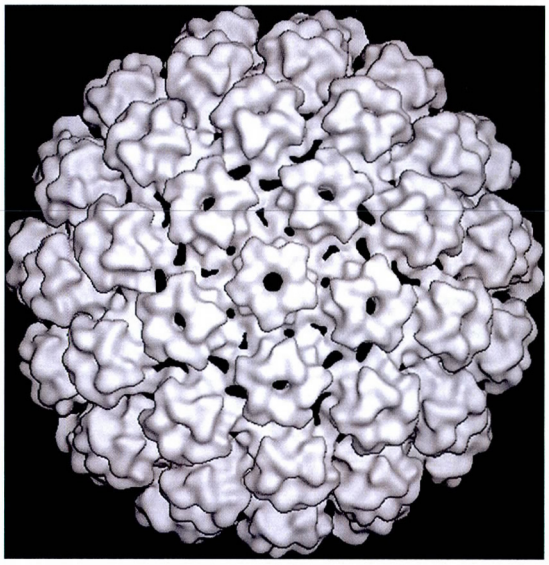


Figure 1.2: Capside du HPV16 [Buck, Day et Trus, 2013].

1.2 Sources de données

« La bioinformatique est l'utilisation de logiciels et de programmes informatiques pour organiser, stocker et analyser l'information et les données biologiques afin d'améliorer notre compréhension des systèmes biologiques » [BioPortail du Gouvernement du Canada, 2008]. Dans le cadre de ce mémoire, les données biologiques analysées sont les génomes des virus du VPH.

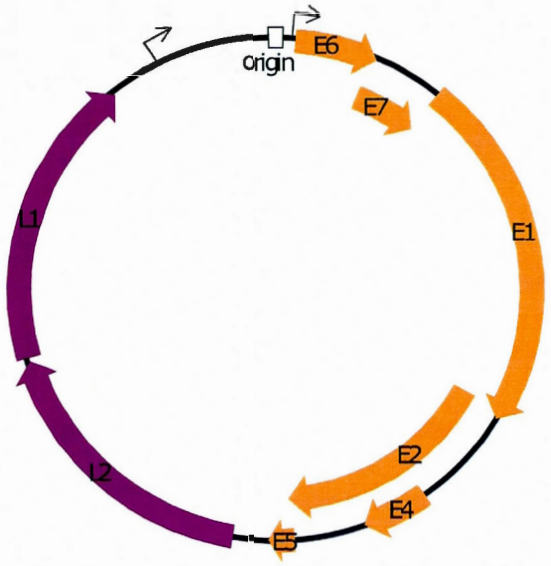


Figure 1.3: Organisation du génome du PV [Van Doorslaer et al., 2013].

Pour analyser ces génomes, l'ordre des bases (ACGT) constituant l'ADN doit être connu. Ces informations sont obtenues par le séquençage de l'ADN. Il existe plusieurs techniques de séquençage, chacune avec de nombreuses variantes. La méthode générale consiste à [Staden, 1979] :

- isoler par des moyens chimiques et mécaniques une certaine quantité de l'ADN à séquencer ;
- fragmenter ces échantillons d'ADN en petits morceaux, puisque les séquenceurs ne peuvent traiter que de quelques dizaines à quelques centaines de bases, selon la technologie employée ;
- amplifier les morceaux d'ADN pour en produire des copies, en employant essentiellement le même mécanisme utilisé par la cellule lors de réplication de l'ADN, mais dans un environnement contrôlé ;
- déterminer la succession des bases dans les fragments d'ADN (le séquençage proprement dit) par des procédés chimiques, chaque fragment représentant une *lecture* («reads») ;
- assembler l'ensemble des lectures pour obtenir la séquence complète par un programme informatique. Un nombre suffisant de lectures correspondant à des portions aléatoires du génome sont générées de manière à ce que chaque portion soit représentée partiellement un certain nombre de fois (de deux à quelques dizaines). Ainsi, certaines lectures se chevauchent, permettant de reconstituer le génome complet.

Toutes ces informations et d'autres sont stockées informatiquement, et semi-publiquement accessibles à partir de plusieurs sources. Quelques exemples parmi tant d'autres :

- «Ensembl» qui maintient de l'information génomique et produit des annotations [Flicek et al., 2013].
- «PDB («Protein Data Bank»)», une source d'information sur les protéines [Berman et al., 2000]. Outre les séquences, la base contient les structures, les références, des outils de comparaison, etc.
- «miRBase», une base de données de micro ARN , contient les séquences, les annotations, la documentation et une certaine classification [Kozomara et Griffiths-Jones, 2011].
- «1000 genomes», qui contient en format plutôt brut le génome de 1092 individus [Via i García et al., 2012].

Une des principales sources de données bioinformatiques est *GenBank* [Benson et al., 2013]. La base de données `Nucleotide` du NCBI contient les séquences d'ADN de près de 260000 espèces. Elle contient aussi les génomes de plus de 3500 virus et en particulier ceux du VPH. Chaque séquence (ou enregistrement de la base de données) est présentée dans un format propriétaire («GenBank flat file format») qui contient minimalement :

- une description (`DEFINITION` ou `TITLE`), par exemple «Human papillomavirus type 16, complete genome» ;
- un identificateur unique (`ACCESSION`), le numéro d'accession, par exemple NC_001526 ;
- le nom scientifique de l'organisme (`ORGANISM`), i.e. le taxon, duquel est issue la séquence, par exemple «Human papillomavirus type 16», et un identificateur unique (`taxID`) correspondant, par exemple 333760 dans ce cas ;
- diverses informations comme la date de soumission, les auteurs, les publications associées, etc ;
- un nombre variable d'annotations (`FEATURES`). Chaque annotation procure de l'information sur un gène ou une autre propriété biologique et la position (`LOCATION`) dans la séquence, par exemple «gene 83..559 /gene="E6"» ;
- la suite de nucléotides constituant la séquence.
- Un exemple est fourni à l'annexe B.1.

Un autre format populaire, mais qui contient seulement des identifiants et des séquences de nucléotides, est le format FASTA [Lipman et Pearson, 1985]. Par exemple, les séquences $s1$=`ACGTGCGATATA` et $s2$=`ACTGCGGATTA` sont représentées au format FASTA de la façon suivante :

```
>s1
```

```
ACGTGCGATATA
>s2
ACTGCGGATTA
```

Il existe aussi des bases de données spécialisées pour certains virus. Le «Papillomavirus Episteme» (PaVE), contient les génomes de référence des PV [Van Doorslaer et al., 2013]. Les responsables de la base de données ont modifié certains des génomes selon, par exemple, le travail déjà effectué par le laboratoire de Los Alamos [Myers, 1994], pour corriger ce qui est considéré comme des erreurs de séquençage. Ils ont également revu les annotations des gènes par un processus semi-automatisé de comparaison avec une base de données maison contenant des représentants typiques des protéines manuellement annotés.

1.3 Méthodes de comparaison de séquences

La génomique comparative est l'analyse des similarités et des différences entre les génomes [Hardison, 2003]. La pertinence de telles analyses repose sur deux faits centraux [Gusfield, 1997] :

- La structure est liée à la fonction : une grande similarité des génomes implique généralement une similarité fonctionnelle ou structurale. Mais l'inverse n'est pas nécessairement vrai : des génomes assez divergents peuvent avoir des structures (et donc des fonctions) similaires.
- Tous les génomes ont un ancêtre commun : ils se sont différenciés au cours de l'histoire par des séries de substitutions, de délétions ou d'insertions de nucléotides. En analysant les similitudes et les différences, l'histoire évolutive des génomes peut être inférée et un arbre d'évolution (arbre phylogénique) reconstruit.

1.3.1 Alignement de séquences

Un alignement (d'ADN ou d'acides aminés) est une façon d'arranger les séquences afin d'identifier les similarités entre elles. Pour procéder à un alignement de deux séquences, il est possible d'y insérer des espaces, appelés *brèches* ou *gaps*, et qui correspondent à des insertions et délétions (*indels*) de nucléotides. Une brèche dans une séquence est symbolisée par un tiret («-»).

Un alignement se représente comme un tableau où les séquences sont dans les lignes, et où les caractères consécutifs sont placés dans les colonnes consécutives. Lorsque deux caractères d'une colonne sont les mêmes, il y a équivalence (*match*), et *mismatch* lorsqu'ils diffèrent, à cause d'un *gap* ou d'une *substitution* (remplacement d'un nucléotide par un autre). Différents

systèmes de pointage (coût) sont utilisés pour quantifier la qualité d'un alignement. Par exemple en assignant la valeur 0 à un *match* et la valeur 1 à un *mismatch* et en appliquant le pointage à l'ensemble de l'alignement, une distance d'édition entre les deux est obtenue, c'est-à-dire le nombre d'opérations requises pour transformer une séquence dans l'autre.

1.3.2 Alignement global de séquences

Un alignement global de séquences fait correspondre chacun des nucléotides d'une séquence avec un nucléotide de l'autre séquence en tentant de maximiser le nombre de matchs. Il est utilisé lorsque la taille des séquences est semblable et les séquences assez similaires sur toute leur longueur. Il permet par exemple d'identifier des régions conservées.

L'alignement global se fait sur l'ensemble des deux séquences, en insérant le nombre de gaps requis pour qu'elles aient la même longueur. Par exemple, un alignement de la séquence $s1$=ACGTGCGATATA et $s2$=ACTGCGGATTA est :

```
s1 :  ACGTGCG-ATATA
      ||  |||| ||  ||
s2 :  AC-TGCGGAT-TA
```

et résulte de la délétion d'un G, de l'insertion d'un G et de la délétion d'un A pour une distance d'édition de 3. Un alignement global de bonne qualité minimise la distance entre les deux séquences (lorsque c'est le pointage utilisé). En général, un coût plus élevé est attribué pour un gap et des coûts différents selon les substitutions de nucléotides. À défaut de savoir celui qui représente l'évolution réelle des séquences, il n'y a pas de meilleur alignement, car il est fonction des paramètres de coût choisis.

Le calcul d'un alignement global au coût minimal peut s'effectuer par un algorithme de programmation dynamique en complexité de temps et d'espace quadratique ($\mathcal{O}(mn)$). La référence à cet algorithme se fait comme étant celui de Needleman-Wunsch [Needleman et Wunsch, 1970] :

Étant donnés :

- $s1, s2$: deux séquences à aligner de longueurs n et m,
- $s1(i), s2(j)$: le ième caractère de $s1$ et le jème de $s2$,
- $\sigma(x, y)$: le coût d'une substitution du caractère x à y,
- d : le coût d'un gap,
- D : une matrice de dimension n par m,

Alors les éléments de la matrice D sont définis par la récurrence :

$$D(i,j) = \max \begin{cases} D(i-1,j-1) + \sigma(s1(i), s2(j)) \\ D(i-1,j) + d \\ D(i,j-1) + d \end{cases}$$

Il est en effet possible de démontrer que $D(i,j)$ ne dépend que de $D(i,j-1)$, $D(i-1,j)$ et $D(i-1,j-1)$.

L'algorithme pour produire l'alignement à l'aide de la récurrence est expliqué en utilisant l'exemple d'alignement précédent des séquences $s1$=`ACGTGCGATATA` et $s2$=`ACTGCGGATTA`, selon l'implémentation réalisée par le programme C++ nw.cpp (A.1). Il utilise deux tableaux : le premier, D, pour mémoriser les valeurs de la récurrence, et le second, T (pour trace), pour rebâtir plus facilement l'alignement. La valeur de d est 2 et la fonction σ vaut 3 lorsque les deux caractères sont égaux et -1 sinon.

Le programme lit d'abord le fichier au format FASTA contenant les deux séquences :

```
>s1
ACGTGCGATATA
>s2
ACTGCGGATTA
```

et initialise le tableau D avec conceptuellement $s1$ comme première colonne et $s2$ comme première ligne. En fait la vraie première ligne $(0,j)$ est initialisée à la valeur $d*j$ et la vraie première colonne $(i,0)$ à la valeur de $d*i$ (tableau 1.1).

Le tableau de trace T est semblable mais initialisé aux valeurs de déplacement : D pour diagonale, U pour haut et L pour gauche (tableau 1.2).

		A	C	T	G	C	G	G	A	T	T	A
	0	-2	-4	-6	-8	-10	-12	-14	-16	-18	-20	-22
A	-2	0	0	0	0	0	0	0	0	0	0	0
C	-4	0	0	0	0	0	0	0	0	0	0	0
G	-6	0	0	0	0	0	0	0	0	0	0	0
T	-8	0	0	0	0	0	0	0	0	0	0	0
G	-10	0	0	0	0	0	0	0	0	0	0	0
C	-12	0	0	0	0	0	0	0	0	0	0	0
G	-14	0	0	0	0	0	0	0	0	0	0	0
A	-16	0	0	0	0	0	0	0	0	0	0	0
T	-18	0	0	0	0	0	0	0	0	0	0	0
A	-20	0	0	0	0	0	0	0	0	0	0	0
T	-22	0	0	0	0	0	0	0	0	0	0	0
A	-24	0	0	0	0	0	0	0	0	0	0	0

Tableau 1.1: Initialisation de la récurrence de l'algorithme de Needleman-Wunch.

		A	C	T	G	C	G	G	A	T	T	A
	DUL	L	L	L	L	L	L	L	L	L	L	L
A	U											
C	U											
G	U											
T	U											
G	U											
C	U											
G	U											
A	U											
T	U											
A	U											
T	U											
A	U											

Tableau 1.2: Initialisation de la trace de l'algorithme de Needleman-Wunch.

L'algorithme remplit ensuite le tableau D selon la récurrence. Les cellules sont numérotées à partir de 0. Alors, par exemple pour la cellule (1,1), le résultat est obtenu par :

- $D(i-1,j-1)+\sigma(s1(i),s2(j)) \equiv D(0,0)+\sigma(s1(1),s2(1)) \equiv D(0,0)+\sigma(A,A) \equiv 0+3 \equiv 3$
- $D(i-1,j) \equiv D(0,1) \equiv -2$
- $D(i,j-i) \equiv D(1,0) \equiv -2$
- alors $D(i,j) \equiv D(1,1) \equiv max(3,-2,-2) \equiv 3$

la valeur **3** se retrouve donc au tableau (tableau 1.3).

		A	C	T	G	C	G	G	A	T	T	A
	0	-2	-4	-6	-8	-10	-12	-14	-16	-18	-20	-22
A	-2	**3**	1	-1	-3	-5	-7	-9	-11	-13	-15	-17
C	-4	1	6	4	2	0	-2	-4	-6	-8	-10	-12
G	-6	-1	4	5	7	5	3	1	-1	-3	-5	-7
T	-8	-3	2	7	5	6	4	2	0	2	0	-2
G	-10	-5	0	5	10	8	9	7	5	3	1	-1
C	-12	-7	-2	3	8	13	11	9	7	5	3	1
G	-14	-9	-4	1	6	11	16	14	12	10	8	6
A	-16	-11	-6	-1	4	9	14	15	17	15	13	11
T	-18	-13	-8	-3	2	7	12	13	15	20	18	16
A	-20	-15	-10	-5	0	5	10	11	16	18	19	21
T	-22	-17	-12	-7	-2	3	8	9	14	19	21	19
A	-24	-19	-14	-9	-4	1	6	7	12	17	19	24

Tableau 1.3: Calcul de la récurrence de l'algorithme de Needleman-Wunch.

Il calcule en même temps les entrées correspondantes du tableau T selon la cellule d'où vient le résultat (le maximum). Donc, pour le même exemple (cellule (1,1)), la valeur de la récurrence, **3**, a été obtenue de la diagonale, et un **D** est donc mémorisé dans le tableau de trace à la même position (tableau 1.4).

Pour retrouver un des alignements optimaux, il s'agit de débuter au coin inférieur droit du tableau de trace (tableau 1.4), et de suivre les indications pour reconstruire l'envers de l'alignement :

- il y a un **D**, les 2 caractères sont alignés et la remontée est selon la diagonale
```
s1 : A
     |
s2 : A
```
- il y a un autre **D**, les 2 caractères sont alignés et la remontée est selon la diagonale
```
s1 : AT
     ||
s2 : AT
```
- il y a un **U**, on introduit un gap dans $s2$ et la remontée est directement vers le haut
```
s1 : ATA
     |||
s2 : AT-
```
- il y aura ensuite un D ce qui signifie que les deux prochains caractères (T) seront alignés, et ainsi de suite.

Il suffit finalement de renverser les chaines de caractères pour obtenir l'alignement.

La solution n'est pas unique, il peut y en avoir plusieurs, au même coût minimum (le tableau de trace contient d'ailleurs toutes les solutions, visibles par la présence de plusieurs symboles dans

		A	C	T	G	C	G	G	A	T	T	A
	DUL	L	L	L	L	L	L	L	L	L	L	L
A	U	D	L	L	L	L	L	L	DL	L	L	DL
C	U	U	D	L	L	DL	L	L	L	L	L	L
G	U	U	U	D	D	L	DL	DL	L	L	L	L
T	U	U	U	D	UL	D	DL	DL	DL	D	DL	L
G	U	U	U	U	D	DUL	D	DL	L	L	DL	DL
C	U	U	DU	U	U	D	DUL	L	L	L	L	L
G	U	U	U	U	DU	U	D	DL	L	L	L	L
A	U	DU	U	U	U	U	U	D	D	L	L	DL
T	U	U	U	DU	U	U	U	DU	U	D	DL	L
A	U	DU	U	U	U	U	U	DU	D	U	D	D
T	U	U	U	DU	U	U	U	DU	U	D	D	UL
A	U	DU	U	U	U	U	U	DU	DU	U	U	D

Tableau 1.4: Calcul de la trace de l'algorithme de Needleman-Wunch.

certaines cellules).

1.3.3 Alignement global multiple de séquences

Un *alignement global multiple* est une généralisation d'un alignement global à plus de deux séquences. Cela peut permettre de mettre en évidence des relations (une histoire évolutive) entre deux séquences qui sont assez divergentes lorsque comparées deux à deux, mais dont l'évolution peut devenir visible avec la présence de séquences intermédiaires.

Le calcul de l'alignement ne se fait généralement pas avec l'algorithme de Needleman-Wunsch, car il devient rapidement pratiquement incalculable : $\mathcal{O}(2^n)$ pour n séquences. Des heuristiques sont plutôt utilisés, ceux-ci étant le plus souvent des algorithmes progressifs [Edgar et Batzoglou, 2006]. En général, la méthode est la suivante :

- Alignement deux à deux de toutes les paires de séquences selon l'algorithme de Needleman-Wunsch.
- Construction d'une matrice de distance entre les toutes les paires de séquences.
- Calcul d'un arbre guide à partir de la matrice de distance. Un algorithme de regroupement hiérarchique comme UPGMA («Unweighted Pair Group Method with Arithmetic Mean» [Sokal et Michener, 1958]) peut être utilisé pour ce faire. Les nœuds les plus proches sont successivement regroupés sous un nœud parent. La distance entre deux nœuds A et B est calculée comme la moyenne de toutes les distances entre les éléments de A et de B.

- Alignement progressif suivant l'arbre guide. Les alignements partiels de séquences sont ajoutés à l'alignement multiple selon les bifurcations de l'arbre guide.

Parmi les méthodes les plus répandues, on retrouve [Edgar et Batzoglou, 2006] :

- «CLUSTALW», qui est apparue en 1994 et est toujours la plus utilisée [Thompson, Higgins et Gibson, 1994]. Elle n'a pas été substantiellement modifiée depuis, et des méthodes plus rapides ou plus exactes existent aujourd'hui.
- «MAFFT» [Katoh et al., 2002] et «MUSCLE» [Edgar, 2004] sont aussi des méthodes basées sur un alignement progressif, mais sont plus rapides et plus exactes.
- «T-COFFEE» [Notredame, Higgins et Heringa, 2000], qui améliore l'alignement progressif en cherchant un alignement qui maximise le consensus. Par exemple, en alignant les séquences (A, B) puis (B, C), on a implicitement un alignement de A et C, mais il peut être différent d'un alignement direct de (A, C).

1.3.4 Alignement local de séquences

Un *alignement local* cherche à identifier les sous-séquences les plus similaires, sans tenir compte de la position. Dans l'exemple précédent, la sous-séquence TGCG est présente dans les deux séquences et est la plus similaire de toutes les paires de sous-séquences. L'alignement local est utilisé lorsque la taille des séquences diffère beaucoup, pour rechercher des motifs partagés, ou pour faire des recherches dans les bases de données. Le logiciel BLAST («Basic Local Alignment Search Tool» [Altschul et al., 1990]) est l'un des plus utilisés en bioinformatique pour la recherche d'alignements locaux.

En plus des substitutions, délétions et insertions, il se produit des réarrangements de plus grande envergure, comme des duplications (répétition d'une sous-séquence) ou des translocations (déplacement d'une sous-séquence) qui touchent des sous-séquences de longueurs importantes. Par exemple, pour des sous-séquences S, T et U, l'alignement global des séquences STU avec SUT ne montrera qu'une similarité partielle de T et pas de similarité pour U (figure 1.4). Avec l'alignement local, les trois sous-séquences correspondantes entre STU et SUT devraient être retrouvées.

Pour les alignements locaux, la référence de base est l'algorithme de Smith–Waterman [Smith et Waterman, 1981]. Comme Needleman-Wunsch, il est aussi basé sur la programmation dynamique, mais la récurrence est différente.

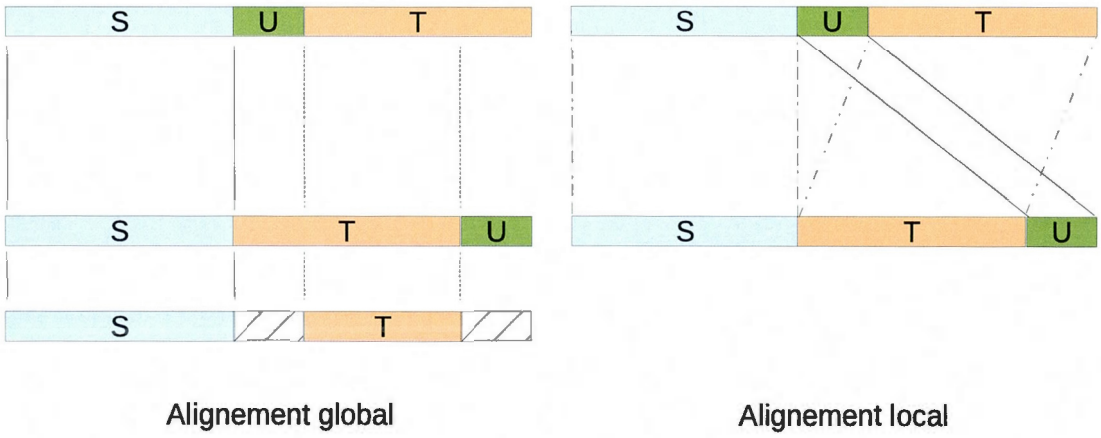


Figure 1.4: Alignement global vs local.

Étant donnés :

- $s1, s2$: deux séquences à aligner,
- $s1(i), s2(j)$: le ième caractère de $s1$ et le jème de $s2$,
- $\sigma(x, y)$: le coût d'une substitution du caractère x à y,
- D : une matrice de dimension n par m,

Alors les éléments de la matrice D sont définis par la récurrence :

$$D(i,j) = \max \begin{cases} 0 \\ D(i-1, j-1) + \sigma(s1(i), s2(j)) \\ D(i-1, j) + \sigma(s1(i), '-') \\ D(i, j-1) + \sigma('-', s2(j)) \end{cases}$$

Les différences par rapport à l'algorithme de Needelman-Wunch sont les suivantes :

- les premières lignes et colonnes sont initialisées à 0 plutôt qu'à un multiple de la pénalité d'un gap ;
- il n'y a pas de valeur négative dans le tableau ;
- pour trouver le meilleur alignement, le départ se fait avec la valeur maximale dans D, pour se terminer lorsqu'une valeur de 0 est atteinte.

1.3.5 Mesure de similarité

La similarité entre deux séquences est le rapport du nombre de caractères égaux (matchs) à la longueur totale de l'alignement global des deux séquences, et varie donc de 0 à 1.

Un alignement de longueur n des séquences $s1$ et $s2$ est représenté par un tableau A à 2 dimensions comportant deux rangées de n colonnes, où la rangée 1 de A contient les caractères de $s1$ ou le caractère «-», et de même pour la rangée 2 avec $s2$. La similarité entre les deux séquences est calculée ainsi :

Étant donnés :

- A un alignement global de longueur n de deux séquences,
- $diff(c1, c2)$: la comparaison de deux caractères

$$diff(c1, c2) = \begin{cases} 1 & \text{si } c1 \equiv c2 \\ 0 & \text{sinon,} \end{cases}$$

alors la similarité de A est :

$$similarité(A) = \frac{1}{n} \sum_{i=1}^{n} diff(A[1, i], A[2, i])$$

Par exemple, pour l'alignement vu précédemment :

```
s1 :  ACGTGCG-ATATA
      || |||| || ||
s2 :  AC-TGCGGAT-TA
```

il y a 10 égalités sur un alignement global de longueur 13. La similarité entre $s1$ et $s2$ est donc de 10/13 ou 0.77.

C'est parfois plutôt la dissimilarité (la distance) qui est requise par certains algorithmes dont il sera question plus loin. Pour l'obtenir, il suffit de faire le complément à 1 de la similarité ($1 - similarité$). Pour l'exemple précédent, la dissimilarité entre $s1$ et $s2$ est donc de $1 - 0.77 \equiv 0.23$.

1.3.6 Regroupement

Une autre façon de comparer les séquences (ou d'autres éléments) entre elles est de les regrouper selon certaines caractéristiques mesurables ou leur similarité. C'est le domaine du regroupement («clustering») qui peut se séparer en deux grandes catégories :

- Le *regroupement partitionné* où une «ligne» est tracée pour séparer les éléments en différents

sous-groupes (figure 1.5 (a) et (c)).

- Le *regroupement hiérarchique* où les éléments les plus similaires sont groupés récursivement ensemble jusqu'à obtenir une hiérarchie qui contient tous les éléments. Le processus peut aussi être à l'inverse : à partir de l'ensemble des éléments et en divisant successivement chaque groupe en un plus petit (figure 1.5 (b)).

Dans les deux cas, l'objectif est le même, à savoir que la similarité des éléments dans les sous-groupes du regroupement soit élevée, et que la similarité des éléments dans les sous-groupes différents soit basse.

Il existe des milliers d'algorithmes de regroupement [Jain, 2010]. Un des plus anciens au niveau du *regroupement partitionné*, et toujours largement utilisé, est K-means, découvert indépendamment dans plusieurs domaines d'application [Steinhaus, 1956; Lloyd, 1982; Ball et Hall, 1965; MacQueen et al., 1967].

Étant donnés :

- $\{x_i\}, i = 1..n$: les n éléments à regrouper,
- K : le nombre de sous-groupes à créer,
- $\{c_k\}, k = 1..K$: les sous-groupes à trouver,
- μ_k : la moyenne du sous-groupes c_k,

et la déviation du sous-groupes c_k par rapport à la moyenne μ_k : $\sum_{x_i \in c_k} ||x_i - \mu_k||^2$,

l'algorithme cherche à minimiser la somme de ces déviations : $\sum_{i=1}^{K} \sum_{x_i \in c_k} ||x_i - \mu_k||^2$.

En changeant la frontières entre les sous-groupes, la déviation de chacun diminue ou augmente. Il s'agit donc de trouver la séparation optimale pour minimiser la déviation totale. La méthode présente deux difficultés :

- le problème est NP-difficile [Drineas et al., 2004] ;
- le nombre de sous-groupes K doit être fixé à l'avance et il est souvent inconnu.

Le regroupement partitionné est utile par exemple pour analyser le résultat de deux expériences dans des conditions différentes. Le but est de vérifier si deux groupes de résultats bien séparés sont obtenus.

Le *regroupement hiérarchique* procède récursivement pour trouver les regroupements imbriqués :

- soit de haut en bas, en partant de l'ensemble des éléments, puis en les divisant récursivement

en sous-groupes plus petits ;

- soit de bas en haut, où initialement chaque élément forme un sous-groupe et sont récursivement réunis en sous-groupes de plus haut niveau pour obtenir au final un regroupement contenant tous les éléments.

Une matrice de similarité entre chaque paire d'éléments est utilisée pour séparer/grouper les sous-groupes les plus différents/similaires et la mesure de similarité utilisée dépend des données et aura une influence sur le résultat.

Il y a aussi différentes façons de calculer la similarité (ou la distance) entre les sous-groupes. Pour deux sous-groupes A et B et une fonction de distance d, on peut utiliser par exemple :

- le chaînage simple : la distance entre les deux éléments les plus proches $\min_{x \in A, y \in B} d(x, y)$;
- le chaînage complet : la distance entre les deux éléments les plus éloignés $\max_{x \in A, y \in B} d(x, y)$;
- le chaînage selon la moyenne : $\frac{1}{|A||B|} \sum_{x \in A} \sum_{y \in B} d(x, y)$

UPGMA, par exemple, utilise le chaînage selon la moyenne. Dans tous les cas, le regroupement hiérarchique produira un résultat (souvent représenté sous forme de dendogramme), soit une hiérarchie de sous-groupes, même s'il n'y a aucune hiérarchie réelle dans les données. Par contre il est intéressant de ne pas avoir à spécifier à l'avance le nombre de sous-groupes. Ce nombre peut être décidé a posteriori, au vu du résultat, en «traçant» une ligne transversale sur le dendogramme à la hauteur désirée, comme par exemple la ligne pointillée «cut off» de la figure 1.5 (b) où il est décidé de faire deux sous-groupes.

1.3.7 Inférence phylogénique

L'inférence phylogénique tente de classer les génomes selon leur histoire évolutive présumée pour produire un arbre. Les feuilles de l'arbre phylogénique correspondent aux génomes connus actuellement, alors que les nœuds internes représentent leurs ancêtres présumés, remontant éventuellement vers un unique ancêtre commun. Les arêtes entre les nœuds, ou *branches*, représentent les liens de parenté. La longueur de ces branches est une représentation du nombre de substitutions entre les génomes : une branche plus longue représente une relation plus éloignée. La figure 1.6 montre un exemple d'arbre phylogénique :

- les feuilles sont les taxons étudiés, les génomes dans ce cas-ci ;
- les branches représentent les relations de parenté entre les taxons ;

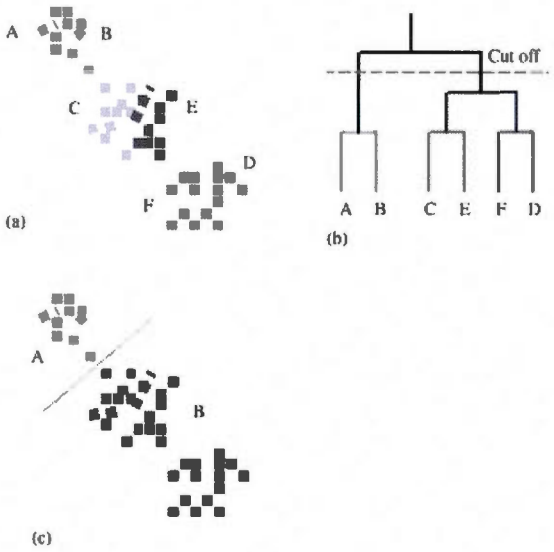


Figure 1.5: «Partition of the data into two classes. (a) Original data clustered into six clusters, (b) the hierarchical tree constructed from the six clusters, and (c) the final two-class partitioning of the data» [Szeto et al., 2003].

- les nœuds internes ($\star$) représentent les ancêtres présumés ;
- la racine (R) est le point de départ.

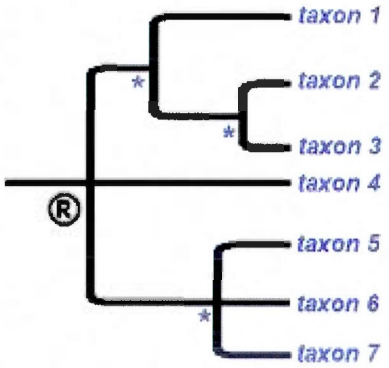


Figure 1.6: Exemple d'arbre phylogénique (1) [`http://www.acanthoweb.fr/fr/content/qu-est-ce-que-la-phylogenie`].

Il existe principalement deux grandes catégories de méthodes de reconstruction, basée soit sur les distances, soit sur les caractères. Toutes deux utilisent à la base un alignement multiple des séquences.

1.3.7.1 Méthodes basées sur les distances

Les méthodes basées sur les distances mesurent la distance (ou la dissimilarité) entre toutes les paires de séquences et reconstruisent l'arbre par un algorithme semblable au regroupement hiérarchique.

Ces méthodes, rapides en temps d'exécution, ont toutefois le désavantage de ne pas tenir compte des scénarios d'évolution de chacun des nucléotides, puisque cette information n'est pas contenue dans la mesure de distance. Outre UPGMA [Sokal et Michener, 1958], une des méthodes de ce type parmi les mieux connues est **Neighbor-Joining** [Saitou et Nei, 1987].

1.3.7.2 Méthodes basées sur les caractères

Les méthodes basées sur les caractères, par contre, tiennent compte de l'évolution. Une de ces méthodes est le maximum de vraisemblance. Elle repose sur un modèle de substitution des nucléotides (les caractères dans ce cas-ci) qui permet de calculer la probabilité qu'un caractère mute en un autre après un certain «temps». La méthode évalue la probabilité qu'un arbre génère l'alignement multiple selon le modèle d'évolution considéré. En fait, elle évalue cette probabilité pour chaque colonne de l'alignement, et la probabilité totale est le produit des probabilités de chaque position. L'arbre avec la meilleure probabilité est celui proposé comme solution. Pour trouver cet arbre il faut théoriquement évaluer tous les arbres, ce qui n'est pas pratiquement possible. Le logiciel **PhyML**, basé sur le maximum de vraisemblance, bâti un premier arbre par une méthode de distance (donc rapidement) et modifie par la suite sa topologie pour en optimiser la vraisemblance [Guindon et Gascuel, 2003; Guindon et al., 2010]. C'est un des logiciels les plus populaires et celui utilisé pour ce mémoire.

Il existe aussi deux autres catégories de méthodes :

- Le maximum de parcimonie (ou évolution minimum) [Fitch, 1971]. La méthode recherche l'arbre qui requiert le minimum de changements pour expliquer les séquences analysées. L'algorithme est implémenté dans plusieurs logiciels, dont **PAUP*** [Swafford, 2002] ou **PHYLIP** [Felsenstein, 1989].
- L'approche Bayésienne, très proche du maximum de vraisemblance, mais moins complexe algorithmiquement, utilise une approche de vraisemblance a posteriori. Le logiciel **MrBayes** implémente cette approche et est parmi les plus utilisés [Ronquist et al., 2012].

1.3.7.3 Support interne

Le support interne («bootstrap» [Efron, 1979]) est un procédé statistique qui peut être utilisé pour estimer la robustesse d'un arbre phylogénique T. Il consiste à échantillonner à nouveau, avec répétitions, les colonnes de l'alignement original ayant produit T, à refaire l'inférence phylogénique et puis à comparer l'arbre obtenu, X, avec T. La comparaison vérifie que chaque sous-arbre de T est présent dans X. En répétant le procédé n fois, une valeur de support interne est obtenue pour chaque sous-arbre de T : c'est le nombre de fois où le sous-arbre a été retrouvé dans X divisé par n. Une valeur de support interne plus près de 1 est un indicateur de la meilleure fiabilité de la prédiction. Ce procédé peut s'appliquer à l'inférence phylogénique car les nucléotides sont présumés évoluer de manière indépendante.

1.3.7.4 Représentation des arbres

Le format *Newick* est très répandu et consiste à utiliser des parenthèses imbriquées et des virgules pour représenter un arbre et la longueur des branches en format texte. Par exemple, l'arbre de la figure 1.7 est décrit par la structure Newick suivante :

(NC_001352 :0.00165252,(EF117891 :0.00164878,EF362754 :6e-08)96 :0.00081764)117 :0.00165474 ;

- la séquence (le nœud terminal ou la feuille) `EF362754`, qui a une longueur de branche de `6e-08` dans l'arbre, est représentée par : `EF362754:6e-08`
- de même pour la séquence `EF117891`, qui a une longueur de branche de `0.00164878`, est représentée par : `EF117891:0.00164878`
- et pour la séquence `NC_001352`, qui a une longueur de branche de `0.00165252`, est représentée par : `EF117891:0.00164878`
- les deux premières séquences forment un sous-arbre et sont séparées par une virgule et regroupées par une paire de parenthèses : *(EF117891:0.00164878,EF362754:6e-08)*
- la valeur de support interne de la bifurcation de ce sous-arbre est de 96 et sa longueur de branche est de `0.00081764`, donc la représentation est :
  (EF117891:0.00164878,EF362754:6e-08) 96:0.00081764
- ce sous-arbre forme un autre sous-arbre avec la troisième séquence, donc regroupés entre parenthèses et séparés par une virgule :
  (NC_001352:0.00165252,(EF117891:0.00164878,EF362754:6e-08)96:0.00081764)
- et ainsi de suite.

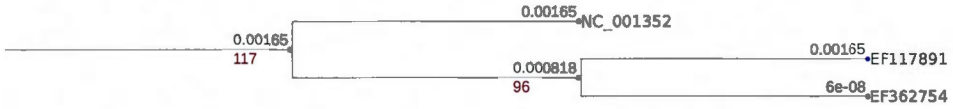


Figure 1.7: Exemple d'arbre phylogénique avec longueurs de branches et valeurs de support interne.

Le format NHX («New Hampshire eXtended») permet d'incorporer des attributs additionnels aux nœuds et aux branches sous forme d'étiquettes [Pavlopoulos et al., 2010]. Les attributs sont introduits par la chaîne de caractères «&&NHX:» suivie des étiquettes constituées d'une paire clé-valeurs; chaque étiquette est séparée par le caractère «:» et le tout est inséré entre crochets «[]». Par exemple en ajoutant l'étiquette «type=HPV2» selon le format NHX «[&&NHX :type=HPV2]», l'arbre précédent s'écrit alors :

```
(NC_001352:0.00165252[&&NHX:type=HPV2],
    (EF117891:0.00164878[&&NHX:type=HPV2],
        EF362754:6e-08[&&NHX:type=HPV2])96:0.00081764);
```

CHAPITRE II

CLASSIFICATION ACTUELLE DU VPH

L'organisation qui fait autorité pour la taxonomie des virus est le *International Committee on Taxonomy of Viruses* (ICTV). Les classes reconnues par le ICTV sont l'ordre, la famille, la sous-famille, le genre et l'espèce. Les *Papillomaviridae* ou PV, forment une famille qui n'est pas assignée à un ordre et qui ne comporte pas de sous-famille. Ils sont donc officiellement classifiés dans la famille *Papillomaviridae*, composée de 30 genres et d'une à 14 espèces selon le genre. La figure 2.1 montre l'arbre taxonomique de l'ensemble de tous les PV. Les VPH se retrouvent dans les genres alpha, beta, gamma, mu et nu.

Historiquement, les PV ont été classifiés en types, sous-types et/ou variants, mais cette classification n'est pas reconnue par le ICTV. L'utilisation des types est cependant très largement répandue dans le domaine. Les différents types se retrouvent à l'intérieur des espèces définies par le ICTV. La définition de sous-type et/ou variant n'est pas très utilisée et fait l'objet de discussions actuellement. Elle ne sera donc pas considérée par la suite. La classification des VPH sera donc évaluée uniquement selon le genre, l'espèce et le type.

Pour éviter toute ambiguïté, le sens dans lequel certains termes sont utilisés dans ce mémoire doit être précisé :

- La **classification** est un regroupement qui est le **résultat** de la disposition des différents éléments en sous-groupes. En ce sens, la figure 2.1 montre une classification. Le terme *arbre taxonomique* pourra aussi être utilisé dans le même sens.
- Un **classificateur** est une méthode permettant de générer une classification. Par exemple un algorithme séparant les nombres pairs et impairs en deux groupes.
- **Classifier** est l'action d'utiliser le classificateur.
- Une **caractéristique de la classification** est une information dérivée de la classification,

ou décrivant la classification.

- Un **critère de classification** est une information que le classificateur utilise pour déterminer quels sont les sous-groupes et dans quels sous-groupes disposer les éléments à classifier.

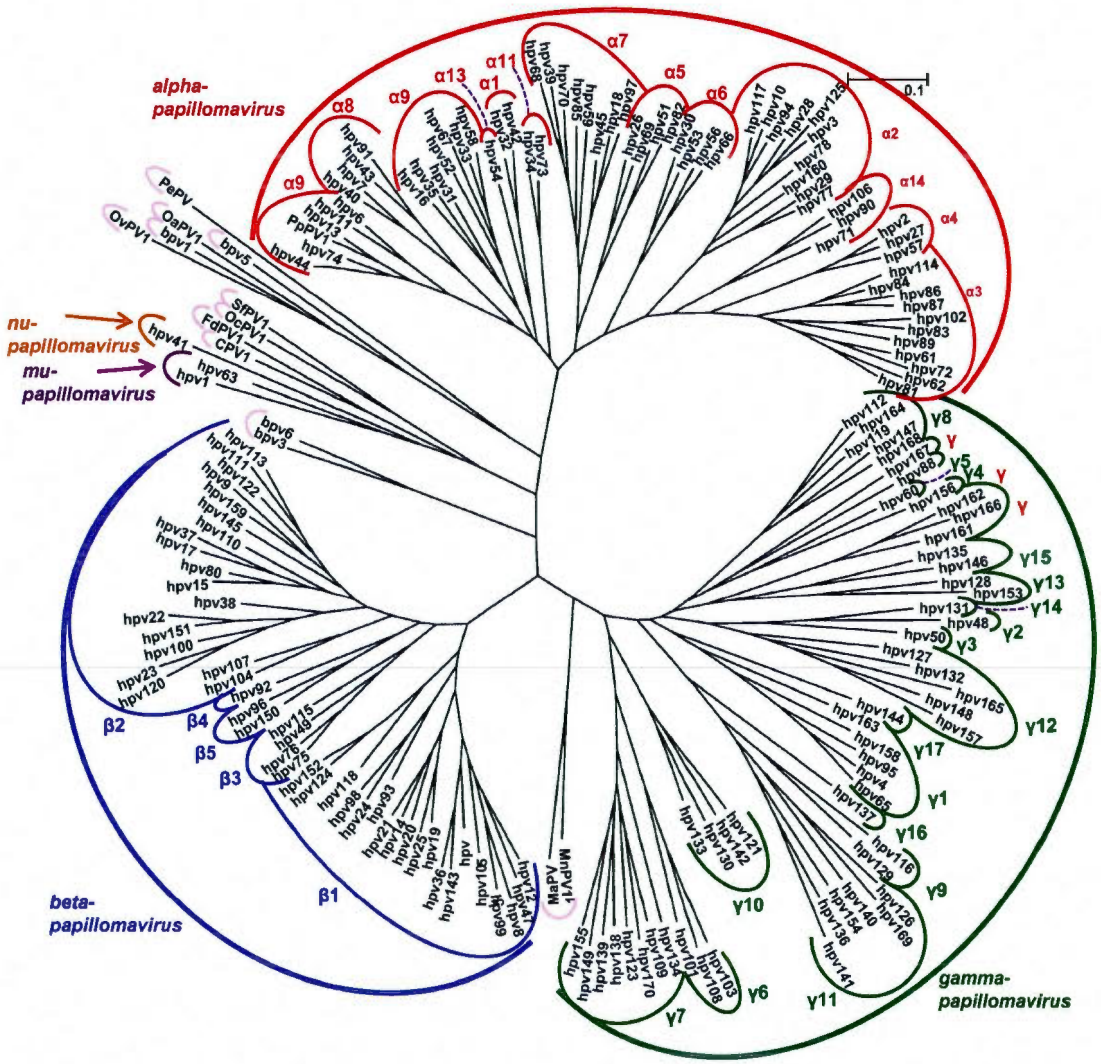


Figure 2.1: Arbre taxonomique de la famille *Papillomaviridae* [de Villiers, 2013].

Comme ce travail vise essentiellement à vérifier si la classification actuelle des centaines de génomes de VPH est congruente avec celle établie en 2003 et en respecte toujours les caractéristiques, il est d'abord nécessaire de décrire précisément ces dernières avant de pouvoir les analyser. Ces caractéristiques sont les suivantes [de Villiers et al., 2004; Bernard et al., 2010] :

- la similarité des séquences, soit le pourcentage global (section 2.1.1), la proximité des séquences

d'une classe (section 2.1.2) et la similarité intra-classes et inter-classes (section 2.1.3) ;

- la position topologique dans l'arbre phylogénique (section 2.1.4) ;
- de plus, il serait évidemment intéressant de pouvoir reproduire la classification actuelle et cette propriété est énoncé comme une caractéristique de reproductibilité (sections 2.1.5 et 2.1.6).

Finalement, pour pouvoir évaluer la conformité à ces critères au chapitre suivant (chapitre 3), il faut d'abord matérialiser la classification actuelle dans des structures de données. Cette matérialisation consiste à faire l'acquisition des données génomiques du VPH (section 2.2), puis à reconstruire l'arbre taxonomique de ces génomes (section 2.3)pour ensuite calculer la similarité entre les génomes (section 2.4).

2.1 Caractéristiques de la classification

2.1.1 Caractéristique de pourcentage de similarité des séquences

La classification des PVs repose essentiellement sur la ressemblance des séquences de nucléotides entre elles. Dans leur article de 2004, «Classification of papillomaviruses», de Villiers et al. présentent la distribution de la similarité du gène L1 de 118 PV (figure 2.2) [de Villiers et al., 2004]. Il y est mentionné que la même distribution est observable au niveau du génome complet du PV. Cependant, même si le virus possède plusieurs gènes, c'est la similarité du **gène L1 uniquement** qui est utilisée pour décrire la classification. La figure «démontre» trois niveaux taxonomiques : les *genres* («genus»), les *espèces* («species») et les *types*, séparés par deux minimum locaux autour de 60% et de 70% de similarité. C'est la prémisse de base, qui sera considérée comme une caractéristique fondamentale de la classification actuelle.

Il s'agit d'une classification hiérarchique, les classes (ou sous-groupes) étant définies ainsi :

- Les classes directement sous la famille *Papillomaviridae* sont les *genres*, identifiés par les lettres grecques (alpha, beta, etc.). La similarité entre les spécimens d'un même *genre* est de plus de 60%, et donc deux spécimens appartenant à des *genres* différents auront une similarité de moins de 60%.
- Les classes sous les *genres* sont les *espèces*, identifiées par le nom du *genre* correspondant concaténé à un entier (alpha1, alpha2, etc.). La similarité entre les spécimens à l'intérieur d'une *espèce* est de plus de 70%, et donc deux spécimens appartenant à des *espèces* différentes d'un même *genre* auront une similarité de 60% à 70%.

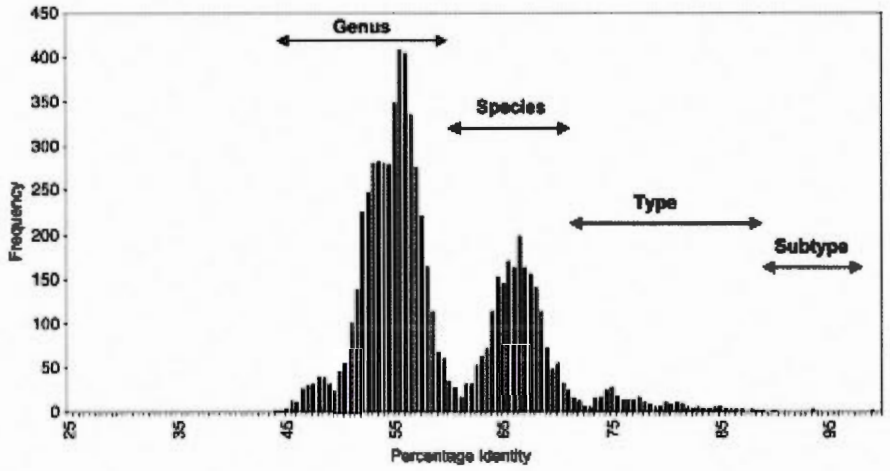


Figure 2.2: «Fig. 2. Distribution de la fréquence des pourcentages de similarité deux à deux des séquences de nucléotides des ORFs du gène L1 de 118 types de virus du papillome» [de Villiers et al., 2004]. Traduction libre.

- Les *types* se retrouvent sous les *espèces* et sont identifiés (pour le VPH) par les lettres «HPV» suivies d'un numéro attribué chronologiquement (HPV144, HPV145, etc.). La similarité entre les spécimens à l'intérieur d'un *type* est de 90% ou plus, et donc deux spécimens appartenant à des *types* différents d'une même *espèce* auront une similarité de 71% à 89%. Un nouveau type de PV est donc reconnu comme tel s'il y a une différence de plus de 10% entre la séquence de son gène L1 et la séquence du gène L1 du PV le plus similaire.

Le tableau 2.1 et la figure 2.3 résument les seuils de similarité.

Classe	% Similarité intra-classes	% Similarité inter-classes
genre	>= 60%	< 60%
espèce	>= 70%	60% - 70%
type	>= 90%	71% - 89%

Tableau 2.1: Caractéristiques de la classification selon la similarité du gène L1.

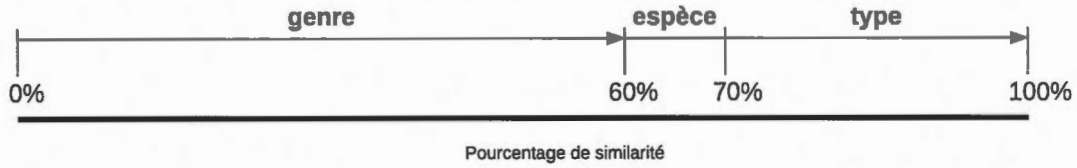


Figure 2.3: Répartition des classes selon la similarité.

Cette caractéristique peut donc s'énoncer ainsi : les séquences à l'intérieur des genres, espèces et types respectent les pourcentages de similarité du tableau 2.1 pour le gène L1.

2.1.2 Caractéristique de proximité des séquences

Dans la classification, la similarité avec la séquence la plus proche est utilisée pour désigner le sous-groupe auquel elle appartient en fonction des seuils de similarité. Ce fait peut donc être considéré et devenir une caractéristique s'énonçant ainsi : deux séquences qui ont une similarité maximale au niveau du gène L1 dépassant les seuils inter-classes sont toujours dans la même classe.

2.1.3 Caractéristique de distribution des pourcentages d'identité intra et inter-classes

Une analyse similaire est reprise par Bernard et al. en 2010, mais cette fois pour 189 PV, et le résultat est reproduit à la figure 2.4 [Bernard et al., 2010]. Elle y confirme ainsi les seuils de délimitation des différentes classes. Elle y ajoute cependant un bémol, appuyé par la figure 2.5. Il s'agit de la superposition de trois histogrammes des distributions spécifiques des pourcentages de similarité des 189 PV (tous les types et un représentant par type) selon :

- toutes les séquences à l'intérieur d'une même espèce (intra-espèces),
- les séquences d'espèces différentes à l'intérieur d'un même genre (inter-espèces),
- les séquences appartenant à des genres différents (inter-genres).

Les deux minimum locaux sont visibles à 60% et 70%, séparant la similitude inter-genres, inter-espèces et intra-espèces. Un léger chevauchement entre ces minimums es aussi visible, ce qui incitent les auteurs à temporiser ces seuils pour les considérer comme des lignes directrices et non des absolus, laissant ainsi une marge non strictement définie. Et ils en tirent la conclusion : « *Thus, assignment of PV types to species and genera cannot be relegated to a computer algorithm, but requires curation (i.e. interpretation based on phylogeny, genome organization, biology and pathogenicity).* »[1] Cette conclusion est intrigante et a partiellement motivé le travail effectué dans ce mémoire. En effet, l'algorithme existe, ne serait-ce que dans l'esprit de quelques spécialistes, puisque la classification actuelle n'a sûrement pas été créée aléatoirement. Cependant, cette phrase montre une certaine confusion entre la classification et le classificateur, si le

1. Bernard, H.-U., R. D. Burk, Z. Chen, K. van Doorslaer, H. zur Hausen et E.-M. de Villiers. 2010. « Classification of papillomaviruses (PVs) based on 189 PV types and proposal of taxonomic amendments ». Virology, vol. 401, no. 1, p. 70 – 79.

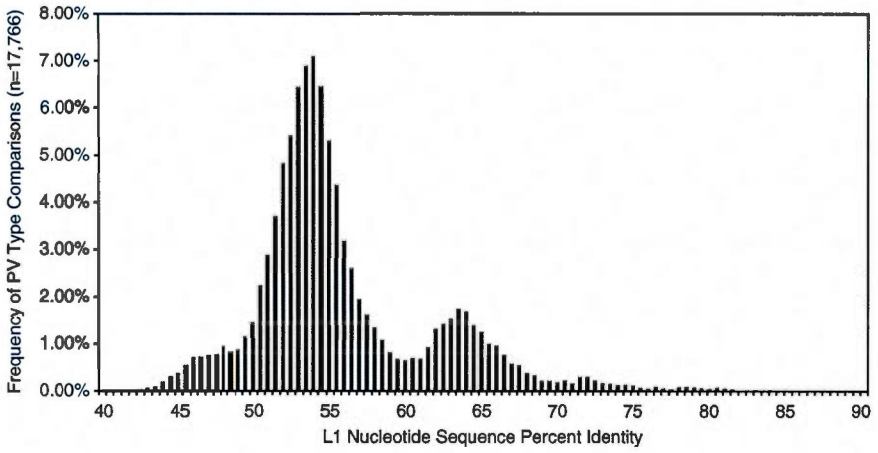


Figure 2.4: «Fig. 1. Distribution des comparaisons deux à deux des séquences de nucléotides de la région L1 de 189 virus du papillome. Les alignements multiples globaux des séquences de la région L1 ont été guidés par l'alignement des acides aminés en utilisant les logiciels MUSCLE v3.7 (Edgar, 2004) et Seaview v4.1 (Galtier et al., 1996). Une matrice de 189 régions L1 comparées avec elles-mêmes renferme un total de 17,766 valeurs de similarités. Les brèches ont été incluses et comptées comme positions valides. L'axe des Y représente le pourcentage du nombre total de comparaisons. L'axe des X montre le pourcentage de similarité de la séquence de nucléotide de la région L1. La figure montre une distribution bimodale dominante avec un chevauchement autour de 60% de similarité des séquences de nucléotides» [Bernard et al., 2010]. Traduction libre.

mot «assignment» est considéré équivalent à «classifier». À partir d'une observation du résultat, le chevauchement des seuils, on infère que le procédé ne peut s'y fier, ce qui s'apparente à un cercle vicieux.

Cette caractéristique est donc que, avec une certaine latitude non spécifiée : les distributions de similarité du gène L1 des séquences à l'intérieur d'une même espèce, des séquences d'espèces différentes à l'intérieur d'un même genre, et des séquences appartenant à des genres différents sont «semblables» à celles de la figure 2.5, avec des minimums locaux «autour» de 60% et 70% et de «légers» chevauchements autour de ceux-ci.

2.1.4 Caractéristique phylogénique

Dans le même article, il est ajouté que, puisque les pourcentages sont des lignes directrices, la classification doit tenir compte également de la position topologique dans l'arbre phylogénique et des propriétés biologiques. Comme les propriétés biologiques ne sont pas traitées dans ce travail, la caractéristique s'énonce ainsi : la classification est congruente avec la phylogénie (présumée)

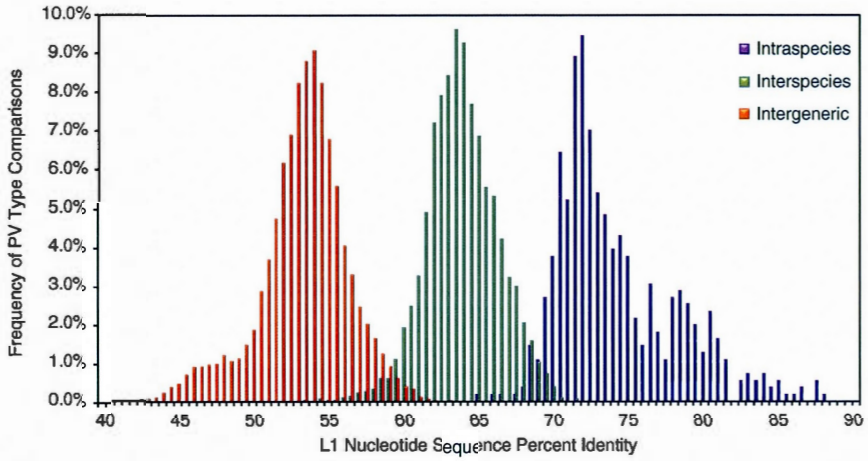


Figure 2.5: «Fig. 2. Comparaisons spécifiques inter-genres, inter-espèces et intra-espèces des séquences de nucléotides de la région L1 selon la matrice d'alignement multiple des séquences. L'alignement global multiple des séquences de la région L1 a été créé tel que décrit dans la légende de la Fig. 1. La même matrice a été utilisée pour évaluer la distribution intra-espèces : comparaison des types de PV à l'intérieur d'une même espèce (161 types de PV à l'intérieur de 149 espèces, 558 comparaisons) ; inter-espèces : comparaison des types de PV à l'intérieur d'un même genre (161 types de PV à l'intérieur de 10 espèces, 3207 comparaisons) ; inter-genres : comparaison de tous les types de PV à l'intérieur de différents genres (189 PV parmi 30 genres, 14,001 comparaisons). L'axe des Y représente le pourcentage du nombre total de comparaisons. L'axe des X montre le pourcentage de similarité de la séquence de nucléotide de la région L1» [Bernard et al., 2010]. Traduction libre.

du gène L1 des VPH.

2.1.5 Caractéristique de déterminisme

Dans le cas où une séquence S se trouve à distance égale de deux autres séquences, chacune dans une classe différente, dans quel sous-groupe devrait se trouver S ?

Si, par exemple, la dissimilarité d'une nouvelle séquence HPVx à classer est de 9% avec HPVy et 9% avec HPVz et que HPVy et HPVz appartiennent à des types différents, alors la dissimilarité de HPVx de 9%, inférieure au seuil intra-type de 10%, implique qu'elle appartiendrait à un type existant, mais deux choix sont possibles.

Cette caractéristique s'énonce ainsi : en fonction du gène L1, les séquences les plus similaires à une autre sont toujours tenues dans un même type, compte tenu des seuils de similarité.

2.1.6 Caractéristique de reproductibilité

La classification du VPH est très largement basée sur la similarité des séquences, mais aussi sur la phylogénie et l'organisation du génome, et de plus, de façon plus vague, sur la biologie et la pathogénicité. Mais pour ces deux derniers points, il y a des incongruités :

- les HPV6 et HPV11 se trouvent dans la même espèce alpha10, mais n'infectent pas dans le même type de tissu humain [Bernard et al., 2010], d'où l'incohérence biologique ;
- le HPV 16 est plus fortement associé au développement de cancer que les HPV31 et HPV35, qui sont pourtant de la même espèce alpha9 [Bernard et al., 2010], d'où l'incohérence du point de vue de la pathogénicité.

Les caractéristiques biologiques et de pathogénicité ne peuvent donc être considérés, puisque non systématiques. Dans tous les cas, pour que la classification soit utile, il faut qu'elle puisse être reproduite. Donc, nonobstant la biologie et la pathogénicité, la caractéristique peut s'énoncer ainsi : la classification actuelle des VPH peut être reproduite à partir des séquences génomiques du gène L1 en utilisant les caractéristiques de similarité ou de phylogénie.

2.2 Acquisition des séquences génomiques du VPH

2.2.1 Méthodologie

2.2.1.1 GenBank

La base de données `Nucleotide` du NCBI contient entre autres les génomes viraux. Chaque génome est associé à un organisme. Un organisme peut être associé à plusieurs génomes. Un organisme est en fait un taxon, par exemple `Human papillomavirus type 18`, lui-même correspondant à un identificateur numérique unique, le `Taxonomy ID` ou `taxID`, soit `333761` pour l'exemple précédent.

Le but est de récupérer tous les génomes complets associés aux taxons du VPH. Un ensemble de programmes développés auparavant est disponible pour télécharger les fichiers au format GenBank, les enregistrer localement dans une base de données SQL, et extraire les séquences génomiques. À l'aide de ces programmes, la méthodologie suivante est appliquée :

- L'ensemble de tous les taxons, disponible dans un fichier nommé `names.dmp`, est téléchargé du NCBI. De ce fichier sont extraits tous les `taxID` correspondant à un taxon contenant la

chaîne de caractères «human papillomavirus». 218 `taxID` sont obtenus.

- Avec les `taxID` sont récupérés en ligne de *GenBank* les `GI` («GenInfo Identifier» qui identifient uniquement une séquence dans la base de données) de la base de données `Nucleotide` correspondant à chaque `taxID`, et dont la longueur est comprise entre 6000 et 10000 nucléotides. 721 `GI` sont obtenus.
- À l'aide des `GI`, les génomes sont téléchargés et stockés dans la base de données locale : 723 enregistrements, au lieu de 721 car les `GI` 1683277 et 17939396 sont segmentés.
- 29 enregistrements sont éliminés, correspondant aux enregistrements segmentés ou incomplets, i.e. ne contenant pas la valeur exacte de chaque nucléotide (N au lieu de A, C, G ou T), laissant ainsi 694 enregistrements valides pour étude.
- Le NCBI choisit certains enregistrements de la base *Nucléotides* comme séquences de référence d'un type de PV et en fait une copie dont le nom débute par les caractères «NC_». De ce fait, 111 enregistrements parmi les 694 sont redondants, et éliminés, pour obtenir finalement 583 séquences génomiques uniques.
- Les séquences génomiques sont alors extraites de la base de données pour chaque gène et pour le génome complet du VPH, chacun dans un fichier au format `FASTA`. Il n'y a en général pas le même nombre de séquences dans chaque fichier. Pour le génome complet, il y a bien 583 séquences mais, par exemple, pour le gène L1, il n'y a que 571 séquences, car il y a 12 séquences qui, soit ne sont pas annotées (la position du gène L1 n'est pas indiquée dans `FEATURES`), soit ne contiennent pas le gène L1.
- Au niveau taxonomique, 45 spécimens se trouvent sous les taxons «unclassified Papillomaviridae» ou «unclassified Gammapapillomavirus» car le ICTV n'a pas encore entériné leur position officielle. Par contre, le PaVE propose une taxonomie au ICTV pour la majorité de ces spécimens. C'est cette classification qui est utilisée dans ce mémoire, obtenue en déplaçant les spécimens de «unclassified» au taxon proposé par le PaVE. Pour ce faire, certains nouveaux taxons (gamma11 à gamma17) ont dû être créés. Il ne reste finalement que 23 spécimens dans la catégorie «unclassified Papillomaviridae».

Les régions génomiques peuvent être extraites de la base de données SQL à l'aide des annotations pour obtenir les fichiers FASTA des gènes et du génome complet à l'aide du script bash fasta_genes_vph-2.sh (A.2) qui appelle itérativement le script fasta_un_gene_vph-2.sh (A.3), qui à son tour utilise le script python biodb_get.py.

Le script fasta_genes_vph-2.sh fournit, pour chaque gène et pour le génome complet, des ex-

pressions régulières au script fasta_un_gene_vph-2.sh, puisque les annotations ne sont pas normalisées et contiennent toutes sortes d'appellations en plus, par exemple, de «L1» : «L1 GENE», «L1 ORF», «ORF L1», etc.

Le script fasta_un_gene_vph-2.sh, quant à lui, cherche l'expression régulière parmi les annotations. En effet, toujours faute de normalisation, l'annotation du gène se trouve dans différents FEATURES, tantôt dans le CDS, tantôt dans le GENE et parfois dans NOTE. Aussi, l'annotation est parfois multiple. Dans ce cas, arbitrairement, le script choisit la séquence la plus longue.

2.2.1.2 PaVE

Les données du PaVE concernent uniquement le PV et peuvent être téléchargées d'un bloc comme un ensemble de fichiers, chacun contenant le génome d'un virus au format GenBank.

2.2.2 Résultats

Il y a 583 séquences génomiques complètes de VPH issues de GenBank, dont :

- 560 sont classées taxonomiquement,
- 548 sont classées taxonomiquement *et* contiennent le gène L1,
- 571 ont une annotation pour le gène L1.

L'ensemble des 583 séquences est utilisé lorsque l'analyse en cours ne requiert pas les relations taxonomiques. Dans le cas contraire, les sous-ensembles de 560, 548 ou 571 séquences sont employés selon la nécessité de la présence du gène L1. Le tableau 2.2 résume la répartition des espèces, types et séquences parmi les 5 genres dans lesquels on retrouve le VPH. Le tableau 2.4 présente l'inventaire détaillé du nombre de séquences pour chaque genre, espèce et type.

Dans la base de données du PaVE, il y au total de 241 séquences génomiques complètes, une seule par type de PV, dont 143 VPH. Chaque type de PV ne contient qu'une seule séquence (considérée comme la séquence de référence du type) et est rebaptisé avec un nom spécifique au PaVE et un pointeur vers la séquence originale du NCBI qui y est associée. Par exemple, le HPV16 est identifié par (`ACCESSION`) «HPV16REF» et la séquence originale «K02718» du NCBI y est associée. Il est important de noter que la séquence génomique originale ainsi que ses annotations peuvent avoir été modifiées dans le PaVE.

	Genre	# Espèce	# Type	# Séquence
	alpha	13	63	464
	beta	5	45	52
	gamma	17	39	39
	mu	2	2	4
	nu	1	1	1
Total	5	38	150	560

Tableau 2.2: Résumé de l'inventaire des séquences du VPH de GenBank. Pour chaque genre, le tableau présente le nombre d'espèces, de types et de séquences.

Il y a 138 séquences de VPH qui se retrouvent à la fois dans PaVE et dans les séquences choisies pour ce mémoire et qui sont annotées pour le gène L1. Sur les 560 séquences de GenBank, la vaste majorité (464) appartient au genre *alpha*. Parmi celles-ci, plus de la moitié (248) sont de l'espèce *alpha9*, dont 6 des 7 types de VPH sont considérés à haut risque, donc plus étudiés et conséquemment plus représentés. Mis à part les ajustements mentionnés plus haut, il n'y a pas eu de revue manuelle systématique de chaque séquence et de leurs annotations. Il y a fort probablement des séquences contenant des erreurs pour les données issues de GenBank et aussi des annotations non retrouvées car non normalisées.

Les données du PaVE ont par contre été complètement révisées, incluant des corrections au niveau de la séquence de nucléotides et une ré-annotation complète, le tout suivi d'un contrôle de qualité manuel. Ces données sont probablement plus fiables, mais moins abondantes (une seule séquence par type) et sujettes à des interprétations particulières, comme il sera souligné plus loin.

2.3 Reconstruction de l'arbre taxonomique

2.3.1 Méthodologie

2.3.1.1 GenBank

L'information taxonomique (`taxID`) de chaque spécimen de VPH est contenue dans chaque enregistrement de la base de données. Les `taxID` sont spécifiques au NCBI et n'ont donc pas une signification universellement reconnue. La hiérarchie entre les taxons peut être visualisée

à l'aide du *taxonomy browser* du NCBI mais n'est cependant pas électroniquement disponible sous forme structurée. L'arbre taxonomique a donc été reconstruit ainsi :

- Le serveur web «Interactive Tree Of Life» (ITOL) [Letunic et Bork, 2011], qui offre le service de rebâtir un arbre en utilisant les `taxID` du NCBI, a été utilisé pour produire l'arbre au format *Newick*.
- La relation séquence-taxon est extraite de la base de données locale par un énoncé SQL :

```
select taxon.ncbi_taxon_id, bioentry.accession
    from bioentry
    join taxon using (taxon_id)
    where bioentry.biodatabase_id = 8
```

- À l'aide d'un script et de la relation séquence-taxon, les noms des séquences sont ajoutés sous les `taxID` correspondants, devenant ainsi les feuilles de l'arbre. À partir de l'identification taxonomique complète, une abréviation est dérivée automatiquement et ajoutée à l'arbre, par exemple «HPV16» pour «Human papillomavirus type 16».

L'environnement «*ETE : a python Environment for Tree Exploration*» (ETE) [Huerta-Cepas, Dopazo et Gabaldon, 2010] a été utilisé pour la manipulation des arbres. Il permet de lire et écrire les arbres au format Newick et de les manipuler interactivement ou par programmation. Une classe *Arbre*, dérivée de la classe originale *Tree* de ETE, a été implémentée (script Arbre.py) pour ajouter différentes méthodes dont *comparer()*. Les différentes fonctionnalités de manipulation des arbres (dans la classe originale ou la classe dérivée) dans le pseudo-code des algorithmes sont représentés par les fonctions et méthodes suivantes :

Arbre(*[nw]*) : Création d'un nouvel arbre vide ou, optionnellement, à partir de la représentation *nw* au format Newick.

ecrire(*nw*) : Enregistre l'arbre au format Newick dans le fichier *nw*.

ajouterEnfant(*[nom]*) : Ajout d'un nœud optionnellement nommé *nom*.

parent(*nom*) : Retourne le nœud parent du nœud nommé *nom*.

chercher(*nom*) : Retourne le nœud (le sous-arbre) nommé *nom*.

parcourir() : Itérateur sur tous les nœuds de l'arbre.

soeurs() : Retourne tous les nœuds sœurs de l'arbre, c'est-à-dire ceux de même hauteur.

estFeuille() : Retourne un booléen indiquant si le nœud est une feuille (un nœud terminal).

hauteur() : Retourne la «hauteur» du nœud dans l'arbre.

lca(*nom1, nom2, ...*) : Retourne le nœud parent le plus bas («lowest commun ancestor») de *nom1* et *nom2* et ...

enleverFeuilles(*nom1, nom2, ...*) : Enlève les feuilles (le nœud correspondant) *nom1* et *nom2* et ... de l'arbre.

$\ell()$: Retourne l'ensemble des feuilles sous le nœud (toutes les feuilles du sous-arbre sous-arbre).

L'algorithme 1 implémenté par le script traiter_arbre_taxa_ncbi.py (A.4) permet d'ajouter les noms de séquences à l'arbre obtenu de ITOL (ITOL.nw). Pour chacune des séquences, la position de son taxon est localisée dans l'arbre et un nœud enfant, qui est la séquence elle-même, est ajouté.

```
nbSeq : le nombre de séquences
taxonAccession[nbSeq] : un tableau de l'association séquence-taxon

T = Arbre(ITOL.nw)

POUR i = 1 JUSQU'À i == nbSeq
    noeud = T.chercher(taxonAccesion[i].taxon)
    noeud.ajouterEnfant(taxonAccesion[i].accession)
FIN POUR

T.ecrire(ITOL.nw)
```

Algorithme 1: Ajout des séquences dans l'arbre taxonomique.

2.3.1.2 PaVE

Un tableau des séquences est téléchargé du PaVE où chaque ligne contient :

- l'identificateur de la séquence (par exemple «HPV1REF»)
- le type (par exemple «HPV1»)
- l'espèce (par exemple «Mupapillomavirus 1»)

Le genre peut être dérivé à partir de l'espèce (par exemple le genre «Mupapillomavirus» pour l'espèce «Mupapillomavirus 1»). L'algorithme 2 implémenté par le script arbrePaVE.py (A.5) reconstruit l'arbre taxonomique à partir de ces quatre informations. Il crée un nouvel arbre, et à chaque changement de genre, espèce, ou type, il ajoute le nœud correspondant, à la bonne hauteur.

```
nbSeq : le nombre de séquences
seq[nbSeq] : le tableau des séquences, trié en ordre de genre,
             espèce et type
T = Arbre()
genre = espece = type = ""
POUR i = 1 JUSQU'À i == nbSeq
    SI genre ≠ seq[i].genre
        genre = seq[i].genre
        T.ajouterEnfant(genre)
    FIN SI
    SI espece ≠ seq[i].espece
        espece = seq[i].espece
        T.ajouterEnfant(espece)
    FIN SI
    SI type ≠ seq[i].type
        type = seq[i].type
        T.ajouterEnfant(type)
    FIN SI
    T.ajouterEnfant(seq[i].sequence)
FIN POUR
T.ecrire(PaVE.nw)
```

Algorithme 2: Reconstruction de l'arbre taxonomique du PaVE.

2.3.2 Résultats

Le résultat final est un arbre dont les feuilles sont les génomes du virus, et dont les nœuds internes de hauteur 0, 1, 2 et 3 correspondent aux classes *famille*, *genre*, *espèce* et *type* respectivement. Un sous-ensemble de cet arbre est montré à la figure 2.6.

Deux arbres taxonomiques sont donc disponibles, au format Newick, de hauteur 4, représentant la hiérarchie des genres, espèces et types, et avec les séquences correspondantes identifiées par leur nom d'accession de GenBank et de PaVE. Ils sont l'incarnation de la classification actuelle du VPH selon GenBank et PaVE.

2.4 Calcul de la similarité

2.4.1 Algorithme

L'algorithme 3, implémenté par le script divergence.py (A.6), illustre la construction de la matrice de dissimilarité D. Chacune des séquences est alignée avec chacune des autres et la dissi-

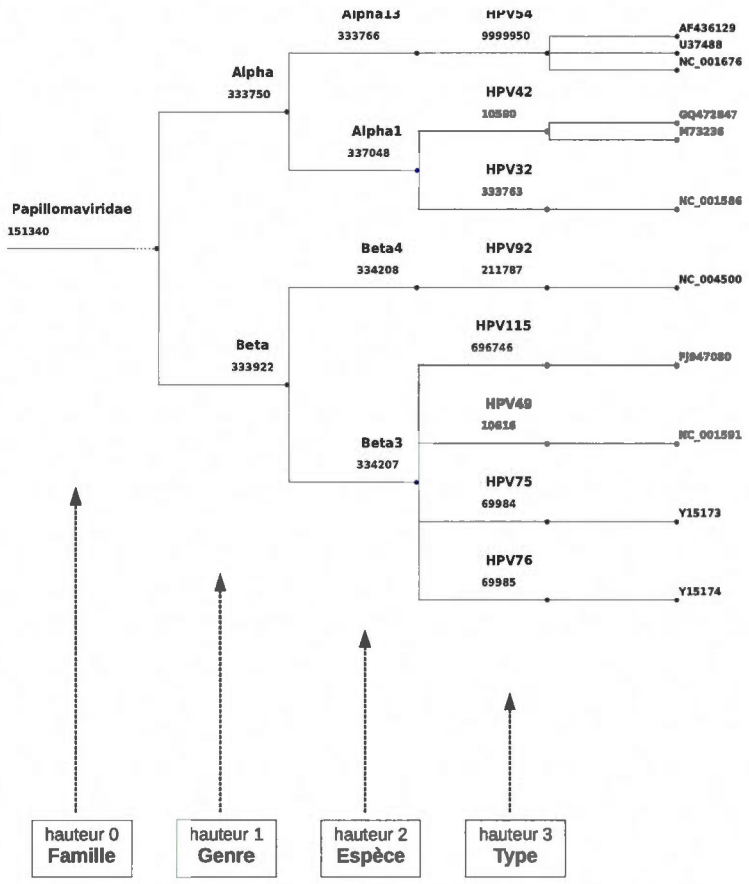


Figure 2.6: Sous-ensemble de l'arbre taxonomique du VPH. Les feuilles sont les génomes. Les nœuds internes de hauteur 0, 1, 2 et 3 correspondent aux classes *famille*, *genre*, *espèce* et *type* respectivement. Les étiquettes des nœuds internes sont l'abréviation du taxon et le `taxID`.

milarité est calculée et enregistrée.

La matrice de similarité est obtenue en complémentant à 1 chacune des valeurs de la matrice D. Le logiciel d'alignement utilisé est *MUSCLE* [Edgar, 2004]. Chaque élément $D[i, j]$ est la dissimilarité entre la séquence i et la séquence j. Il s'agit d'une matrice symétrique dont la diagonale est 0 puisque l'alignement de deux séquences ne dépend pas de l'ordre :

- $disimilarité(s1, s2) = disimilarité(s2, s1)$, et
- $disimilarité(s1, s1) = 0$

La matrice de dissimilarité est sauvegardée au format JSON (`http://www.json.org/`).

```
dissimilarité: fonction de calcul de la dissimilarité d'un alignement
alignement : fonction d'alignement global
D : matrice de dissimilarité
nbSeq : nombre de séquences
seq[nbSeq] : tableau des séquences

D = 0

POUR i = 1 JUSQU'À i == nbSeq - 1
    POUR j = i + 1 JUSQU'À j == nbSeq
        A = alignement(seq[i], seq[j])
        D[i, j] = D[j, i] = dissimilarité(A)
    FIN POUR
FIN POUR

ÉCRIRE D
```

Algorithme 3: Mesure de la dissimilarité.

2.4.1.1 Mise en garde

La méthodologie utilisée pour le calcul de la similarité dans l'article original n'est pas relatée en détails : « Distribution de la fréquence du pourcentage de l'identité paire à paire résultant de la comparaison des séquences du gène L1 » (figure 2.2) [de Villiers et al., 2004] (traduction libre).

C'est ce qui a été fait ici (avec MUSCLE) en faisant un alignement global de chaque séquence avec chacune des autres pour ensuite compter les divergences paires à paires.

Dans l'autre article ([Bernard et al., 2010]), plus élaboré sur le sujet, les auteurs indiquent avoir procédé à partir d'un alignement global (avec MUSCLE) de l'ensemble des séquences, pour ensuite calculer la matrice de similarité à partir de cet alignement (figure 2.4). En fait chaque paire de séquence est extraite de l'alignement pour ensuite calculer les similarités paires à paires. Le procédé est semblable, mais pas identique.

En effet, les alignements deux à deux indépendamment du reste sont plus précis et en général plus courts, car ils ne tiennent pas compte des autres séquences.

La matrice de distance résultante est certainement différente selon l'approche, mais proportionnellement semblable, comme la section 3.3.3 permettra de le constater.

	DQ090005	U31793	AF092932	NC_001583	NC_001587	...
DQ090005	0.0000000	0.4137931	0.4216182	0.3979461	0.4361766	...
U31793	0.4137931	0.0000000	0.3596435	0.3277202	0.3790123	...
AF092932	0.4216182	0.3596435	0.0000000	0.3229773	0.3662935	...
NC_001583	0.3979461	0.3277202	0.3229773	0.0000000	0.3687461	...
NC_001587	0.4361766	0.3790123	0.3662935	0.3687461	0.0000000	...
⋮	⋮	⋮	⋮	⋮	⋮	⋱

Tableau 2.3: Sous-ensemble de la matrice de dissimilarité du gène L1 des VPH de GenBank.

2.4.2 Résultats

Chacune des 571 séquences du gène L1 a été alignée globalement (avec MUSCLE) avec chacune des 570 autres séquences pour produire une matrice de dissimilarité. Un sous-ensemble est présenté au tableau 2.3. Une matrice de dissimilarité à également été calculée pour les autres gènes, pour le génome complet ainsi que pour les séquences du PaVE.

Il faut remarquer que même si les séquences génomiques complètes dupliquées ont été éliminées, il pourrait se retrouver au niveau d'un gène en particulier des séquences identiques et donc une dissimilarité nulle à ce niveau.

Alpha			Beta			Gamma			Mu			Nu		
esp.	*type*	#	*esp.*	*type*	#	*esp.*	*type*	#	*esp.*	*type*	#	*esp.*	*type*	#
1	32	1	1	RTRX7	1	1	4	1	1	—	1	1	41	1
	42	2		5	3		65	1		1a	2			
	Σ =2	Σ =3		8	1		95	1		Σ =2	Σ =3			
2	XS2	1		12	1		Σ =3	Σ =3	2	63	1			
	3	1		14	1	2	48	1	Σ =2	ΣΣ =3	ΣΣ =4			
	10	1		19	1	3	50	1						
	28	1		20	1	4	60	1						
	29	1		21	1	5	88	1						
	77	1		24	1	6	101	1						
	94	3		25	1		103	1						
	117	1		36	1		108	1						
	125	1		47	1		Σ =3	Σ =3						
	Σ =9	Σ =11		93	1	7	109	1						
3	61	2		98	1		123	1						
	62	1		99	1		134	1						
	72	1		105	1		138	1						
	81	1		118	1		139	1						
	83	1		124	1		149	1						
	84	1		143	1		155	1						
	86	1		Σ =19	Σ =21		Σ =7	Σ =7						
	87	1	2	SIBX8	1	8	112	1						
	89	1		9	1		119	1						
	102	1		15	1		147	1						
	114	1		17	1		Σ =3	Σ =3						
	Σ =11	Σ =12		22	1	9	116	1						
4	2	6		23	1		129	1						
	27	2		37	1		Σ =2	Σ =2						
	57	3		38	2	10	121	1						
	Σ =3	Σ =11		80	1		130	1						
5	26	1		100	1		133	1						
	51	1		104	2		142	1						
	69	1		107	1		Σ =4	Σ =4						
	82	2		110	1	11	126	1						
	Σ =4	Σ =5		111	1		136	1						
6	30	1		113	1		140	1						
	53	15		120	4		141	1						
	56	7		122	1		Σ =4	Σ =4						
	66	11		145	1	12	132	1						
	Σ =4	Σ =34		151	1		148	1						
7	cand85	1		Σ =19	Σ =24		Σ =2	Σ =2						
	18	26	3	49	1	13	128	1						
	39	1		75	1		153	1						
	45	13		76	1		Σ =2	Σ =2						
	59	2		115	1	14	131	1						
	68	4		Σ =4	Σ =4	15	135	1						
	70	1	4	92	1		146	1						
	97	3	5	96	1		Σ =2	Σ =2						
	Σ =8	Σ =51		150	1	16	137	1						
8	7	1		Σ =2	Σ =2	17	144	1						
	40	1	Σ =5	ΣΣ =45	ΣΣ =52	Σ =17	ΣΣ =39	ΣΣ =39						
	43	1												
	91	1												
	Σ =4	Σ =4												
9	16	106												
	31	23												
	33	23												
	35	28												
	52	23												
	58	37												
	67	8												
	Σ =7	Σ =248												
10	6	41												
	11	26												
	13	2												
	44	2												
	74	1												
	Σ =5	Σ =72												
11	34	1												
	73	1												
	Σ =2	Σ =2												
13	54	3												
14	71	5												
	90	2												
	106	1												
	Σ =3	Σ =8												
Σ =13	ΣΣ =63	ΣΣ =464												

Tableau 2.4: Inventaire des séquences du VPH de GenBank selon les classes taxonomiques. Pour chaque genre (alpha, beta, etc.), le tableau présente 3 colonnes : la colonne *esp.* est le numéro de l'espèce à ajouter au genre (par exemple 1 sous alpha correspond à l'espèce alpha1), la colonne *type* identifie le type que l'on doit préfixer de «HPV» (par exemple 32 correspond à HPV32), et sous la colonne # le nombre de séquences correspondant au *type*. À chaque changement d'espèce, préfixé du symbole $\Sigma =$, se trouve le total du nombre de types et de séquences de l'espèce (si plus de un type). À chaque changement de genre, préfixé du symbole $\Sigma =$, se trouve le total du nombre d'espèces du genre (si plus d'une espèce), suivie du nombre total de types et de séquences du genre, préfixés de $\Sigma\Sigma =$.

CHAPITRE III

VÉRIFICATION DE LA CONFORMITÉ DE LA CLASSIFICATION DU VPH AUX CARACTÉRISTIQUES INFORMATIQUES

Les caractéristiques de la classification ont été établies il y a plusieurs années. La classification elle-même a débuté avec un nombre limité de séquences. Depuis, le nombre de virus complètement séquencés n'a cessé d'augmenter et les caractéristiques de la classification n'ont pas été modifiées au fur et à mesure de ces ajouts.

Il est à-propos de poser la question à savoir si la répartition des séquences dans les différentes classes taxonomiques respecte toujours les caractéristiques sur lesquelles est basée la classification. Pour ce faire, chacune des caractéristiques énoncées au chapitre précédent est vérifiée.

3.1 Vérification de la caractéristique de pourcentage de similarité des séquences

Il s'agit de vérifier que la distribution de la similarité deux à deux de l'ensemble des séquences du gène L1 des 571 VPH actuels est similaire à celle de la figure 2.2 ou 2.4 avec un minimum local autour de 60% séparant les genres des espèces, et de 70% pour la séparation des types, ou tel qu'énoncé plus tôt : les séquences à l'intérieur des genres, espèces et types respectent les pourcentages de similarité du tableau 2.1 pour le gène L1.

3.1.1 Algorithme

Pour débuter par une comparaison visuelle, il faut reproduire le graphe. Ce graphe est en fait un histogramme de la matrice triangulaire supérieure (ou inférieure) de similarité, sans la diagonale, présentant la distribution des pourcentages de similarité. C'est ce que produit le script distSimilarite.py (A.7).

Il accepte comme paramètres :

- une matrice de dissimilarité,
- le nom du fichier à produire,
- la liste des séquences à inclure ou à exclure,
- le titre du graphique,

et utilise le logiciel *R* [R Development Core Team, 2008] pour :

- lire la matrice,
- enlever ou extraire les séquences voulues,
- faire le complément à un pour obtenir une matrice de similarité,
- faire l'histogramme (fonction *hist*) de la matrice triangulaire supérieure sans la diagonale pour ne pas avoir la répétition des comparaisons symétriques,
- produire le graphique.

3.1.2 Résultats pour le gène L1

La figure 3.1 montre la distribution de la similarité deux à deux des 571 séquences du VPH du gène L1 de GenBank (les séquences «unclassified» sont conservées puisque qu'il n'y a pas de comparaison aux classes taxonomiques). L'allure du graphe est assez différente de celle de la figure 2.4 bien que le minimum local vers 63% et vers 73% puisse se deviner. Notamment, il y a une fréquence plutôt élevée, à la droite, autour de 100% d'identité. En effet, il y a un grand nombre de VPH identiques ou presque identiques au niveau du gène L1 et ceci a pour effet de perturber le graphe. Avec n paires de séquences identiques qui diffèrent de x% d'une autre séquence, la fréquence pour le pourcentage d'identité x sera augmentée de n et il y aura n fréquences de plus à 100% d'identité.

Puisqu'il s'agit de l'élément fondamental sur lequel repose la classification, il serait très souhaitable de reproduire la distribution de la figure 3.1, et dans l'impossibilité de le faire, d'expliquer pourquoi. Dans ce but, différents scénarios sont explorés :

- En ne conservant qu'une séquence parmi celles dont la similarité est de 99% et plus, il est possible de tenter de limiter l'effet des séquences très similaires (script genereDistSimilarite-Distance01.sh). La figure 3.2a est alors obtenue. Il n'y a plus de minimum local autour de 60% et l'allure est toujours différente de celle attendue. Les séquences très similaires ne semblent donc pas être la cause de la divergence des graphes.

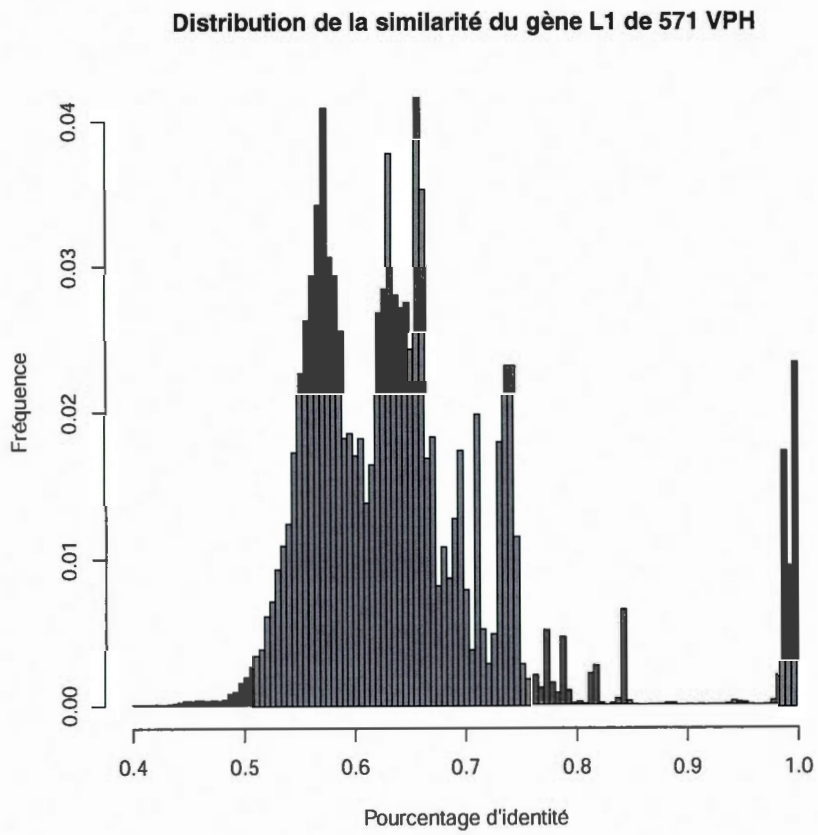
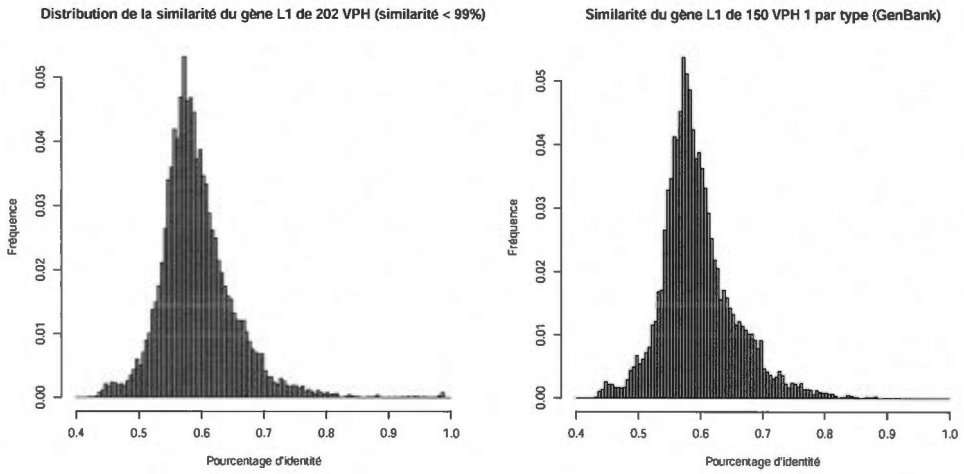


Figure 3.1: Histogramme de la similarité des comparaisons deux à deux du gène L1 de 571 VPH de GenBank.

- Une autre hypothèse est que la sur-représentation de certaines classes biaise la fréquence de similarité, puisque les séquences d'une classe sont plus similaires entre elles. Par exemple le type *HPV16* (genre *alpha*, espèce *alpha9*) compte 106 séquences alors que la plupart des types n'en comptent qu'une seule. Il s'agit alors plutôt de calculer la distribution des fréquences de similarité en ne gardant qu'une seul séquence choisie pseudo-aléatoirement par type de VPH. Le résultat est alors, par exemple, la figure 3.2b. Là non plus, le résultat cherché n'est pas obtenu.
- Il est possible que le hasard n'ait pas été favorable, car en se basant sur le tableau 2.4, il y a théoriquement 1.0626065e+27 façons différentes de choisir un ensemble de séquences avec un seul représentant pour chacun des types de VPH. Avec un échantillon pseudo-aléatoire de 1000 ensembles de telles séquences, la figure 3.3 est obtenue. Il s'agit de 1000 histogrammes,



(a) Gène L1 de 202 VPH de GenBank avec similarité $< 99\%$.

(b) Gène L1 de 153 VPH de GenBank avec un représentant par type.

Figure 3.2: Distribution de la similarité de deux sous-ensembles de séquences de VPH de GenBank.

correspondants aux 1000 ensembles de séquences, superposés sur le même graphe. Chaque ensemble de séquences compte 150 séquences ou un peu moins car le gène L1 n'est pas identifié dans chaque génome. Toutes les distributions sont semblables et le minimum local autour de 60% n'est toujours pas présent. Il est donc permis de conclure avec une certaine confiance que la distribution cherchée n'est pas présente dans les données.

- Il reste aussi l'hypothèse que l'absence de séquences pour les autres espèces animales que l'humain, qui sont a priori toutes moins similaires, ne permette pas de reproduire la courbe souhaitée, mais il sera montré plus bas que ce n'est pas le cas.

D'autres hypothèses pourraient être encore élaborées, mais l'utilisation d'un autre jeu de données, celui du PaVE, a d'abord été expérimentée afin de reproduire la courbe souhaitée. Avec les 138 séquences de VPH qui se retrouvent à la fois dans PaVE et dans les séquences GenBank choisies pour le mémoire, la figure 3.4a est produite. Les minimums locaux recherchés sont présents, bien qu'il soient situé à 63% et 73% plutôt que 60 et 70. Cela permet aussi de confirmer que les types humnains de PV sont suffisants pour générer cette distribution (qui infirme la dernière hypothèse).

En ayant ainsi trouvé une combinaison de séquences reproduisant presque la courbe voulue, il

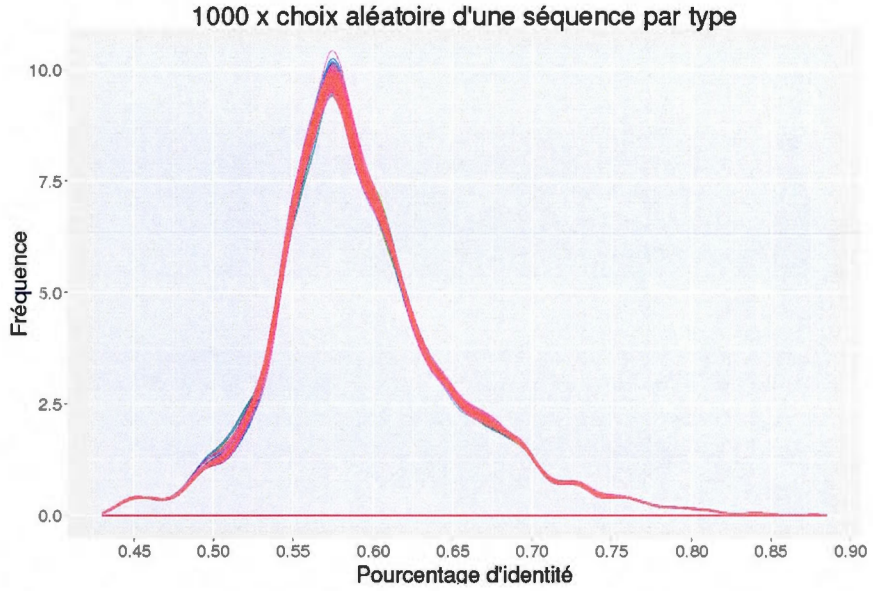


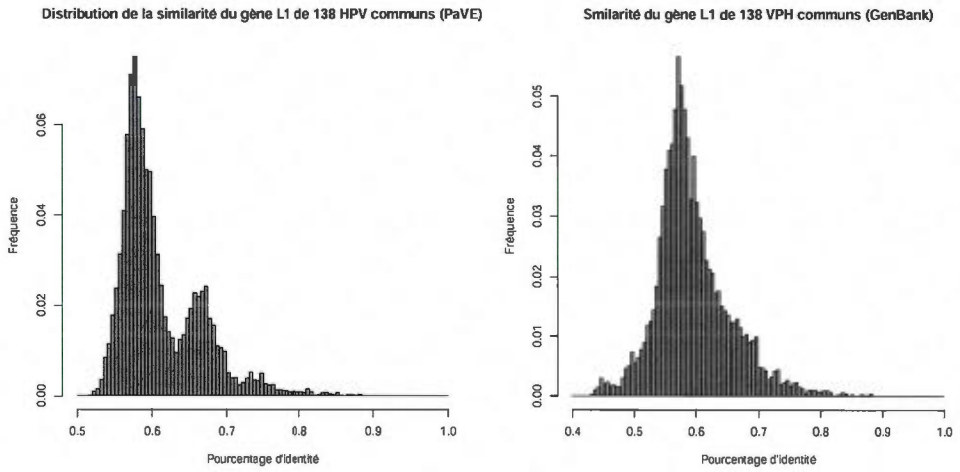
Figure 3.3: 1000 histogrammes de la similarité du gène L1 d'une séquence par type de VPH.

est maintenant possible de refaire le graphique avec les mêmes séquences génomiques originales correspondantes dans GenBank. La figure 3.4b montre le résultat, où le minimum local est encore absent.

La différence se trouve donc obligatoirement dans les données, et non dans la sélection des séquences représentantes du type.

Pour le démontrer, la séquence de nucléotides du gène L1 de PaVE est comparée avec la séquence de GenBank pour chacun des 138 types. Les séquences sont alignées deux à deux avec MUSCLE [Edgar, 2004], puis les différences comptées de la même façon que lors du calcul des matrices de dissimilarités. Il y a 39 des 138 VPH qui présentent des différences de similarité. Deux de ces séquences (HPV70(U21941) : une divergence, et HPV5(NC_001531) : trois divergences) ont été corrigées par le PaVE à l'aide d'information de diverses sources et les différences attribuées à des erreurs de séquençage. Le tableau B.1 énumère les 37 séquences restantes. Il faut d'abord noter que la dissimilarité est due uniquement à des gaps, mises à part les séquences NC_001526 et NC_001357 où les quelques mismatchs supplémentaires peuvent être attribués à des corrections manuelles effectuées par le PaVE.

En examinant le résultat de la comparaison dont voici un exemple partiel :



(a) 138 VPH de PaVE.

(b) 138 VPH de GenBank.

Figure 3.4: Distribution de la similarité du gène L1 de 138 VPH avec le données du PaVE et de GenBank.

```
comparer:  HPV2REF NC_001352
HPV2REF    ----------------------------------------------ATGGCTTTGTGGCGGCCT ...
           ##############################################
NC_001352 ATGTCTTGTGGCCTAAACGACGTAAACGTGTCCACTATTTCTTTGCAGATGGCTTTGTGGCGGCCT ...
48/1533 divergences (0.031)
1485 matches, 0 mismatches, 48 gap, score: 4359
```

il apparait que tous les gaps sont regroupés au début, et que les séquences du PaVE sont toutes plus courtes, à l'exception du HPV54REF, bizarrement, qui commence plutôt au premier codon d'initiation contrairement à GenBank. Dans tous ces cas, le gène L1 débute au second codon d'initiation (ATG). La moyenne de dissimilarité est de 0.066, le minimum est 0.004 et le maximum est 0.222.

3.1.3 Résultats pour le génome complet

La distribution de similarité du génome complet des 583 VPH de GenBank se retrouve illustrée à la figure 3.5. Cette distribution n'est pas du tout semblable celle du gène L1 (figure 3.1), alors qu'elle devrait l'être selon de Villiers et al (2003).

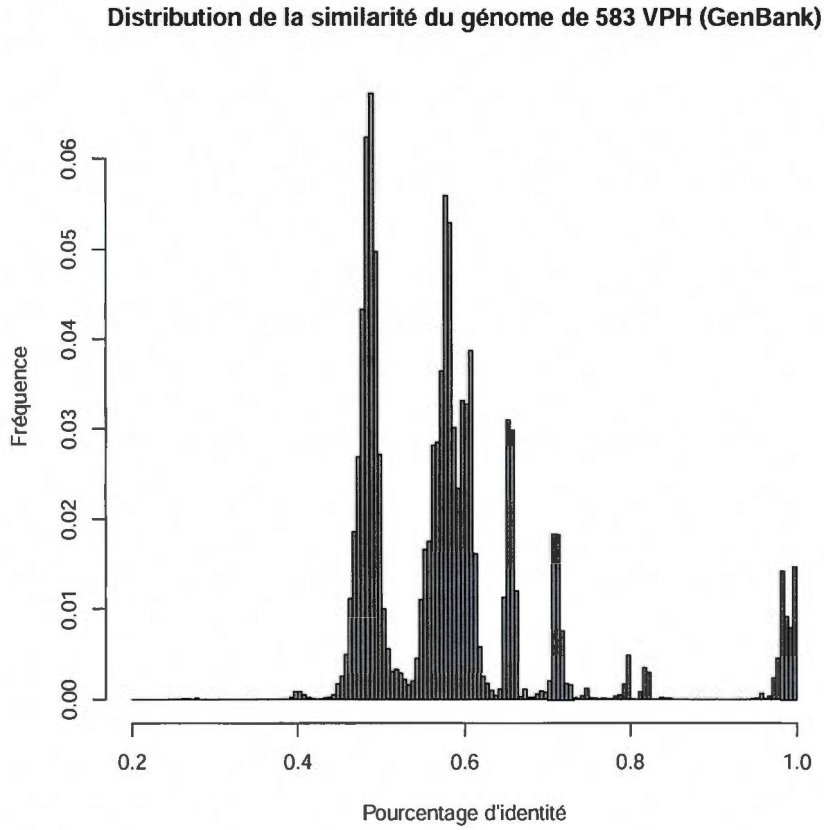
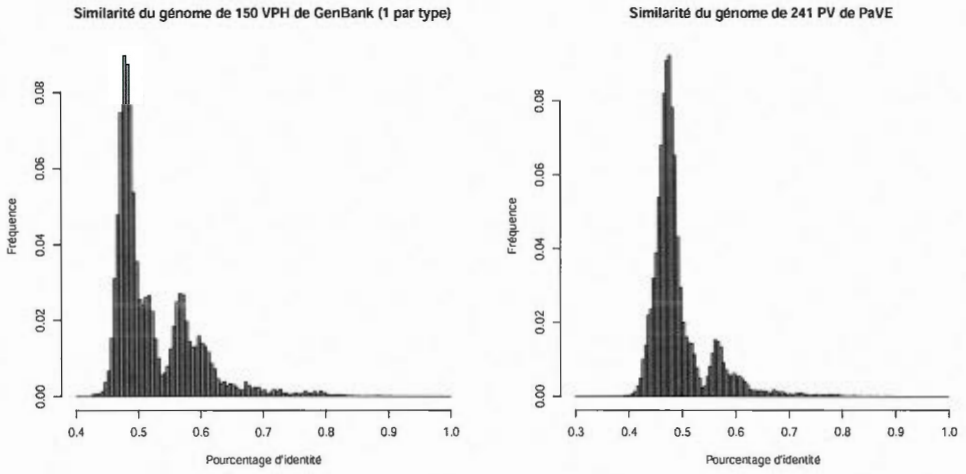


Figure 3.5: Distribution de la similarité du génome complet de 583 VPH de GenBank.

Par contre, toujours avec les données de GenBank, mais avec une seule séquence par type de VPH (figure 3.6a), ou celles du PaVE incluant tous les types de PV (figure 3.6b) les courbes sont semblables avec un minimum local autour de 53%.

3.1.4 Discussion

À propos des divergences de distribution au niveau du gène L1, après vérification, elles découlent d'un choix effectué par le PaVE, qui considère ce codon d'initiation comme le plus probable [Buck, Day et Trus, 2013]. Par contre, une autre étude, quoique moins récente, suggère que la capacité d'un VPH de produire les deux versions de la protéine pourrait être associée avec le niveau de dangerosité du virus [Webb, Cox et Edwards, 2005]. Quoiqu'il en soit, il n'est peut-être pas souhaitable qu'une interprétation de cette nature ait une influence sur la classification.



(a) 150 génomes de VPH de GenBank.

(b) 241 génomes de PV de PaVE.

Figure 3.6: Comparaison de la similarité du génome complet des VPH de GenBank avec un représentant par type et des génomes complets des PV de PaVE.

D'autant plus que le choix de ce codon d'initiation ne semble pas être très répandu chez les autres chercheurs, puisque les annotations du gène L1 des séquences de GenBank utilisent plutôt le premier codon d'initiation.

3.2 Vérification de la caractéristique de proximité des séquences

Cette caractéristique est la suivante : deux séquences qui ont une similarité maximale au niveau du gène L1 dépassant les seuils inter-classes sont toujours dans la même classe.

3.2.1 Algorithme

L'algorithme 4, implémenté par le script pourcentage_arbre.py (A.8), vérifie les pourcentages d'identité intra-classes. Il parcourt les feuilles de l'arbre, et pour chacune, vérifie qu'elle a le même parent le plus bas («lowest commun ancestor») que la feuille la plus similaire, tout en tenant compte des seuils de similarité.

3.2.2 Résultats

En appliquant l'algorithme 4 aux 548 séquences du gène L1, le résultat montre que les séquences NC_001356, V01116 et NC_004500 ne respectent pas la proximité (tableau 3.1) :

```
T = Arbre(arbre.nw)

POUR chaque f ∈ ℓ(T)
    x = feuille la plus similaire à f
    SI T.parent(f) ≠ T.parent(x) ALORS
        a = T.lca(f, x)
        SI similarité(f,x) est hors des seuils pour a.hauteur() ALORS
            signaler non conformité de f
        FIN SI
    FIN SI
FIN POUR
```

Algorithme 4: Vérification de la proximité.

- Les séquences NC_001356 et V01116 sont identiques au niveau du gène L1, mais pas au même endroit dans l'arbre. Il s'agit là probablement d'une imprécision au niveau de la taxonomie. Pour la suite, la séquence NC_001356 a été replacée sous le taxon HPV1a.
- Les séquences NC_004500 et FN677755 appartiennent à des espèces différentes (beta4 et beta5) mais avec une similarité de 71%, elles dépassent légèrement le seuil de similarité entre des espèces différentes à l'intérieur d'un genre (60 à 70%).

Séquence	Séquence la plus similaire	% Similarité	Ancêtre commun
NC_001356(Mu1)	V01116(HPV1a)	100%	Mu1
V01116(HPV1a)	NC_001356(Mu1)	100%	Mu1
NC_004500(HPV92)	FN677755(HPV150)	71%	Beta

Tableau 3.1: Résultats de la vérification de la caractéristique de proximité du gène L1 de 548 VPH.

Sans tenir compte des séquences NC_004500 et FN677755, généralement donc, étant donné deux séquences qui ont une similarité maximale, elles sont dans la même classe.

3.3 Vérification de la caractéristique de distribution des pourcentages d'identité intra et inter-classes

Les pourcentages de similarité définissant les classes sont maintenant considérés comme des lignes directrices plutôt que des absolus. Il est quand même à-propos de vérifier que : les distributions de similarité du gène L1 des séquences à l'intérieur d'une même espèce, des séquences d'espèces

différentes à l'intérieur d'un même genre, et des séquences appartenant à des genres différents sont «semblables» à celles de la figure 2.5, avec des minimums locaux «autour» de 60% et 70% et de «légers» chevauchements autour de ceux-ci.

3.3.1 Algorithmes

L'algorithme 5, implémenté par le script pourcentage_arbre.py (A.8), vérifie les pourcentages d'identité intra-classes. Pour chaque nœud interne de l'arbre, il est vérifié que la similarité de chaque feuille avec toutes les autres feuilles à l'intérieur du même nœud respecte le seuil intra-classes, compte tenu de la hauteur du nœud dans l'arbre.

L'algorithme 6, implémenté par le même script pourcentage_arbre.py (A.8), vérifie les pourcentages d'identité inter-classes. Pour chaque nœud interne de l'arbre, il est vérifié que la similarité de chaque feuille avec toutes les autres feuilles dans les autres nœuds de même hauteur respecte le seuil inter-classes, compte tenu de la hauteur du nœud dans l'arbre.

```
similarité(x,y) : la similarité entre x et y
T = Arbre(arbre.nw)

POUR chaque N ∈ T.parcourir()
    SI ¬ N.estFeuille()
        POUR chaque f ∈ ℓ(N)
            reste = ℓ(N) \ f
            POUR chaque x ∈ reste
                SI similarité(f, x) < seuil intra-classes
                                        pour N.hauteur()
                    signaler la non conformité intra-classes de f et x
                FIN SI
            FIN POUR
        FIN POUR
    FIN SI
FIN POUR
```

Algorithme 5: Vérification des pourcentages d'identité *intra*-classes.

3.3.2 Résultats

En appliquant les algorithmes 5 et 6 aux 548 séquences du gène L1, un très grand nombre de résultats est obtenu, et chacun ne peut donc pas être présenté. Un sommaire des pourcentages **intra**-classes apparaît au tableau 3.2 et celui des pourcentages **inter**-classes au tableau 3.3. Il est visible que, pour les pourcentages de similarité intra-classes, les seuils minima sont dépassés dans

```
similarité(x,y) : la similarité entre x et y
T = Arbre(arbre.nw)

POUR chaque N ∈ T.parcourir()
    SI ¬ N.estFeuille()
        POUR chaque f ∈ ℓ(N)
            inter = ∅
            POUR chaque S ∈ (N).soeurs()
                inter = inter ∪ S
            FIN POUR
            POUR chaque x ∈ inter
                SI similarité(f, x) ne respecte pas les seuils
                                    inter-classes pour N.hauteur()
                    signaler la non conformité inter-classes de f et x
                FIN SI
            FIN POUR
        FIN POUR
    FIN SI
FIN POUR
```

Algorithme 6: Vérification des pourcentages d'identité *inter*-classes.

les trois classes, mais seulement pour un nombre assez petit ($< 0,07$) de paires de séquences. Pour la similarité inter-classes, 8% des paires de séquences au niveau des genres dépassent la similarité maximale de 60%, 10% des paires sont surtout inférieures au minimum de 60% au niveau des espèces et 14% sont inférieures au minimum pour les types.

Un histogramme de la distribution des similarités intra-espèces, inter-espèces et inter-genres des 548 VPH est présenté à la figure 3.7. Le graphe a une allure semblable à celui de la figure 2.5 :

- sans tenir compte des pourcentages d'identité de 95 à 100% qui reflètent la présence de séquences très similaires,
- sans tenir compte de la hauteur des courbes car elles ne sont pas normalisées dans la figure 3.7, et il y a plus de comparaisons inter-genres, que inter-espèces, que intra-espèces.

classe	seuils	min	nb	comp	%
genre	,60 – 1,00	,48	5909	105254	,06
espèce	,70 – 1,00	,57	2459	34452	,07
type	,90 – 1,00	,88	36	8716	,00

Tableau 3.2: Pourcentages **intra**-classes de la similarité du gène L1 de 548 VPH. Pour chaque type de classe, la colonne *seuils* donne l'intervalle des pourcentages de similarité intra-classes, la colonne *min* contient la similarité minimale observée, la colonne *nb* le nombre de fois où la similarité entre une paire de séquences est inférieure au seuil minimum, la colonne *comp* le nombre de comparaisons effectuées, et la colonne % le ratio $nb/comp$.

classe	seuils	min	nb1	max	nb2	comp	%
genre	,00–,60	,43	0	,64	7520	90344	,08
espèce	,60–,70	,48	11804	,72	2272	141604	,10
type	,71–,89	,57	7414	,88	0	51472	,14

Tableau 3.3: Pourcentages **inter**-classes de la similarité du gène L1 de 548 VPH. Pour chaque type de classe, la colonne *seuils* donne l'intervalle des pourcentages de similarité inter-classes, la colonne *min* contient la similarité minimale observée, la colonne $nb1$ le nombre de fois où la similarité entre une paire de séquences est inférieure au seuil minimum, la colonne *max* la valeur maximale observée, la colonne $nb2$ le nombre de fois où la similarité entre une paire de séquences est supérieure au seuil maximum, la colonne *comp* le nombre de comparaisons effectuées, et la colonne % le ratio $(nb1 + nb2)/comp$.

Les chevauchements sont toutefois encore plus étendus, surtout au niveau inter-genres et inter-espèces, qui s'étendent jusqu'à un peu moins de 50% de similarité, ce qui est consistant avec les résultats du tableau 3.3.

D'un autre côté, ici aussi, les effets des 37 types dont les gènes L1 sont plus courts sont visibles, tel que montré à la figure 3.8. Le chevauchement entre les classes est plus étendu et pourrait bien expliquer les chevauchements de la figure précédente.

En effet, en excluant les 37 types dont les gènes sont différents, il reste 113 types (276 séquences) dans GenBank, et la figure 3.9 montre les mêmes frontières et les mêmes chevauchements que la figure 2.5.

Finalement, toujours en ce qui concerne le gène L1, la figure 3.10 montre qu'il est possible de reproduire avec les données du PaVE la même distribution que la figure 2.5 :

- la figure 3.10a montre les 143 gènes L1 du PaVE, donc incluant les gènes plus courts.
- la figure 3.10b montre l'ensemble des 241 PV de références.

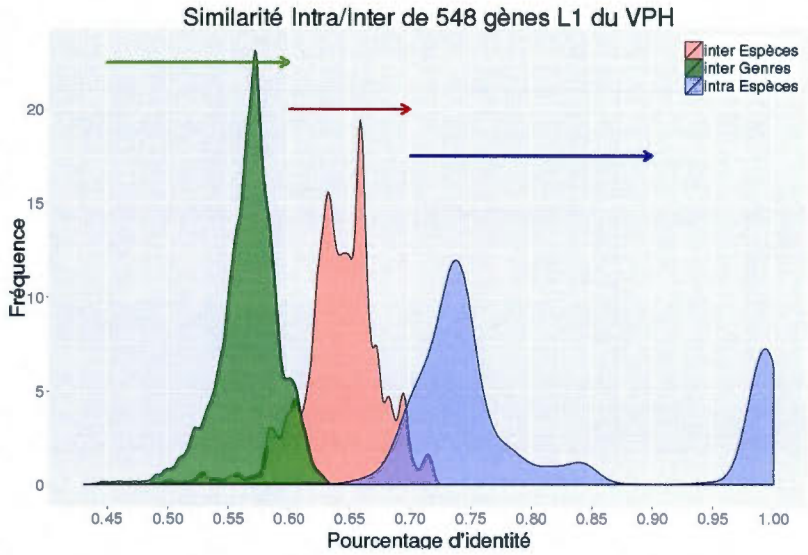


Figure 3.7: Histogramme de la fréquence de similarité intra-espèces, inter-espèces et inter-genres de 548 gènes L1 VPH.

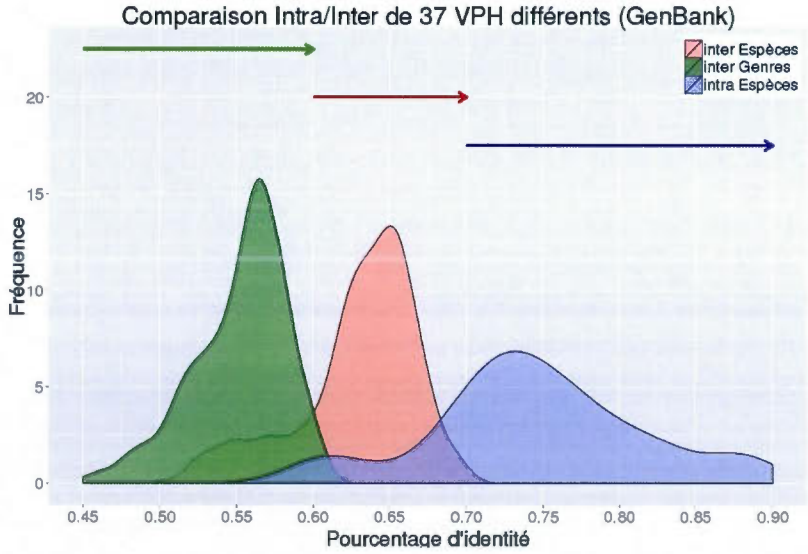


Figure 3.8: Histogramme de la fréquence de similarité intra-espèces, inter-espèces et inter-genres de 37 gènes L1 VPH de GenBank qui n'ont pas la même annotation dans GenBank que dans le PaVE.

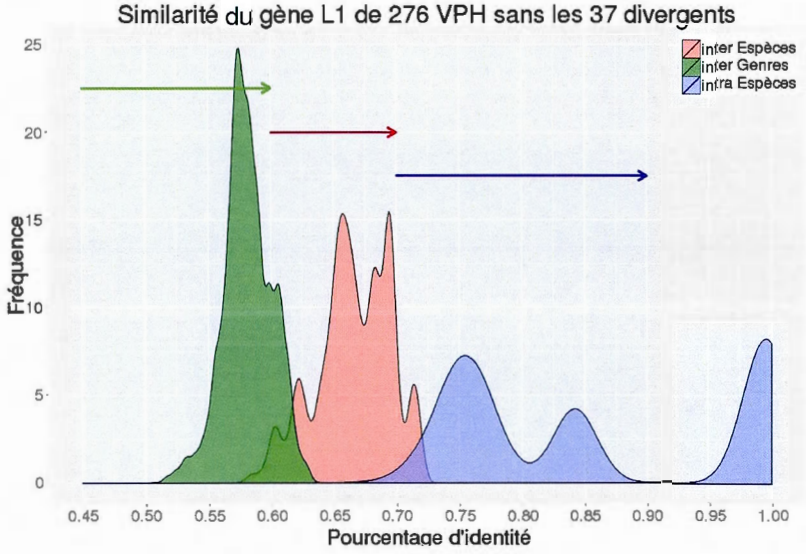
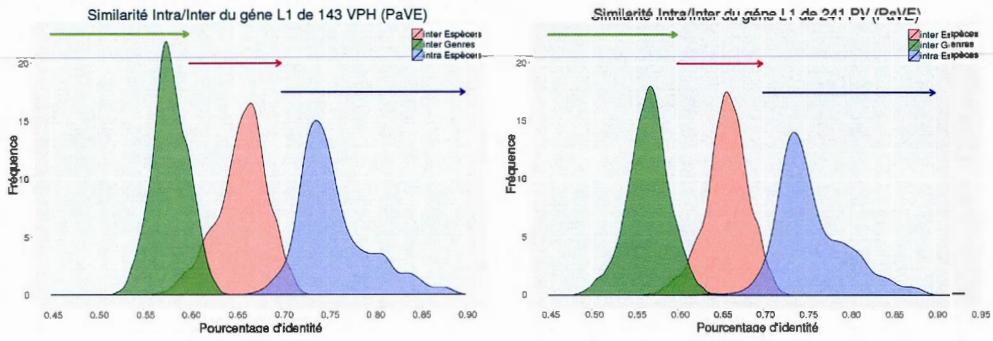


Figure 3.9: Histogramme de la fréquence de similarité intra-espèces, inter-espèces et inter-genres de 276 gènes L1 du VPH de GenBank, sans les types dont le gène est différent dans le PaVE.



(a) 143 gènes L1 de VPH de PaVE.

(b) 241 gènes L1 de PV de PaVE.

Figure 3.10: Histogrammes de la fréquence de similarité intra-espèces, inter-espèces et inter-genres des PV de référence.

Pour le génome complet, la figure 3.11 montre que la séparation entre les groupes est plus nette, surtout au niveau inter-espèces et inter-genres où le chevauchement est pratiquement absent. Les résultats sont sensiblement les mêmes en choisissant un seul représentant par type dans GenBank (figure 3.12a), les 143 VPH de référence de PaVE (figure 3.12b) ou l'ensemble des PV de PaVE (figure 3.12c).

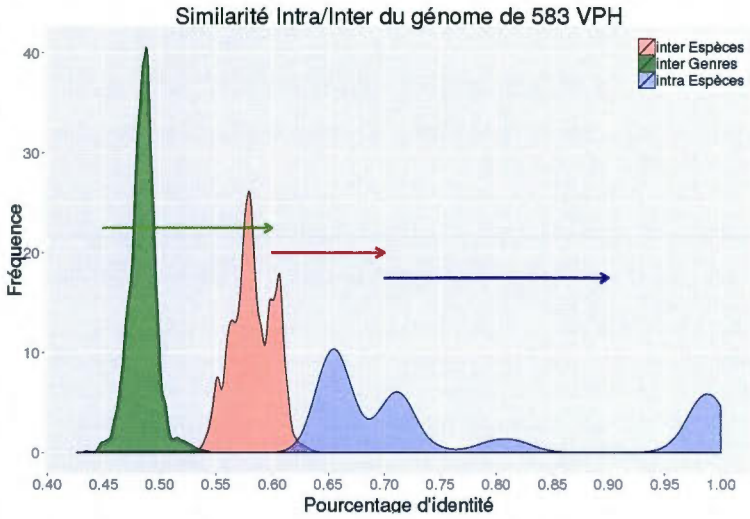
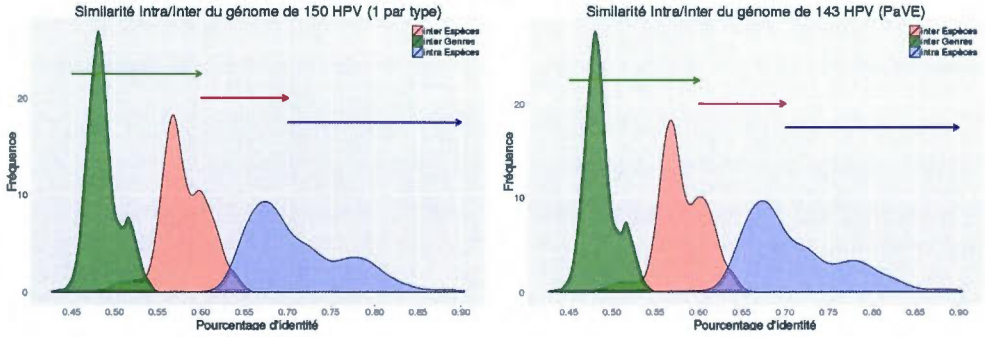
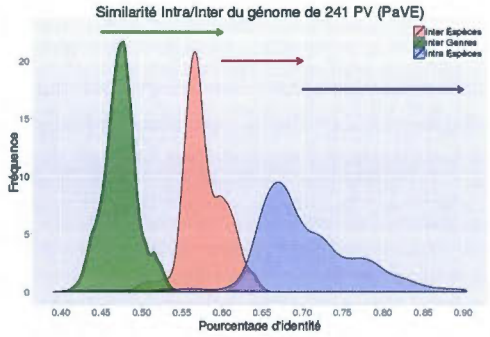


Figure 3.11: Histogramme de la fréquence de similarité intra-espèces, inter-espèces et inter-genres de 583 génomes complets de VPH.



(a) 150 VPH de GenBank (1 par type). (b) 143 VPH de référence (PaVE).



(c) 241 PV de référence (PaVE).

Figure 3.12: Distribution de la similarité des génomes selon différents sous-ensembles.

3.3.3 Discussion

Il existe des divergences relativement importantes par rapport au respect des seuils de pourcentage de similarité, et ce fait a déjà été relevé [Bernard et al., 2010]. Cependant, dû aux annotations différentes du gène L1, les divergences sont amplifiées. Par contre, la présence de séquences très similaires à un moindre effet. Aussi, en utilisant le génome complet, les chevauchements, bien que délimités par des seuils différents, sont beaucoup moins importants, ce qui indique qu'il serait peut-être préférable d'utiliser le génome complet pour ce type de comparaisons.

Également, concernant la mise en garde de la section 2.4.1.1 sur la méthodologie du calcul de la matrice de similarité, il est visible qu'il ne semble pas y avoir d'effets notables sur les résultats par la figure 3.10. Les deux figures ont été produites avec les données du PaVE, fort probablement les même que celles utilisées par Bernard et al, mais avec l'algorithme 3. Et elles sont tout à fait comparables avec la figure 2.5 de Bernard et al.

La figure 3.10 montre aussi qu'il y a pas ou peu d'impact en ne considérant que le sous-ensemble de PV humains plutôt que tous les PV, au niveau de la distribution des pourcentages de similarité.

3.4 Vérification de la caractéristique phylogénique

Depuis récemment, surtout à cause du chevauchement des frontières entre les classes, il est recommandé que la classification tienne compte également de la position topologique dans l'arbre phylogénique. Il s'agit donc vérifier que : la classification est congruente avec la phylogénie (présumée) du gène L1 des VPH.

Pour ce faire, il s'agit de :

- reconstruire l'arbre phylogénique,
- et le comparer avec l'arbre taxonomique.

3.4.1 Méthodologie et algorithmes

3.4.1.1 Reconstruction de l'arbre phylogénique

Le procédé pour reconstruire l'arbre est le suivant :

- Génération de l'alignement multiple de toutes les séquences :
  - Les séquences de nucléotides sont alignées globalement avec le logiciel MUSCLE [Edgar, 2004] :
    - un fichier FASTA contenant l'ensemble des séquences est fourni en entrée.
    - un fichier FASTA contenant l'alignement global est obtenu en sortie.
- Extraction des zones informatives de l'alignement global :
  - Le programme Gblocks élimine les régions mal alignées ou divergentes d'un alignement global permettant ainsi une meilleure inférence phylogénique [Castresana, 2000].
  - L'alignement global est soumis au logiciel Gblocks avec les paramètres permissifs («relax selection») suivants :
    - «Minimum Number Of Sequences For A Flank Position» : (nombre de séquences / 2) + 5, soit un peu plus de la moitié ;
    - «Maximum Number Of Contiguous Nonconserved Positions» : 10 ;
    - «Minimum Length Of A Block» : 5 ;
    - «Allowed Gap Positions» : «With Half» ;
  - Le logiciel Gblocks produit un alignement global de plus petite taille avec des régions phylogénétiquement plus significatives ;
  - Ce résultat est converti au format «phylip» à l'aide du script convert_msa,py, grâce à l'utilisation de bioPython `http://www.biopython.org/`.
- Inférence de l'arbre phylogénique :
  - l'arbre phylogénique est produit par le logiciel PhyML [Guindon et Gascuel, 2003] :
    - l'alignement multiple au format «phylip» est fourni en entrée, ainsi que le nombre de réplications de mesure de support interne («boostrap») à effectuer, soit 128 dans ce cas-ci ;
    - Le Bovine Papillomavirus type 1, BPV1 (NC_001522) a été ajouté comme élément de différenciation («outgroup») selon la coutume (par exemple [Burk, Chen et Van Doorslaer, 2009]) ;
    - La version parallèle du logiciel PhyML (phyml-mpi) a été utilisée sur les grappes de calcul «guillimin» (`http://www.hpc.mcgill.ca/`) et/ou «colosse» (`http://www.calculquebec.ca/index.php/en/resources/compute-servers/colosse`) de Calcul Canada, en utilisant 32 processeurs, pour réduire la durée des calculs ;
    - Le script bash generer-jobs-phyml-gb.sh (A.9) a été utilisé pour générer les travaux à soumettre à l'ordonnanceur MOAB (`http://www.adaptivecomputing.com/products/`

`hpc-products/moab-hpc-basic-edition/`);

- Le logiciel PhyML produit un fichier contenant l'arbre ayant le maximum de vraisemblance dans un fichier au format Newick.

3.4.1.2 Comparaison de l'arbre phylogénique et taxonomique

Une méthode très connue de comparaison d'arbres est la *distance de Robinson et Foulds* [Robinson et Foulds, 1981]. Elle ne s'applique cependant qu'à des arbres binaires, ce qui n'est pas le cas de l'arbre taxonomique. Pour comparer les arbres phylogéniques et taxonomiques, l'algorithme 7, implémenté par le script comparer_arbre.py (A.10) qui utilise la classe Arbre (A.11) et sa méthode Arbre.comparer(), est donc plutôt utilisé. En parcourant chaque nœud interne du deuxième arbre en post-ordre (gauche, droite, parent), si l'ancêtre le plus bas des mêmes feuilles dans le premier arbre a les mêmes feuilles que le nœud du deuxième arbre, ces sous-arbres sont éliminés. L'algorithme reprend alors au tout début, tant qu'un sous-arbre peut être éliminé. L'algorithme termine lorsqu'il n'y a plus de changement, ce qui est le cas également lorsque tous les sous-arbres ont été éliminés. Dans ce dernier cas les arbres sont considérés équivalents.

Pour illustrer l'algorithme, la figure 3.13 montre un sous-ensemble de l'arbre taxonomique et de l'arbre phylogénique reconstruit. Le nœud ancêtre le plus bas des cinq feuilles colorées de l'arbre phylogénique dans l'arbre taxonomique est HPV2. Ce nœud a six feuilles (la séquence X55964 est en plus) et les arbres ne seraient donc pas considérés équivalents.

En réalité, dans ce cas précis, le gène L1 n'est pas annoté pour la séquence X55964 et cette feuille serait éliminée au début du programme de comparaison. Ces deux sous-arbres seraient alors considérés équivalents, éliminés de leurs arbres respectifs, et l'algorithme reprendrait la comparaisons avec les deux arbres ainsi élagués.

3.4.2 Résultats

L'arbre phylogénique des gènes L1 des 560 VPH a été reconstruit et le résultat comparé à l'arbre taxonomique. A priori, les arbres sont considérés équivalents. Cependant, les valeurs de support interne de l'arbre phylogénique sont plutôt mauvaises en général, surtout lorsque les longueurs de branches sont près de zéro. Avec le grand nombre de séquences, incluant des séquences de gènes L1 identiques ou très similaires, l'algorithme de reconstruction doit faire des choix de bifurcations qui ne reposent pas sur un signal clair.

```
T1 = Arbre(arbre1.nw)
T2 = Arbre(arbre2.nw)
changement = VRAI

TANT QUE changement
    changement = FAUX
    POUR chaque t2 ∈ T2.parcourir(post-ordre)
        SI ¬ changement
            SI ¬ t2.estFeuille()
                f2 = ℓ(T2)
                A1 = T1.lca(f2)
                f1 = ℓ(A1)
                SI f1 ≠ f2
                    RETOURNER FAUX
                FIN SI
                T1.enleverFeuilles(f1)
                T2.enleverFeuilles(f2)
                changement = VRAI
            FIN SI
        FIN SI
    FIN POUR
FIN TANT QUE
RETOURNER VRAI
```

Algorithme 7: Comparaison d'arbres.

Il est possible de ne pas en tenir compte en éliminant les bifurcations dont le support interne est inférieur à 85% par exemple. Éliminer une telle bifurcation à un nœud n consiste à enlever n et rattacher les enfants de n au parent de n.

Après une telle transformation, les arbres sont considérés équivalents par l'algorithme, mais pas au-delà de cette valeur de support (86% et plus). La figure 3.14 montre le résiduel de l'arbre en enlevant les bifurcations qui n'ont pas un support interne d'au moins 86%, après que le programme eut éliminé les sous-arbres équivalents.

La divergence est causée par le sous-arbre gamma11 qui compte quatre séquences dans l'arbre taxonomique, mais cinq séquences dans le sous-arbre phylogénique correspondant au plus proche ancêtre des quatre séquences. La séquence supplémentaire est le «NC_017995» qui est sous gamma16 dans l'arbre taxonomique.

En augmentant la valeur de support interne minimum à 99 %, les divergences sont plus grandes entre les arbres. À 100%, à peu près aucun sous-arbre commun ne peut être éliminé, mais les

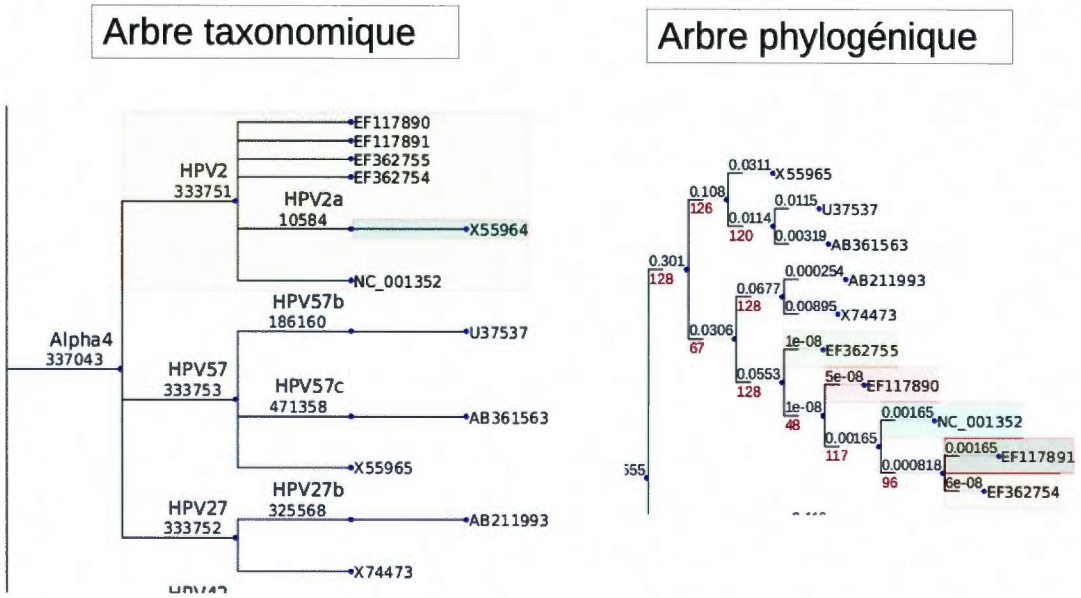


Figure 3.13: Comparaison des arbres phylogéniques et taxonomiques.

séquences qui causent la divergence sont difficilement identifiables.

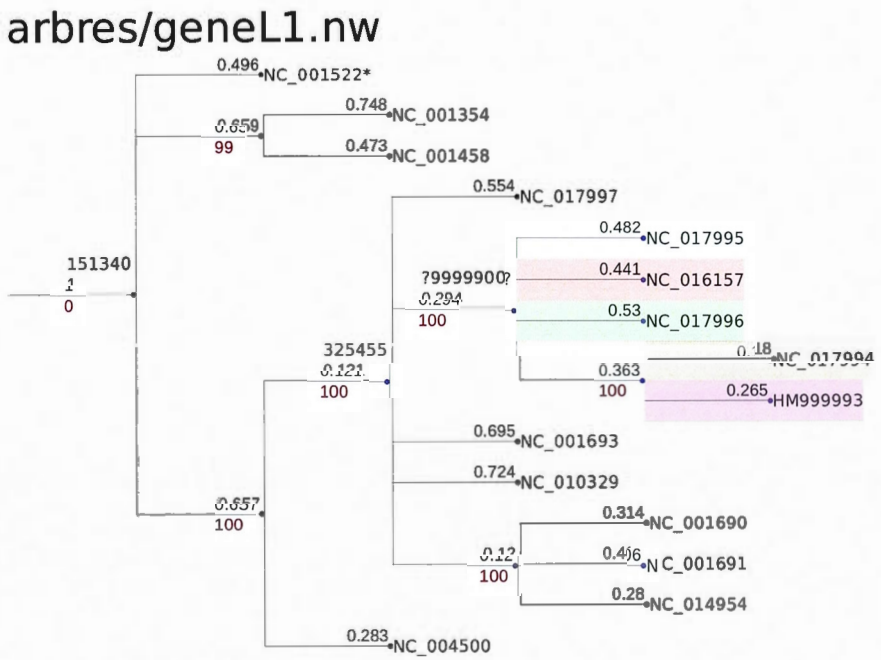
Par contre, il est intéressant de noter que si le génome complet des 560 VPH est utilisé, et que les mêmes algorithmes sont appliqués, les arbres taxonomique et phylogénique sont toujours considérés équivalents. Et ceci peu importe que toutes les bifurcations soient conservées, ou que les bifurcations qui n'ont pas un support interne de 100% soient éliminées.

3.4.3 Discussion

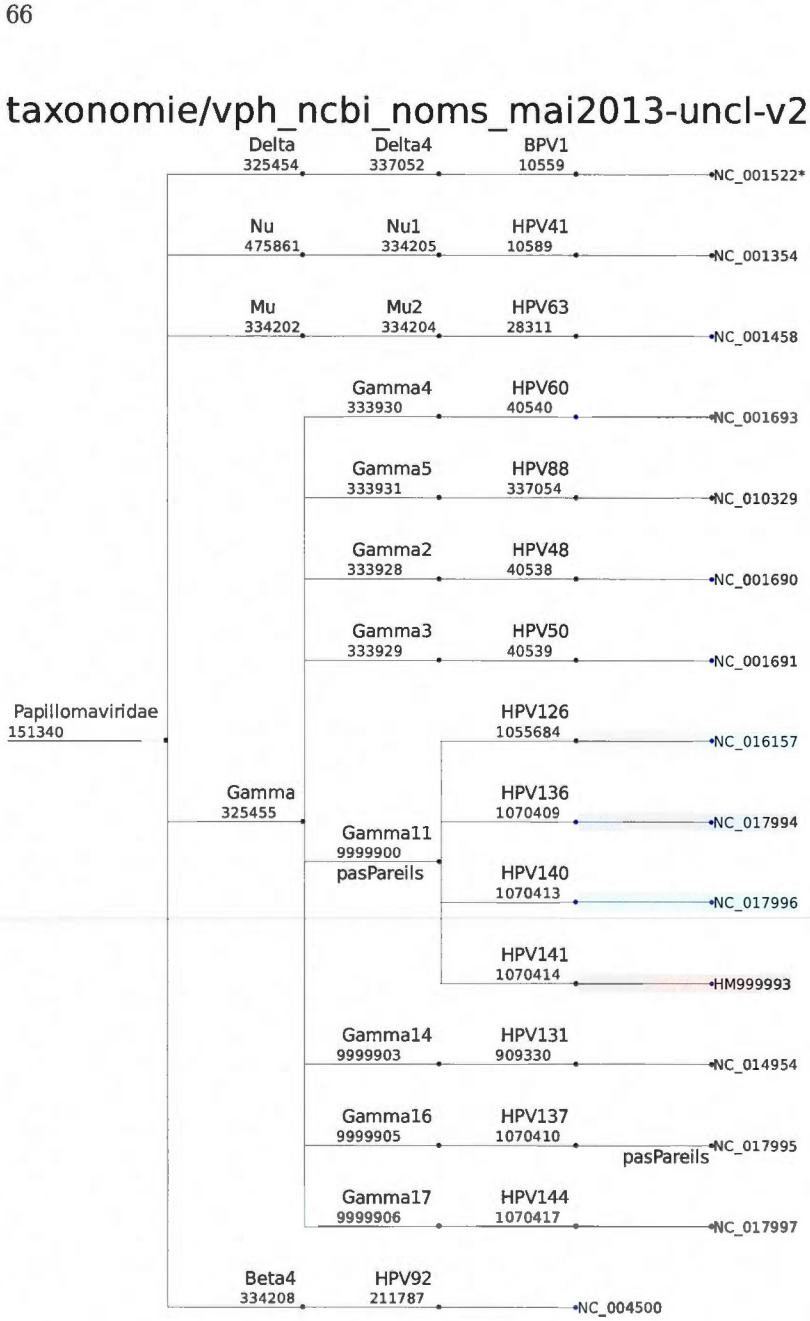
Selon l'inférence phylogénique du gène L1, la séquence NC_017995 devrait se trouver sous gamma11 plutôt que de constituer une nouvelle espèce directement sous le genre gamma. Mais l'arbre du génome complet ne montre pas cette différence. La classification est congruente avec la phylogénie (présumée) du génome complet des VPH, mais diverge légèrement au niveau du gène L1.

Il faut cependant noter que la séquence NC_017995 est sous «unclassified papillomavirus» dans GenBank. L'étiquette gamma16 lui est attribuée par le PaVE, mais n'a as encore été entérinée par le ICTV. Aussi, au niveau du gène L1, la distance entre NC_017995 et les autres séquences sous gamma11 varie de 0.359 à 0.370, donc une similarité variant de 63% à 64.1%. Elle n'atteint pas le seuil de similarité de 70% fixé pour une même espèce, ce qui pourrait bien expliquer

pourquoi une nouvelle espèce a été créée.



(a) Une partie de l'arbre phylogénique du gène L1.



(b) Une partie de l'arbre taxonomique du gène L1.

Figure 3.14: Résultat de la comparaison des arbres phylogénique et taxonomique du gène L1. Les sous-arbres considérés équivalents ont été éliminés pour ne laisser que ceux divergents.

3.5 Vérification de la caractéristique de déterminisme

La classification comportant des séquences ayant la même similarité avec deux classes différentes serait problématique, ce qui peut s'illustrer par un exemple. Soient les séquences $a1$=ACGT et $a2$=GTAC, et un seuil d'appartenance à une classe fixé à 50% ou plus de similarité :

- $a1$=ACGT / $a2$=GTAC les séquences $a1$ et $a2$ ont une similarité de 0% (1/5), inférieure au seuil de 50%. $a1$ et $a2$ seraient donc dans des classes distinctes.
- la séquence $s1$=GTGT : $a1$=ACGT / $s1$=GTGT / $a2$=GTAC a une similarité de 50% (2/4) avec $a1$ et de 50% (2/4) avec $a2$, et se trouverait à briser la caractéristique de déterminisme en étant placée dans une des deux classes.

Avec les données de GenBank et celles de PaVE, il est possible d'identifier si de tels cas existent.

3.5.1 Algorithme

L'algorithme 8, implémenté et incorporé dans le script dissim_arbre.py (A.12), est utilisé pour déterminer si des cas de non déterminisme sont présents. Pour chaque feuille de l'arbre (nœud terminal), il vérifie que les feuilles les plus similaires et à égale distance sont dans le même sous-arbre, ou dans un autre sous-arbre approprié en respectant les seuils de similarité.

3.5.2 Résultats

L'algorithme a été appliqué à l'arbre taxonomique de GenBank et du PaVE et montre qu'une telle condition n'est pas présente. C'est-à-dire qu'avec les classifications actuelles, s'il y a plusieurs séquences à distance minimum d'une autre séquence, elles se trouvent soit dans la même classe, soit dans des classes taxonomiques différentes respectant les seuils de similarité.

L'algorithme souligne tout de même le cas de la séquence NC_004500 (de type HPV92 et d'espèce Beta4) qui est à distance de 0.2866 de la séquence FN677755 (de type HPV150 et d'espèce Beta5) alors que la distance devrait être de 0.3 pour les espèces différentes.

L'analyse des résultats montre aussi un cas intéressant, soulignant la possibilité de séquences à distance égales :

- HPH109REF (gamma7) est à égale distance (0.35153797865662273) de HPV65REF et HPV4REF

```
T = Arbre(arbre.nw)

POUR chaque f ∈ ℓ(T)
    v = feuille la plus proche de f
    voisins = {feuilles les plus proches de f} \ v
    d = dissimilarité(f, v)
    POUR chaque x ∈ voisins
        a = T.lca(f, x)
        h = a.hauteur()
        SI (h == 2 ET d < 0.1) OU
           (h == 1 ET d < 0.3) OU
           (h == 0 ET d < 0.4)
            signaler la condition de non déterminisme
        FIN SI
    FIN POUR
FIN POUR
```

Algorithme 8: Évaluation du déterminisme.

(toutes deux de gamnma1),

- et la distance entre HPV65REF et HPV4REF est de 0.1628056628056628.

Ceci ne pose tout de même pas d'ambiguïté puisque HPV65REF et HPV4REF font partie de la même espèce et qu'en plus la distance avec HPH109REF est supérieure à 0.3.

Même si aucune démonstration directe de la condition de non déterminisme de la classification n'a pu être faite avec les données actuelles, elle demeure théoriquement possible, et de plus en plus probable avec la croissance du nombre de séquences.

L'algorithme vérifie la condition de non-déterminisme avec l'ensemble des séquences. Il est possible qu'une telle condition existe avec un sous-ensemble (par exemple en enlevant une séquence qui empêche la condition d'égalité de se produire), mais cela n'a pas été vérifié.

3.6 Vérification de la caractéristique de reproductibilité

Il s'agit de vérifier que la classification actuelle des VPH peut être reproduite à partir des séquences génomiques du gène L1 en utilisant les caractéristiques de similarité ou de phylogénie. À la section 3.4, il a été montré que la caractéristique phylogénique est suffisante pour reproduire la classification en utilisant le génome complet. Il reste donc à savoir si c'est possible en utili-

sant les caractéristiques de similarité. À prime abord, cela semble difficile : l'appartenance des séquences à une classe dépend de l'ordre dans lequel elles sont ajoutées, donc de la chronologie. De plus, en regroupant les séquences avec la plus similaire, il est possible d'en venir petit à petit à dévier du seuil de similarité de la classe pour certaines séquences.

Ces deux situations peuvent s'illustrer par un exemple. Soient les séquences précédentes $a1$=ACGT et $a2$=GTAC qui se trouvent dans des classes différentes $c1$ et $c2$ respectivement, dont le seuil de similarité est de 50%, et deux nouvelles séquences à classer $s1$=GTGC et $s2$=GAGA :

- en ajoutant dans l'ordre $s1$ puis $s2$, on obtient pour $s1$:

```
          c1    c2
        ACGT  GTAC
          |   || |
s1=     GTGC  GTGC
```

  et $s1$, avec une similarité de 75% (3/4), est classifiée dans $c2$:

```
c1    c2
GTGC  GTAC
      GTGC
```

- puis en ajoutant $s2$:

```
          c1    c2
        ACGT  GTAC
              GTGC
          |   | |
s2=     GAGA  GAGA
```

  et $s2$, avec une similarité de 50% (2/4), est classifiée dans $c2$:

```
c1    c2
ACGT  GTAC
      GTGC
      GAGA
```

- les deux nouvelles séquences se trouvent dans la même classe $c2$ que $a2$, mais la similarité entre $a2$ et $s2$ n'est plus que de 25% (1/4) :

```
a2=GTAC
  |
s2=GAGA
```

  en-dessous du seuil de 50%, soit une déviation du seuil de similarité.

- si $s2$ est d'abord ajoutée :

```
          c1    c2
        ACGT  GTAC
          |   |
s2=     GAGA  GAGA
```

  pour une similarité de 25% (1/4) inférieure au seuil de 50% de chacune des classes. Il faut donc créer une nouvelle classe pour $s2$, ce qui montre que le processus est non reproductible car dépendant de l'ordre d'insertion des séquences.

3.6.1 Algorithme

Pour déterminer si la classification peut être reproduite en considérant les caractéristiques de similarité comme critère de classification, un classificateur est développé. Il est basé sur l'algorithme 9, implémenté dans le script classerVPH.py (A.13). Avec un arbre constitué de la première séquence classée à la hauteur 4 (les hauteurs 0, 1, 2, 3 et 4 équivalent respectivement à la famille, genre, espèce, type et séquence), les séquences suivantes sont ajoutées une à une,

à partir de la séquence la plus similaire. Au besoin, un nouveau type, espèce ou genre est créé selon les seuils de similarité de la classification.

```
                    PROCÉDURE ajouterNoeud(nom, arbre, nbHaut, nbBas)
                        POUR i = 1 JUSQU'À i == nbHaut
                            arbre = arbre.parent()
                        POUR i = 1 JUSQU'À i == nbBas
                            arbre = arbre.ajouterEnfant()
                        arbre.ajouterEnfant(nom)
                    FIN PROCÉDURE
nbSeq : nombre de séquences à classer
seq[nbSeq] : tableau des séquences à classer
T = Arbre()
ajouterNoeud(seq[1], T, 0, 3)
POUR i = 2 JUSQU'À i == nbSeq
    x = une des séquences les plus similaire de seq[i] dans T
    d = similarité(seq[i], x)
    SI d >= 0.9
        nbSaut = 0
    SINON SI d >= 0.71
        nbSaut = 1
    SINON SI d >= 0.60
        nbSaut = 2
    SINON
        nbSaut = 3
    FIN SI
    noeudV = T.chercher(x)
    ajouterNoeud(seq[i], T, nbSaut + 1, nbSaut)
FIN POUR
```

Algorithme 9: Évaluation de la reproductibilité.

3.6.2 Résultats

À l'aide d'un ensemble de six séquences du PaVE, la non reproductibilité de la classification est démontrée. Le tableau 3.4 énumère les six séquences dans l'ordre où elles ont été ajoutées pour produire les arbres.

La figure 3.15a montre les arbres correspondants. Dans les deux cas :

- HPV11REF et HPV13REF sont groupés dans une même espèce (hauteur du parent = 2),
- HPV32REF, HPV16REF et HPV31REF sont groupés dans une autre espèce (la hauteur du parent est 2) du même genre (la hauteur du parent est 1).

Séquence ajoutée	Séquence la plus proche	Pourcentage de similarité	Position relative
HPV11REF	Sans objet	Sans objet	hauteur 4
HPV13RE	HPV11REF	0.759736	meme espece
HPV32RE	HPV13REF	0.696047	meme genre
HPV16RE	HPV32REF	0.710749	meme espece
HPV31RE	HPV16REF	0.777559	meme espece
HPV1REF	HPV31REF	0.610608	meme genre

(a) Correspond à la figure 3.15a

Séquence ajoutée	Séquence la plus proche	Pourcentage de similarité	Position relative
HPV1REF	Sans objet	Sans objet	hauteur 4
HPV11REF	HPV1REF	0.587291	autre genre
HPV13REF	HPV11REF	0.759736	meme espece
HPV32REF	HPV13REF	0.696047	meme genre
HPV16REF	HPV32REF	0.710749	meme espece
HPV31REF	HPV16REF	0.777559	meme espece

(b) Correspond à la figure 3.15b

Tableau 3.4: Reconstruction de deux arbres avec les mêmes séquences, mais dans un ordre différent. La colonne de gauche énumère les séquences ajoutées dans l'ordre, la suivante la séquence la plus similaire actuellement dans l'arbre, puis le pourcentage de similarité et la dernière colonne indique la position relative où la séquence a été ajoutée.

Par contre, HPV1REF est dans le même genre que les autres séquences dans le cas de la figure 3.15a, mais dans un autre genre dans le cas de la figure 3.15b. Ceci démontre le caractère non reproductible de la classification selon la similarité.

Pour montrer un autre exemple, à partir des 143 VPH du PaVE dans un ordre quelconque (en fait dans l'ordre lexicographique du genre, espèce et type de la séquence), l'arbre obtenu ne contient qu'un seul genre. À partir d'un autre sous-ensemble de 100, 3 genres différents sont créés.

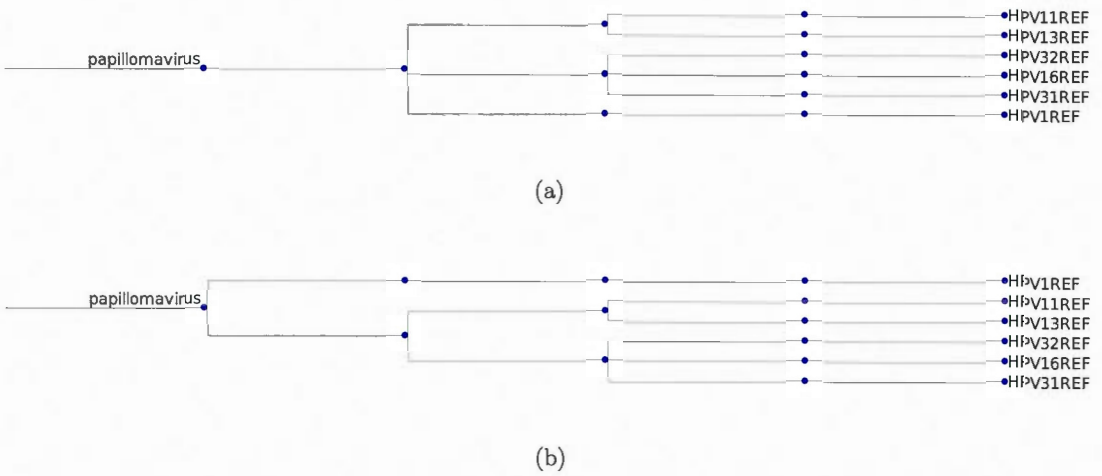


Figure 3.15: Reconstruction de deux arbres taxonomiques de six séquences dans un ordre différent.

3.6.3 Discussion

Ce résultat n'est pas étonnant puisque la mesure de distance utilisée n'est pas absolue, mais relative aux autres séquences. Le classement d'une nouvelle séquence est donc entièrement dépendant des séquences déjà classées, comme en fait foi le tableau 3.4 où la séquence HPV1REF se retrouve dans un nouveau genre à part des autres séquences lorsqu'elle est la première à être classée.

En fait l'algorithme est un peu naïf et ne reflète pas la réalité historique de la construction de la classification, qui ne s'est pas faite en ajoutant chronologiquement une séquence à la suite des autres. Elle a débutée avec quelques séquences et a été raffinée en considérant simultanément un ensemble de séquences [de Villiers, 2013]. La classification actuelle est basée sur une analyse phylogénique datant de 1995 proposant des super-groupes et des sous-groupes [Chan et al., 1995]. Il est fort probable que l'algorithme 9 démarrant avec cet arbre pour ajouter de nouvelles séquences donnerait un meilleur résultat.

La méthodologie pour produire la classification est donc plutôt :

- inférence phylogénique,
- désignation de sous-arbres comme genres et espèces selon les seuils de pourcentages,
- désignation d'un nouveau type si la similarité est inférieure à 90%.

CHAPITRE IV

VALIDATION DE LA CLASSIFICATION DU VPH EN TANT QUE REGROUPEMENT HIÉRARCHIQUE

La classification du VPH est basée sur la similarité des séquences et constitue en fait un regroupement hiérarchique. Une caractéristique importante des regroupements est que les éléments à l'intérieur d'un sous-groupe soient similaires (compacité), et ceux de sous-groupes différents soient distincts (séparation). Il est donc intéressant de s'intéresser à des mesures de compacité et de séparation, typiques dans ce genre de regroupements [Handl, Knowles et Kell, 2005].

Le problème de déviation des seuils de similarité soulevé à la section 3.6 et illustré à la figure 4.1 peut conduire à un chevauchement des frontières entre les sous-groupes, c'est-à-dire qu'une séquence à l'intérieur d'un sous-groupe est autant ou plus similaire à une séquence d'un autre sous-groupe qu'à certaines séquences du même sous-groupe.

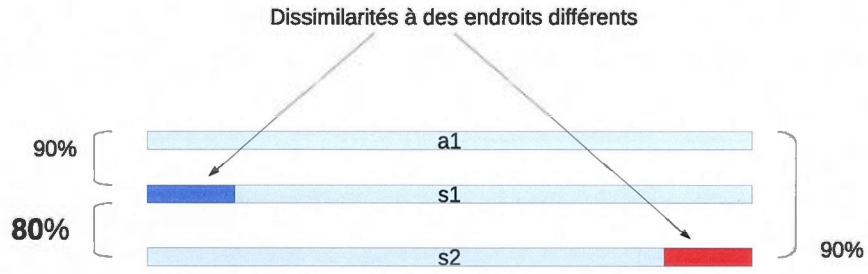


Figure 4.1: Déviation progressive des seuils de similarité. À mesure que des séquences sont ajoutées à une classe, la similarité entre certaines paires de séquences peut diminuer. Ici, $a1$ et $s1$ sont similaires à 90%, de même que $a1$ et $s2$. Par contre $s1$ et $s2$ ne sont similaires qu'à 80%.

4.1 Mise à l'échelle multidimensionnelle de la similarité des séquences du VPH

Pour avoir une idée approximative de la compacité et de la séparation, il est possible de représenter visuellement dans le plan la distance entre les séquences à l'aide d'une mise à l'échelle multidimensionnelle (MDS ou «Classical Multidimensional Scaling» ou encore «Principal Coordinates Analysis» [Gower, 1966]) de la matrice de dissimilarité. C'est d'ailleurs une étape exploratoire qu'il est recommandé de faire avant de catégoriser de nouveaux échantillons biologiques, pour tenter de visualiser les tendances des regroupements [Handl, Knowles et Kell, 2005].

Cette transformation comporte généralement des distorsions et le résultat ne peut pas s'interpréter au pied de la lettre. La distance entre n points peut être représentée exactement dans un espace à $n-1$ dimensions. En réduisant le nombre de dimensions, des erreurs sont introduites dans le cas général, ce qui peut s'illustrer par deux exemples :

- La distance entre deux points est représentée exactement en une dimension, sur une droite. En réduisant le nombre de dimension à zéro, la distorsion est évidente puisque les deux points sont confondus en un seul.
- La distance entre trois points est représentée exactement en deux dimensions, sur le plan. Par exemple, la figure 4.2a représente exactement dans le plan les trois sommets a, b et c d'un triangle isocèle avec une base de longueur 2 et les autres côtés de longueur 10. La matrice de distance entre les trois points est :

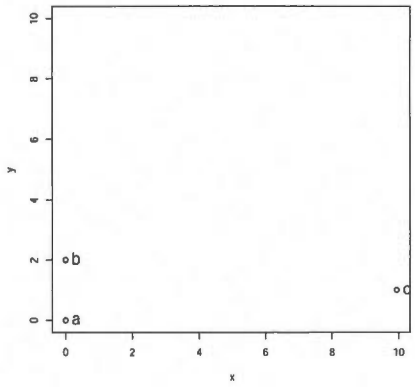
	a	b	c
a	0	2	10
b	2	0	10
c	10	10	0

La mise à l'échelle multidimensionnelle en deux dimensions préserve évidemment les distances, comme on voit à la figure 4.2b. L'échelle (les axes) est cependant souvent modifiée. La mise à l'échelle en une dimension ne préserve plus les distances, comme le montre la figure 4.2c. Les points a et b sont confondus. Il est quand même possible de supposer à partir de cette figure que a et b sont plus près l'un de l'autre que de c.

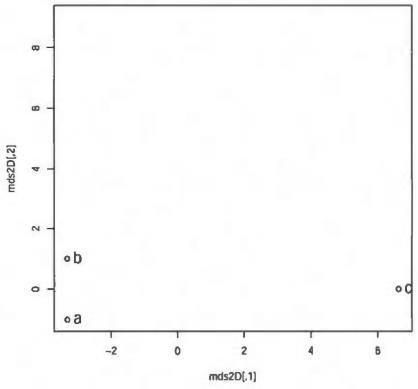
4.1.1 Résultats

La figure 4.3, réalisée en partie avec le logicial *R* [R Development Core Team, 2008], montre cette mise à l'échelle de huit régions génomiques différentes des VPH, soient les gènes L1, L2, E1, E2, E4, E6, E7 et le génome complet. Les figures de la colonne de gauche montrent le MDS de

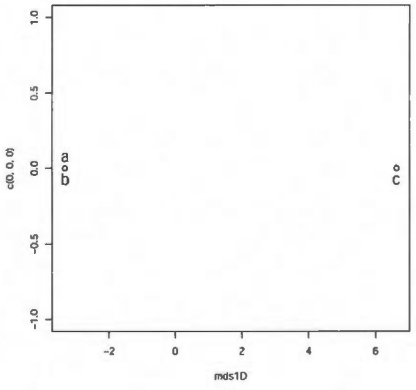
chaque région selon l'ensemble des séquences de GenBank, alors que celles de la colonne de droite illustrent le MDS avec les données du PaVE. Cela permet de visualiser l'effet de l'augmentation du nombre de séquences sur la séparation des classes. Il faut noter que la différence de longueur des 37 gènes L1 pourrait aussi avoir une influence sur la position des points dans les graphiques. L'enveloppe convexe des points de chaque genre est surimposée.



(a) 3 sommets a, b et c d'un triangle isocèle.

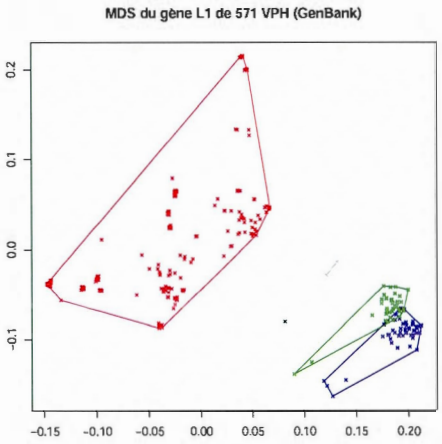


(b) Projection 2D des 3 sommets.

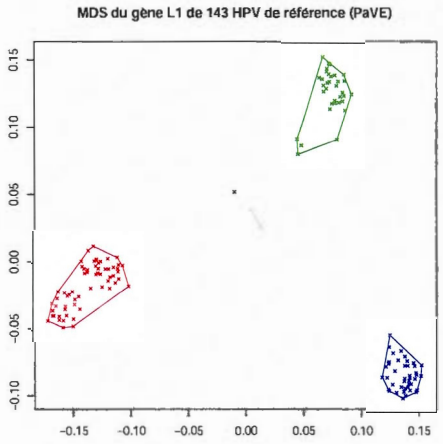


(c) Projection 1D des 3 sommets.

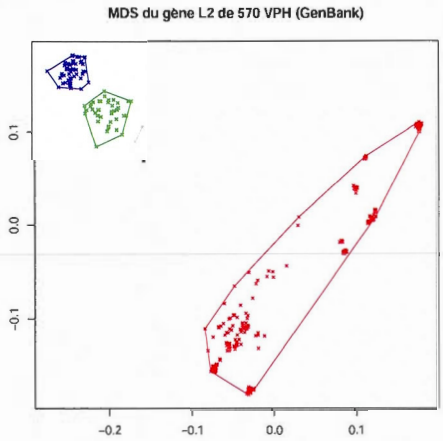
Figure 4.2: Illustration d'une mise à l'échelle multidimensionnelle.


MDS du gène L1 de 571 VPH (GenBank)
0.2
0.1
0.0
-0.1
-0.15
-0.10
-0.05
0.00
0.05
0.10
0.15
0.20

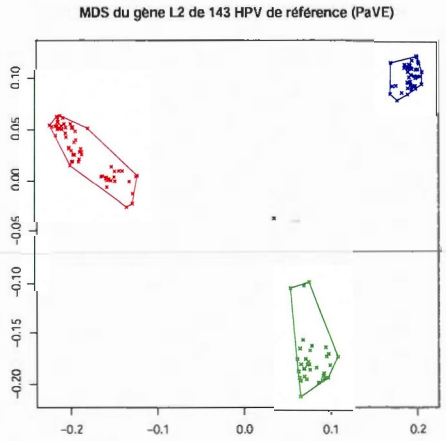
(a)


MDS du gène L1 de 143 HPV de référence (PaVE)
0.15
0.10
0.05
0.00
-0.05
-0.10
-0.15
-0.10
-0.05
0.00
0.05
0.10
0.15

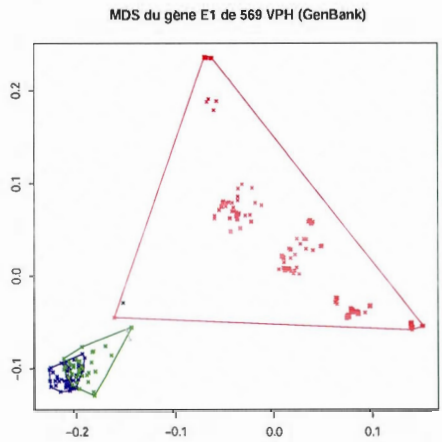
(b)


MDS du gène L2 de 570 VPH (GenBank)
0.1
0.0
-0.1
-0.2
-0.1
0.0
0.1

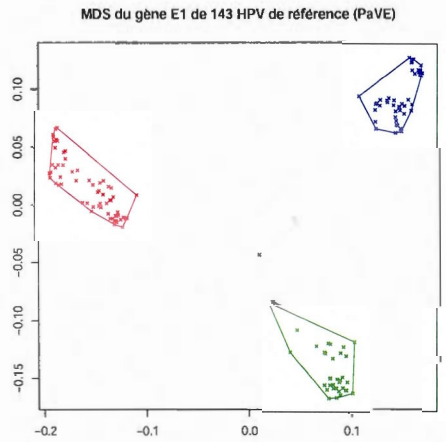
(c)


MDS du gène L2 de 143 HPV de référence (PaVE)
0.10
0.05
0.00
-0.05
-0.10
-0.15
-0.20
-0.2
-0.1
0.0
0.1
0.2

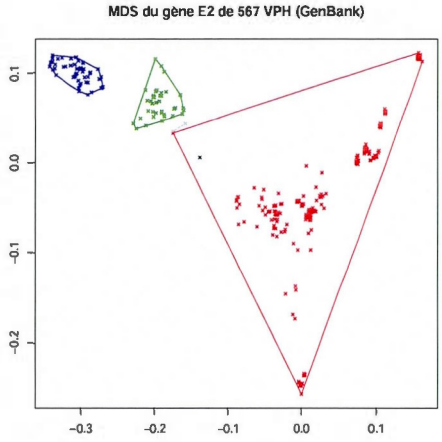
(d)


MDS du gène E1 de 569 VPH (GenBank)
0.2
0.1
0.0
-0.1
-0.2
-0.1
0.0
0.1

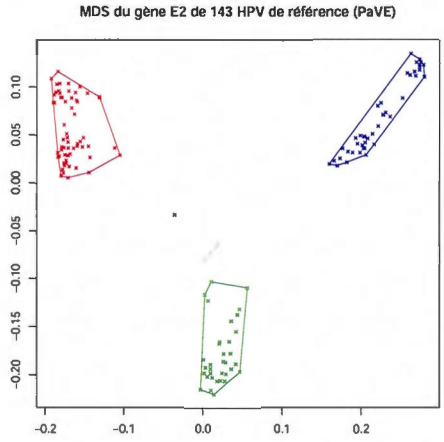
(e)


MDS du gène E1 de 143 HPV de référence (PaVE)
0.10
0.05
0.00
-0.05
-0.10
-0.15
-0.2
-0.1
0.0
0.1

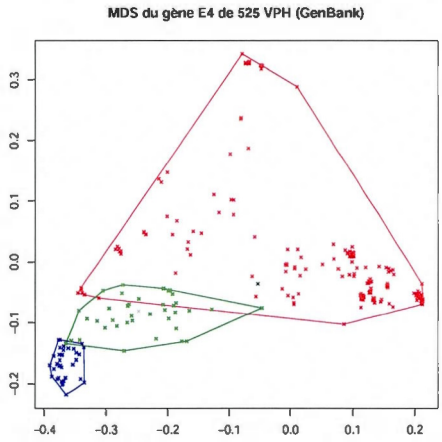
(f)


MDS du gène E2 de 567 VPH (GenBank)

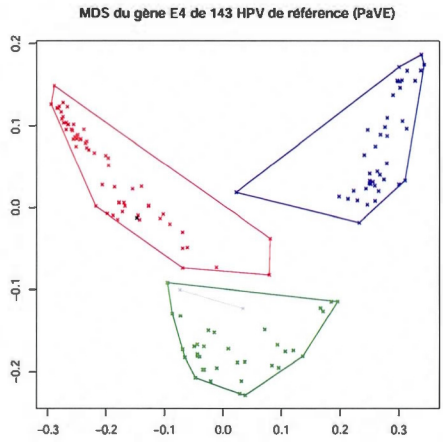
(g)


MDS du gène E2 de 143 HPV de référence (PaVE)

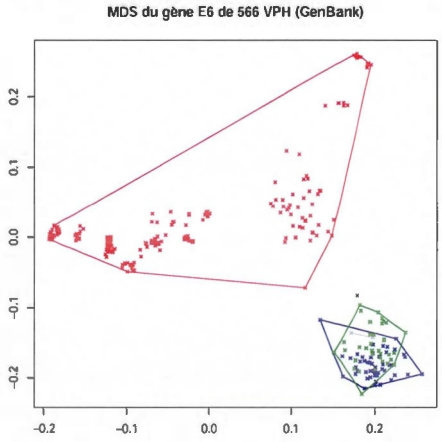
(h)


MDS du gène E4 de 525 VPH (GenBank)

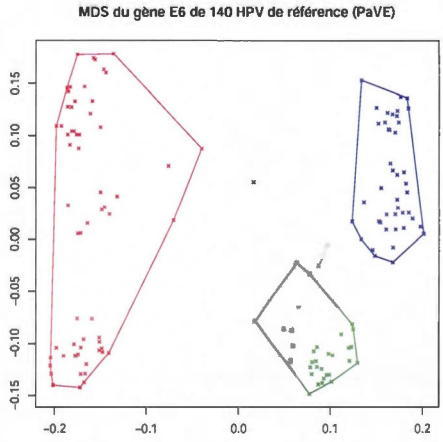
(i)


MDS du gène E4 de 143 HPV de référence (PaVE)

(j)


MDS du gène E6 de 566 VPH (GenBank)

(k)


MDS du gène E6 de 140 HPV de référence (PaVE)

(l)

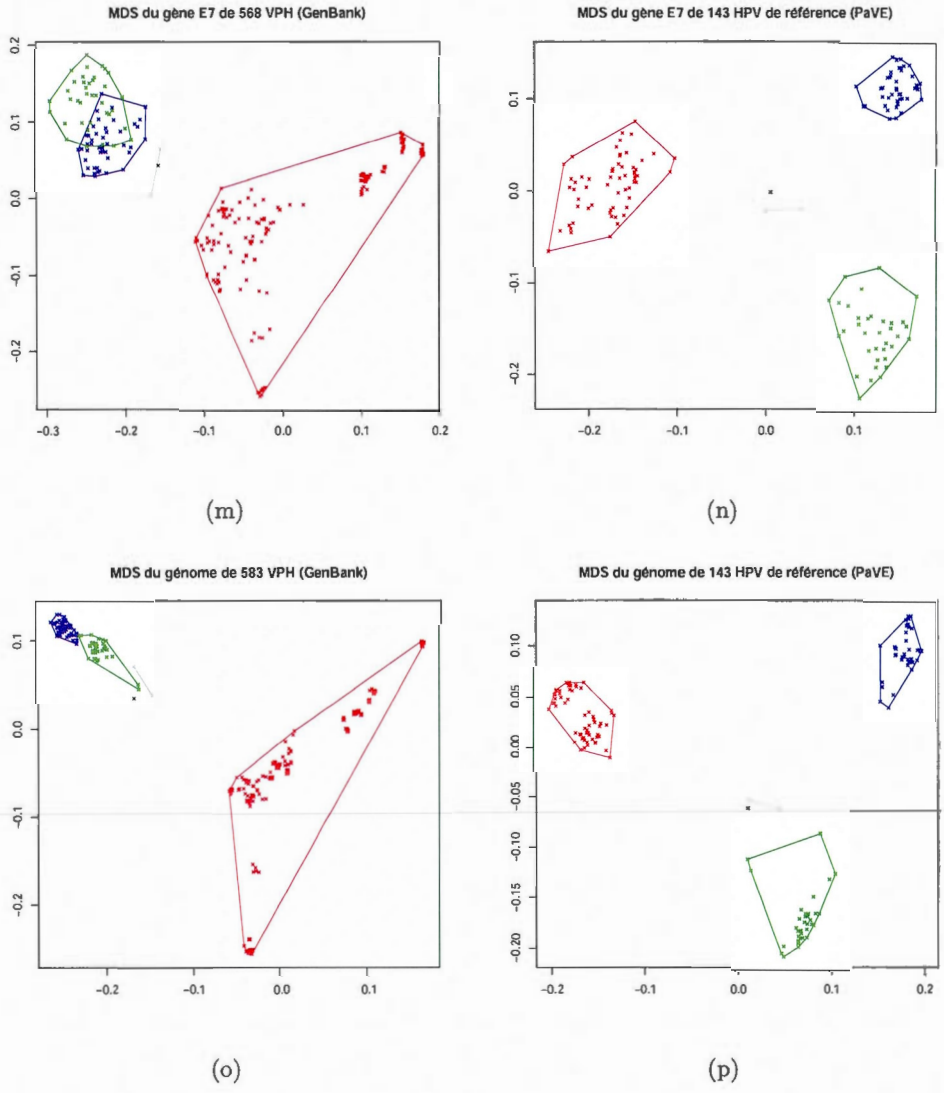


Figure 4.3: Mise à l'échelle multidimensionnelle (MDS) de 8 régions génomiques différentes des VPH, soient les gènes L1, L2, E1, E2, E4, E6, E7 et le génome complet. Les figures de la colonne de gauche (a, c, e, g, i, k, m et o) montrent le MDS de chaque région selon l'ensemble des séquences de GenBank, alors que celles de la colonne de droite illustrent le MDS avec les données de PaVE. L'enveloppe convexe des points de chaque genre est surimposée. Les genres alpha beta, gamma, mu et nu sont respectivement en rouge, bleu, vert, gris et noir.

Comme le genre alpha est relativement bien séparé des autres, la mise à l'échelle multidimensionnelle est reprise pour visualiser la séparation des espèces à l'intérieur de ce genre. La figure 4.4 montre les résultats pour le gène L1 et pour le génome complet, à la fois pour les données

de GenBank et pour les données du PaVE. L'enveloppe convexe des points de chaque espèce est surimposée.

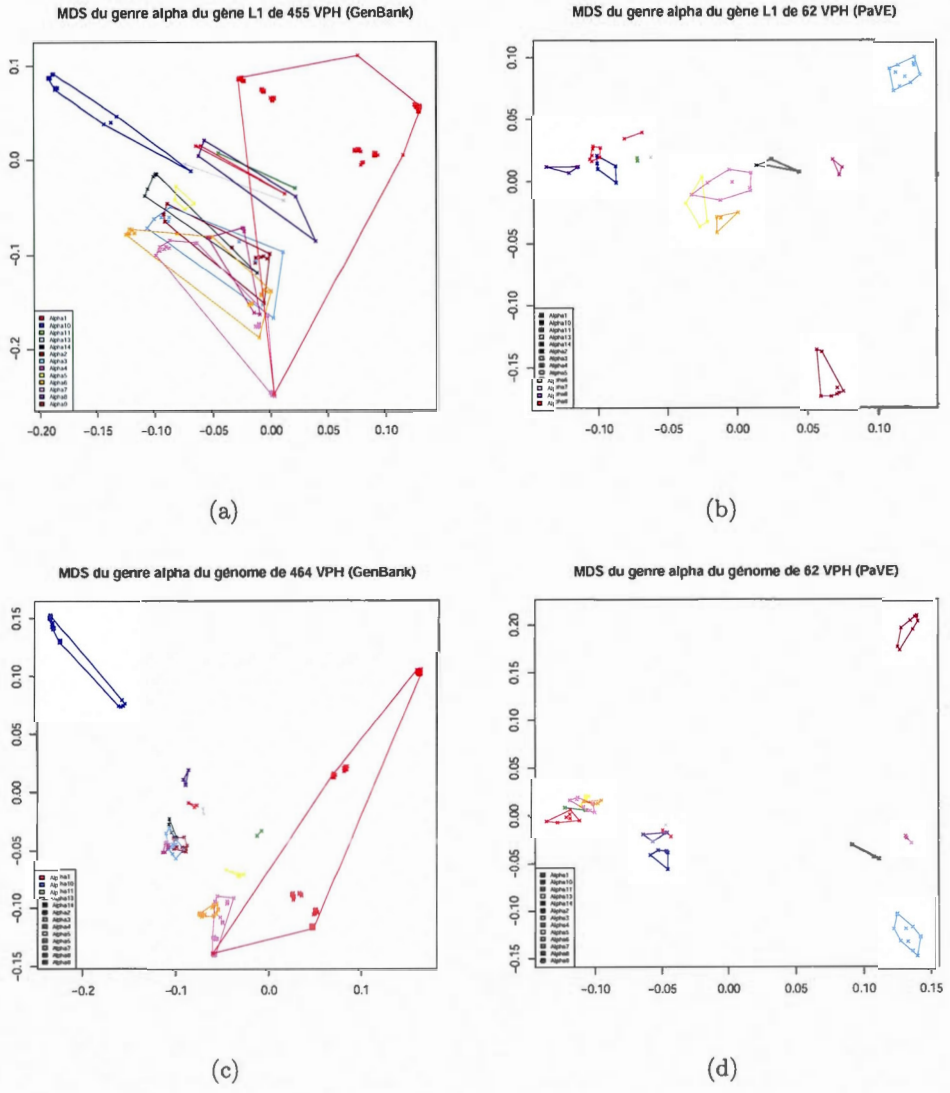


Figure 4.4: Mise à l'échelle (MDS) du gène L1 et du génome du genre alpha. Les figures de la colonne de gauche (a et c) montrent le MDS de chaque région selon l'ensemble des séquences de GenBank, alors que celles de la colonne de droite illustrent le MDS avec les données du PaVE. L'enveloppe convexe des points de chaque espèce est surimposée.

4.1.2 Discussion

Malgré les distorsions, il y a quand même une relativement bonne séparation des genres pour les données (moins nombreuses) du PaVE, car il n'y a aucun chevauchement. Les genres sont moins compacts cependant pour les gènes E4 et E6. Vue l'étendue des enveloppes convexes pour ces deux gènes, il pourrait vraisemblablement y avoir des séquences d'un genre plus semblables à des séquences d'un autre genre qu'à certaines séquences éloignées à l'intérieur du même genre.

Pour les données de GenBank, il y a toujours un chevauchement sauf pour les gènes L2 et E2, et même dans ces cas, les genres sont plus proches. Les groupes sont aussi beaucoup plus étendus. Certaines séquences des genres beta et gamma pourraient être plus semblables entre elles que des séquences à l'intérieur de chaque genre. Il est possible d'avoir plus de chevauchements de frontières entre les genres pour ces données.

En ce qui concerne l'espèce alpha, la projection de la figure 4.4, ne révèle pas une séparation évidente de toutes les espèces, autant dans GenBank que dans le PaVE. Par contre, le moins grand nombre dans le PaVE semble diminuer le chevauchement des frontières. Il faut remarquer qu'il n'y a que 62 types de VPH dans le PaVE, alors que le tableau 2.2 en montre 63 pour GenBank. Le type supplémentaire est le HPVXS2 tel que nommé au NCBI, classé sous l'espèce alpha2, et la séquence génomique est identifiée par KC138720. De toute évidence, ce génome n'a pas encore été traité par le PaVE.

4.2 Validation de regroupement

En fonction des mêmes critères, une collection d'objets peut-être regroupée différemment selon le classificateur utilisé et/ou ses paramètres. Il doit donc exister un moyen de décider du meilleur résultat et les mesures de validation de regroupements peuvent y contribuer. La «meilleure» valeur d'une mesure peut être une indication du meilleur regroupement. Il existe des dizaines de mesures de validation, chacune mettant l'emphase sur une ou plusieurs caractéristiques du regroupement. Ces mesures se séparent en deux catégories principales :

Les mesures de validation externe qui évaluent un regroupement par rapport à des données externes, un étalon («gold standard») représentant le «vrai» regroupement, avec lequel la mesure tente d'établir un consensus.

Les mesures de validation interne qui évaluent un regroupement uniquement avec les données internes.

4.2.1 Mesures de validation externe

La mesure de validation externe doit quantifier la similarité entre deux regroupements : un hypothétique et un étalon. Pour ce faire, une des façons de procéder est de d'abord représenter les regroupements par paires [Achtert et al., 2012; Santos et Embrechts, 2009].

Un regroupement contient plusieurs sous-groupes, chacun étant un sous-ensemble d'éléments d'un univers U. Le regroupement peut être représenté comme l'ensemble de ses sous-groupes. Ainsi, par exemple, le regroupement étalon E d'un univers $U = \{a, b, c, d, e\}$ de cinq éléments, contenant les deux sous-groupes $\{a, b, c\}$ et $\{d, e\}\}$ est représenté par $E = \{\{a, b, c\}, \{d, e\}\}$. La représentation par paires d'un regroupement est l'ensemble de toutes les paires possibles de chacun de ses sous-groupes. Pour le même exemple, la représentation par paires de E est $pE = \{\{a, b\}, \{a, c\}, \{b.c\}, \{d, e\}\}$, soit toutes les paires possibles de $\{a, b, c\}$ et $\{d, e\}$.

La mesure est établie en fonction de la présence ou de l'absence des paires entre deux regroupements représentés par paires. Pour illustrer la suite, un autre regroupement de U est utilisé avec lequel la similarité sera mesurée : $X = \{\{a, b\}, \{c, d, e\}\}$. Sa représentation par paires est : $pX = \{\{a, b\}, \{c, d\}, \{c, e\}, \{d, e\}\}$.

Dans ce contexte, pour établir les mesures, les définitions suivantes sont requises :

- un **vrai positif** est un élément de pX qui est aussi dans pE, par exemple $\{a, b\}$;
- un **faux positif** est un élément de pX qui n'est pas dans pE, par exemple $\{c, d\}$;
- un **faux négatif** est un élément qui n'est pas dans pX mais qui est dans pE, par exemple $\{a, c\}$;
- un **vrai négatif** est un élément qui n'est ni dans pX ni dans pE, par exemple $\{a, d\}$;
- VP est le nombre de vrais positifs ;
- FP est le nombre de faux positifs ;
- FN est le nombre de faux négatifs ;
- VN est le nombre de vrais négatifs.

À l'aide de ces définitions, quelques unes des mesures parmi les plus répandues s'énoncent ainsi :

- **«F-measure»** [Rijsbergen, 1979] est définie par $F = \frac{2VP}{2VP + FP + FN}$. La définition équivalente suivante est souvent rencontrée dans la littérature : $F = \frac{2PR}{P + R}$ où $P = \frac{VP}{VP + FP}$ est la précision, et $R = \frac{VP}{VP + FN}$ le rappel. Il est facilement démontrable par quelques transformations algébriques que les deux sont équivalentes. Il s'agit d'une moyenne pondérée de la

pureté (P) et de la complétude (R) et s'applique à l'ensemble du regroupement, non à chaque sous-groupe [Handl, Knowles et Kell, 2005].

- **«Rand Index»** [Rand, 1971] est définie par $R = \frac{VP + VN}{VP + FP + FN + VN}$.
- **«Jaccard coefficient»** [Jaccard, 1908] est définie par $R = \frac{VP}{VP + FP + FN}$.

La valeur de chacun des indices varie entre 0 et 1, où 1 signifie la concordance parfaite des regroupements comparés.

Pour la classification du VPH, il n'y a pas d'étalon avec lequel établir la comparaison. Elle constitue en quelque sorte l'étalon lui-même et les mesures de validation externe ne sont pas directement applicables.

4.2.2 Mesures de validation interne

Ces mesures de validation évaluent un regroupement uniquement à partir des données internes, c'est-à-dire à la façon dont les éléments sont répartis entre les sous-groupes. Elles cherchent généralement à mesurer [Liu et al., 2010; Handl, Knowles et Kell, 2005] :

- La compacité, c'est-à-dire combien les éléments à l'intérieur d'un sous-groupe sont près les uns des autres. Une forme de variance ou la dissimilarité moyenne entre toutes les paires d'éléments sont communément utilisées. Par exemple, le RMSD («Root Mean Square Distance») d'un regroupement E d'un univers U, composé des sous-groupes E_i est défini comme :

$$\sqrt{\frac{1}{N} \sum_{E_i \in E} \sum_{x \in E_i} \delta(x, \mu_i)}$$

  où N est le nombre d'éléments dans U, $\delta()$ est la distance entre deux éléments et μ_i est le centroïde du sous-groupe E_i.
- La séparation, c'est-à-dire à quel point les sous-groupes sont séparés les uns des autres. La moyenne pondérée des distances entre les sous-groupes, où la distance entre deux sous-groupes est par exemple la distance entre leurs centroïdes, pourrait être utilisée.

Il est généralement désirable que les sous-groupes d'un regroupement soient à la fois compacts et bien séparés. Plusieurs mesures de validation interne combinent donc la mesure des deux aspects. Parmi celles-ci, il existe par exemple :

- **«Dunn Index»** [Dunn, 1974] est défini ainsi :

$$\min_{E_i \in E} \left(\min_{E_j \in E} \left(\frac{dist(E_i, E_l)}{\max_{E_k \in E} diam(E_k)} \right) \right)$$

où $dist(E_i, E_l)$ est, par exemple, la distance minimale entre toutes les paires de E_i et E_l (ce pourrait être une autre mesure comme la distance entre les centroïdes), et $diam(E_k)$ est, par exemple, la distance maximum entre les éléments à l'intérieur de E_k. Une valeur plus grande indique normalement une meilleure compacité et séparation. Le calcul de cet indice produit une valeur unique qui s'applique à l'ensemble du regroupement.

- **«Davies–Bouldin Index»** [Davies et Bouldin, 1979] est défini ainsi :

$$\frac{1}{k} \sum_{E_i \in E} \max_{\substack{E_j \in E \\ E_j \neq E_i}} \left(\frac{\sigma_i + \sigma_j}{dist(\mu_i, \mu_j)} \right)$$

où k est le nombre de sous-groupes, μ_i est le centroïde de E_i, $dist(\mu_i, \mu_j)$ est la distance entre les centroïdes μ_i et μ_j, et σ_i est la distance moyenne entre les éléments de E_i et le centroïde μ_i. Une valeur plus petite indique normalement une meilleure compacité et séparation. Cet indice est également une valeur unique qui s'applique à l'ensemble du regroupement.

- **«Silhouette index»** [Rousseeuw, 1987] est défini ainsi :
  - la largeur Silhouette («Silhouette width») d'un élément est :

$$s(x) = \frac{b(x) - a(x)}{max(b(x), a(x))}$$

où $a(x)$ est la distance moyenne entre x et tous les autres éléments du même sous-groupe que x, et $b(x)$ est la distance moyenne entre x et le sous-groupe le plus proche (celui dont la distance moyenne avec x est minimale),

  - l'indice Silhouette d'un sous-groupe E_i est la moyenne de la largeur Silhouette de chacun de ses éléments :

$$s(E_i) = \frac{1}{|E_i|} \sum_{x \in E_i} s(x)$$

  - l'indice Silhouette global d'un regroupement E est la moyenne de l'indice Silhouette de chacun de ses sous-groupes :

$$s(E) = \frac{1}{k} \sum_{E_i \in E} s(E_i) \equiv \frac{1}{k} \sum_{E_i \in E} \left(\frac{1}{|E_i|} \sum_{x \in E_i} \frac{b(x) - a(x)}{max(b(x), a(x))} \right)$$

où k est le nombre de sous-groupes.

- L'indice est dans l'intervalle ouvert $]-1,1[$. Normalement une valeur près de -1 indique qu'un élément est dans le mauvais sous-groupe et une valeur près de 1 indique une meilleure compacité et séparation. Les bornes de l'intervalle ne sont normalement jamais atteintes puisque cela supposerait que $b(x)$ ou $a(x)$ est nul, impliquant que des éléments sont identiques. Le calcul de cet indice produit une valeur pour chaque élément, chaque sous-groupe et pour le regroupement global.

La procédure pour obtenir un bon regroupement d'un ensemble de données pourrait être la suivante :

- procéder à une mise à l'échelle multidimensionnelle des distances entre les données afin de visualiser des patrons de regroupements dans le but de sélectionner des classificateurs appropriés ;
- sélectionner un certain nombre de classificateurs ;
- sélectionner un certain nombre de mesures de validation.

Alors, le «meilleur» regroupement, ou la meilleure classification, est obtenu ainsi :

```
    POUR chaque classificateur
        POUR chaque paramètre du classificateur
            classifier
            POUR chaque mesure de validation
                mesurer
            FIN POUR
        FIN POUR
FIN POUR

RETOURNER la classification qui a le meilleur indice
```

Dans le cas de la classification du VPH, il n'y a pas de choix de classificateur ou de paramètres. La classification est fixée. Les mesures de validation interne qui donnent un indice global pour l'ensemble d'un regroupement comme l'indice de *Dunn* ne sont donc pas directement utilisables. Le but de ce mémoire n'est pas de trouver une meilleur regroupement, mais de voir la robustesse de la classification actuelle et d'identifier des sous-groupes problématiques. Dans ce but, l'indice *Silhouette* pourrait s'avérer utile puisqu'il s'applique à chaque élément ainsi qu'à chaque sous-groupe, mais il ne donne pas une réponse absolue à la question de savoir si la distance maximale

entre les éléments d'un sous-groupe est toujours plus petite que la distance avec un élément à l'extérieur du sous-groupe, puisque la valeur un n'est jamais atteinte.

CHAPITRE V

DÉFINITION D'UNE NOUVELLE MESURE DE VALIDATION DE REGROUPEMENT : L'INDICE DE COHÉSION

La classification du VPH est unique, et donc une alternative n'est pas disponible pour fins de comparaison. Alors, les mesures de validation externe ne sont pas directement applicables. Par ailleurs, le classificateur et ses paramètres ne sont pas modifiables, rendant ainsi certaines mesures de validation interne inapplicables directement. Dans le but d'évaluer la robustesse de la classification et éventuellement d'identifier des sous-groupes problématiques, un nouvel indice est utilisé. Il est dérivé de l'indice *Silhouette* [Rousseeuw, 1987] mais utilise plutôt une mesure discrète. Alors que l'indice *Silhouette* utilise la distance moyenne entre les éléments comme base de comparaison, le nouvel indice utilise une valeur booléenne, rendant l'interprétation plus facile et éliminant le biais lié à la distance. Le nouvel indice mesure le degré de compacité et de séparation selon que la similarité minimale intra-classes est ou non inférieure à la similarité maximale inter-classes, pour chaque paire de séquences et pour chaque classe.

5.1 Description de l'indice de cohésion

Pour représenter la cohésion d'un groupe, un indice est défini :

Étant donnés :

- T : un arbre,
- $\ell(T)$: l'ensemble des feuilles de l'arbre T,
- T_i : un sous-arbre de T,
- $sim(x, y)$: une mesure de similarité entre 2 feuilles x et y quelconques de T,
- $\sigma(x, y, T_i)$: une valeur discrète selon que la similarité des feuilles x et y (normalement à l'intérieur de T_i) est strictement plus grande que la similarité entre x et chacune des

autres feuilles en dehors de T_i

$$\sigma(x, y, T_i) = \begin{cases} 1 & \text{si } sim(x, y) > \max\limits_{z \in \ell(T) \setminus \ell(T_i)} sim(x, z) \\ 0 & \text{sinon,} \end{cases}$$

- $facteur(x, T_i)$: un facteur d'appartenance d'une feuille x à T_i calculé comme la moyenne de la valeur σ de x par rapport à chacune des autres feuilles de T_i

$$facteur(x, T_i) = \frac{1}{|\ell(T_i)| - 1} \sum_{y \in \ell(T_i) \setminus x} \sigma(x, y, T_i)$$

Alors l'indice de cohésion de T_i est défini par la fonction $indice(T_i)$ comme étant la moyenne du facteur d'appartenance à T_i de chacune de ses feuilles :

$$indice(T_i) = \frac{1}{|T_i|} \sum_{x \in \ell(T_i)} facteur(x, T_i)$$

Ainsi, un indice de cohésion de 1 indique que la similarité entre n'importe lesquels des éléments d'une classe est toujours plus grande que la similarité entre n'importe lesquels des éléments en dehors de la classe, ce qui mesure effectivement la compacité et la séparation. Un indice de cohésion inférieur à 1 indique que le critère n'est pas respecté pour certains éléments d'une classe, i.e. qu'ils sont plus similaires avec des éléments à l'extérieur de la classe qu'avec certains éléments à l'intérieur.

L'indice de cohésion est applicable autant à un regroupement hiérarchique que partitionné. Dans ce dernier cas, l'arbre n'a que deux niveaux et les sous-groupes sont directement rattachés à la racine. Pour un regroupement hiérarchique, l'indice est applicable à chacun des nœuds internes. Les mêmes feuilles se retrouveront alors à l'intérieur de différents sous-groupes, selon le niveau hiérarchique considéré. Aussi, dans ce cas, un nœud pourrait avoir un indice de 1 malgré que ses nœuds enfants aient un indice inférieur à 1.

Bien que l'indice soit défini à l'aide d'une notation se référant aux arbres, il est important de mentionner qu'il ne s'agit là que d'une représentation pratique des regroupements. Une notation ensembliste ou autre aurait pu être utilisée. Ainsi, l'indice de cohésion ne dépend pas directement de la topologie d'un sous-arbre autrement que pour l'ensemble des feuilles qu'il contient. L'indice de cohésion d'un sous-arbre donné sera toujours le même peu importe l'arrangement topologique de ses sous-arbres.

Il faut noter que la définition de l'indice ne permet pas de le calculer pour l'arbre complet. En effet, la fonction σ n'est pas définie dans ce cas, puisque $\ell(T) \setminus \ell(T_i) \equiv \ell(T) \setminus \ell(T) \equiv \emptyset$ et que le maximum n'existe donc pas. Cela est naturel, car pour mesurer la distance minimale avec les feuilles à l'extérieur du nœud, un tel nœud doit exister.

Pour représenter l'indice de cohésion global, la moyenne des indices de chacun des sous-arbres est plutôt utilisée, pondérée par le nombre d'éléments de chaque sous-arbre.

En ajoutant :

- N : le nombre de sous-arbres de T,

alors l'indice de cohésion global de T est :

$$indice(T) = \frac{\sum_{i=1}^{N} indice(T_i)|\ell(T_i)|}{\sum_{i=1}^{N} |\ell(T_i)|}$$

5.2 Algorithme

L'algorithme 10, implémenté par le script dissim_arbre.py (A.12), est utilisé pour le calcul de l'indice de cohésion. Pour chaque nœud interne de l'arbre, la distance entre chaque paire de feuilles à l'intérieur du nœud, ainsi que la distance minimale entre chaque feuille et les feuilles à l'extérieur du nœud sont disponibles. Pour chaque feuille, lorsque la distance à l'interne avec une autre feuille est inférieure à la distance minimale à l'externe, la feuille voit son *facteur* augmenter de un. Le *facteur* final de la feuille, qui est le *facteur* divisé par le nombre de feuilles internes moins un, est ajouté à l'*indice* du nœud. L'*indice* final du nœud est ensuite divisé par le nombre de feuilles.

Quant à l'indice de cohésion global de l'arbre, il est calculé dans un tableur à partir de l'ensemble des indices de cohésion de chacun des nœuds, obtenu par le même algorithme (en fait l'information vient de la sortie standard du script dissim_arbre.py).

Le calcul de l'indice pour le nœud HPV94 du gène L2 est illustré avec les données de GenBank. Le tableau 5.1 montre les étapes du calcul, qui sont décrites ainsi :

- Le nœud compte trois séquences : `AB201226`, `AJ620211` et `GU117628`.

```
T = Arbre(arbre.nw)

POUR chaque N ∈ T.parcourir()
    feuillesInter = ℓ(T) \ ℓ(N)
    feuillesIntra = ℓ(N)
    nbFeuillesIntra = nombre d'éléments de feuillesIntra
    indice = 0
    SI ¬ N.estFeuille()
        POUR chaque f ∈ feuillesIntra
            minInter =  distance minimum entre f et feuillesInter
            facteur = 0
            POUR chaque x ∈ feuillesIntra \ f
                SI similarité(f, x) < minInter
                    facteur = facteur + 1
                FIN SI
            FIN POUR
            indice = indice + (facteur / (nbFeuillesIntra - 1))
        FIN POUR
        indice = indice / nbFeuillesIntra
     FIN SI
 FIN POUR
```

Algorithme 10: Calcul de l'indice de cohésion.

- Il s'agit de comparer la similarité de chacune des séquences entre elles et de vérifier qu'elle est plus grande que celle avec n'importe quelle séquence en dehors du HPV94. En utilisant la dissimilarité, c'est dire que la dissimilarité de chacune des séquences entre elles est plus petite qu'avec n'importe quelle séquence en dehors du HPV94.
- En débutant avec la première séquence `AB201226`, la séquence la plus similaire à l'inter (en dehors du nœud HPV94) est `NC_001576` avec une dissimilarité de 0.158528.
- La dissimilarité entre `AB201226` et `AJ620211`, une autre séquence du même nœud, est 0.002174, donc inférieure à 0.158528. Ainsi les séquences `AB201226` et `AJ620211` sont plus similaires entre elles qu'avec n'importe quelle autre séquence de l'arbre. La fonction σ vaut donc 1 dans ce cas.
- La dissimilarité entre `AB201226` et `GU117628`, la troisième séquence du même nœud, est 0.186693, donc supérieure 0.158528. Ainsi la séquence `AB201226` est plus similaire avec `NC_001576`, qui est à l'extérieur du groupe, qu'avec avec une séquence du groupe. La fonction σ vaut donc 0 dans ce cas.
- Le *facteur* de la feuille `AB201226` est la somme des valeurs de σ soit $(1 + 0)$ divisé par le

nombre de séquences moins 1, soit (3 - 1). Le *facteur* vaut donc (1 + 0) / (3 - 1) = 0.5.

- Le calcul est refait pour les deux autres feuilles AJ620211 et GU117628 pour lesquels on obtient respectivement des *facteurs* de 0.5 et 1.0.
- Finalement, l'indice de cohésion du nœud HPV94 pour le gène L2 est la somme des *facteurs* (0.5 + 0.5 + 1) divisé par le nombre de séquences (3), soit 2/3 ou 0.667, comme indiqué au tableau 5.2.

Séquence	Séq inter	Dissim	Séq intra	Dissim	σ	Facteur	Indice cumul.
AB201226	NC_001576	0.158528	AJ620211	0.002174	1		
			GU117628	0.186693	0		
						(1+0)/2 = 0.5	0.5
AJ620211	NC_001576	0.157820	AB201226	0.002174	1		
			GU117628	0.184755	0		
						(1+0)/2 = 0.5	1
GU117628	NC_001576	0.198966	AB201226	0.186693	1		
			AJ620211	0.184755	1		
						(1+1)/2 = 1	2

Tableau 5.1: Exemple de calcul de l'indice de cohésion.

5.3 Résultats

Le tableau 5.2 montre, pour chacun des gènes ainsi que pour le génome complet (les différentes régions génomiques), les 30 taxons (les lignes ombrées du tableau) pour lesquels l'indice de cohésion est inférieur à 1 pour au moins une région. La figure 5.1 présente le diagramme de l'occurrence d'indices inférieurs à 1 pour chaque région génomique. La moyenne est 11.11.

L'information de même nature est fournie au tableaux 5.3 et 5.2 (moyenne de 8.11) pour les VPH du PaVE.

Le tableau 5.4 quant à lui présente l'indice de cohésion global pour chacune des régions génomiques pour GenBank et pour le PaVE. La figure 5.3 présente la même information graphiquement.

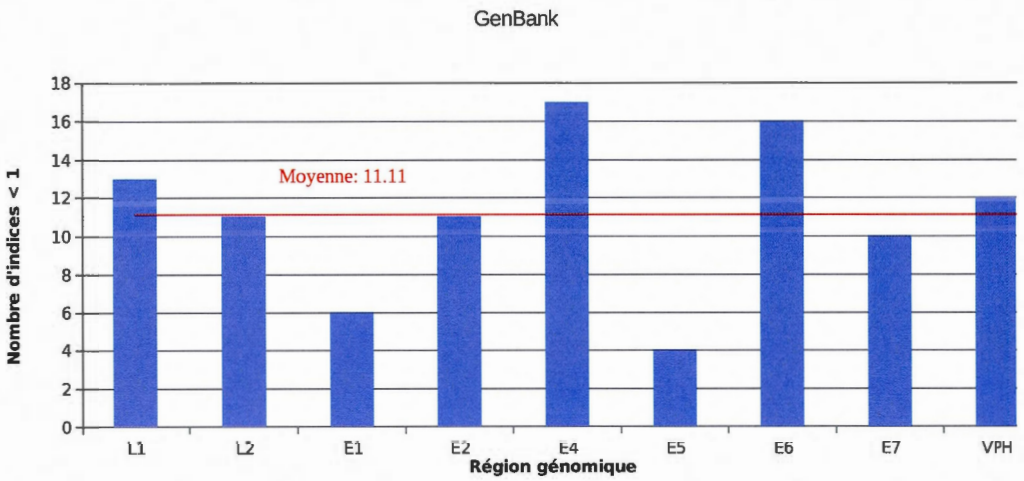


Figure 5.1: Diagramme de l'occurrence d'indices de cohésion inférieurs à 1 pour chaque gène et pour le génome complet (GenBank). La moyenne est de 11.11.

5.4 Discussion

D'abord, aucun gène n'a un indice de cohésion de 1 pour l'ensemble des taxons (il n'y a aucune colonne vide), que ce soit pour les données de GenBank ou du PaVE. Autrement dit, le meilleur indice n'est pas toujours obtenu avec la même région. Il n'est pas vraiment informatif d'analyser la moyenne de chaque taxon (ligne) et/ou de chaque gène (colonne), puisque certains gènes, notamment E5, ne sont pas présents dans tous les genres, et il n'y a pas le même nombre de séquences par taxon. L'indice de cohésion global est plus intéressant à cet effet puisqu'il tient compte du nombre de séquences.

Au niveau des indices individuels pour les données de GenBank, pour le gène L1, sur lequel est basé la classification, les espèces alpha 3, 7, 8 et 9, beta 2 et gamma 6 ont les indices les plus faibles, inférieurs à 0.9. Le genre gamma au complet à un indice inférieur à 0.5 et 13 taxons ont un indice inférieur à 1. Alors, en principe, E1 serait plus approprié comme base de classification (figure 5.1), avec «seulement» 6 taxons dont l'indice est inférieur à 1. Mais, dans ce cas :

- alpha a un meilleur indice avec le génome complet (VPH) ;
- alpha6 a un meilleur indice avec n'importe laquelle des autres régions ;

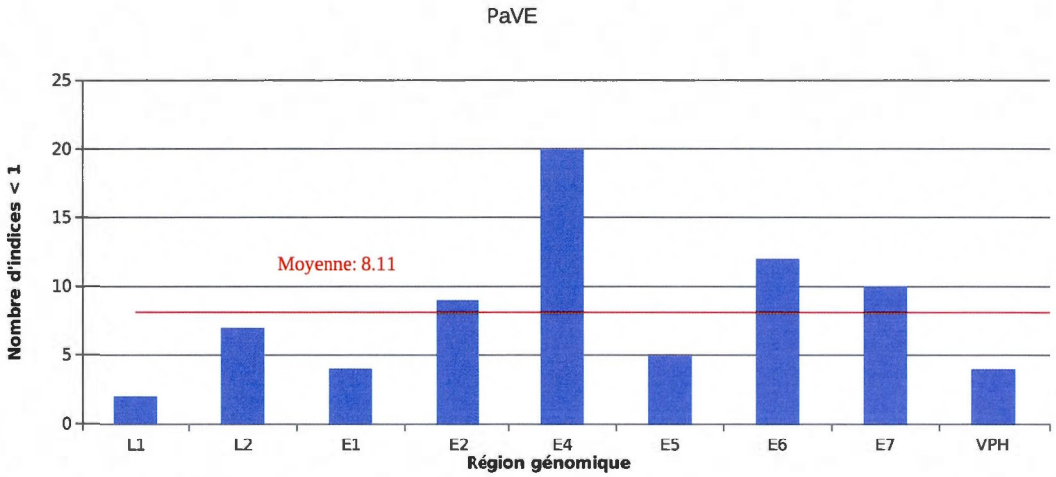


Figure 5.2: Diagramme de l'occurrence d'indices de cohésion inférieurs à 1 pour chaque gène et pour le génome complet (PaVE). La moyenne est de 8.11.

- alpha9 a un meilleur indice avec L1, L2, E2, E6, E7 ou VPH ;
- gamma a un meilleur indice avec E2 ;
- HPV53 a un meilleur indice avec n'importe laquelle des autres régions.

Sauf pour le gène E4, le genre alpha à un indice supérieur à 0.91. Le plus grand nombre de séquences de ce genre pourrait en partie expliquer ce résultat. Avec beta2 et gamma, ce sont les seuls taxons pour lesquels aucune région n'obtient un indice de 1 (toujours sans considérer E5).

L'indice global de GenBank montre que le génome complet est meilleur (0.994) que E1 (0.963).

Pour les indices individuels avec les données du PaVE, seulement beta2 et gamma sont inférieurs à 1 pour le gène L1. C'est le gène qui à le moins d'occurrences (figure 5.2). Le gène L1 pourrait donc être considéré comme la meilleure région, ce qui n'est pas surprenant puisque c'est celle utilisée pour la classification. Cependant l'indice global montre que l'indice du génome complet est meilleur (0.973) que celui de L1 (0.934).

Donc, dans les deux cas (GenBank et PaVE), en utilisant le génome complet, il y a moins de séquences qui sont plus similaires à des séquences à l'extérieur des sous-groupes qu'à l'intérieur.

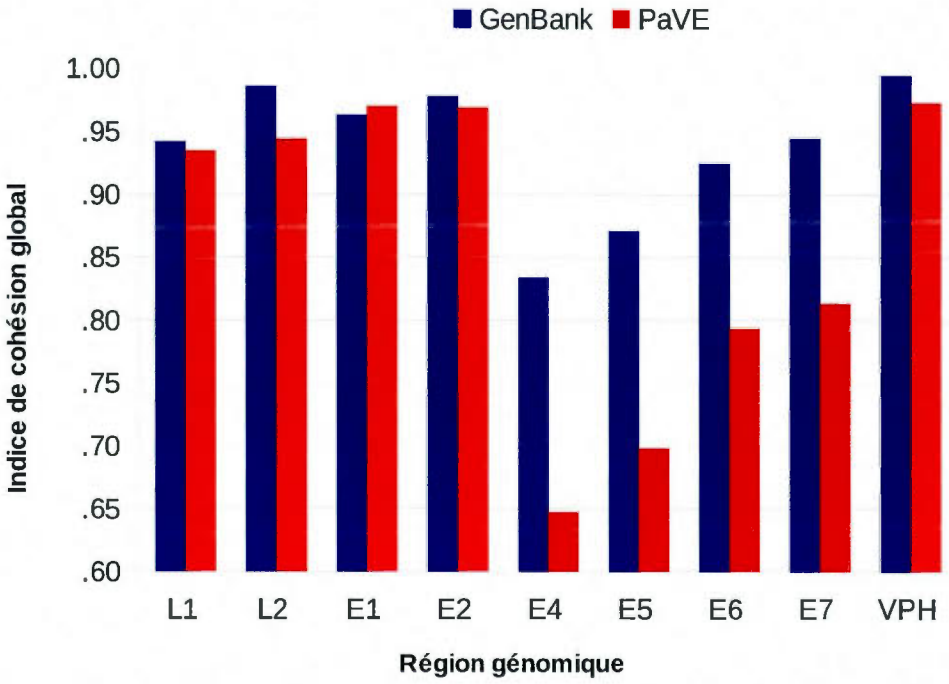


Figure 5.3: Indice de cohésion global de chaque région génomique selon la classification taxonomique de GenBank et du PaVE.

Alors que les gènes L1, L2, E1, E2 et le VPH ont des indices globaux élevés, la figure 5.3 met en évidence la faible valeur relative de l'indice pour E4, E5, E6 et E7. Un regroupement à partir de la similarité de ces régions pourrait bien mener à une classification différente. À cet effet, plusieurs études ont montré une incongruité phylogénique entre l'évolution des différents gènes (par exemple [Narechania et al., 2005; Burk, Chen et Van Doorslaer, 2009]).

Taxon	L1	L2	E1	E2	E4	E5	E6	E7	VPH
Alpha	0.965 455	0.998 454	0.992 454	0.988 453	0.756 431	0.987 298	0.936 454	0.918 453	0.998 454
Alpha1							0.333 3		
Alpha2	0.973 11				0.982 8				
Alpha3	0.818 12	0.833 12			0.538 12		0.977 12	0.712 12	0.833 12
Alpha5		0.833 4			0.333 3	0.833 3		0.917 4	0.833 4
Alpha6	0.987 34		0.954 33	0.986 34					
Alpha7	0.769 50	0.998 50			0.919 49	0.995 49			0.998 50
Alpha8	0.750 4						0.667 4		
Alpha9	0.866 243	0.999 243	0.838 243	0.941 243	0.717 235	0.537 239	0.869 243		0.999 243
Alpha10					0.926 72				
Alpha14	0.905 7	0.905 7			0.810 7				0.905 7
Beta	0.993 50			0.918 50	0.750 39		0.739 49	0.807 49	
Beta1	0.965 19	0.985 19		0.985 19	0.553 12		0.706 18	0.768 18	0.995 21
Beta2	0.875 24	0.775 24	0.976 24	0.942 24	0.867 21		0.801 24	0.620 24	0.975 24
Beta3							0.833 4	0.667 4	
Beta5							0.000 2		
Gamma	0.490 39	0.752 39	0.806 39	0.928 39	0.518 38		0.502 36	0.535 39	0.846 39
Gamma6	0.833 3				0.000 2				
Gamma7					0.738 7		0.905 7	0.929 7	
Gamma8					0.500 3				
Gamma9				0.500 2	0.000 2		0.500 2		
Gamma11					0.250 4			0.917 4	
Gamma12		0.500 2		0.500 2			0.000 2		0.500 2
HPV1a									0.500 2
HPV35				0.929 28					
HPV42							0.000 2		
HPV53			0.933 15						
HPV56				0.857 7					
HPV94		0.667 3							0.667 3
HPV120							0.917 4		

Tableau 5.2: Indices de cohésion des gènes et du génome complet des VPH de GenBank. Pour chacun des gènes et pour le génome complet, le tableau présente les taxons dont les indices de cohésion sont différents de 1, pour au moins un gène ou pour le génome complet. Lorsqu'il n'y a pas la valeur de l'indice, c'est qu'il est à 1. Par exemple pour le taxon *alpha1* l'indice est 1 pour toutes les régions génomiques sauf pour le gène E6 dont l'indice est 0.333. Les nombres sous les indices indiquent le nombre de séquences appartenant au taxon correspondant.

Taxon	L1	L2	E1	E2	E4	E5	E6	E7	VPH
Alpha		0.999 62	0.996 62	0.990 62	0.572 62	0.585 41	0.807 62	0.834 62	
Alpha3		0.855 11			0.518 11	0.500 2	0.982 11	0.718 11	
Alpha5					0.750 4	0.333 3		0.917 4	
Alpha6					0.833 4				
Alpha7		0.982 8			0.786 8	0.982 8			
Alpha8					0.583 4		0.750 4		
Alpha9			0.833 7	0.905 7	0.738 7	0.429 7	0.738 7		
Alpha10					0.700 5				
Alpha14					0.667 3				
Beta				0.959 43	0.735 43		0.734 43	0.812 43	
Beta1		0.984 18		0.977 18	0.507 18		0.683 18	0.778 18	0.993 18
Beta2	0.938 18	0.791 18	0.974 18	0.967 18	0.820 18		0.775 18	0.631 18	0.984 18
Beta3				0.917 4	0.833 4		0.833 4	0.667 4	
Gamma	0.519 35	0.760 35	0.824 35	0.925 35	0.448 35		0.522 32	0.531 35	0.829 35
Gamma7					0.600 5		0.950 5	0.950 5	
Gamma9				0.500 2	0.000 2		0.500 2		
Gamma11					0.333 4			0.917 4	
Gamma12		0.667 3		0.833 3	0.833 3		0.500 3		0.667 3
Gamma13					0.000 2				
Mu					0.000 2				

Tableau 5.3: Indices de cohésion des gènes et du génome complet des VPH du PaVE. Pour chacun des gènes et pour le génome complet, le tableau présente les taxons dont les indices de cohésion sont différents de 1, pour au moins un gène ou pour le génome complet. Lorsque'il n'y a pas la valeur de l'indice, c'est qu'il est à 1. Par exemple pour le taxon *alpha* l'indice est 1 pour le gène L1 et pour le VPH. Les nombres sous les indices indiquent le nombre de séquences appartenant au taxon correspondant.

	L1	L2	E1	E2	E4	E5	E6	E7	VPH
GenBank	0.941	0.985	0.963	0.977	0.834	0.87	0.924	0.944	0.994
PaVE	0.934	0.944	0.97	0.969	0.648	0.698	0.794	0.813	0.973

Tableau 5.4: Indice de cohésion global de chaque région génomique selon la classification taxonomique de GenBank et du PaVE.

CONCLUSION ET PERSPECTIVES

La classification actuelle du VPH a été analysée sous différents angles et les principales conclusions se résument ainsi :

Le pourcentage de similarité des séquences utilisé comme caractéristique de base de classification ne permet pas d'obtenir des frontières nettes entre les classes. De plus, pour voir apparaître ces frontières à la comparaison, il est nécessaire de choisir un sous-ensemble de séquences peu similaires, et surtout, une annotation spécifique du gène L1. L'utilisation du génome complet permet d'éviter cette dernière difficulté. Aussi, non seulement il y a des chevauchements entre les frontières, mais les seuils de similarité ne sont pas absolus. Par exemple, les frontières seraient à 63% et 73% plutôt que 60% et 70%. Finalement, la frontière séparant les types (10%) n'apparaît jamais clairement et ne devrait peut-être pas être considérée comme un élément caractérisant le type.

La caractéristique de similarité, interprétée plutôt comme critère de classification, ne permet pas de la reproduire. En ce sens, la similarité ne constitue pas une méthode de classification. En fait, compte tenu des autres conclusions, il apparaît peu probable de pouvoir reproduire la classification avec un classificateur basé uniquement sur la similarité.

L'arbre phylogénique du gène L1 est, à une exception près, identique à l'arbre taxonomique, donc à la classification actuelle, mais avec un assez faible degré de confiance. L'arbre du génome complet est quant à lui identique, avec un bon degré de confiance, suggérant ainsi qu'il est préférable d'utiliser le génome complet pour l'inférence phylogénique. L'inférence basée sur les caractères, selon le maximum de vraisemblance, en utilisant le génome complet, est un classificateur permettant de régénérer la classification actuelle, à partir de l'ensemble des séquences publiques de VPH disponibles à ce jour. Les seuils de similarité peuvent être utilisés ensuite pour désigner certains sous-arbres comme genres, espèces et types, à des fins taxonomiques. C'est la seule des caractéristiques qui peut être considérée comme un critère de classification. Il est donc possible de générer exactement la classification actuelle en utilisant uniquement un algorithme

informatique, ce qui est contraire à l'opinion de certains des spécialistes du domaine.

L'indice de cohésion montre que certaines séquences sont plus similaires avec des séquences à l'extérieur d'un sous-groupe qu'avec certaines séquences à l'intérieur d'un sous-groupe de la classification. C'est une indication que certains sous-groupes ne sont pas bien séparés ou pas assez compacts. C'est d'autant plus vrai avec un plus grand nombre de séquences. Bien que le travail n'ait pas été fait, les valeurs différentes de l'indice de cohésion selon le gène utilisé, laissent à penser que la classification selon d'autres régions génomiques que le gène L1 pourrait être différente.

L'augmentation du nombre de génomes séquencés a un effet sur la compacité et la séparation des sous-groupes, mais peu sur les arbres phylogéniques inférés qui demeurent équivalents à l'arbre taxonomique, surtout avec le génome complet. La différence est dans la vraisemblance des bifurcations lorsque les séquences sont très similaires.

Historiquement, l'emphase a été mise sur les pourcentages de similarité pour présenter la classification du VPH (et des autres PV). Or, il apparaît qu'il s'agit plus d'une constatation a posteriori : étant donné la classification, voici une de ses caractéristiques. La réciproque : étant donné ses caractéristiques, voici la classification, ne peut pas être établie. Il serait souhaitable de distinguer officiellement la classification du classificateur pour lever toute confusion à cet égard.

La phylogénie prend en compte plus d'informations que la simple mesure de similarité, qui résume l'information génomique en un seul chiffre. Éventuellement, avec une mesure de similarité plus sophistiquée, le regroupement, utilisé comme classificateur, pourrait s'avérer une avenue intéressante. Cette mesure pourrait tenir compte, par exemple, des caractéristiques fonctionnelles, physiques ou chimiques des portions de génomes. L'indice de cohésion, quant à lui, pourrait être enrichi en intégrant l'information des différentes régions génomiques.

À un autre niveau, la question peut se poser à savoir si cette classification est la «meilleure». La réponse tient évidement à l'objectif poursuivi. En ce sens, la classification actuelle est bonne puisqu'elle a favorisé, entre autres, le développement des tests de dépistage et de vaccins pour quatre des types de virus les plus nocifs. D'un autre côté, elle pourrait sans doute être améliorée pour éliminer les incongruités au niveau de la biologie et de la pathogénicité (ref. 2.1.6). Le classificateur pourrait par exemple tenir compte des caractéristiques carcinogènes des séquences

[Diallo et al., 2009]. L'évolution divergente de certains gènes pourrait également être analysée pour détecter le transfert horizontal de matériel génétique entre les virus [Boc, Philippe et Makarenkov, 2010].

APPENDICE A

LISTAGE DES PROGRAMMES

A.1 nw.cpp

Calcul d'un alignement selon l'algorithme de Needleman-Wunch.

```
// calcul d'un alignement selon l'algorithme de Needleman-Wunch

#include <cstdlib>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <iostream>
#include <string>

using namespace std;

#define moveDiag 1
#define moveUp 2
#define moveLeft 4

/* ************************************************************************
* fonction afficheScore
* affiche la matrice de score
* PRÉCONDITIONS:
*   m: espace memoire de nLig*nCol
*   nLig: nombre de lignes
*   nCol: nombre de colonnes
*   lig: entete de ligne
*   col: entete de colonne
* ALGORITHME:
*   affiche la matrice precede de l'entete de ligne et de colonne
* POSTCONDITIONS:
*   affiche la matrice sur stdout
*/

void afficheScore (int *m, int nLig, int nCol, string lig, string col) {
    int i, j;

    printf("      ");
    for (j=0; j<nCol-1; j++) {
        printf("%6c", col[j]);
    }
    printf("\n");

    printf("      ");
    for (i=0; i<nLig; i++) {
```

```
        if (i > 0) printf("%6c", lig[i-1]);
        for (j=0; j<nCol; j++) {
            printf("%6d", m[i*nCol + j]);
        }
        printf("\n");
    }
    printf("\n");

}

/* ************************************************************************
* fonction afficheTrace
* affiche la matrice de trace
* PRÉCONDITIONS:
*   m: espace memoire de nLig*nCol
*   nLig: nombre de lignes
*   nCol: nombre de colonnes
*   lig: entete de ligne
*   col: entete de colonne
* ALGORITHME:
*   affiche la matrice precede de l'entete de ligne et de colonne en format
*   lisible avec U pour haut D pour diagonale et L pour gauche
* POSTCONDITIONS:
*   affiche la matrice sur stdout
*/

void afficheTrace (int *m, int nLig, int nCol, string lig, string col) {
    int i, j;

    printf("␣␣␣␣␣␣␣␣");
    for (j=0; j<nCol-1; j++) {
        printf("%4c", col[j]);
    }
    printf("\n");

    printf("␣␣␣␣");
    string moveC;
    for (i=0; i<nLig; i++) {
        if (i > 0) printf("%4c", lig[i-1]);
        for (j=0; j<nCol; j++) {
            int move = m[i*nCol + j];
            if (move & moveDiag) moveC += 'D';
            if (move & moveUp) moveC += 'U';
            if (move & moveLeft) moveC += 'L';
            printf("%4s", moveC.c_str());
            moveC.clear();
        }
        printf("\n");
    }
    printf("\n");

}

/* ************************************************************************
* fonction max
* retourne le maximum de 3 valeurs et une indication de sa provenance
* PRÉCONDITIONS:
*   a: un tableau d.entiers
*   *index: un entier
* ALGORITHME:
*   calcule le maximum des 3 valeurs
*   et indique sa position dans un vecteur de bits (1, 2, 4) ou une combinaison
```

```
*   lors d'égalités, il y a plusieurs indicateurs
* POSTCONDITIONS:
*   *index contient l'indicateur
*/

int max (int *a, int *index) {
    if (a[0] > a[1]) {
        if (a[0] > a[2]) {
            *index = 1;
            return a[0];
        }
        // a[0] <= a[2]
        *index = 4;
        if (a[0] == a[2]) *index = 5;
        return a[2];
    }
    //a[0] <= a[1]
    if (a[1] > a[2]) {
        *index = 2;
        if (a[0] == a[1]) *index = 3;
        return a[1];
    }
    //a[1] <= a[2]
    *index = 4;
    if (a[1] == a[2]) *index = 6;
    if (a[0] == a[1]) *index = 7;
    return a[2];

}

/* *****************************************************************************
* fonction calculeNW
* calcule un alignement global selon l'algorithme de Needleman-Wunch
* PRÉCONDITIONS:
*   nomA: nom de la premiere sequence
*   sA: premiere sequence
*   nomB: nom de la deuxieme sequence
*   sB: deuxieme sequence
*   unSeulAlignement : booleen si vrai imprime un seul alignement, sinon imprime
*                      tous les alignements optimaux
* ALGORITHME:
*    Needleman-Wunch
* POSTCONDITIONS:
*   resultat ecrit sur stdout
*/

void calculeNW (string nomA, string sA, string nomB, string sB,
                bool unSeulAlignement) {
    /*
    sA sera l'étiquette des lignes
    sB sera l'étiquette des colonnes
    */
    const int d = -2; //gap penalty
    int lA = sA.length() + 1; //+1 pour ligne initiale
    int lB = sB.length() + 1; //+1 pour colonne initiale
    int *score, *trace; //matrices de lA lignes par lB colonnes
    int move[3];
    int index;
    int size = lA * lB;
    if ((score = (int *) calloc(size, sizeof(int))) == NULL) exit(-2);
    if ((trace = (int *) calloc(size, sizeof(int))) == NULL) exit(-3);
```

```
for (int i=0; i<lB; i++) { //1ere ligne
    score[i] = i * d;
    trace[i] = moveLeft;
}

for (int j=0; j<lA; j++) { //1ere colonne
    score[j* lB] = j * d;
    trace[j* lB] = moveUp;
}

trace[0] = -1; //pour le fun...
afficheScore(score, lA, lB, sA, sB);
afficheTrace(trace, lA, lB, sA, sB);

for (int i=1; i<lA; i++) {
    for (int j=1; j<lB; j++) {
        //diagonale
        move[0] = score[(i-1)*lB + (j-1)] + (sA[i-1] == sB[j-1] ? 3 : -1);
        move[1] = score[(i-1)*lB + j] + d; //haut
        move[2] = score[i * lB + (j-1)] + d;//gauche
        score[i * lB + j] = max(move, &index);
        trace[i * lB + j] = index;
    }
}

afficheScore(score, lA, lB, sA, sB);
afficheTrace(trace, lA, lB, sA, sB);

//traceback
string sAalign, sBalign;
bool encore = true;
int diff = 0;
int gapA = 0;
int gapB = 0;

while (encore) {

    int i = lB -1;
    int j = lA -1;

    encore = false;

    while (j > 0 || i > 0) {
        int courant = i + j*lB;
        switch (trace[courant]) {
            case 7:
            case 5:
            case 3:
                if (! encore) {
                    encore = true;
                    trace[courant] &= ~1;
                }
                //pas de break;
            case 1: //diag
                i--; j--;
                sAalign += sA[j];
                sBalign += sB[i];
                break;
            case 6:
                if (! encore) {
                    encore = true;
                    trace[courant] &= ~4;
```

```
                    }
                    //pas de break;
                case 4: // left
                    i--;
                    sAalign += '-';
                    sBalign += sB[i];
                    break;
                case 2: //up
                    j--;
                    sAalign += sA[j];
                    sBalign += '-';
                    break;
                default://bug
                    exit(-99);
            }

        }
        cout << nomA << endl;
        cout << string(sAalign.rbegin(), sAalign.rend()) << endl; //envers
        cout << nomB << endl;
        cout << string(sBalign.rbegin(), sBalign.rend()) << endl; //envers

        if (unSeulAlignement) {
            break;
        }
        cout << endl;
    }

    free(score);
    free(trace);
}

/* *****************************************************************************
* programme principal
* calcule un alignement global selon l'algorithme de Needleman-Wunch
* PRÉCONDITIONS:
*   2 sequences au format FASTA sur stdin
* ALGORITHME:
*    Needleman-Wunch
* POSTCONDITIONS:
*   alignement au format FASTA sur stdout
*/

main(int argc, char**argv) {
    string lignes[4];
    string buff;
    int i = -1;

    getline(cin, buff);
    if (buff[0] != '>') {
        cerr << "fichier␣invalide:␣caractère␣>␣attendu␣au␣début\n";
        return -1;
    }
    while (!cin.eof() && i < 4) {
        if (buff[0] == '>') { //en-tête
            if (i > 1) {
                cerr << "fichier␣invalide:␣trop␣de␣caractères␣>\n";
                return -1;
            }
            i++;
            lignes[i] += buff;
            i++;
```

```
        }
        else {
            lignes[i] += buff;
        }
        getline(cin, buff);
    }

    calculeNW(lignes[0], lignes[1], lignes[2], lignes[3], true);

    return 0;
}
```

A.2 fasta_genes_vph-2.sh

Extraire le FASTA des gènes et des protéines du VPH sur stdout.

```
#!/bin/bash
#extrait le fasta des genes et des proteines du VPH sur stdout
bd="vph"

genes=(E1 E2 E4 E5 E5A E5B E6 E7 L1 L2)

#expression regulieres pour tenter de retrouver le max d'info sur les genes
reGenes=("E1A.*E1B|.*[eE]1(?!A|B|\^|\-).*" ".*[eE]2.*" ".*(?<!\-|\^)[eE]4.*"
         ".*[eE]5(?!A|B|a|b).*" ".*[eE]5[aA].*" ".*[eE]5[bB].*" ".*[eE]6.*"
         ".*[eE]7.*" ".*[lL]1(?!A|B|a|b).*" ".*[lL]2.*")
pfixGene="gene"
pfixProteine="proteine"

for i in ${!genes[*]}
do
    fasta_un_gene_vph-2.sh $bd $ids ${reGenes[$i]} -featseq >>
                                                    $pfixGene${genes[$i]}
    fasta_une_proteine_vph-2.sh $bd $ids ${reGenes[$i]} -feattrans >>
                                                    $pfixProteine${genes[$i]}
done
```

A.3 fasta_un_gene_vph-2.sh

Retourner sur stdin le FASTA d'un gène du VPH

```
#!/bin/bash
if [ -z "$4" ]
    then
        echo "retourne␣sur␣stdin␣les␣fasta␣des␣gènes␣du␣VPH␣␣associés␣aux␣id"
        echo "Usage:␣$0␣nom_bd_logique␣fichier_de_id␣gene␣-featseq␣-feattrans"
        exit -1
fi

bd=$1
ids=$2
gene=$3
type=$4
baseParams="-id␣-gi␣-featdb␣xx␣-annot␣xx"

for id in $(cat $ids)
    #recherche dans les differents FEATURES
    do
        #dans gene
```

```
biodb_get.py $baseParams -feat gene -featq gene -featqval $gene $type
             $bd $id > tmp
source="gene/gene"
nbLignes=$(wc -l < tmp)
if [ "$nbLignes" -le 1 ] #on a seulement le id ou rien du tout
    then
        #dans le CDS/gene
        biodb_get.py $baseParams -feat CDS -featq gene -featqval
                     $gene $type $bd $id > tmp
        source="CDS/gene"
        nbLignes=$(wc -l < tmp)
        if [ "$nbLignes" -le 1 ]
            then
                # dans le CDS/product
                biodb_get.py $baseParams -feat CDS -featq product
                             -featqval $gene $type $bd $id > tmp
                source="CDS/product"
                nbLignes=$(wc -l < tmp)
            if [ "$nbLignes" -le 1 ]
                then
                    #dans le gene/note
                    biodb_get.py $baseParams -feat gene -featq note
                                 -featqval $gene $type $bd $id > tmp
                    source="gene/note"
                    nbLignes=$(wc -l < tmp)
                if [ "$nbLignes" -le 1 ]
                    then
                        #dans le CDS/note
                        biodb_get.py $baseParams -feat CDS -featq note
                                     -featqval $gene $type $bd $id > tmp
                        source="CDS/note"
                        nbLignes=$(wc -l < tmp)
                        if [ "$nbLignes" -le 1 ]
                            then
                                echo "Not␣found␣$id␣$gene" >&2
                                continue
                        fi
                    fi
                fi
        fi
fi
bonneLigne='3p'
if [ "$nbLignes" -gt 3 ]
then
    echo "trop␣de␣lignes␣($nbLignes)␣gene␣$id␣$gene" >&2
    if [ "$nbLignes" -ne 5 ]
    then
        echo ">>>␣rejeté" >&2
        continue
    fi
    #on garde la plus longue
    lg3=$(sed -n '3p' < tmp | wc -c)
    lg5=$(sed -n '5p' < tmp | wc -c)

    if [ "$lg5" -gt "$lg3" ]
    then
        bonneLigne='5p'
    fi
    echo "lg3:␣$lg3,␣lg5:␣$lg5,␣on␣garde␣$bonneLigne" >&2
fi

echo -n ">"
```

```
        sed -n '1p' < tmp | tr -d '"' #"
        sed -n "$bonneLigne" < tmp | tr -d '"' | fold #"
        echo "Info: $id $gene $source" >&2
    done
```

A.4 traiter_arbre_taxa_ncbi.py

Ajouter le nom des séquences (accession) sous les taxons correspondants

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
'''
Usage: %s arbre_taxonomique format taxon-accession

Lit l'arbre phylogénique au format spécifié et:

1) ajoute le nom des séquences (accession) sous les taxons correspondants
2) enlève les taxons qui n'ont pas de séquences terminales
3) le nouvel arbre est enregistré dans le répertoire courant sous le nom
new_tree.nw
'''

from ete2 import Tree
import sys

if len(sys.argv) == 4 :
    racine = Tree(sys.argv[1], format=int(sys.argv[2]))
else :
    print  __doc__ % sys.argv[0]
    sys.exit()

f = open(sys.argv[3], 'r')
for ligne in f:
    #pour chaque taxon
    champs = ligne.split()
    if len(champs) != 2 :
        print 'Fichier %s invalide (nombre de champs)' % sys.argv[3]
        sys.exit()
    taxon = champs[0]
    accession = champs[1]
    print taxon, accession
    liste = racine.search_nodes(name=taxon)
    if len(liste) == 0:
        print 'taxon', taxon, 'non trouve'
        continue
    elif len(liste) > 1 :
        print 'plusieurs taxonx trouvés pour %s' % taxon
        sys.exit()
    noeud = liste[0]
    #ajouter la séquence
    nouveau = noeud.add_child(name=accession, dist=0.0, support=100)
    nouveau.add_features(estSequence=True)

feuilles = racine.get_leaves()
aDetruire = []
for f in feuilles :
    if not 'estSequence' in f.features :
        aDetruire.append(f)
for f in aDetruire :
    f.delete()
```

```
racine.write(format=int(sys.argv[2]), outfile="new_tree.nw", features=[])
```

A.5 arbrePaVE.py

Produire un arbre newick à partir du fichier csv trié en ordre croissant des 4 colonnes.

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
'''
Usage: %s fichier-csv-trie-au-bon-format nom_arbre
produit un arbre newick à partir du fichier csv trié en ordre des colonnes 4:
    colonne 1: le genre
    colonne 2: l'espêce
    colonne 3: le type
    colonne 4: la séquence
Le résultat est écrit dans nom_arbre
'''

from Arbre import Arbre
import sys
import csv

if len(sys.argv) == 3 :
    fichier = open(sys.argv[1], 'r')
else :
    print "Usage: %s fichier-csv-trie-au-bon-format nom_arbre" % sys.argv[0]
    sys.exit()

racine=Arbre()
racine.name='Papillomaviridae'
t=racine
t.add_features(hauteur='0')
t.add_features(nomTaxon='Papillomaviridae')

lignesCvs= csv.reader(fichier, delimiter=',')

genre = espece = tipe = ""

for l in lignesCvs :
    if genre != l[0] :
        genre = l[0]
        genreCourant = racine.add_child(name=genre)
        genreCourant.add_features(hauteur='1')
        genreCourant.add_features(nomTaxon=genre)

    if espece != l[1] :
        espece = l[1]
        especeCourant = genreCourant.add_child(name=espece)
        especeCourant.add_features(hauteur='2')
        especeCourant.add_features(nomTaxon=espece)
    if tipe != l[2] :
        tipe = l[2]
        tipeCourant = especeCourant.add_child(name=tipe)
        tipeCourant.add_features(hauteur='3')
        tipeCourant.add_features(nomTaxon=tipe)

    feuille = tipeCourant.add_child(name=l[3])
    feuille.add_features(hauteur='3')
    feuille.add_features(nomTaxon=l[4])

racine.write(format=2, outfile=sys.argv[2], features=[])
```

A.6 divergence.py

Calculer la dissimilarité des séquences entre elles.

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
'''
Usage: %s fichier_fasta

Calcule la dissimilarité des séquences entre elles

fichier_fasta: fichier contenant toutes les séquences au format FASTA

en sortie: le même nom dwe fichier avec l'extension '.dissim.txt' au format JSON
           représentant une matrice carrée contenant où les étiquettes de lignes
           et de colonnes sont les noms de séquences, et les cellules
           a dissimilarité entre elles.
'''
from Bio import Entrez
from Bio import SeqIO
from BioSQL import BioSeqDatabase
import os
import sys
import datetime
import argparse
import fileinput
import subprocess
from pprint import pprint
import json

if len(sys.argv) == 2 :
    records = SeqIO.parse(sys.argv[1], "fasta")
    nomFichierOut = os.path.basename(sys.argv[1]) + ".dissim.txt"
else :
    print __doc__ % sys.argv[0]
    sys.exit()

verbose = True
debut = datetime.datetime.now()
print 'début:␣' ,debut

#mettre les séquences dans une liste
sequences=[]
for rec in records :
    sequences.append(rec)

nbSeq = len(sequences)
nbAlign = 0
dissimilarite = {}

#pour chaque séquence
for i in range(0, nbSeq - 1) :
    if sequences[i].name not in dissimilarite :
        dissimilarite[sequences[i].name] = {}
    #divergence avec elle-même
    dissimilarite[sequences[i].name][sequences[i].name] = 0.0
    #pour chaque séquence qui reste
    for j in range(i + 1, nbSeq):
        if verbose :
            print "alignement:", sequences[i].name, sequences[j].name
        #faire l'alignement
        nw = subprocess.Popen(["muscle", "-quiet"], stdout=subprocess.PIPE,
```

```
                                stdin=subprocess.PIPE, stderr=subprocess.STDOUT)
            output = nw.communicate(input=sequences[i].format("fasta") +
                                    sequences[j].format("fasta"))[0]
            #récupérer le resultat
            blocs = output.split('>')
            lignes = blocs[1].splitlines()
            nomA = lignes[0]
            del lignes[0]
            alA = ''.join(lignes)
            lignes = blocs[2].splitlines()
            nomB = lignes[0]
            del lignes[0]
            alB = ''.join(lignes)
            lgAl = len(alA)
            if (lgAl != len(alB)) :
                print "alignement %s %s de longueur invalide\n" % (nomA, nomB)
                sys.exit()

            #calculer la dissimilarité
            lgNom = max(len(nomA), len(nomB))
            alDiff = ""
            nbDiff = 0
            score = 0
            matches = 0
            mismatches = 0
            gaps = 0
            for pos in range(0, lgAl) :
                if alA[pos] != alB[pos] :
                    alDiff += '#'
                    nbDiff += 1
                    if alA[pos] == '-' or alB[pos] == '-' :
                        gaps += 1
                    else :
                        mismatches += 1
                else :
                    alDiff += ' '
                    matches += 1

            score = matches * 3 + mismatches * -1 + gaps * -2
            percDiff = nbDiff /float(lgAl)
            dissimilarite[sequences[i].name][sequences[j].name] = percDiff

            # on enregistre aussi la symétrique
            if sequences[j].name not in dissimilarite :
                dissimilarite[sequences[j].name] = {}
                dissimilarite[sequences[j].name][sequences[j].name] = 0.0
            dissimilarite[sequences[j].name][sequences[i].name] = percDiff

            nbAlign += 1

            if verbose :
                print "%-*s %s" % (lgNom, nomA, alA)
                print "%-*s %s" % (lgNom, "", alDiff)
                print "%-*s %s" % (lgNom, nomB, alB)
                print "%d/%d divergences (%.3f)" % (nbDiff, lgAl, percDiff)
                print "%d matches, %d mismatches, %d gap, score: %d" % \
                      (matches, mismatches, gaps, score)
                print
    #enregistrer au format JSON trié
    json.dump(dissimilarite, open(nomFichierOut,'w'), sort_keys=True, indent=2)

    fin = datetime.datetime.now()
```

```
print 'fin:␣', fin
duree = fin - debut
dureeSecondes = (duree.microseconds + (duree.seconds +
                                duree.days * 24 * 3600) * 10**6) / float(10**6)
print "durée:␣%.2f␣secondes,␣%d␣séquences,␣%d␣alignements,␣%.2f␣align/sec␣" % \
    (dureeSecondes, nbSeq, nbAlign, nbAlign/dureeSecondes)
```

A.7 distSimilarite.py

Produire un histogramme de la distribution de la matrice de dissimilarité.

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
'''
Usage: %s matriceDeDissimilarite nomPlot sequencesAinclure options titre

Produit un histogramme de la distribution de la matrice.

matriceDeDissimilarite : fichier JSON des dissimilarités
nomPlot:    nom du fichier de résultat auquel sera ajout '.pdf'
sequencesAinclure : fichier contenant le noms des séquences à inclure ou exclure
options: 1==inclure seulement les séquences présentes dans sequencesAinclure
         0==enlever les séquences présentes dans sequencesAinclure
titre: titre à ajouter au graphique
'''

import os
import sys
import datetime
import argparse
import fileinput
import subprocess
from pprint import pprint
import json
import numpy

# ##########################
#pour plotter histogramme de la distribution de la matrice de similarite
scriptRhisto = '''
args <- commandArgs(trailingOnly = TRUE)
print(args)
if (length(commandArgs(trailingOnly = TRUE)) != 5) {
    print("mauvais arguments1")
    quit(save = "no", status = -1, runLast = FALSE)
}
library(rjson)
library(ggplot2)
library(scales)

#convert from JSON into a list in R
#le fichier doit avoir un CR a la derniere ligne
tmp=fromJSON(file=args[1])

#coerce this to a data.frame
dissim=do.call(rbind,tmp)
dissimN = apply(dissim, c(1,2),as.numeric)
rownames(dissimN)=rownames(dissim)
colnames(dissimN)=colnames(dissim)

#les sequences voulues / non voulues
seq1=read.table(args[3],stringsAsFactors=FALSE)
```

```
seq2=do.call(c,seq1)
if (args[4] == '0') {
    #enlever les sequences
    dissimAtraiter= dissimN[!rownames(dissimN) %in% seq2,
                            !colnames(dissimN) %in% seq2]
} else {
    #garder seulement les sequences voulues
    seqAtraiter = intersect(seq2, row.names(dissimN))
    dissimAtraiter= dissimN[seqAtraiter, seqAtraiter]
}

#ne garder que la matrice triangulaire superieure, sans la diagonale
dissimAtraiter[lower.tri(dissimAtraiter, diag=TRUE)] <- NA

#matrice de similarite plutot que dissimilarite
dissimAtraiter = 1 - dissimAtraiter
fich=paste(args[2], "3.csv", sep="")
write.table(dissimAtraiter, file=fich, sep=",", row.names=TRUE, col.names=TRUE,
            quote=FALSE)

leMin = min(dissimAtraiter[upper.tri(dissimAtraiter, diag=FALSE)])
leMin = floor(leMin*10) * 20

#faire l'histogramme
pdf(paste(args[2], ".pdf", sep=""))

h=hist(dissimAtraiter, breaks=c(leMin:200)/200, plot=FALSE)
h$density = h$counts/sum(h$counts)
plot(h,freq=F, border=colors()[153], col=colors()[200], main=args[5],
     xlab="Pourcentage d'identité", ylab="Fréquence")

length(h$breaks) = length(h$breaks) - 1
res=cbind(h$breaks, h$counts, h$density)
colnames(res) = c("breaks", "count", "percent")
format(res, scientific=FALSE)
options(scipen=20)
write.table(res, file=paste(args[2], "Tableau.csv", sep=""), sep=",",
            quote=FALSE,row.names=FALSE)

dev.off()
warnings()
quit(save = "no")
'''

# ########################

if len(sys.argv) != 6 :
    print __doc__ % sys.argv[0]
    sys.exit()

R = subprocess.Popen("R␣␣--vanilla␣--quiet␣--slave␣--no-save␣--args" \
                     "%s␣%s␣%s␣%s␣'%s'" % (sys.argv[1], sys.argv[2], sys.argv[3],
                     sys.argv[4], sys.argv[5]),
                     shell=True, stdout=subprocess.PIPE, stdin=subprocess.PIPE,
                     stderr=subprocess.STDOUT)
output = R.communicate(input=scriptRhisto)[0] #attend la fin automatiquement
print output
```

A.8 pourcentage_arbre.py

Vérifier les seuils de pourcentages selon les valeurs prédéfinies.

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
'''
Usage: %s arbre format fichier_dissimilarite affichage(1 ou 0)

Vérifie les seuils de pourcentages selon les valeurs prédéfinies

arbre: fichier Newick de l'arbre à analyser
format: format de l'arbre à analyser (voir ETE)
fichier_dissimilarite : matrice de dissimilarité au format JSON
affichage: si 1, l'arbre est affiché
'''
import sys
import logging
import json
import unicodedata
import operator
from Arbre import Arbre

#seuils dans l'ordre famille, genre, espèce et type
seuils = [
    {"classe": "famille", "intra" : 0.0, "minInter": 0.0, "maxInter": 1.0 },
    {"classe": "genre", "intra" : .6, "minInter": 0.0, "maxInter": .6 },
    {"classe": "espece", "intra" : .7, "minInter": 0.6, "maxInter": .71 },
    {"classe": "type", "intra" : .9, "minInter": 0.71, "maxInter": .90 }
    ]

#mesures dans l'ordre famille, genre, espèce et type
mesures = [
    {"minIntra":2.0, "nbDepasseIntra":0, "nbCompIntra":0, "pourcDepIntra":0.0,
        "minInter":2.0, "nbDepasseMinInter":0, "maxInter":-1.0,
        "nbDepasseMaxInter":0, "nbCompInter":0, "pourcDepInter":0.0},
    {"minIntra":2.0, "nbDepasseIntra":0, "nbCompIntra":0, "pourcDepIntra":0.0,
        "minInter":2.0, "nbDepasseMinInter":0, "maxInter":-1.0,
        "nbDepasseMaxInter":0, "nbCompInter":0, "pourcDepInter":0.0},
    {"minIntra":2.0, "nbDepasseIntra":0, "nbCompIntra":0, "pourcDepIntra":0.0,
        "minInter":2.0, "nbDepasseMinInter":0, "maxInter":-1.0,
        "nbDepasseMaxInter":0, "nbCompInter":0, "pourcDepInter":0.0},
    {"minIntra":2.0, "nbDepasseIntra":0, "nbCompIntra":0, "pourcDepIntra":0.0,
        "minInter":2.0, "nbDepasseMinInter":0, "maxInter":-1.0,
        "nbDepasseMaxInter":0, "nbCompInter":0, "pourcDepInter":0.0}
    ]

# ####################
# trouve le minimum de dissimilarité entre la séquence 'nom' et l'ensemble de
# séquences 'feuilles' à l'aide de la matrice de dissimilarité 'matDissim'
# le résultats est une liste de [nom, dissimilarite] des séquences les plus près
def minDissim (nom, matDissim, feuilles) :
    minSeq = []
    if nom in matDissim :
        #toutes les mesures pour cette séquence
        valeurs = sorted(matDissim[nom].items(), key=operator.itemgetter(1))
        #valeurs==[nom, dist]
        for i in range(0, len(valeurs)) :
            val = valeurs[i]
            # on veut le minimum, pas la dissimilarité avec elle-même
            if val[0] != nom  and val[0] in feuilles :
                #il y a peut-etre plusieurs elements minimums
```

```
                if len(minSeq) > 0 and val[1] > minSeq[0][1] :
                    break
                minSeq.append(val)
    return minSeq


# ####################
# vérifiew les pourcenatages de similarité de l'arbre 'racine' selon la matrice
# dissimilarité 'dissim'
# le résultats est affiché sur stdout

def dissimIntraInter (dissim, racine) :
    while len(racine.children) == 1 :
        racine = racine.children[0]

    tousNomsFeuilles = set()
    tousNoeudsFeuilles = set(racine.get_leaves())
    for f in tousNoeudsFeuilles :
        #on enlève les feuilles pour lesquelles il n'y a pas de mseure pour ne
        #pas avoir à faire ces tests dans le reste de la fonction
        if f.name not in dissim:
            if 'pasDeMesure' not in f.features :
                f.add_features(pasDeMesure=True)
        else :
            tousNomsFeuilles.add(f.name)


    #verifier la proximite
    for f in tousNomsFeuilles :
        voisins = minDissim(f, dissim, tousNomsFeuilles)
        for v in voisins :
            if not racine.memeSousArbreMinimum(f, v[0]) :
                ancCommun = racine.ancetreCommunParNom([f, v[0]])
                h = int(ancCommun.hauteur) + 1
                sim = 1.0 - v[1]
                if not (seuils[h]["minInter"] <= sim < seuils[h]["maxInter"]):
                    t1Parent = racine.trouverTaxonParent(f)
                    t2Parent = racine.trouverTaxonParent(v[0])
                    print "proximite non conforme %s(%s), plusProche:," \
                          "%s(%s), %.10f, ancCommun %s(%s)" % \
                          (f, t1Parent,v[0], t2Parent, sim, ancCommun.nomTaxon,
                           ancCommun.name)

    #verifier les seuils de pourcentage intraclasse et interclasse
    for noeud in racine.traverse() :
        if noeud.name == '1070417' :
            pass
        if noeud.is_leaf() or int(noeud.hauteur) > 3:
            continue
        h = int(noeud.hauteur)
        assert(0 <= h <= 3)
        maxNoeud = [-1.0, "", ""]
        #seulement celles avec une mesure
        feuillesIntra = set(noeud.get_leaf_names()) & tousNomsFeuilles
        feuillesInter = set()
        for s in noeud.get_sisters() :
            feuillesInter.update(set(s.get_leaf_names()))
        feuillesInter &= tousNomsFeuilles #seulement celles avec une mesure

        #intra-classes et inter-classes
        minIntra = 2.0 #valeur minimum de la similarite intra
        minInter = 2.0 #valeur minimum de la similarite inter
```

```
        maxInter = -1.0 #valeur maximum de la similarite inter
        reste = feuillesIntra
        nbCompIntra = 0
        nbCompInter = 0
        nbDepasseIntra = 0 #nombre de comparaison qui depassent le seuil
        nbDepasseMinInter = 0 #nombre de comparaison qui depassent le seuil
        nbDepasseMaxInter = 0 #nombre de comparaison qui depassent le seuil
        for fi in feuillesIntra :
            reste = reste - set([fi]) #pas faire comparaisons symetriques
            for fj in reste:
                nbCompIntra += 1
                sim = 1.0 - dissim[fi][fj]
                if sim < minIntra :
                    minIntra = sim
                if sim < seuils[h]["intra"] :
                    nbDepasseIntra += 1

            for fj in feuillesInter:
                nbCompInter += 1
                sim = 1.0 - dissim[fi][fj]
                if sim < minInter :
                    minInter = sim
                if sim > maxInter :
                    maxInter = sim
                if sim < seuils[h]["minInter"] :
                    nbDepasseMinInter += 1
                if sim >= seuils[h]["maxInter"] :
                    nbDepasseMaxInter += 1
        if nbDepasseIntra > 0 :
            #intra-classe
            print "3 intra-classe noeud %s(%s), minIntra %f, nbDepasseIntra %d"\
                  ", sur %d : %f" % (noeud.nomTaxon,noeud.name,minIntra,
                  nbDepasseIntra,nbCompIntra, nbDepasseIntra/float(nbCompIntra))
        if nbDepasseMinInter > 0  or nbDepasseMaxInter > 0 :
            print "4 inter-classe noeud %s(%s), minInter %f, nbDepasseMinInter"\
                  " %d : %f maxInter %f, nbDepasseMaxInter %d : %f,"\
                  " sur %d comparaisons" % (noeud.nomTaxon, noeud.name,
                  minInter, nbDepasseMinInter,
                  nbDepasseMinInter/float(nbCompInter), maxInter,
                  nbDepasseMaxInter, nbDepasseMaxInter/float(nbCompInter),
                  nbCompInter)
            pass

        #faire les statistiques
        if minIntra < mesures[h]["minIntra"] :
            mesures[h]["minIntra"] = minIntra
        if minInter < mesures[h]["minInter"] :
            mesures[h]["minInter"] = minInter
        if maxInter > mesures[h]["maxInter"] :
            mesures[h]["maxInter"] = maxInter
        mesures[h]["nbCompIntra"] += nbCompIntra
        mesures[h]["nbDepasseIntra"] += nbDepasseIntra
        mesures[h]["nbCompInter"] += nbCompInter
        mesures[h]["nbDepasseMinInter"] += nbDepasseMinInter
        mesures[h]["nbDepasseMaxInter"] += nbDepasseMaxInter

    print "============================\nSommaire:"
    i = 0
    formatMesure = "{0:7} {1:12} {2:12} {3:12} {4:12} {5:12} {6:14} {7:14} "\
                   "{8:14} {9:14} {10:14} {11:14} {12:14} {13:14}"
    print formatMesure.format("classe", "seuilIntra", "minIntra", "depIntra",
          "compIntra", "pcDepIntra", "seuilMinInter", "minInter", "depMinInter",
```

```
            "seuilMaxInter", "maxInter", "depMaxInter", "compInter", "pcDepInter")
    for m in mesures :
        if m["nbCompIntra"] > 0 :
            m["pourcDepIntra"] = m["nbDepasseIntra"]/float(m["nbCompIntra"])
        if m["nbCompInter"] > 0 :
            m["pourcDepInter"] = (m["nbDepasseMinInter"] +
                                  m["nbDepasseMaxInter"])/float(m["nbCompInter"])
        print formatMesure.format(seuils[i]["classe"], seuils[i]["intra"],
            m["minIntra"], m["nbDepasseIntra"], m["nbCompIntra"],
            m["pourcDepIntra"], seuils[i]["minInter"], m["minInter"],
            m["nbDepasseMinInter"], seuils[i]["maxInter"],m["maxInter"],
            m["nbDepasseMaxInter"], m["nbCompInter"], m["pourcDepInter"])

        i += 1

# ##############################################################################
logging.basicConfig(format='%(levelname)s:%(message)s', level=logging.DEBUG)

if len(sys.argv) == 5 :
    arbre = Arbre(sys.argv[1], format=int(sys.argv[2]))
    matDissim = json.load(open(sys.argv[3]))
    affichage = sys.argv[4] == '1'
else :
    print __doc__ % sys.argv[0]
    sys.exit()

arbre.baptiserNoeuds()
dissimIntraInter(matDissim, arbre)

if affichage :
    arbre.mettreTitre(sys.argv[1] + " et " + sys.argv[3])
    arbre.afficher(support=False, longBranche=False)
```

A.9 generer-jobs-phyml-gb.sh

Script de génération des travaux à l'ordonnanceur MOAB.

```
#!/bin/bash
genes=(E1 E2 E4 E5 E5A E5B E6 E7 L1 L2)
pfixGene="gene"

header1="#!/bin/bash
#PBS -N laJob       #Nom de la tâche
#PBS -A hvg-164-aa  #Identifiant Rap; ID
#PBS -l procs=32    #Nombre de noeuds et nombre de processus par noeud
#PBS -l walltime=12:00:00:00    #Durée en secondes
#PBS -o laJob.out
#PBS -e laJob.err
#PBS -M bruno.daigle@gmail.com
#PBS -m bea
#PBS -S /bin/bash

# Commande à exécuter
cd \"\${PBS_O_WORKDIR}\"
date
"

#pour chaque gene
for i in ${!genes[*]}
do
    fichier=$pfixGene${genes[$i]}.muscle-gb
```

```
    job="job-phyml-"$fichier

    echo "${header1//laJob/$job}" > $job.sh
    echo mpiexec ./phyml-mpi -i $fichier.phy --no_memory_check -b 128 >> $job.sh
    echo "date" >> $job.sh
    msub $job.sh
done

#pour le genome complet
fichier=VPH.muscle-gb
job="job-phyml-"$fichier
echo "${header1//laJob/$job}" > $job.sh
echo mpiexec ./phyml-mpi -i $fichier.phy --no_memory_check -b 128 >> $job.sh
echo "date" >> $job.sh
msub $job.sh
```

A.10 comparer_arbre.py

Comparer deux arbres.

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
'''
Usage: %s arbre format autreArbre autreFormat [options]

Compare deux arbres

arbre: le premier arbre
format: le format du premier arbre
autreArbre: le deuxième arbre
autreFormat: le format du deuxième arbre

options: deux caractères parmi {0, 1}, par exemple '11'
    si options[0] est 1 le bootstrap sera normalisé de 128 à 100
    si options[1] est 1 l'arbre sera nivelé quand le bootstrap est < 100
'''

import sys
import logging
import time
from Arbre import Arbre

logging.basicConfig(format='%(levelname)s:%(message)s', level=logging.DEBUG)

if len(sys.argv) == 5 or len(sys.argv) == 6:
    racine = Arbre(sys.argv[1], format=int(sys.argv[2]))
    autreArbre = Arbre(sys.argv[3], format=int(sys.argv[4]))
else :
    print __doc__ % sys.argv[0]
    sys.exit()

normaliserBS = False
nivelerBS = False
if len(sys.argv) == 6:
    options = sys.argv[5]
    normaliserBS = options[0] == '1'
    nivelerBS = options[1] == '1'

#on ne garde que les feuilles présentes dans les 2 arbres
diff = racine.difference(autreArbre)
if len(diff) > 0 :
```

```
    print sys.argv[1], '-' , sys.argv[3], len(diff), 'feuilles:', diff
    racine.enleverFeuilles(diff)

diff = autreArbre.difference(racine)
if len(diff) > 0 :
    print sys.argv[3], '-' , sys.argv[1], len(diff), 'feuilles:', diff
    autreArbre.enleverFeuilles(diff)

if normaliserBS :
    racine.normaliserBootstrap(128)

if nivelerBS :
    racine.nivelerSelonBootstrap(85)

pareils = racine.comparer(autreArbre)
print '>>>>>>>>>>>>>>>> les arbres sont: ',
if pareils :
    print '"ÉGAUX"'
else:
    print 'différents'

racine.afficher(support=True, longBranche=True)
time.sleep(2)
autreArbre.afficher()

racine.write(format=1, outfile="new_tree1.nw", features=[])
```

A.11 Arbre.py

Classe dérivée de Tree() (ETE) qui implémentes diff.érentes fonctionnalités de manipulation des arbrees, dont la comparaison et l'affichage.

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
'''
classe dérivée de Tree() qui ajoute certaines méthodes
'''

from ete2 import Tree, faces, AttrFace, NodeStyle, TreeStyle, TextFace
import sys
import logging
import os
from os import environ
from itertools import combinations

#fonction utilitaire pour éliminer des noeuds de la liste aDetruire
def detruireNoeud(aDetruire) :
    for node in aDetruire:
        listeEnfants = node.get_children()
        for enfant in listeEnfants :
            #on ajoute la longeur de branche du noeud qui disparait aux enfants
            #immediats
            enfant.dist += node.dist
        node.delete()

#fonction utilitaire pour affichage, en call back, surtout pour débugguer
def my_layout(node):
    if node.is_leaf():
        # If terminal node, draws its name
        if 'pasDeMesure' not in node.features :
            name_faces = AttrFace("name", fsize=8)
```

```
        else :
            name_faces = AttrFace("name", fsize=8,
                                  text_prefix="␣pas␣de␣mesure␣pour␣",
                                  fstyle='italic')

        faces.add_face_to_node(name_faces, node, column=0,
                               position="branch-right")
        if 'plusPres' in node.features :
            faces.add_face_to_node(TextFace(node.plusPres, fsize=8), node,
                                           column=1,  position="branch-right")

    else:
        if node.name != Arbre.sansNom :
            if 'nomTaxon' in node.features :
                name_faces = AttrFace("nomTaxon", text_suffix="␣",
                                      text_prefix="␣", fsize=10)
                faces.add_face_to_node(name_faces, node, column=0,
                                       position="branch-top")
            name_faces = AttrFace("name", fsize=8)
            # Adds the name face to the image at the preferred position
            faces.add_face_to_node(name_faces, node, column=0,
                                   position="branch-top")

    if 'maxSousArbre' in node.features :
        faces.add_face_to_node(TextFace("dMax:␣%.1f%%" % \
                                        (node.maxSousArbre[0]*100), fsize=8),
                                        node, column=0,  position="branch-top")
        faces.add_face_to_node(TextFace(node.maxSousArbre[1], fsize=7), node,
                                        column=0,  position="branch-bottom")
        faces.add_face_to_node(TextFace(node.maxSousArbre[2], fsize=7), node,
                                        column=0,  position="branch-bottom")
    if 'cohesion' in node.features :
        faces.add_face_to_node(TextFace("cohesion:␣%.4f" % (node.cohesion),
                                        fsize=8), node, column=0,
                                        position="branch-bottom")
    if 'pasPareils' in node.features :
        faces.add_face_to_node(TextFace("pasPareils", fsize=10), node, column=0,
                               position="branch-bottom")

#la nouvelle classe elle-même dérivée de Tree()
class Arbre(Tree) :
    nonResolu = 'nonResolu'
    sansNom = 'NoName'
    nomAuto = 'n'
    _titre =''
    elaguer = True

    #par défaut, lors de l'affichage le nom sera celui du fichier utilisé pour
    #le créer
    def __init__(self, *args, **kw) :
        super(Arbre, self).__init__(*args, **kw)
        if len(args) >= 1 and len(args[0]) >= 1 :
            self._titre = args[0]

    #changer le titre pour l'affichage
    def mettreTitre(self, titre) :
        self._titre = titre

    #élimine les noeuds dont les noms sont dans 'feuilles' qui normalement
    #seraient des feuilles. Les parents qui deviennet des feuilles sont
    #enlevés également
    def enleverFeuilles (self, feuilles) :
```

```
        for feuille in feuilles :
            temp = self.search_nodes(name=feuille)
            if len(temp) != 1 :
                logging.warning('enleverFeuilles: probleme: %s', feuille)
            else :
                f = temp[0]
                #si le parent (qui est un noeud interne) n'a qu'une feuille, il
                #deviendra une feuille qu'il faut donc enlever
                while len(f.up.children) == 1 :
                    f = f.up
                f.detach()

    #assigne un nom à chaque noeud qui n'en n'a pas
    def baptiserNoeuds(self) :
        noeudsAnonymes = self.search_nodes(name=Arbre.sansNom)
        if len(noeudsAnonymes) > 0 :
            logging.debug("%s noeuds sans nom", len(noeudsAnonymes))

        for i, n in enumerate(noeudsAnonymes) :
            #tous les noeuds doivent avoir un nom car c'est le repère
            n.name = Arbre.nomAuto + str(i)

    #normalise la valeur de bootstrap de 'facteur' à 100
    def normaliserBootstrap(self, facteur) :
        for node in self.traverse():
            if not node.is_leaf():
                node.support = int((node.support / facteur) * 100)

    #enlève les noeuds dont le bootstrap est inférieur à 'limite'
    def nivelerSelonBootstrap(self, limite) :
        aDetruire = []
        for node in self.traverse():
            if not node.is_leaf() and node.support < limite \
               and not node.is_root():
                aDetruire.append(node)
        detruireNoeud(aDetruire)

    #retourne le noeud parent du noeud dont le nom est 'nom'
    def trouverNoeudParent (self, nom) :

        liste = self.search_nodes(name=nom)
        if len(liste) != 1 :
            print 'BUGGGG', nom, liste
        assert len(liste) == 1
        if len(liste) == 0:
            #print nom, ': non trouve'
            return None
        noeud = liste[0]
        #assert not noeud.is_root()
        if noeud.is_root() :
            return None
        return noeud.up

    #retourne le nom du taxon associé au noeud courant
    def trouverNomTaxon (self) :
        if 'nomTaxon' in self.features :
            return self.nomTaxon
        else :
            return "taxon??"

    #retourne le nom du noeud parent du noeud nommé 'nom'
    def trouverNomParent (self, nom) :
```

```
        # à modifier pour utiliser trouverNoeidParent
        liste = self.search_nodes(name=nom)
        if len(liste) != 1 :
            print 'BUGGGG', nom, liste
        assert len(liste) == 1
        if len(liste) == 0:
            print nom, ': non trouve'
            return 'noName'
        noeud = liste[0]
        if noeud.is_root() :
            return nom
        return noeud.up.name

    #retourne le nom du taxon du noeud parent du noeud nommé 'nom'
    def trouverTaxonParent (self, nom) :
        noeud = self.trouverNoeudParent(nom)
        if noeud is not None : #and 'nomTaxon' in noeud.features :
            return noeud.trouverNomTaxon()
        return "is None %s" % (nom)

    #initialise à la chaine vide le nom de tous les noeuds
    def initialiserNomNoeudInterne(self) :
        for node in self.traverse():
            if not node.is_leaf() :
                #node.name = Arbre.nonResolu
                node.name = ""

    #élimine les noeuds de plus haut niveau dont le nom est 'noName'
    def ecimerNoName(self) :
        #bug avec ete2 ? quand écrit arbre, il efface le nom de la racine
        #donc on ajoute un noeud avant d'écrire et on l'enlève en lisant ...
        while self.name == Arbre.sansNom and len(self.children) == 1 :
            self = self.children[0]

    #retourne l'ancêtre commun le plus bas des noeuds dont les noms sont dans la
    #liste 'feuilles'. Dans le cas où la liste ne contient qu'un seul élément,
    #on retourne l'ancêtre le plus bas qui contient une bifurcation
    def ancetreCommunParNom (self, feuilles) :
        listeNoeuds = []
        for f in feuilles :
            #trouver le noeud de chaque nom de feuille
            noeuds = self.search_nodes(name=f)
            assert(len(noeuds) == 1)
            listeNoeuds.append(noeuds[0])
        assert(len(listeNoeuds) >= 1)

        if len(listeNoeuds) == 1 :
            #s'il n'y a qu'une seule feuille, on recherche le parent le plus
            #loin qui n'a qu'un seul descendant
            commAncestor = listeNoeuds[0]
            while len(commAncestor.up.children) == 1 :
                commAncestor = commAncestor.up
        else :
            commAncestor = self.get_common_ancestor(listeNoeuds)
        #print 'commAncestor: ', commAncestor.name
        return commAncestor

    #élimine le sous-arbre 'noeud' si aucun des noeuds sous lui n'a été
    #marqué avec l'attribut 'pasPareils'
    def elaguerFeuilles (self, noeud) :
        aDetruire = []
        if 'pasPareils' in noeud.features and noeud.pasPareils :
```

```
            #c'est un noeud non resolu, on ne peut eliminer le sous-arbre
            return False
        for node in noeud.iter_descendants():
            aDetruire.append(node)
            if 'pasPareils' in node.features and node.pasPareils :
                #c'est un noeud non resolu, on ne peut eliminer le sous-arbre
                return False
        if not noeud.is_root() :
            while len(noeud.up.children) == 1 :
                #on enleve les parents qui n'ont qu'un enfant
                aDetruire.append(noeud)
                noeud = noeud.up
        detruireNoeud(aDetruire)
        return True

    #ajoute un attribut ('paspareils') aux noeuds de la liste 'setNoms'
    def marquerNoeud (self, setNoms) :

        for nom in setNoms :
            noeuds= self.search_nodes(name=nom)
            if len(noeuds) != 1 :
                logging.warning('marquerNoeud: probleme: %s', nom)
                print setNoms
            else :
                noeud = noeuds[0]
                if 'pasPareils' not in noeud.features :
                    noeud.add_features(pasPareils=True)
                else :
                    noeud.pasPareils = True

    #retourne une liste de feuilles qui sont dans arbre mais pas dans autreArbre
    def difference(self, autreArbre) :

        a = self.get_leaf_names()
        b = autreArbre.get_leaf_names()
        return list(set(a) - set(b))

    #compare 2 arbre et retourne vrai ou faux selon qu'ils sont "égaux" ou non
    #on cherche les sous-arbres égaux et on les enlève jusqu'à ce que l'arbre
    #soit vide ou qu'il n'y ait plus de progrès
    def comparer (self, autreArbre) :
        divergences = {}
        pareils = True
        self.ecimerNoName()
        autreArbre.ecimerNoName()
        autreArbre.baptiserNoeuds()
        self.initialiserNomNoeudInterne()

        #on va modifier les arbres, on utilise une copie
        arbreA = self.copy()
        arbreB = autreArbre.copy()

        changement = True
        while changement :
            changement = False
            for noeud in autreArbre.traverse(strategy='postorder') :
                if noeud.is_leaf() :
                    print 'skip traitement de feuille autreArbre', noeud.name
                    continue
                if len(noeud.children) == 1 :
                    print "noeud intermediaire inutile: ", noeud.name
                    continue
```

```
                feuillesAutreArbre = noeud.get_leaf_names()
                ancetre = self.ancetreCommunParNom(feuillesAutreArbre)
                feuillesAncetre = ancetre.get_leaf_names()
                setA = set(feuillesAncetre)
                setB = set(feuillesAutreArbre)
                if noeud.name == '337049' :
                    pass
                if setA != setB :
                    pareils = False
                    autreArbre.marquerNoeud([noeud.name])
                    print '================================ divergence avant: ',\
                        ancetre.name, 'noeud autreArbre', noeud.name
                    if ancetre.name == "" :
                        ancetre.name = "?" + noeud.name + "?"
                    #if ancetre not in divergences :
                    #    divergences[ancetre] = []
                    #divergences[ancetre].append(setB - setA)
                    #divergences[ancetre].append(setA - setB)
                    #autreArbre.marquerNoeud(divergences[ancetre][0])
                    #autreArbre.marquerNoeud(divergences[ancetre][1])

                else :
                    if not ancetre.is_leaf() :
                        if ancetre.name == "" :
                            ancetre.name = noeud.name
                    if Arbre.elaguer :
                        #on enlève les feuilles lorsque les arbres sont
                        #equivalents et qu'il n'y a pas de noeud interne
                        #non resolu
                        if autreArbre.elaguerFeuilles(noeud) :
                            self.elaguerFeuilles(ancetre)
                            changement = True
                            break

        return pareils

    #pour affichage
    def colorierNoeuds(self, noeuds, couleur) :
        if noeuds is not None :
            nst = NodeStyle()
            nst["fgcolor"] = couleur
            nst["faces_bgcolor"] = couleur
            nst["size"] = 10
            for n in noeuds :
                n.set_style(nst)

    #pour affichage
    def colorierNoms(self, noms, couleur) :
        if noms is not None :
            for n in noms :
                noeudsArbre = self.search_nodes(name=n)
                if len(noeudsArbre) == 0 :
                    logging.warning("%s non trouvé", n)
                self.colorierNoeuds(noeudsArbre, couleur)

    #affiche l'arbre
    def afficher(self, background=True, feuilles=False, topologie=True,
                 support=False, longBranche=False, noeuds=None, noms=None,
                 png=None) :

        ts = TreeStyle()
        # Do not add leaf names automatically
```

```
        ts.show_leaf_name = feuilles
        ts.force_topology = topologie
        ts.show_branch_support = support
        ts.show_branch_length = longBranche
        ts.show_scale = False
        ts.layout_fn = my_layout

        if noeuds is not None :
            for l in noeuds :
                self.colorierNoeuds(l[1], l[0])
        if noms is not None :
            for l in noms :
                self.colorierNoms(l[1], l[0])
        ts.title.add_face(TextFace(self._titre, fsize=20), column=1)

        if background :
            child_pid = os.fork()
            if child_pid == 0 :
                #dans l'enfant
                self.show(tree_style=ts)
                sys.exit()
        else :
            self.show(tree_style=ts)
        if png is not None :
            ts2 = TreeStyle()
            ts2.show_leaf_name = feuilles
            ts2.force_topology = topologie
            ts2.show_branch_support = support
            ts2.show_branch_length = longBranche
            ts2.show_scale = False
            self.render(png, tree_style=ts)

    #affichage en mode texte
    def imprimer(self) :
        print self.get_ascii(show_internal=True)

# ##############################################################################
#Programme principal
if __name__ == "__main__":
    print 'tests',
    arb = ["(GQ845442,JQ963485)NoName;",
           "(JQ963485,GQ845442)NoName;",
           "(JQ963484,(FN598907,(GQ845442,JQ963485)NoName)NoName)NoName;",
           "(JQ963484,((GQ845442,JQ963485)NoName,FN598907)NoName)NoName;",
           "(JQ963484,((FN598907,JQ963485)NoName,GQ845442)NoName)NoName;"]
    attendu = [[0,0,True], [1,1,True], [2,2,True], [3,3,True], [0,1,True],
               [2,3,True], [4,4,True], [2,4,False]]

    for test in attendu :
        print ".", test,
        t1 = Arbre(arb[test[0]])
        t2 = Arbre(arb[test[1]])
        assert(t1.comparer(t2) == test[2])
    print
    print len(attendu), "tests réussis"
```

A.12 dissim_arbre.py

Calculer l'indice de cohésion.

```
#! /usr/bin/env python
```

```
# -*- coding: utf-8 -*-
'''
Usage: %s arbre format fichier_dissimilarite affichage(1 ou 0)

Calcul de l'indice de cohésion

arbre: fichier Newick de l'arbre à analyser
format: format de l'arbre à analyser (voir ETE)
fichier_dissimilarite : matrice de dissimilarité au format JSON
affichage: si 1, l'arbre est affiché
'''
import sys
import logging
import json
import unicodedata
import operator
from Arbre import Arbre

# ######################
# trouve le minimum ou le maximum selon la fonction 'comp' de dissimilarité
# entre la séquence 'nom' et l'ensemble de
# séquences 'feuilles' à l'aide de la matrice de dissimilarité 'matDissim'
# le résultats est un tuple [nom, dissimilarite]

def compareDissim (seq, ensSeq, matDissim, comp, valInit) :
    if seq not in matDissim :
        # pas d'info sur cette séquence
        return None
    valCourant = valInit
    #pour une seule séquence le max ou min avec elle-même est 0
    seqLatch = (seq, 0.0)
    for seq2 in ensSeq :
        if seq2 in matDissim[seq] :
            d = matDissim[seq][seq2]
            if comp(d, valCourant) :
                valCourant = d
                seqLatch = (seq2, d)
    return seqLatch

# ######################
# trouve le minimum de dissimilarité entre la séquence 'nom' et l'ensemble de
# séquences 'feuilles' à l'aide de la matrice de dissimilarité 'matDissim'
# le résultats est une liste de [nom, dissimilarite] des séquences les plus près

def minDissim (nom, matDissim, feuilles) :
    minSeq = []
    if nom in matDissim :
        valeurs = sorted(matDissim[nom].items(), key=operator.itemgetter(1))
        # on veut le minimum, pas la dissimilarité avec la séquence elle-même
        # ni celle avec une séquence qui n'est pas dans l'arbre courant
        # e.g certaines n'ont pas le gène L1 ou E5 ou ...
        for i in range(0, len(valeurs)) :
            val = valeurs[i]
            if val[0] != nom  and val[0] in feuilles :
                #il y a peut-etre plusieurs elements minimums
                if len(minSeq) > 0 and val[1] > minSeq[0][1] :
                    break
                minSeq.append(val)
    return minSeq

# ######################
# calcul de l'indice de cohésion de l'arbre 'racine'
```

```
# à l'aide de la matrice de dissimilarité 'matDissim'
# les résultats sont sur stdout
#
# Pour chaque nud interne de l'arbre, la distance entre chaque paire de
# feuilles à l'intérieur du nud, ainsi que la distance minimale entre chaque
# feuille et les feuilles à l'extérieur du nud sont disponibles. Pour chaque
# feuille, lorsque la distance à l'interne avec une autre feuille est inférieure
# à la distance minimale à l'externe, la feuille voit son facteur
# augmenter de 1. Le \emph{facteur} final de la feuille, qui est le facteur
# divisé par le nombre de feuilles internes moins 1, est ajouté à l'indice du
# nud.
def dissimIntraInter (dissim, racine) :
    while len(racine.children) == 1 :
        racine = racine.children[0]

    if racine.is_leaf() :
        #rien à faire
        return

    tousNomsFeuilles = set()
    tousNoeudsFeuilles = set(racine.get_leaves())
    for f in tousNoeudsFeuilles :
        #on enlève les feuilles popur lesquelles il n'y a pas de mseure pour ne
        #pas avoir à faire ces tests dans le reste de la fonction
        if f.name not in dissim:
            if 'pasDeMesure' not in f.features :
                f.add_features(pasDeMesure=True)
        else :
            tousNomsFeuilles.add(f.name)

    for noeud in racine.traverse() :
        if noeud == racine : #noeud.is_root()
            #inutile de faire le traitement sur l'arbre complet puisque
            #inter sera vide
            continue

        if noeud.is_leaf() :
            continue

        maxNoeud = [-1.0, "", ""]
        #seulement celles avec une mesure
        feuillesIntra = set(noeud.get_leaf_names()) & tousNomsFeuilles
        feuillesInter = tousNomsFeuilles - feuillesIntra #complément

        cohesion = 0.0
        nbFeuillesIntra = len(feuillesIntra)
        for g in feuillesIntra :
            maxIntra = compareDissim(g, feuillesIntra, dissim, operator.gt, 0.0)
            minIntra = compareDissim(g, feuillesIntra - set([g]), dissim,
                                     operator.lt, float("inf"))
            minInter = compareDissim(g, feuillesInter, dissim, operator.lt,
                                     float("inf"))
            print ">>>>>>> Noeud: %s(%s): " % (noeud.trouverNomTaxon(),\
                                               noeud.name),
            if minInter is None or maxIntra is None or minIntra is None:
                print ",--absent--, %s, %s(%s), maxIntra:, --, --, minIntra:,"\
                      ", --, --, minInter, --, --, --" % \
                      (g, noeud.trouverNomTaxon(), noeud.name)
                continue
            else :
                print ",",
                if minInter[1] <= maxIntra[1] :
```

```
                    print '++++++++++++++++++++',
                if minInter[1] <= minIntra[1] :
                    print >> sys.stderr, 'bizarre', noeud.name, g, 'minInter:',\
                            minInter[0], minInter[1], 'minIntra:', minIntra[0],\
                            minIntra[1]

                print ",%s,␣%s(%s),␣maxIntra:,␣%s,␣%.10f,␣minIntra:,␣%s,"\
                      "␣%.10f,␣minInter:,␣%s(%s),␣%.10f" % (g, \
                      noeud.trouverNomTaxon(), noeud.name, maxIntra[0],\
                      maxIntra[1], minIntra[0], minIntra[1],\
                      racine.trouverTaxonParent(minInter[0]),\
                      minInter[0], minInter[1])
                if maxIntra[1] > maxNoeud[0] :
                    maxNoeud[0] = maxIntra[1]
                    maxNoeud[1] = maxIntra[0]
                    maxNoeud[2] = g

            voisins = minDissim(g, dissim, feuillesInter)
            if len(voisins) > 0 :
                plusProcheInter = voisins[0]
            else :
                plusProcheInter = None
            coefficientFeuille = 0.0
            for f in feuillesIntra - set([g]) :
                #calcul du coefficient d'appartenance de la feuille g au groupe
                #noeud
                if plusProcheInter is not None and g in dissim and \
                                                            f in dissim[g] :
                    deltaGF = dissim[g][f]
                    if deltaGF <= plusProcheInter[1] :
                        coefficientFeuille += 1
                else :
                    nbFeuillesIntra -= 1
            if nbFeuillesIntra > 1 :
                cohesion += coefficientFeuille / (nbFeuillesIntra - 1)

        if maxNoeud[0] >= 0.0 and nbFeuillesIntra > 1:
            if nbFeuillesIntra == 2 :
                # pas besoin de spécifier quelles sont les feuilles dont la
                # distance est max
                maxNoeud[1] = maxNoeud[2] = ""
            if 'maxSousArbre' not in noeud.features :
                noeud.add_features(maxSousArbre=maxNoeud)
            else :
                noeud.maxSousArbre = maxNoeud
        if nbFeuillesIntra > 1 :
            cohesion /= nbFeuillesIntra
            if 'cohesion' not in noeud.features :
                noeud.add_features(cohesion=cohesion)
            else :
                noeud.cohesion = cohesion
            print "cohesion␣Noeud:,␣%s(%s)␣:,␣%f,␣nbFeuillesIntra:,␣%d" % \
                  (noeud.trouverNomTaxon(), noeud.name, cohesion,\
                  nbFeuillesIntra)

# ##########################

logging.basicConfig(format='%(levelname)s:%(message)s', level=logging.DEBUG)

if len(sys.argv) == 5 :
    arbre = Arbre(sys.argv[1], format=int(sys.argv[2]))
    matDissim = json.load(open(sys.argv[3]))
```

```
    affichage = sys.argv[4] == '1'
else :
    print __doc__ % sys.argv[0]
    sys.exit()

arbre.baptiserNoeuds()
dissimIntraInter(matDissim, arbre)

if affichage :
    arbre.mettreTitre(sys.argv[1] + "␣et␣" + sys.argv[3])
    arbre.afficher(support=False, longBranche=False)
```

A.13 classerVPH.py

Classer les VPH selon la similarité.

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-
'''
Usage: %s matrice_dissimilarite liste_sequences [arbre_de_depart format]

produit un arbre selon les criteres de classification du VPH avec les sequences
de 'liste_sequences' et la matrice de dissimilarite
'matrice_dissimilarite' format JSON

'''

from Arbre import Arbre
import json
import sys

# ######################
# ajoute un noeud nommé 'nom' dans un arbre 'racine', à partir de la position
# relative 'racine' en faisant un déplacement de 'nbUp' noeuds parents puis
# en ajoutant 'nbDown' noeuds enfants

def ajouterFeuille(nom, racine, nbUp, nbDown) :
    courant = racine
    for n in range(0, nbUp) :
        courant = courant.up
    for n in range(0, nbDown) :
        courant = courant.add_child()

    courant.add_child(name=nom)

# ######################
# vérifie si, dans un arbre 'racine', il y a dans les noeuds soeurs du parent
# du noeud 'plusProche' qui contiennent une feuille qui est à la même distance
# ou plus près que 'minimum' de la feuille 'f' à l'aide de la matrice de
# dissimilarité 'matDissim'

def verifierAutrePlusProche(f, plusProche, minimum, feuilles, matDissim,
                            racine) :
    dissimF = matDissim[f]
    parentPlusProche = racine.search_nodes(name=plusProche)[0].up
    soeursPlusProche = parentPlusProche.get_leaf_names()

    for v in feuilles :
        if v == plusProche :
            continue
        delta = abs(dissimF[v] - minimum)
```

```
        if delta <= 0.0005 :
            #print "petit delta %s %s %f" %(f, v, delta),
            if v not in soeursPlusProche :
                print "autre choix pour %s, %s delta %f" %(f, v, delta)


# ######################
# trouve la feuille la plus proche de 'f' parmi les 'feuilles' à l'aide de la
# matrice de dissimilarité 'matDissim'

def trouverPlusProche(f, feuilles, matDissim) :
    minimum = 2.0 #les valeurs ont entre 0 et 1
    dissimF = matDissim[f]
    for v in feuilles :
        if dissimF[v] < minimum :
            minimum = dissimF[v]
            plusProche = v
    return plusProche, minimum


# ####################################################
# on crée un arbre et on ajoute séquentiellement chaque séquence du fichier en
# entrée selon la séquence la plus proche et en créant de nouveaux noeuds en
# fonction des seuils de similarité.

if 3 <= len(sys.argv) <= 5 :
    fSequences = open(sys.argv[2], 'r')
    matDissim = json.load(open(sys.argv[1]))
else :
    print  __doc__ % sys.argv[0]
    sys.exit()

feuilles = []
if len(sys.argv) > 3 :
    if len(sys.argv) == 5 :
        #on commence avec cet arbre
        racine = Arbre(sys.argv[3], format=int(sys.argv[4]))
        feuilles = racine.get_leaf_names()
    else :
        print  __doc__ % sys.argv[0]
        sys.exit()
else :
    #on cree un nouvel arbre
    racine=Arbre()
    racine.name = 'papillomavirus'
    #on ajoute la premiere sequence
    f = fSequences.next().split('\n')[0]
    ajouterFeuille(f, racine, 0, 3)
    feuilles.append(f)


for s in fSequences :
    f = s.split('\n')[0]
    if f in feuilles :
        #deja classee
        continue
    plusProche, dissim = trouverPlusProche(f, feuilles, matDissim)
    verifierAutrePlusProche(f, plusProche, dissim, feuilles, matDissim, racine)

    similarite = 1 - dissim
    print "ajout %s, +proche %s, similarite %f" % (f, plusProche, similarite),
```

```
        similarite = 1 - dissim
        if similarite >= 0.90 :
            print "meme type"
            nbUp = 0
        elif similarite >= 0.71 :
            print "meme espece"
            nbUp = 1
        elif similarite >= 0.60 :
            print "meme genre"
            nbUp = 2
        else :
            print "autre genre"
            nbUp = 3

        noeudsPlusProche = racine.search_nodes(name=plusProche)
        if len(noeudsPlusProche) == 0 :
            print f, "non trouve dans arbre"
        else:
            ajouterFeuille(f, noeudsPlusProche[0], nbUp + 1, nbUp)
            feuilles.append(f)

racine.afficher()
racine.write(format=2, outfile='tmp.nw', features=[])
```

APPENDICE B

EXEMPLES

B.1 Exemple de fichier GenBank

```
LOCUS       NC_017995              7236 bp    DNA             VRL 04-JUN-2012
DEFINITION  Human papillomavirus type 137, complete genome.
ACCESSION   NC_017995
VERSION     NC_017995.1  GI:389656416
DBLINK      BioProject:PRJNA167867
KEYWORDS    .
SOURCE      Human papillomavirus type 137
  ORGANISM  Human papillomavirus type 137
            Viruses; Papillomaviridae; Human papillomavirus type 137.
REFERENCE   1  (bases 1 to 7236)
  AUTHORS   Bottalico,D., Chen,Z., Dunne,A., Ostoloza,J., McKinney,S., Sun,C.,
            Schlecht,N.F., Fatahzadeh,M., Herrero,R., Schiffman,M. and Burk,R.D.
  TITLE     The oral cavity contains abundant known and novel human
            papillomaviruses from the Betapapillomavirus and Gammapapillomavirus
            genera
  JOURNAL   J. Infect. Dis. 204 (5), 787-792 (2011)
   PUBMED   21844305
REFERENCE   2  (bases 1 to 7236)
  TITLE     Direct Submission
  JOURNAL   Submitted (02-JUN-2012) National Center for Biotechnology
            Information, NIH, Bethesda, MD 20894, USA
REFERENCE   3  (bases 1 to 7236)
  AUTHORS   Chen,Z., Sun,C., Bottalico,D. and Burk,R.
  TITLE     Direct Submission
  JOURNAL   Submitted (29-JUL-2010) Microbiology and Immunology, Albert Einstein
            College of Medicine of Yeshiva University, 1300 Morris Park Ave,
            Ullmann 515, Bronx, NY 10461, USA
COMMENT     PROVISIONAL REFSEQ: This record has not yet been subject to final
            NCBI review. The reference sequence is identical to HM999989.
            COMPLETENESS: full length.
FEATURES             Location/Qualifiers
     source          1..7236
                     /mol_type="genomic DNA"
                     /collection_date="2009"
                     /country="USA"
                     /isolate="NJ2801H"
                     /db_xref="taxon:1070410"
                     /host="Homo sapiens"
                     /organism="Human papillomavirus type 137"
     gene            1..429
                     /db_xref="GeneID:12983976"
                     /locus_tag="A400_gp1"
```

```
                     /gene="E6"
     CDS             1..429
                     /product="transforming protein"
                     /codon_start="1"
                     /locus_tag="A400_gp1"
                     /db_xref="GI:389656417"
                     /db_xref="GeneID:12983976"
                     /translation="MEQIFPTKLNDYCSYFEISFFDLSLKCIFCRHYLTLVDLAKFHDK
                     DLSLVWRGNICYACCDKCICCSARYEANKHFQCTFKTDSLHSIVQKPLQEIDIRCYYCL
                     RVLTLVEKFDLIAQGKPTCLIRGYFRAPCTECLRKELY"
                     /gene="E6"
                     /protein_id="YP_006393295.1"

      ...
      ...

     gene            5235..6785
                     /db_xref="GeneID:12983975"
                     /locus_tag="A400_gp7"
                     /gene="L1"
     CDS             5235..6785
                     /product="major capsid protein"
                     /codon_start="1"
                     /locus_tag="A400_gp7"
                     /db_xref="GI:389656423"
                     /db_xref="GeneID:12983975"
                     /translation="MAVWVPNKGRLYLPPQRPVAKVLSTDDYIVGTDLYFHSSTDRLLT
                     VGHPFFDVLSTDQNTVDVPKVSGNQFRVFRLNLPDPNQFALIDTSIYNPEHERLVWRLV
                     GIEIDRGGPLGIGSTGHPLFNKLQDTENPSVYNGLISDQKDNRMNVAFDPKQNQLFIVG
                     CKPAVGQHWDKAEPCPNTRPPPGSCPPLKLVHSTIEDGDMSDIGLGNINFSDLSDDKSS
                     APLEIINSKCKWPDFALMTKDLFGDSAFFFGRREQLYARHQWCRDGLVGDAIPDEHFYF
                     NPNGQDPKPPQYQLGSSIYFTIPSGSLTSSESNIFGRPYWLHRAQGANNGIAWGNQLFV
                     TLLDNTHNTNFTISVSTESQTTYDKNKFKVYLRHAEEIEIEIVCQLCKVPLEADILAHL
                     YAMDPSILDNWQLAFVPAPPQTLEDTYRYIRSMATMCPADVPPKEPEDPYKDLHFWTIN
                     LTDRFTSELDQTPLGKRFLYQMGLLTGNKRLRTDYIGSPVAKRRRTVKSSKRKKSSAK"
                     /gene="L1"
                     /protein_id="YP_006393301.1"
ORIGIN
        1 atggagcaga tatttccaac gaagctaaat gattattgct cttactttga aataagtttt
       61 tttgacttat ctttaaaatg tatcttctgt agacattatt tgactttggt agatttagca
      121 aaatttcatg ataaagacct ttctttagtt tggaggggta atatttgtta tgcttgttgt
      181 gataaatgta tttgttgtag tgctagatac gaagctaata agcattttca atgtacattt
      241 aaaactgata gtttgcattc tatagttcag aagcctttgc aagaaattga tattcgttgt
      301 tattactgtt tacgtgtttt aacattggtg gaaaaatttg atttgattgc tcaagggaag
      361 cctacttgtt taattagagg atactttaga gcgccttgta cagagtgttt gcgaaaagaa
      421 ttgtattaga atgaggggta attctccaac tgttgaaaat attgaactag atcttgagcc
      481 tctggtttta cctgcaaatc tattaagcaa cgaatctttg tcatcagatg aggaattaga
      541 ggaggagcgt gattatagtc cttttcaaat tgacagctat tgtcattctt gtcaaaggag
      601 agttagagtt tgtgttgtgg ctcgcgctgg agccattcaa tcactggaag tacatttatt
      ...
      ...
     6901 aatcctcctg tgtttatttg gcacttcaaa acatatctat agacaaaggc tgaagattgg
     6961 acagctgtgc cttttggcta tcaaaacaac cactttcggt aagtaaaaag gcgccaaaat
     7021 gtcaaacaaa ccgttatcgg tcgtcttcag tgagttggct tgtaaaaacg aatcgtatga
     7081 gtcaacaatc atgcctttgg ctgtcagcct ttgtaccggg agtggtaact tccttggcat
     7141 ttccttctga ttgttggcaa caattatgat ttcaataaaa agcaaccggt agaggtacaa
     7201 tataaaagag ctgagcttgg gagaattttt ggccag
//
```

B.2 VPH dont le gène L1 diffère de GenBank à PaVE

PaVE	GenBank	Div/Tot	Dissim	Match	Mismatch	Gap
HPV1REF	NC_001356	18/1527	0.012	1509	0	18
HPV2REF	NC_001352	48/1533	0.031	1485	0	48
HPV3REF	X74462	84/1599	0.053	1515	0	84
HPV10REF	NC_001576	84/1596	0.053	1512	0	84
HPV16REF	NC_001526	85/1599	0.053	1514	1	84
HPV18REF	NC_001357	187/1707	0.110	1520	4	183
HPV19REF	X74470	87/1641	0.053	1554	0	87
HPV27REF	X74473	327/1785	0.183	1458	0	327
HPV30REF	X74474	21/1527	0.014	1506	0	21
HPV34REF	NC_001587	78/1587	0.049	1509	0	78
HPV41REF	NC_001354	210/1752	0.120	1542	0	210
HPV42REF	M73236	81/1590	0.051	1509	0	81
HPV43REF	AJ620205	84/1596	0.053	1512	0	84
HPV45REF	X74479	78/1620	0.048	1542	0	78
HPV52REF	X74481	78/1590	0.049	1512	0	78
HPV54REF	NC_001676	6/1500	0.004	1494	0	6
HPV56REF	X74483	105/1605	0.065	1500	0	105
HPV57REF	X55965	48/1533	0.031	1485	0	48
HPV58REF	D90400	78/1575	0.050	1497	0	78
HPV67REF	D21208	87/1590	0.055	1503	0	87
HPV72REF	X94164	84/1605	0.052	1521	0	84
HPV74REF	AF436130	78/1587	0.049	1509	0	78
HPV77REF	Y15175	186/1698	0.110	1512	0	186
HPV81REF	AJ620209	84/1599	0.053	1515	0	84
HPV87REF	AJ400628	183/1698	0.108	1515	0	183
HPV89REF	AF436128	48/1557	0.031	1509	0	48
HPV91REF	AF419318	276/1785	0.155	1509	0	276
HPV94REF	AJ620211	84/1599	0.053	1515	0	84
HPV97REF	DQ080080	78/1602	0.049	1524	0	78
HPV99REF	FM955838	84/1632	0.051	1548	0	84
HPV100REF	FM955839	108/1629	0.066	1521	0	108
HPV101REF	NC_008189	117/1665	0.070	1548	0	117
HPV102REF	DQ080083	432/1944	0.222	1512	0	432
HPV104REF	FM955840	78/1587	0.049	1509	0	78
HPV105REF	FM955841	18/1563	0.012	1545	0	18
HPV106REF	DQ080082	183/1704	0.107	1521	0	183
HPV117REF	GQ246950	84/1602	0.052	1518	0	84

Tableau B.1: Liste des VPH dont la séquence de nucléotides du gène L1 diffère de GenBank à PaVE. Les colonnes dans l'ordre sont : *PaVE* - identificateur PaVE de la séquence, *GenBank* - le numéro d'accession de la séquence correspondante de GenBank, *Div/Tot* - le nombre de caractères divergents / la longueur de l'alignement, *Dissim* - la dissimilarité soit le résultat de *Div/Tot*, *Match* - le nombre de caractères égaux, *Mismatch* - le nombre de caractères différents à l'exception des gaps, *Gap* - le nombre de gaps.

BIBLIOGRAPHIE

Achtert, E., S. Goldhofer, H.-P. Kriegel, E. Schubert et A. Zimek. 2012. « Evaluation of clusterings – metrics and visual support ». *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, vol. 0, p. 1285–1288.

Altschul, S. F., W. Gish, W. Miller, E. W. Myers et D. J. Lipman. 1990. « Basic local alignment search tool ». *Journal of Molecular Biology*, vol. 215, no. 3, p. 403–410.

Antonsson, A., O. Forslund, H. Ekberg, G. Sterner et B. G. Hansson. 2000. « The ubiquity and impressive genomic diversity of human skin papillomaviruses suggest a commensalic nature of these viruses ». *Journal of Virology*, vol. 74, no. 24, p. 11636–11641.

Ball, G., et D. Hall. 1965. « Isodata, a novel method of data analysis and pattern classification ». *Technical report/Stanford research institute.*

Benson, D. A., M. Cavanaugh, K. Clark, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell et E. W. Sayers. 2013. « Genbank ». *Nucleic Acids Research*, vol. 41, no. D1, p. D36–D42.

Berman, H. M., J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov et P. E. Bourne. 2000. « The protein data bank ». *Nucleic Acids Research*, vol. 28, no. 1, p. 235–242.

Bernard, H.-U. 2005. « The clinical importance of the nomenclature, evolution and taxonomy of human papillomaviruses ». *Journal of Clinical Virology*, vol. 32, Supplement, no. 0, p. 1–6.

Bernard, H.-U., R. D. Burk, Z. Chen, K. van Doorslaer, H. zur Hausen et E.-M. de Villiers. 2010. « Classification of papillomaviruses (PVs) based on 189 PV types and proposal of taxonomic amendments ». *Virology*, vol. 401, no. 1, p. 70–79.

BioPortail du Gouvernement du Canada. 2008. « La bioinformatique ». En ligne. <`http://www.biofondations.gc.ca/francais/View.asp?x=739`>. Consulté le 2013.04.01.

Boc, A., H. Philippe et V. Makarenkov. 2010. « Inferring and validating horizontal gene transfer events using bipartition dissimilarity ». *Systematic biology*, vol. 59, no. 2, p. 195–211.

Buck, C. B., P. M. Day et B. L. Trus. 2013. « The papillomavirus major capsid protein L1 ». *Virology*. no. 0, p. –. Sous presse.

Burk, R. D., Z. Chen et K. Van Doorslaer. 2009. « Human papillomaviruses : genetic basis of carcinogenicity ». *Public Health Genomics*, vol. 12, no. 5-6, p. 281–290.

Castresana, J. 2000. « Selection of conserved blocks from multiple alignments for their use in phylogenetic analysis ». *Molecular Biology and Evolution*, vol. 17, no. 4, p. 540–552.

Chan, S. Y., H. Delius, A. L. Halpern et H. U. Bernard. 1995. « Analysis of genomic sequences of 95 papillomavirus types : uniting typing, phylogeny, and taxonomy. ». *Journal of Virology*, vol. 69, no. 5, p. 3074–83.

Cooper, G., et R. Hausman. 2004. *The cell : a molecular approach.* ASM Press.

Davies, D. L., et D. W. Bouldin. 1979. « A cluster separation measure ». *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PAMI-1, no. 2, p. 224–227.

de Villiers, E.-M. 2013. « Cross-roads in the classification of papillomaviruses ». *Virology.* no. 0, p. –. Sous presse.

de Villiers, E.-M., C. Fauquet, T. R. Broker, H.-U. Bernard et H. zur Hausen. 2004. « Classification of papillomaviruses ». *Virology*, vol. 324, no. 1, p. 17–27.

Diallo, A. B., D. Badescu, M. Blanchette et V. Makarenkov. 2009. « A whole genome study and identification of specific carcinogenic regions of the human papilloma viruses ». *Journal of Computational Biology*, vol. 16, no. 10, p. 1461–1473.

Dolja, V. V., E. V. Koonin *et al.* 2011. « Common origins and host-dependent diversity of plant and animal viromes ». *Current opinion in virology*, vol. 1, no. 5, p. 322–331.

Drineas, P., A. Frieze, R. Kannan, S. Vempala et V. Vinay. 2004. « Clustering large graphs via the singular value decomposition ». *Machine learning*, vol. 56, no. 1-3, p. 9–33.

Dunn, J. C. 1974. « Well-separated clusters and optimal fuzzy partitions ». *Journal of cybernetics*, vol. 4, no. 1, p. 95–104.

Edgar, R. 2004. « MUSCLE : a multiple sequence alignment method with reduced time and space complexity ». *BMC Bioinformatics*, vol. 5, no. 1, p. 113–131.

Edgar, R. C., et S. Batzoglou. 2006. « Multiple sequence alignment ». *Current Opinion in Structural Biology*, vol. 16, no. 3, p. 368–373.

Efron, B. 1979. « Bootstrap methods : Another look at the jackknife ». *The Annals of Statistics*, vol. 7, no. 1, p. pp. 1–26.

Felsenstein, J. 1989. « PHYLIP - phylogeny inference package (version 3.2) ». *Cladistics*, vol. 5, p. 164–166.

Fitch, W. M. 1971. « Toward defining the course of evolution : Minimum change for a specific tree topology ». *Systematic Zoology*, vol. 20, no. 4, p. pp. 406–416.

Flicek, P., I. Ahmed, M. R. Amode, D. Barrell, K. Beal, S. Brent, D. Carvalho-Silva, P. Clapham, G. Coates, S. Fairley, S. Fitzgerald, L. Gil, C. García-Girón, L. Gordon, T. Hourlier, S. Hunt, T. Juettemann, A. K. Kähäri, S. Keenan, M. Komorowska, E. Kulesha, I. Longden, T. Maurel, W. M. McLaren, M. Muffato, R. Nag, B. Overduin, M. Pignatelli, B. Pritchard, E. Pritchard, H. S. Riat, G. R. S. Ritchie, M. Ruffier, M. Schuster, D. Sheppard, D. Sobral, K. Taylor, A. Thormann, S. Trevanion, S. White, S. P. Wilder, B. L. Aken, E. Birney, F. Cunningham, I. Dunham, J. Harrow, J. Herrero, T. J. P. Hubbard, N. Johnson, R. Kinsella, A. Parker, G. Spudich, A. Yates, A. Zadissa et S. M. J. Searle. 2013. « Ensembl 2013 ». *Nucleic Acids Research*, vol. 41, no. D1, p. D48–D55.

Gower, J. C. 1966. « Some distance properties of latent root and vector methods used in multivariate analysis ». *Biometrika*, vol. 53, no. 3-4, p. 325–338.

Guindon, S., J.-F. Dufayard, V. Lefort, M. Anisimova, W. Hordijk et O. Gascuel. 2010. « New algorithms and methods to estimate maximum-likelihood phylogenies : Assessing the performance of phyml 3.0 ». *Systematic Biology*, vol. 59, no. 3, p. 307–321.

Guindon, S., et O. Gascuel. 2003. « A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood ». *Systematic Biology*, vol. 52, no. 5, p. 696–704.

Gusfield, D. 1997. *Algorithms on strings, trees, and sequences : computer science and computational biology*. New York, NY, USA : Cambridge University Press.

Handl, J., J. Knowles et D. B. Kell. 2005. « Computational cluster validation in post-genomic data analysis ». *Bioinformatics*, vol. 21, no. 15, p. 3201–3212.

Hardison, R. C. 2003. « Comparative genomics ». *PLoS Biol*, vol. 1, no. 2, p. e58.

Huerta-Cepas, J., J. Dopazo et T. Gabaldon. 2010. « ETE : a python environment for tree exploration ». *BMC Bioinformatics*, vol. 11, no. 1, p. 24.

Jaccard, P. 1908. « Nouvelles recherches sur la distribution florale ». *Bulletin de la Sociète Vaudense des Sciencæs Naturelles*, vol. 44, p. 223–270.

Jain, A. K. 2010. « Data clustering : 50 years beyond K-means ». *Pattern Recognition Letters*, vol. 31, no. 8, p. 651–666.

Katoh, K., K. Misawa, K.-I. Kuma et T. Miyata. 2002. « Mafft : A novel method for rapid multiple sequence alignment based on fast fourier transform ». *Nucleic Acids Research*, vol. 30, no. 14, p. 3059–3066.

Kozomara, A., et S. Griffiths-Jones. 2011. « miRBase : integrating microRNA annotation and deep-sequencing data ». *Nucleic Acids Research*, vol. 39, no. suppl 1, p. D152–D157.

Letunic, I., et P. Bork. 2011. « Interactive tree of life v2 : online annotation and display of phylogenetic trees made easy ». *Nucleic Acids Research*, vol. 39, no. suppl 2, p. W475–W478.

Lipman, D. J., et W. R. Pearson. 1985. « Rapid and sensitive protein similarity searches ». *Science*, vol. 227, no. 4693, p. 1435–1441.

Liu, Y., Z. Li, H. Xiong, X. Gao et J. Wu. 2010. « Understanding of internal clustering validation measures ». In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, p. 911–916.

Lloyd, S. 1982. « Least squares quantization in pcm. ». *IEEE Transactions on Information Theory*, vol. IT-28, no. 2 pt 1, p. 129–137.

MacQueen, J., *et al.* 1967. « Some methods for classification and analysis of multivariate observations ». In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. T. 1, p. 14. California, USA.

Muñoz, N., F. X. Bosch, S. de Sanjosé, R. Herrero, X. Castellsagué, K. V. Shah, P. J. Snijders et C. J. Meijer. 2003. « Epidemiologic classification of human papillomavirus types associated with cervical cancer ». *New England Journal of Medicine*, vol. 348, no. 6, p. 518–527.

Muñoz, N., X. Castellsagué, A. B. de González et L. Gissmann. 2006. « Chapter 1 : HPV in the etiology of human cancer ». *Vaccine*, vol. 24, Supplement 3, no. 0, p. S1–S10.

Myers, G. 1994. *Human papillomaviruses, 1994 : a compilation and analysis of nucleic acid and amino acid sequences*. Theoretical Biology and Biophysics Group T-10, Los Alamos National Laboratory.

Narechania, A., Z. Chen, R. DeSalle et R. D. Burk. 2005. « Phylogenetic incongruence among oncogenic genital alpha human papillomaviruses ». *Journal of virology*, vol. 79, no. 24, p. 15503–15510.

Needleman, S. B., et C. D. Wunsch. 1970. « A general method applicable to the search for similarities in the amino acid sequence of two proteins ». *Journal of Molecular Biology*, vol. 48, no. 3, p. 443–453.

Notredame, C., D. G. Higgins et J. Heringa. 2000. « T-coffee : a novel method for fast and accurate multiple sequence alignment ». *Journal of Molecular Biology*, vol. 302, no. 1, p. 205–217.

Pavlopoulos, G. A., T. G. Soldatos, A. Barbosa-Silva et R. Schneider. 2010. « A reference guide for tree analysis and visualization ». *BioData mining*, vol. 3, no. 1, p. 1.

R Development Core Team. 2008. *R : A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Rand, W. M. 1971. « Objective criteria for the evaluation of clustering methods ». *Journal of the American Statistical Association*, vol. 66, no. 336, p. 846–850.

Rijsbergen, C. J. V. 1979. *Information Retrieval*. Newton, MA, USA : Butterworth-Heinemann, 2nd édition.

Robinson, D. F., et L. R. Foulds. 1981. « Comparison of phylogenetic trees ». *Mathematical Biosciences*, vol. 53, no. 1, p. 131–147.

Ronquist, F., M. Teslenko, P. van der Mark, D. L. Ayres, A. Darling, S. Höhna, B. Larget, L. Liu, M. A. Suchard et J. P. Huelsenbeck. 2012. « Mrbayes 3.2 : efficient bayesian phylogenetic inference and model choice across a large model space ». *Systematic Biology*, vol. 61, no. 3, p. 539–542.

Rousseeuw, P. 1987. « Silhouettes : A graphical aid to the interpretation and validation of cluster analysis ». *Journal of Computational and Applied Mathematics*, vol. 20, no. C, p. 53–65.

Saitou, N., et M. Nei. 1987. « The neighbor-joining method : a new method for reconstructing phylogenetic trees. ». *Molecular biology and evolution*, vol. 4, no. 4, p. 406–425.

Santos, J. M., et M. Embrechts. 2009. « On the use of the adjusted rand index as a metric for evaluating supervised classification ». In *Proceedings of the 19th International Conference on Artificial Neural Networks : Part II*. Coll. « ICANN '09 », p. 175–184, Berlin, Heidelberg. Springer-Verlag.

Schiffman, M. H., et P. Castle. 2003. « Epidemiologic studies of a necessary causal risk factor : Human papillomavirus infection and cervical neoplasia ». *Journal of the National Cancer Institute*, vol. 95, no. 6, p. E2.

Smith, T., et M. Waterman. 1981. « Identification of common molecular subsequences ». *Journal of Molecular Biology*, vol. 147, no. 1, p. 195–197.

Sokal, R., et C. Michener. 1958. « A statistical method for evaluating systematic relationships ». *Univ. Kans. Sci. Bull.*, vol. 38, p. 1409–1438.

Staden, R. 1979. « A strategy of DNA sequencing employing computer programs. ». *Nucleic acids research*, vol. 6, no. 7, p. 2601–2610.

Steinhaus, H. 1956. « Sur la division des corp materiels en parties ». *Bull. Acad. Polon. Sci.*, vol. 4, no. 12, p. 801–804.

Swafford, D. 2002. *PAUP*. Phylogenetic Analysis Using Parsimony (*and other methods). Version 4.* Sunderland, Massachussets : Sinauer Associates.

Szeto, L. K., A. W.-C. Liew, H. Yan et S. sen Tang. 2003. « Gene expression data clustering and visualization based on a binary hierarchical clustering framework ». *Journal of Visual Languages & Computing*, vol. 14, no. 4, p. 341–362.

Thompson, J., D. Higgins et T. Gibson. 1994. « Clustal w : Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice ». *Nucleic Acids Research*, vol. 22, no. 22, p. 4673–4680.

Tota, J. E., M. Chevarie-Davis, L. A. Richardson, M. deVries et E. L. Franco. 2011. « Epidemiology and burden of HPV infection and related diseases : Implications for prevention strategies ». *Preventive Medicine*, vol. 53, Supplement 1, no. 0, p. S12–S21.

Van Doorslaer, K., Q. Tan, S. Xirasagar, S. Bandaru, V. Gopalan, Y. Mohamoud, Y. Huyen et A. A. McBride. 2013. « The papillomavirus episteme : a central resource for papillomavirus sequence data and analysis ». *Nucleic Acids Research*, vol. 41, no. D1, p. D571–D578.

Via i García, M., *et al.* 2012. « An integrated map of genetic variation from 1,092 human genomes ». *Nature*, vol. 491, p. 56–65.

Webb, E., J. Cox et S. Edwards. 2005. « Cervical cancer-causing human papillomaviruses have an alternative initiation site for the L1 protein ». *Virus Genes*, vol. 30, no. 1, p. 31–35.

Zheng, Z.-M., et C. C. Baker. 2006. « Papillomavirus genome structure, expression, and post-transcriptional regulation ». *Frontiers in bioscience : a journal and virtual library*, vol. 11, p. 2286–2302.