Emmanuel Cornet
Stage de maîtrise

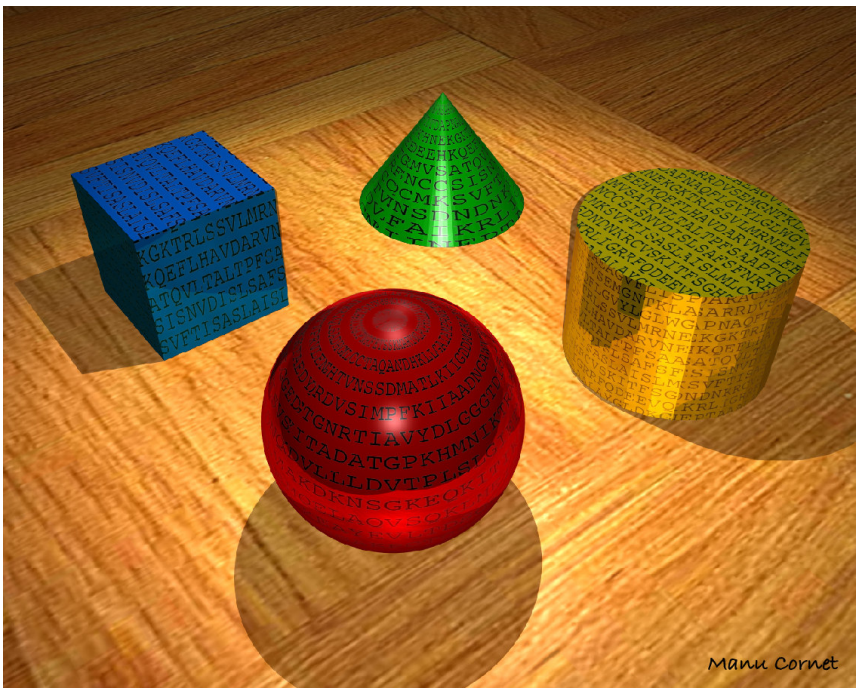
École normale supérieure
MMFAI

Sujet du stage

Étude des systèmes de recombinaison chez les bactéries par génomique comparative

Résumé : Les gènes impliqués dans les mécanismes de la recombinaison homologue sont relativement bien connus chez *Escherichia coli* et *Bacillus subtilis*. Nous avons, dans un premier temps, tenté de généraliser ces connaissances aux autres bactéries dont le génome est entièrement connu, en identifiant les gènes existant dans chaque espèce et en les localisant. Cette étude a permis de construire des arbres phylogénétiques pour chacun des gènes étudiés, et apporté des résultats sur les vitesses d'évolution comparées de ces gènes.

Mots-clefs : recombinaison homologue, génétique, bactéries, comparaison, bioinformatique



*Sous la direction d'**Eduardo Rocha***

Atelier de bioinformatique
12 rue Cuvier, 75005 Paris
wwwabi.snv.jussieu.fr

Été 2004

Remerciements

Je souhaite remercier en tout premier lieu Eduardo Rocha qui m'a encadré et soutenu avec un enthousiasme, une pédagogie et une disponibilité exemplaires. Merci également à toute l'équipe de l'Atelier de bioinformatique, qui m'a accueilli très chaleureusement dans une atmosphère de bonne humeur permanente. Merci enfin à Ariel Lindner et à François Taddei, sans qui ce projet n'aurait certainement pas vu le jour.

Ce document (PDF et PostScript), ainsi que l'image de couverture (JPEG), les arbres phylogénétiques produits et les programmes originaux sont disponibles à l'adresse

`www.eleves.ens.fr/home/cornet/Stage`

Table des matières

Introduction

L'astronomie a probablement commencé lorsque les Babyloniens ont cartographié le ciel. On ne dira certainement pas, plus tard, que la biologie a commencé avec la génétique contemporaine, mais il est indéniable que notre époque voit s'installer une accumulation impressionnante de données biologiques, génétiques en particulier. Tycho Brahe a rassemblé une quantité incalculable de mesures concernant les mouvements des planètes visibles depuis la Terre (et parfois apparemment désordonnés) avant que Johannes Kepler n'en déduise ses lois bien connues des physiciens ; de même, le défi est à présent de mener l'enquête sur l'énorme quantité d'informations génétiques dont nous disposons, car le séquençage d'un génome est un processus bien plus rapide que l'analyse des séquences produites.

Bien qu'une partie de ce défi soit simplement un travail de classification des données séquencées, l'enjeu est nettement plus vaste, car derrière ces simples chaînes de caractères (quatre caractères différents pour l'ADN, vingt pour les acides aminés[1]) réside l'immense complexité des mécanismes de biologie moléculaire.

L'un d'eux, la recombinaison homologue[2], occupe une place fondamentale chez tout être vivant, et en particulier chez les bactéries. Citons quelques exemples de son influence.

- Pour toutes les espèces, la recombinaison homologue est l'un des principaux moteurs du brassage génétique, une source inépuisable d'inspiration pour former des individus originaux, et surtout un énorme réservoir d'adaptations potentielles[3].
- La recombinaison homologue est impliquée dans la réparation des dommages subis par l'ADN. C'est, chez la levure par exemple, la principale stratégie de réparation des cassures dites « double brin ».
- Elle participe au contrôle de l'expression de certains gènes, qui dépend de réarrangements (donc de recombinaisons) à un moment précis.
- Elle est profondément impliquée dans la plupart des mécanismes de transfert horizontal[4].
- Un mauvais fonctionnement de la recombinaison peut conduire à des pathologies comme le daltonisme ou certaines déficiences du système immunitaire (venant de l'incapacité des lymphocytes à réarranger correctement les gènes d'immunoglobulines).

Nous resituerons d'abord le problème étudié dans son contexte théorique en rappelant les notions principales nécessaires à cette étude[5] ainsi que le principe des principaux algorithmes utilisés. Nous reviendrons ensuite sur la démarche expérimentale suivie lors de nos recherches, avant de présenter les résultats obtenus.

[1] Les notions d'ADN, d'acide aminé et de base azotée seront expliquées dans la première partie de ce document.
[2] On parle parfois de recombinaison générale.
[3] Les notions relatives à la théorie de l'évolution seront exposées brièvement à la section I.3 du premier chapitre.
[4] Cette notion sera elle aussi expliquée à la section I.3 du premier chapitre.
[5] La lecture de ce document ne nécessite aucune connaissance préalable en biologie moléculaire ou en génétique. Les notions nécessaires sont expliquées au moment où elles sont utiles.

Première partie

Théorie

Y a-t-il encore des esprits assez naïfs pour s'imaginer que les théories servent à être crues ? Les théories servent à irriter les philistins, à séduire les esthètes et à faire rire les autres. Le propre des vérités confondantes est d'échapper à l'analyse.

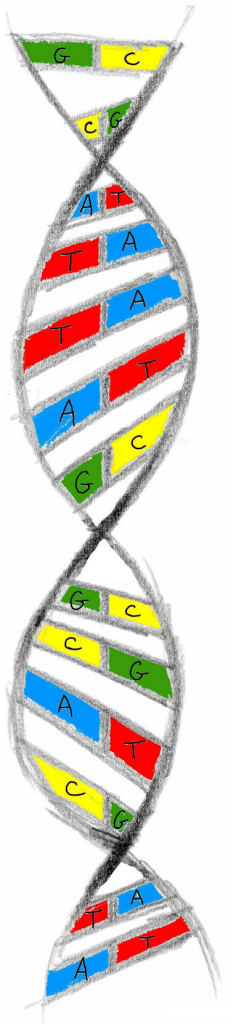
Amélie NOTHOMB, *Le Sabotage amoureux*, p. 24.

Chapitre 1

Contexte biologique

I Quelques notions de génétique

I.1 ADN, génome et séquences



L'ensemble du matériel génétique d'un individu est contenu dans une immense molécule (d'une longueur de l'ordre du mètre chez l'homme) : l'**ADN** (acide désoxyribonucléique).

L'ADN possède une structure en double hélice (voir la figure ci-contre) constituée d'un squelette externe (succession de groupements phosphates et de sucres) sur lequel viennent s'appuyer des bases azotées, arrangées par paires : `A` (adénosine) s'apparie toujours avec `T` (thymidine) et `G` (guanosine) avec `C` (cytidine).

Cet arrangement par paires est tel que la donnée de l'un des deux brins seulement suffit à déduire la structure de toute la molécule. Après avoir orienté correctement l'ADN, on peut représenter la totalité du matériel génétique d'un individu par un ensemble de **séquences** elles-mêmes formées d'une succession unique des lettres `A`, `T`, `G` et `C`.

Par exemple, on peut faire correspondre au fragment d'ADN ci-contre (en l'orientant de haut en bas et en choisissant le brin commençant en haut, à gauche) la séquence suivante :

`GCTAATCGCACAT`

Par conséquent, on peut suivre la molécule d'ADN sur toute sa longueur et garder en mémoire la succession des quatre lettres pour obtenir finalement une séquence extrêmement longue (environ quatre millions de bases pour la bactérie *Escherichia coli*, quatre milliards pour l'homme) appelée **génome** de l'individu. Cette opération constitue le **séquençage** d'un génome.

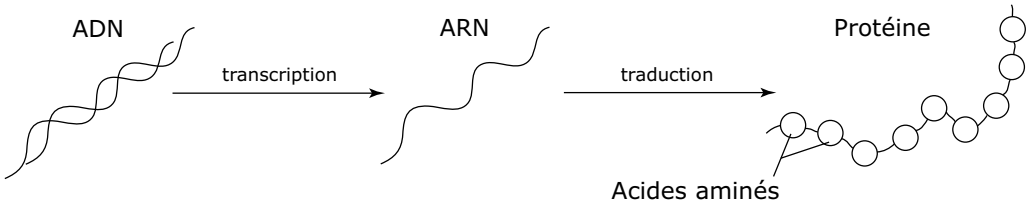
Le génome d'un individu (représenté par la séquence de son ADN) peut en réalité être vu comme une « banque de données », appelée à fournir en temps utile toutes les informations nécessaires à l'ensemble des mécanismes biologiques contribuant au bon fonctionnement de l'organisme.

Mais un fragment d'ADN n'est pas directement fonctionnel : il nécessite d'être transformé, grâce à différents processus chimiques (transcription puis traduction), en une ou plusieurs **protéines**[1].

Les protéines étant elles-mêmes constituées d'acides aminés, le code génétique fait correspondre, de façon non bijective, un acide aminé à chaque groupe de trois bases azotées ou **codon**[2].

[1] Le produit de la transcription, l'ARN, peut également remplir directement certaines fonctions.

[2] Cette correspondance n'est pas bijective car les $4^3 = 64$ façons de composer un codon parmi quatre lettres sont plus nombreuses que le nombre (20) d'acides aminés. Plusieurs codons peuvent ainsi correspondre à un même acide aminé, d'où l'expression de « redondance du code génétique ».



Par exemple, `AGC` code pour la sérine. Chaque acide aminé pouvant lui-même être représenté par une lettre[3], par exemple `S` pour la sérine, on peut transformer une séquence d'ADN comportant $3n$ bases en une séquence de n acides aminés[4]. Cette nouvelle séquence ne contient pas autant d'information, au sens strict, que la séquence d'ADN (puisqu'à un acide aminé peut correspondre plusieurs codons), mais elle est absolument équivalente du point de vue biologique.

I.2 Et les bactéries ?

Les bactéries sont des organismes unicellulaires et **procaryotes** : la cellule, qui constitue la totalité d'un individu, ne possède pas de noyau[5] et le matériel génétique n'est pas isolé du reste de la cellule.

Le génome des bactéries est d'une taille modeste par rapport à celui des principaux mammifères (dont l'homme). L'ensemble de son matériel génétique est regroupé dans un chromosome (parfois deux). Celui-ci est généralement circulaire, contrairement aux chromosomes humains, par exemple, qui sont linéaires. Une bactérie peut aussi posséder un ou plusieurs plasmides, sortes de chromosomes supplémentaires qui ne sont pas indispensables au bon développement de l'organisme[6].

I.3 Génétique et évolution

Toutes les espèces vivantes voient leur patrimoine génétique évoluer constamment. En effet, une séquence donnée d'ADN ne reste pas figée au cours du temps. Mis à part les brassages génétiques naturels chez les espèces sexuées, l'ADN subit en permanence des **mutations**. Celles-ci peuvent avoir des causes très variées : erreur lors d'un processus biologique (réplication, par exemple), irradiation, etc. Ces mutations sont extrêmement rares à notre échelle de temps, mais suffisamment nombreuses pour proposer, à des échelles plus grandes, un remaniement perpétuel des génomes de toutes les espèces.

Parmi ces mutations, certaines sont sans effet, certaines sont si néfastes que l'individu qui en résulte n'est pas viable, mais certaines prédisposent leur bénéficiaire à une meilleure adaptation à son milieu de vie. Il survivra, s'alimentera ou se reproduira mieux que les autres et transmettra plus favorablement son patrimoine génétique : c'est ce processus, extrêmement lent (mais pas aussi progressif qu'on pourrait le croire), que l'on nomme **sélection naturelle**, à la base de la théorie de l'**évolution**[7].

Au niveau de l'ADN, les mutations peuvent être de trois types : insertion (ajout d'une base azotée, qui s'intercale entre deux bases existantes), délétion (suppression d'une base) et substitution (remplacement d'une base par une autre). La pression de sélection favorise ou défavorise telle ou telle mutation, et le cours de l'histoire des espèces varie en conséquence.

En particulier, lorsque deux groupes d'individus de la même espèce se séparent pour vivre indépendamment, ils évoluent de façons différentes et deviennent finalement deux espèces distinctes : c'est le phénomène de **spéciation**.

Dans l'histoire d'un gène, on distingue deux types d'événements principaux : la spéciation, qui est la déclinaison du même gène en deux versions différentes, appartenant à deux espèces distinctes, et la duplication, qui est un phénomène de recopie du gène en un second exemplaire, toujours au

[3] À ce sujet, voir l'annexe B.

[4] Nous éludons ici le problème de la transcription en ARN et la transformation des thymidines en uridines, mais il n'est pas nécessaire pour la compréhension de la démarche.

[5] Le noyau d'une cellule est un compartiment qui contient, entre autres, l'ADN, chez les organismes **eucaryotes**.

[6] Ces plasmides sont également d'excellents vecteurs pour le transfert horizontal.

[7] Remarquons que, même s'il est devenu courant, ce terme n'est pas particulièrement bien choisi. Darwin lui-même proposait « descendance avec modification », moins impressionnant mais probablement plus correct. Le choix du mot « évolution » ne doit toutefois pas inciter à faire l'amalgame entre évolution et progrès.

sein d'une même espèce ; cette copie suit généralement un cours d'évolution légèrement différent de la première version.

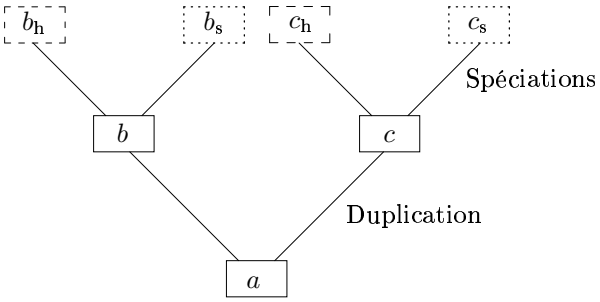


FIG. 1.1 – Un exemple : un gène a qui se serait d'abord dupliqué en b et c, chacun de ces gènes subissant ensuite une spéciation entre l'homme (h) et le singe (s).

On dit que des gènes descendant d'un même ancêtre commun sont **homologues** ; c'est le cas par exemple de tous les gènes représentés sur la figure 1.1. On dit que deux gènes homologues ayant divergé entre eux après une spéciation sont **orthologues** ; par exemple, les gènes b_h et b_s. On dit que deux gènes homologues ayant divergé après une duplication sont **paralogues** ; par exemple, les gènes b et c.

Néanmoins, chez les bactéries en particulier, un gène ne se transmet pas obligatoirement par descendance. Il est vrai que le moyen le plus logique et le plus fréquent pour propager un gène est simplement de le transmettre à ses descendants (verticalement, sur la figure 1.1). Mais il existe des cas où certains gènes sont échangés entre des bactéries de la même génération (éventuellement d'espèces différentes), car les bactéries ont, de façon générale, la capacité d'absorber et d'intégrer à leur patrimoine génétique des fragments d'ADN venus de l'extérieur. On parle dans ce cas de **transfert horizontal**.

Tous ces processus d'évolution et de remodelage des génomes rendent leur analyse directe particulièrement délicate, et la longueur moyenne du génome d'une bactérie ne facilite pas cette analyse. C'est pourquoi l'outil informatique peut nous être d'une aide inestimable, et même devenir notre principal moyen d'analyse des données issues du séquençage.

II La recombinaison homologue

II.1 Principe

Pour expliquer succinctement en quoi consiste la recombinaison homologue, et bien que nous ayons travaillé uniquement sur les bactéries, nous allons prendre l'exemple de l'homme.

L'homme est une espèce **diploïde** : chacun des vingt-trois chromosomes de nos cellules existe en deux exemplaires. Cependant, l'une des copies provient de la mère et l'autre du père, de sorte que les deux copies ne sont pas tout à fait identiques[8].

Les gonades (ovaires et testicules) produisent les gamètes (ovules et spermatozoïdes) grâce à une division cellulaire particulière : la **méiose**. Celle-ci forme deux gamètes, qui sont haploïdes (ne contiennent qu'une seule version de chacun des vingt-trois chromosomes), à partir d'une cellule normale, diploïde. Les deux chromosomes de chaque paire se répartissent donc entre les deux « cellules-filles ».

Au cours de cette division, les chromosomes semblables se rassemblent par paires et, leurs contenus étant extrêmement similaires, on assiste à un rapprochement deux à deux des séquences qui se ressemblent, comme si les bras des chromosomes se « collaient » l'un à l'autre.

C'est au cours de ce processus qu'apparaît la recombinaison homologue. Rappelons que le complexe formé des deux « bras » d'un chromosome est en fait constitué de quatre brins d'ADN,

[8] Les différences restent tout de mêmes minimes d'un point de vue quantitatif ; en effet, on sait que les patrimoines génétiques de deux humains quelconques sont identiques à environ 99 %.

puisque chaque bras est formé d'une double hélice d'ADN (deux brins). On observe très souvent l'apparition d'un *crossing over* (figure 1.2), qui entraîne un échange de matériel génétique une fois que les deux molécules se sont séparées.

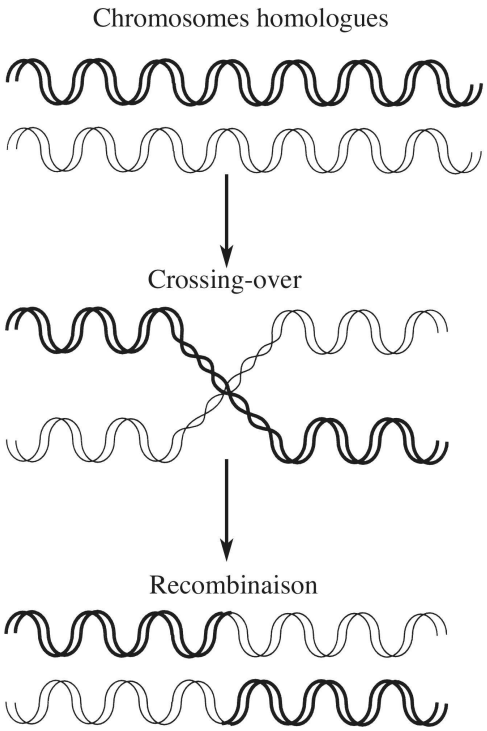


FIG. 1.2 – Recombinaison entre deux molécules d'ADN.

Ce phénomène n'est pas accidentel. Il est déclenché, régulé, contrôlé par quelques protéines dédiées.

Insistons sur le fait qu'il ne peut y avoir de recombinaison homologue qu'entre deux brins d'ADN *très similaires* (sinon l'appariement n'est pas possible).

II.2 Mécanisme

Tous les détails moléculaires qui régissent le processus de recombinaison ne sont pas connus à ce jour. Cependant, de nombreux travaux à l'échelle moléculaire permettent d'établir des modèles de la recombinaison.

Le modèle le plus cohérent de la recombinaison homologue a été proposé par Robin Holliday en 1964 (voir la figure 1.3). Il est basé sur

- la formation d'un **pont** entre deux molécules d'ADN ;
- le glissement par migration le long de la molécule hétéroduplex (double hélice d'ADN formée par deux simples brins de provenances différentes), appelé **migration de branchement** ;
- la coupure de l'intermédiaire ainsi formé au niveau des brins d'ADN, de manière à produire différents types de molécules recombinantes.

Dans le cas de deux modèles linéaires, le déplacement peut se poursuivre jusqu'à la séparation complète des deux molécules (figure 1.3.(d)) : on parle de la **résolution de la structure de Holliday**.

Dans le cas de molécules d'ADN circulaires (chez les bactéries), cette structure intermédiaire est résolue sous l'action d'endonucléases (enzymes qui coupent à l'intérieur de l'ADN). Pour produire des molécules recombinantes, la structure de Holliday présentée en (c) peut aussi être résolue d'une

autre manière, présentée sur les figures 1.3.(e), (f), (g) et (h). Une rotation de 180° de la partie inférieure de (e) produit (f). Par une coupure enzymatique au niveau de deux brins seulement, cette structure peut être résolue en deux doubles hélices indépendantes. La coupure pouvant avoir lieu verticalement ou horizontalement, on obtient deux types de molécules recombinantes (figures (g) et (h)).

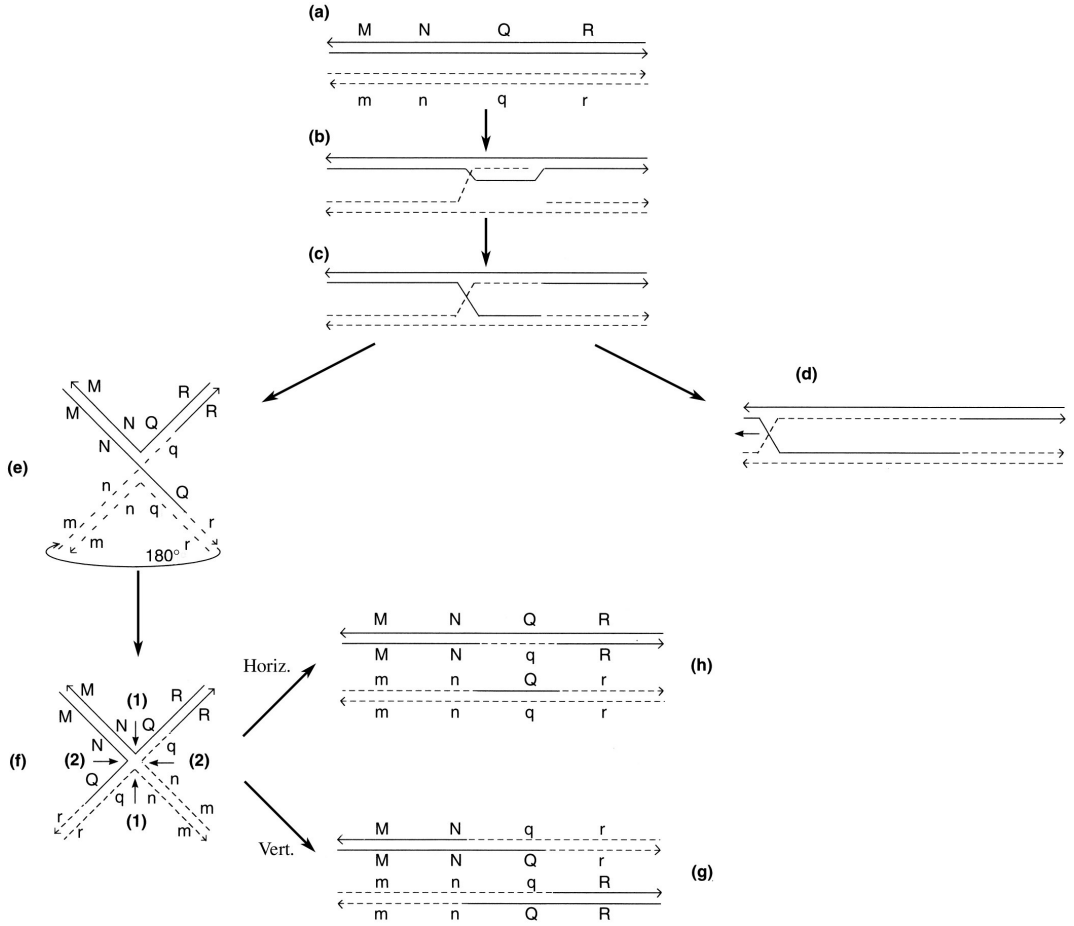


FIG. 1.3 – Modèle de la recombinaison homologue.

Notre connaissance des propriétés physico-chimiques de l'ADN ne nous laisse nullement prévoir que deux molécules puissent se casser et se réunir de façon passive. En effet, la dissection génétique des processus de la recombinaison montre que les phages[9], les bactéries et les levures possèdent des enzymes qui catalysent les étapes biochimiques de la recombinaison. Il est fort possible qu'il en soit de même chez les autres organismes. Nos connaissances les plus approfondies sur les mécanismes enzymatiques de la recombinaison ont été acquises chez *Escherichia coli*, où plusieurs gènes (par exemple *recA*, *recB*, *recC* et *recD*) ont été caractérisés.

Nous n'avons malheureusement pas la possibilité de détailler ici tous les mécanismes impliqués dans la recombinaison homologue, mais nous pouvons résumer le rôle des protéines les plus importantes :

- Les protéines RecB, RecC et RecD forment un complexe RecBCD qui possède une double activité : hélicase (sépare deux brins d'ADN) et nucléase (coupe un brin entre deux bases). Le complexe suit la molécule d'ADN (figure 1.4, à gauche) et la « déroule » localement (activité hélicase) ; les expériences génétiques montrent que le complexe coupe l'un des deux brins au niveau de sites particuliers de huit paires de bases `GCTGGTGG`, dits sites **Chi**[10].

[9]Les phages sont des virus qui parasitent les bactéries.

[10]Nous avons éludé le problème d'orientation de l'ADN en 5′ et 3′ ; ici la séquence reconnue est 5′ `GCTGGTGG` 3′. Chez

- La protéine RecA agit sur le substrat préparé par le complexe RecBCD ; c'est elle qui effectue la réaction fondamentale de la recombinaison : l'hybridation entre deux molécules d'ADN. RecA se lie à l'ADN simple brin sur toute sa longueur (figure 1.4, au milieu), et catalyse l'invasion par ce brin d'un ADN bicaténaire. Le brin d'ADN déplacé forme une **boucle de déplacement** (aussi appelée boucle en D).
- En ce qui concerne le déplacement de la jonction de Holliday, on a mis en évidence l'implication de deux protéines RuvA et RuvB. On a pu voir, par microscopie électronique, que RuvA était fixée au point d'échange (exactement au croisement des deux molécules d'ADN) et que deux anneaux hémisphériques de RuvB l'entouraient ; la figure 1.4 (à droite) présente un mécanisme d'action possible de ces deux protéines.
- Pour la résolution de la jonction de Holiday, plusieurs enzymes de clivage des brins d'ADN ont été identifiées, comme RuvC, RecG et Rus (parfois nommée RusA) ; elles participent à différentes voies de clivage.

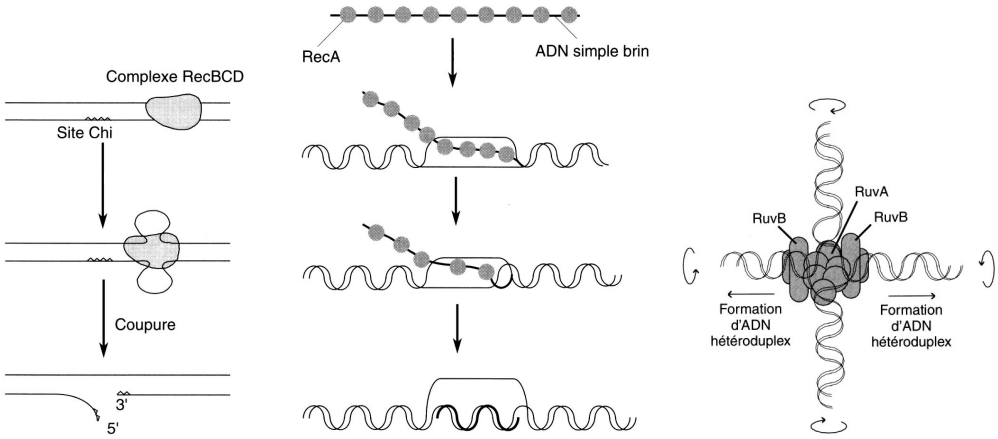


FIG. 1.4 – Modèles des mécanismes d'action des protéines RecA, RecB, RecC, RecD, RuvA et RuvB.

Hormis son rôle fondamental dans la diversité génétique, la recombinaison homologue participe à l'une des fonctions les plus vitales pour une cellule : la réparation de l'ADN endommagé.

Il a été mis en évidence que les mutants RecA chez la bactérie et la levure (ces mutants ne possèdent pas de gène *recA* fonctionnel) étaient extrêmement sensibles aux radiations ou à d'autres produits qui endommagent l'ADN.

C'est principalement la protéine RecA qui intervient ici : en catalysant le rapprochement d'un double brin d'ADN sain et d'un double brin dont un fragment est endommagé, elle rend possible la reconstitution de la séquence perdue en prenant la molécule saine comme modèle.

II.3 Et les bactéries ?

La recombinaison homologue ne s'effectue pas, chez les bactéries, dans le même contexte que chez les eucaryotes.

- Lorsque du matériel génétique étranger (un phage par exemple) entre dans la bactérie, et s'il comporte une ou plusieurs séquences suffisamment similaires à certaines parties du chromosome bactérien, il peut y avoir recombinaison homologue, et donc échange de matériel génétique.
- Lorsque deux zones du chromosome bactérien sont très similaires (issues d'une duplication, par exemple), il peut là aussi y avoir recombinaison homologue avec échange de matériel génétique.

Bacillus subtilis, le complexe AddA–AddB est l'équivalent fonctionnel de RecBCD.

- Le dernier cas intervient lors de la réplication[11]. Lors de la copie de chacun des brins d'ADN, un dysfonctionnement peut apparaître et stopper le processus, en n'ayant non seulement dupliqué qu'une partie du chromosome, mais surtout en laissant certaines zones incomplètes (celles où les protéines de réplication[12] travaillaient juste avant de s'arrêter). La bactérie interprète alors cette situation comme un endommagement de son matériel génétique, et on peut observer des cas de recombinaison homologue entre brins jumeaux pour réparer les zones incomplètes. Remarquons que cela ne termine pas pour autant le processus de réplication, mais peut lui permettre de recommencer sur des bases correctes. Notons également qu'il n'y a pas, dans ce cas, de modification du matériel génétique puisque les deux brins impliqués sont strictement identiques.

Malgré ces disparités fondamentales, il ne semble pas exister de différence essentielle entre le mécanisme de recombinaison homologue chez les bactéries et chez les eucaryotes.

[11] La réplicaton est un processus de copie de l'ADN en un second exemplaire à l'occasion d'une division cellulaire.
[12] L'ADN-polymérase, principalement.

III Objet du stage

Les mécanismes régissant la recombinaison homologue, et en particulier les gènes codant pour les protéines impliquées dans ce phénomène, sont relativement bien connus dans le cas d'*Escherichia coli* et de quelques autres bactéries telles que *Bacillus subtilis*. Le but de ce stage a été de tenter d'étendre ces connaissances à d'autres bactéries dont le génome est connu totalement, en recherchant chez ces bactéries des gènes équivalents à ceux que l'on connaît chez *E. coli*.

Nous avons exclusivement travaillé sur des séquences d'acides aminés, et non sur des séquences d'ADN directement. Nous avons également travaillé sur les *génomes complets* de tous les organismes étudiés ; nous avons exclu de ceux-ci les archéobactéries, moins intéressantes dans une première approche.

Nous avons ainsi porté notre étude sur vingt-trois gènes[13] et sur cent dix-sept génomes bactériens (avec entre une et quatre souches de chacune de ces espèces). Les gènes étudiés ont une longueur de l'ordre du millier de paires de bases, et la longueur des génomes entiers est, elle, de l'ordre du million de paires de bases.

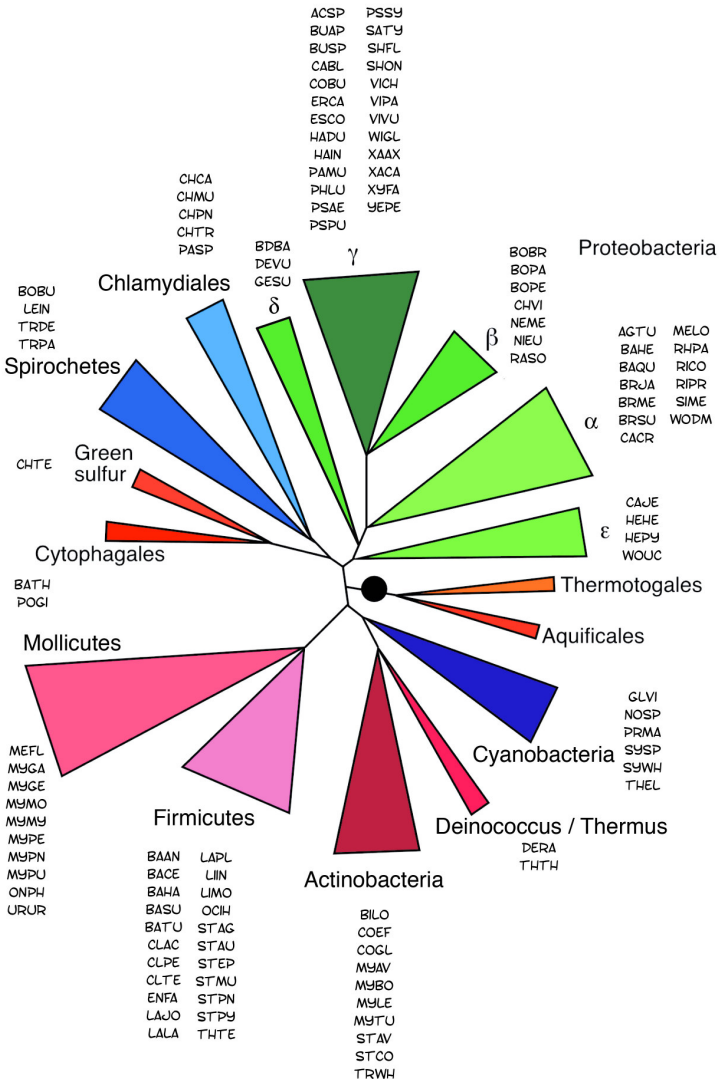


FIG. 1.5 – Les différents groupes de bactéries étudiés. Le laboratoire d'accueil utilise un codage très pratique, qui fait correspondre à chaque nom de génome un code unique à quatre lettres. Les noms complets de tous ces organismes figurent dans l'annexe A.

[13] Trois gènes parmi les vingt-trois, qui se sont avérés peu représentés (seulement quelques espèces), ont été écartés des résultats expérimentaux afin de clarifier ces derniers.

L'imaginaire nourrit le savoir qui le nourrit en retour. Le premier, sans l'autre, devient stupide, par ignorance. L'auttre, sans le premier, devient stérile, par absence de motivation.

Michel Rio, *La Terre gaste*, p. 57.

Chapitre 2

Méthodes bioinformatiques et algorithmique

La taille de ce rapport ne nous permet pas de rentrer dans tous les détails concernant les algorithmes effectivement utilisés dans les programmes courants en bioinformatique. Il nous a paru nécessaire, cependant, d'exposer les bases des algorithmes d'alignement de deux séquences, d'alignements multiples et de construction d'arbres phylogénétiques, même si les programmes dont nous nous sommes servi en implémentent parfois des variantes plus raffinées.

I Quelques prérequis

La méthode la plus simple pour comprendre la fonction d'une séquence d'ADN (ou d'acides aminés, *ie* une protéine) est de la comparer à d'autres, dont on connaît déjà la fonction. Comparer deux séquences revient à chercher des indices montrant qu'elles ont divergé d'un ancêtre commun par des processus de mutation et de sélection. Elles ont, dans ce cas, toutes les chances de coder pour des protéines qui remplissent les mêmes fonctions au sein de l'organisme, ou au moins d'en être des vestiges qui ne sont plus fonctionnels.

La comparaison de deux séquences s'avère fastidieuse, d'abord parce qu'elles peuvent être très longues (et c'est précisément l'intérêt de l'outil informatique de nous fournir une capacité de calcul suffisante) mais aussi parce qu'elles n'ont pas, la plupart du temps, la même longueur. En effet, ces séquences ayant divergé entre elles, en particulier par insertions et délétions, elles sont souvent de tailles sensiblement différentes tout en présentant de nombreuses similitudes locales.

Prenons un exemple simple : l'alignement des séquences[1]

ERGPSTVA		ERGP-ST--VA
RGPSSTWA	donnerait	-RGPSST--WA
ERGSTLKVA		ERG--STLKVA

où « - » représente un *gap*. Notons que la lettre W en fin de séquence a été alignée avec la lettre V plutôt qu'avec K ou L parce que les acides aminés qu'elles représentent sont relativement semblables, et ont par conséquent une probabilité plus élevée d'être apparentés.

Dès lors, il est plus aisé de détecter les similitudes et les disparités entre des séquences de tailles différentes. C'est ainsi que, lorsque l'on dispose de multiples séquences déjà alignées entre elles, une comparaison deux à deux de ces séquences peut nous renseigner sur l'éloignement relatif des espèces qu'elles représentent. En effet, en supposant que le taux de mutations auquel est soumise une espèce donnée est globalement constant on peut, en examinant le nombre de « lettres » différentes entre deux espèces, connaître les durées relatives de divergence de ces espèces ; on construit alors la **matrice des distances**, qui caractérise l'éloignement de chaque espèce par rapport à toutes les

[1] Les lettres ci-après correspondent chacune à un acide aminé. Il existe vingt acides aminés différents ; le code à une lettre utilisé ici figure dans l'annexe B, page c de la partie Annexes. Il existe aussi un code à trois lettres pour chaque acide aminé, mais il est évidemment beaucoup moins efficace lorsque l'on traite de longues protéines. Il va sans dire que les séquences à traiter en temps normal sont beaucoup plus longues : de l'ordre du millier de bases, soit quelques centaines d'acides aminés.

autres. Par exemple, si l'on dispose de trois espèces α, β et γ telles que, pour un gène donné, α et β comportent beaucoup moins de différences entre elles qu'avec γ, on peut en déduire que les espèces α et β se sont séparées bien après que la branche γ eut divergé de la branche $\alpha + \beta$. Le nombre de disparités entre les séquences peut même donner une idée des longueurs relatives des différentes branches.

Le but de tout ceci est d'établir un **arbre phylogénétique** des espèces étudiées. La figure 2.1 est un exemple d'arbre phylogénétique : celui des hominidés[2].

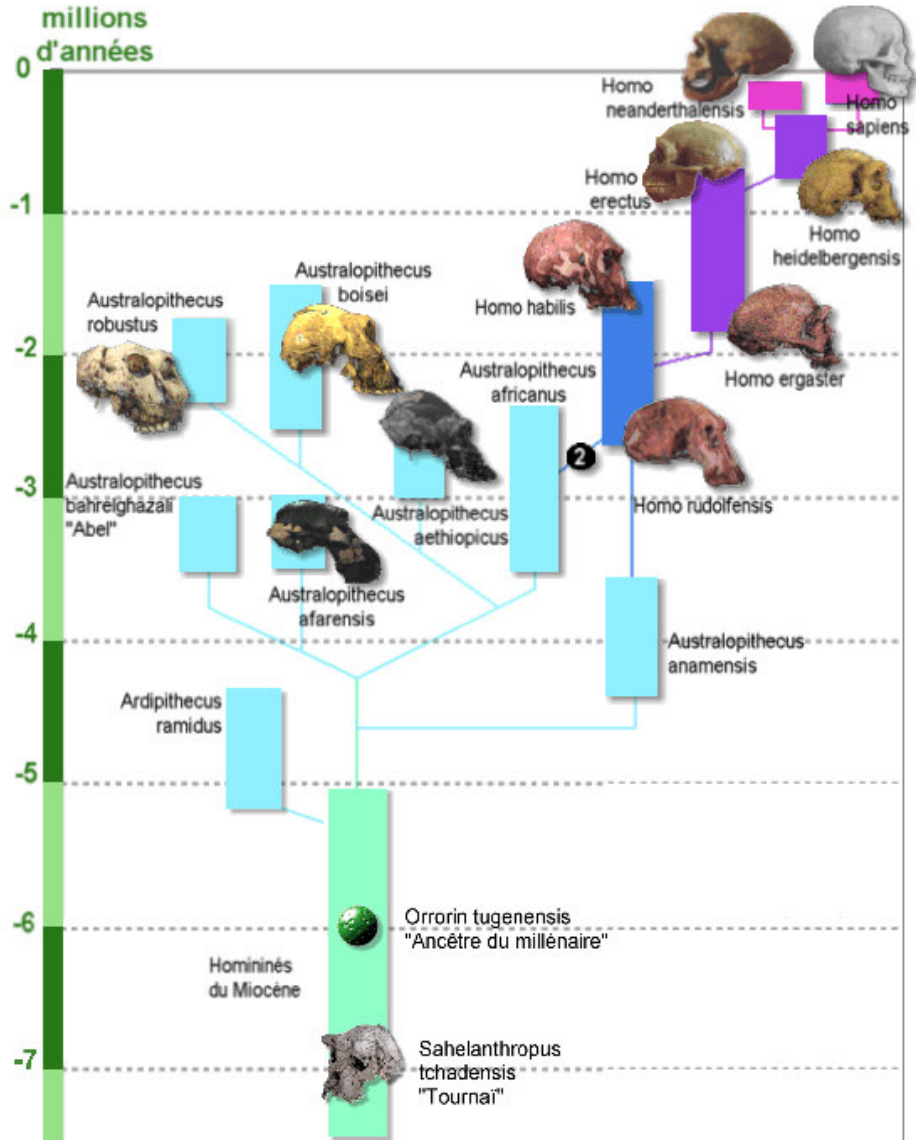


FIG. 2.1 – Un exemple d'arbre phylogénétique : celui des hominidés.

Une étape supplémentaire peut être de calculer les valeurs de ***bootstrap*** de chaque embranchement afin d'évaluer sa robustesse : un programme construit, à partir de l'alignement, mille autres arbres, obtenus en choisissant aléatoirement n colonnes parmi le nombre total m de colonnes de l'alignement, chacune pouvant être choisie plusieurs fois (par défaut, n est pris égal à m). Le programme examine ensuite, pour chaque embranchement de l'arbre, le nombre de fois où il apparaît tel quel sur l'ensemble des mille arbres. La valeur obtenue, dite valeur de *bootstrap*, permet de se faire une idée de la fiabilité (ou « robustesse ») de l'embranchement.

[2] Si les arbres que nous présenterons ne seront bien sûr pas aussi bien illustrés (ils se contenteront de fournir des résultats bruts), leur construction globale sera identique. En revanche, ils comporteront bien plus d'espèces.

II Alignement

II.1 *Scoring*

Avant d'expliquer plus en détail le fonctionnement des algorithmes d'alignement de plusieurs séquences ou de construction d'un arbre phylogénétique, expliquons succinctement comment l'on peut savoir, lorsque l'on dispose de deux séquences bien alignées, si elles ont une chance ou non de dériver d'un ancêtre commun.

II.1.a De quoi s'agit-il ?

Le travail le plus immédiat en analyse de séquences est de déterminer si deux séquences sont apparentées. Pour cela, on commence par les aligner, puis on se demande si cet alignement vient du fait que les séquences ont effectivement un lien de parenté, ou s'il est simplement dû au hasard. C'est dans ce but que l'on attribue des « scores » aux alignements : plus un score est élevé, plus il est probable que les séquences alignées soient apparentées.

Le score total que l'on attribue à un alignement est essentiellement la somme des scores attribués à chaque couple d'acides aminés[3]. Implicitement, on considère donc que la mutation d'un site est indépendante de celle de tous les autres. Ceci est une bonne approximation dans le cas de l'ADN et des protéines (séquences d'acides aminés)[4].

Dans ce modèle probabiliste, on considère la probabilité que deux séquences soient apparentées, par rapport à la probabilité qu'elles ne le soient pas, ou plus exactement le logarithme de cette quantité. Implicitement à nouveau, on s'attend à observer des identités et des substitutions conservatives plus souvent dans de bons alignements que dans des séquences aléatoires, et cela contribue à augmenter le score ; à l'inverse, on s'attend à observer des changements non conservatifs plus rarement dans de bons alignements que dans des séquences aléatoires, et cela contribue à diminuer le score.

II.1.b Matrices de substitution

Notons x et y deux séquences, x_i le i^{e} symbole de x et y_j le j^{e} symbole de y. Ces symboles seront pris dans un alphabet $\mathcal{A} = \{a, b, \ldots\}$, soit $\{\mathtt{A}, \mathtt{T}, \mathtt{G}, \mathtt{C}\}$ dans le cas de l'ADN, et les vingt acides aminés dans le cas des protéines. Considérons pour l'instant deux séquences parfaitement alignées, sans *gaps*, et de longueurs identiques.

On cherche à construire un score qui puisse donner une idée de la probabilité que les deux séquences soient apparentées. Il suffit d'attribuer une probabilité à cet alignement dans chacun des deux cas (lien de parenté ou non), puis de faire le rapport de ces deux probabilités.

Si l'on suppose d'abord que les deux séquences n'entretiennent aucun lien de parenté entre elles et qu'une lettre a apparaît à une fréquence p_a (modèle « 1 »)[5], la probabilité associée à l'alignement est simplement le produit des probabilités associées à chacun des acides aminés, soit

$$\mathrm{P}_1(x, y) = \prod_i p_{x_i} \prod_j p_{y_j}$$

Supposons maintenant que deux symboles a et b ont une probabilité q_{ab} de dériver d'un ancêtre commun c (c pouvant être identique à a et/ou à b). Dans ce modèle « 2 », la probabilité associée à l'alignement devient

$$\mathrm{P}_2(x, y) = \prod_i q_{x_i y_i}$$

Le rapport de ces deux probabilités est donc

$$\frac{\mathrm{P}_2(x, y)}{\mathrm{P}_1(x, y)} = \frac{\prod_i q_{x_i y_i}}{\prod_i p_{x_i} \prod_j p_{x_j}} = \prod_i \frac{q_{x_i y_i}}{p_{x_i} p_{y_i}}$$

[3] Ce sont des séquences d'acides aminés qui sont le plus fréquemment étudiées, mais l'on peut également travailler directement sur les bases azotées (ADN) ; tout ce qui est exposé dans cette partie est valable dans les deux cas.

[4] Ce modèle est cependant tout à fait inadapté au cas de l'ARN, mais ceci dépasse largement le cadre de notre étude.

[5] A priori, les p_a, $a \in \mathcal{A}$ ne sont pas égales, car certains acides aminés sont beaucoup plus fréquents que d'autres.

Pour obtenir un système de *scoring* additif, il faut considérer le logarithme de cette quantité,

$$\mathrm{S} = \sum_i \ln\left(\frac{q_{x_i y_i}}{p_{x_i}\, p_{y_i}}\right) = \sum_i s(x_i, y_i) \qquad \text{où} \qquad s(a,b) = \ln\left(\frac{q_{ab}}{p_a\, p_b}\right)$$

Les coefficients $s(a,b)$ peuvent être rassemblés en une matrice ; pour les acides aminés, c'est une matrice 20×20. Une telle matrice est appelée matrice de *scoring* ou **matrice de substitution**. Elle résume donc, pour chaque couple de caractères, la probabilité qu'ils descendent d'un même ancêtre ; ces valeurs sont **empiriques**, même si elles sont bien sûr grandement déterminées par les connaissances disponibles en évolution des séquences d'acides aminés.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	4	−1	−1	−2	0	−1	−1	0	−2	−1	−1	−1	−1	−2	−1	1	0	−3	−2	0
R	−1	5	0	−1	−3	1	0	−2	0	−3	−2	2	−1	−3	−2	−1	−1	−3	−2	−2
N	−1	0	6	1	−2	0	0	0	1	−3	−3	0	−2	−3	−2	1	0	−4	−2	−3
D	−2	−1	1	6	−3	0	2	−1	−1	−3	−3	−1	−3	−3	−1	0	−1	−4	−3	−3
C	0	−3	−2	−3	9	−3	−3	−2	−3	−1	−1	−3	−1	−2	−3	−1	−1	−2	−2	−1
Q	−1	1	0	0	−3	5	2	−2	1	−3	−2	1	0	−3	−1	0	−1	−2	−1	−2
E	−1	0	0	2	−3	2	5	−2	0	−3	−3	1	−2	−3	−1	0	−1	−3	−2	−2
G	0	−2	0	−1	−2	−2	−2	6	−2	−3	−4	−1	−2	−3	−2	0	−2	−2	−3	−3
H	−2	0	1	−1	−3	1	0	−2	7	−3	−3	−1	−1	−1	−2	−1	−2	−2	2	−3
I	−1	−3	−3	−3	−1	−3	−3	−3	−3	4	2	−3	1	0	−3	−2	−1	−2	−1	3
L	−1	−2	−3	−3	−1	−2	−3	−4	−3	2	4	−2	2	0	−3	−2	−1	−2	−1	1
K	−1	2	0	−1	−3	1	1	−1	−1	−3	−2	4	−1	−3	−1	0	−1	−3	−2	−2
M	−1	−1	−2	−3	−1	0	−2	−2	−1	1	2	−1	5	0	−2	−1	−1	−1	−1	1
F	−2	−3	−3	−3	−2	−3	−3	−3	−1	0	0	−3	0	6	−4	−2	−2	1	3	−1
P	−1	−2	−2	−1	−3	−1	−1	−2	−2	−3	−3	−1	−2	−4	7	−1	−1	−4	−3	−2
S	1	−1	1	0	−1	0	0	0	−1	−2	−2	0	−1	−2	−1	4	1	−3	−2	−2
T	0	−1	0	−1	−1	−1	−1	−2	−2	−1	−1	−1	−1	−2	−1	1	4	−2	−2	0
W	−3	−3	−4	−4	−2	−2	−3	−2	−2	−2	−2	−3	−1	1	−4	−3	−2	10	2	−3
Y	−2	−2	−2	−3	−2	−1	−2	−3	2	−1	−1	−2	−1	3	−3	−2	−2	2	6	−1
V	0	−2	−3	−3	−1	−2	−2	−3	−3	3	1	−2	1	−1	−2	−2	0	−3	−1	4

FIG. 2.2 – Un exemple de matrice de substitution : la matrice BLOSUM60, systématiquement utilisée dans nos recherches. Les valeurs ont été étalées et arrondies à l'entier le plus proche pour accélérer les calculs par ordinateur. Remarquons que les coefficients diagonaux sont strictement positifs (substitution d'un acide aminé par lui-même).

II.1.c Et les *gaps* ?

Il faut un système qui puisse pénaliser les *gaps*. Le coût associé à un *gap* de longueur ℓ suit une loi affine :

$$\mathcal{C}(\ell) = \alpha\,(\ell - 1) + \beta$$

β est appelé « coût d'ouverture d'un *gap* » et α, « coût d'extension d'un *gap* » ; β est généralement choisi supérieur à α pour pénaliser la création d'un *gap* plus fortement que l'allongement d'un *gap* déjà ouvert.

Pour un modèle plus simple, on peut également poser $\mathcal{C}(\ell) = \alpha\,\ell$ (loi linéaire).

On a donc créé un système complet de *scoring* qui permettra, en temps voulu, d'attribuer une « note » à un alignement donné pour se faire une idée sur les liens possibles unissant les séquences qui le composent.

II.2 Alignement global de deux séquences : algorithme de Needleman–Wunsch

Le système de *scoring* étant donné, il nous faut désormais un alogrithme qui puisse aligner deux séquences de la meilleure façon possible. Celui que nous allons exposer produit un alignement qualifié de « global » (contrairement aux alignements « locaux », qui ne nous intéressent pas ici). Il s'agit de l'alignement de Needleman-Wunsch (1970), mais la version améliorée que nous allons décrire a été introduite par Gotoh (1982).

II.2.a Principe

Si l'on met de côté le cas où les deux séquences à aligner sont de la même longueur et extrêmement semblables, il faut introduire des *gaps* aux endroits adéquats dans l'une et/ou l'autre des séquences afin d'obtenir un score optimal.

L'idée est de construire l'alignement de façon récursive, en faisant croître la longueur des sous-séquences alignées. Nous supposons en effet que toutes les colonnes sont indépendantes deux à deux : le score global d'un alignement est composé d'une somme de termes indépendants. Le meilleur score jusqu'à un certain point est ainsi le meilleur score jusqu'à l'étape précédente ajouté au score du pas supplémentaire ; autrement dit, le problème satisfait au principe d'optimalité de Bellman et l'on peut donc adopter une méthode de programmation dynamique.

Soient x et y les deux séquences à aligner, de longueurs respectives n et m ; on notera x_i le i^{e} symbole de la séquence x. On construit une matrice F, indexée par i et j (un indice pour chacune des deux séquences), où $\mathrm{F}_{ij} = \mathrm{F}(i,j)$ est le score de l'alignement optimal entre les segments initiaux $x_{1\cdots i}$ de x jusqu'à x_i et $y_{1\cdots i}$ de y jusqu'à y_i.

Voici comment l'on construit F récursivement. On commence par initialiser $\mathrm{F}(0,0) = 0$ et on remplit la matrice de haut en bas et de droite à gauche. Si à la fois $\mathrm{F}(i-1,j-1)$, $\mathrm{F}(i-1,j)$ et $\mathrm{F}(i,j-1)$ sont connus, alors on peut calculer $\mathrm{F}(i,j)$. En effet, si l'on suppose que l'on a effectué l'alignement jusqu'à (x_i, y_j), c'est que

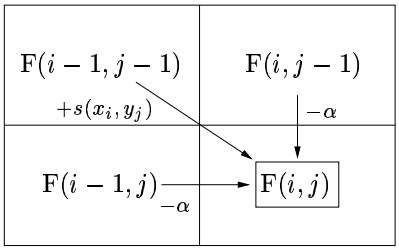
- soit x_i est aligné avec y_j, auquel cas $\mathrm{F}(i,j) = \mathrm{F}(i-1,j-1) + s(x_i,y_j)$, où l'on rappelle que $s(x_i,y_j)$ est le score de l'alignement des deux symboles x_i et y_j, donné par la matrice de *scoring* choisie ;
- soit x_i est aligné avec un *gap*, auquel cas $\mathrm{F}(i,j) = \mathrm{F}(i-1,j) - \alpha$ (pour simplifier, on choisit une loi affine pour le décompte des *gaps*) ;
- soit enfin c'est y_j qui est aligné avec un *gap*, et alors $\mathrm{F}(i,j) = \mathrm{F}(i,j-1) - \alpha$.

Le meilleur score obtenu au stade (i,j) est le maximum de ces trois valeurs. Autrement dit

$$\mathrm{F}(i,j) = \max \begin{cases} \mathrm{F}(i-1,j-1) + s(x_i,y_j) \\ \mathrm{F}(i-1,j) - \alpha \\ \mathrm{F}(i,j-1) - \alpha \end{cases}$$

Si ces valeurs ne sont pas distinctes, on choisit l'une quelconque d'entre elles.

On peut ainsi remplir entièrement la matrice F en calculant chacun des coefficients à partir à partir de l'une des trois valeurs qui le précèdent (au nord, à l'ouest et au nord-est), selon celle qui donne le meilleur résultat.



Pendant que l'on remplit la matrice on trace, pour chaque case, une flèche qui « remonte » vers la case qui a servi à calculer sa valeur (voir l'exemple complet d'alignement, plus bas). On garde ainsi une trace des chemins suivis.

Pour que le remplissage de la matrice soit complet, il faut également définir la valeur des coefficients $\mathrm{F}(0,j)$ et $\mathrm{F}(i,0)$ (première colonne et première ligne), où les $\mathrm{F}(i-1,j-1)$ ne sont pas définis. Les valeurs $\mathrm{F}(0,j)$ correspondent à un alignement du préfixe y_j avec uniquement des *gaps*. On peut donc raisonnablement poser $\mathrm{F}(0,j) = -j\,\alpha$. De même, $\mathrm{F}(i,0) = -i\,\alpha$.

Le coefficient $\mathrm{F}(n,m)$ est par conséquent le meilleur score possible obtenu en alignant x avec y. Pour obtenir l'alignement lui-même, il faut repartir de cette case $\mathrm{F}(n,m)$ et le construire à l'envers, en remontant à la case précédente grâce aux flèches tracées au fur et à mesure. À partir de $\mathrm{F}(i,j)$, on ajoute les deux symboles x_i et y_j si la flèche est orientée vers le nord-ouest, x_i et un *gap* si elle se dirige vers l'ouest, et un *gap* et y_j si elle pointe vers le nord. Remarquons que si le score optimal est

unique, l'alignement optimal ne l'est pas toujours (cas où l'on a choisi arbitrairement entre deux façons d'obtenir le même résultat, ceci suffisamment de fois pour obtenir un ou plusieurs chemins alternatifs).

II.2.b Exemple

L'exemple suivant montre comment on été alignées les séquences `HEAGAWGHEE` et `PAWHEAE` en prenant pour base la matrice BLOSUM50, dont la restriction à ces deux séquences est la suivante :

	H	E	A	G	A	W	G	H	E	E
P	−2	−1	−1	−2	−1	−4	−2	−2	−1	−1
A	−2	−1	**5**	0	**5**	−3	0	−2	−1	−1
W	−3	−3	−3	−3	−3	**15**	−3	−3	−3	−3
H	**10**	0	−2	−2	−2	−3	−2	**10**	0	0
E	0	**6**	−1	−3	−1	−3	−3	0	**6**	**6**
A	−2	−1	**5**	0	**5**	−3	0	−2	−1	−1
E	0	**6**	−1	−3	−1	−3	−3	0	**6**	**6**

Voici comment se construit la matrice F, où l'on a pris $\alpha = 8$:

		H	E	A	G	A	W	G	H	E	E
	0 ←	**−8** ←	**−16** ←	−24 ←	−32 ←	−40 ←	−48 ←	−56 ←	−64 ←	−72 ←	−80
	↑ ↖	↖	↖		↖					↖	↖
P	−8	−2	−9	**−17** ←	**−25**	−33 ←	−42 ←	−49 ←	−57	−65	−73
	↑	↑ ↖	↖		↖						
A	−16	−10	−3	−4 ←	−12	**−20** ←	−28 ←	−36 ←	−44 ←	−52 ←	−60
	↑	↑	↑ ↖	↖	↖	↖					
W	−24	−18	−11	−6	−7	−15	**−5** ←	**−13** ←	−21 ←	−29 ←	−37
	↑ ↖	↖	↖	↖	↖		↑ ↖	↖			
H	−32	−14	−18	−13	−8	−9	−13	−7	**−3** ←	−11 ←	−19
	↑	↑ ↖			↑ ↖	↖		↑ ↖	↖	↖	
E	−40	−22	−8 ←	−16	−16	−9	−12	−15	−7	**3**	−5
	↑	↑	↑ ↖			↖	↖	↖	↑	↑ ↖	
A	−48	−30	−16	−3 ←	−11	−11	−12	−12	−15	**−5**	2
	↑	↑	↑	↑ ↖	↖	↖	↖	↖	↖	↖	
E	−56	−38	−24	−11	−6	−12	−14	−15	−12	−9	**1**

et cela donne l'alignement

```
HEAGAWGHE-E
--P-AW-HEAE
```

II.2.c Complexité

On stocke en mémoire $(n+1)\times(m+1)$ nombres (n et m sont les longueurs des deux séquences), et chacun de ces nombres se calcule en temps constant (trois sommes et un maximum). Cet algorithme a donc une complexité de $O(n \times m)$ en temps et en espace. m et n étant en général comparables, on peut donc retenir une complexité de $\mathbf{O(n^2)}$ en temps comme en espace.

II.3 Alignements multiples

Autant aligner manuellement deux séquences de taille raisonnable ne pose pas de grande difficulté pour un biologiste expérimenté, autant l'alignement manuel de multiples séquences est une tâche extrêmement difficile. Les méthodes d'alignement multiple automatique sont un important sujet de recherche en bioinformatique.

Commençons par introduire quelques notations.

- m désigne l'alignement multiple utilisé ;
- m_i est la colonne i de l'alignement m ;
- m_i^j représente le symbole de la colonne i pour la séquence j ;
- $s(a, b)$ est, comme précédemment, le score obtenu en alignant les deux symboles a et b.

II.3.a *Scoring*

Il existe plusieurs façons d'attribuer un score à un alignement multiple, mais la méthode standard consiste simplement à faire la somme des scores des alignements par paires, d'où son nom, « score SP » pour « somme de paires ».

Là encore, une matrice de substitution doit être choisie, comme par exemple la matrice BLOSUM60 présentée page 16.

Le score SP pour la colonne m_i est défini par

$$\mathrm{S}(m_i) = \sum_{k<\ell} s(m_i^k, m_i^\ell)$$

On peut comptabiliser les *gaps* soit en posant $s(a, -) = s(-, a) = \alpha$ et $s(-, -) = 0$, soit en les prenant en compte séparément (par exemple pour une loi affine).

II.3.b Ce qu'il faudrait faire

La façon la plus rigoureuse de construire un alignement multiple serait de généraliser le système de *scoring* utilisé pour les alignements de deux séquences.

On considère toujours qu'il y a une indépendance parfaite (horizontale et verticale) entre tous les symboles de l'alignement. On suppose également que la prise en compte des *gaps*, qui suit une loi linéaire, est incluse dans les coefficients $s(a, b)$. On peut alors exprimer simplement le score global d'un alignement comme la somme des scores de ses colonnes :

$$\mathrm{S}(m) = \sum_i \mathrm{S}(m_i)$$

Si l'on définit $\alpha(i_1, i_2, \ldots, i_\mathrm{N})$ comme le score maximal d'un alignement des N sous-séquences se terminant par $x^1_{i_1}, x^2_{i_2}, \ldots, x^\mathrm{N}_{i_\mathrm{N}}$, la généralisation se fait de la façon suivante :

$$\alpha(i_1, i_2, \ldots, i_\mathrm{N}) = \max \begin{cases} \alpha(i_1-1, i_2-1, \ldots, i_\mathrm{N}-1) & + & \mathrm{S}(x^1_{i_1}, x^2_{i_2}, \ldots, x^\mathrm{N}_{i_\mathrm{N}}) \\ \alpha(i_1, i_2-1, \ldots, i_\mathrm{N}-1) & + & \mathrm{S}(-, x^2_{i_2}, \ldots, x^\mathrm{N}_{i_\mathrm{N}}) \\ \alpha(i_1-1, i_2, i_3-1, \ldots, i_\mathrm{N}-1) & + & \mathrm{S}(x^1_{i_1}, -, \ldots, x^\mathrm{N}_{i_\mathrm{N}}) \\ & \vdots & \\ \alpha(i_1-1, i_2-1, \ldots, i_\mathrm{N}) & + & \mathrm{S}(x^1_{i_1}, x^2_{i_2}, \ldots, -) \\ \alpha(i_1, i_2, i_3-1, \ldots, i_\mathrm{N}-1) & + & \mathrm{S}(-, -, \ldots, x^\mathrm{N}_{i_\mathrm{N}}) \\ & \vdots & \\ \alpha(i_1, i_2-1, \ldots, i_{\mathrm{N}-1}-1, i_\mathrm{N}) & + & \mathrm{S}(-, x^2_{i_2}, \ldots, -) \\ & \vdots & \end{cases}$$

où apparaissent toutes les combinaisons de *gaps* possibles sauf celle composée de N *gaps*. Il y a donc $2^\mathrm{N} - 1$ combinaisons possibles, et autant de quantités à calculer ! L'initiation, la terminaison et la reconstitution de l'alignement sont similaires au cas $\mathrm{N} = 2$.

Ce n'est bien sûr pas cette approche (programmation dynamique) qui est retenue puisqu'elle conduit à une complexité exponentielle. On peut utiliser cet algorithme exact (qui garantit de trouver le meilleur alignement) en deçà de 15 ou 20 séquences de quelques dizaines d'acides aminés. Toutefois, dans le cas général, on préfère se tourner vers des algorithmes heuristiques, dits d'« alignement progressif », moins précis mais beaucoup plus efficaces.

II.3.c Alignement progressif

Le principe est de construire une succession d'alignement par paires : on choisit d'abord deux séquences et on les aligne. Cet alignement étant fixé, une troisième séquence est choisie puis alignée avec le premier alignement, et ainsi de suite. Par « alignement fixé », on entend qu'un *gap* ajouté

à l'une des séquences qui le composent est également ajouté à toutes les autres, au même endroit, et que ceci est la seule opération permise sur l'alignement.

La question principale est de savoir dans quel ordre choisir les séquences, puisque cet ordre influence notablement le résultat final. L'idéal est de commencer par choisir deux séquences très semblables puis de progresser petit à petit vers des séquences de plus en plus différentes de l'alignement de départ. Pour cela,

1. on aligne les $N(N-1)/2$ paires de séquences parmi les N séquences au total et on garde en mémoire tous les scores, que l'on convertit en « distances » entre séquences (les modalités de conversion, *ie* la loi qui lie ces deux grandeurs varie avec les algorithmes) ;
2. on construit un **arbre guide** (*guide tree*) à partir de ces distances[6] ;
3. on procède à l'alignement progressif, en commençant par les feuilles et en alignant à chaque étape les couples de frères (cela revient à aligner par ordre décroissant de similarité) jusqu'à ce que toutes les séquences aient été alignées.

L'arbre guide est construit sur le même principe qu'un arbre phylogénétique, mais il est beaucoup moins précis et ne sert que de support à l'alignement multiple.

Une remarque sur cet algorithme : on sait déjà comment aligner deux séquences isolées, mais comment aligner entre eux deux alignements, ou une séquence isolée avec un alignement « fixé » de plusieurs autres ? La méthode est simple : on construit, à partir de l'alignement à traiter (comportant un nombre quelconque de séquences), une séquence « consensus », dont chaque symbole est simplement le symbole le plus fréquent dans la colonne correspondante. Une méthode un peu plus complexe et plus utilisée est de garder en mémoire la fréquence d'apparition des 20 acides aminés dans chaque colonne grâce à une matrice de consensus.

Du point de vue de la complexité de cet algorithme, c'est l'étape 1 qui demande le plus de temps. On a vu en effet que l'on pouvait aligner deux séquences de longueur moyenne ℓ en un temps $O(\ell^2)$. Il faut ici effectuer $N(N-1)/2 = O(N^2)$ alignements, où N est le nombre de séquences à aligner. Comme nous le verrons dans la suite, l'étape 2 est moins gourmande en temps ($O(N^2)$). Quant à l'étape 3, elle nécessite d'opérer à $N-1$ alignements (le nombre de nœuds dans un arbre binaire complet à N feuilles), coûtant chacun $O(\ell^2)$, d'où une complexité de $O(\ell^2 \times N)$. La complexité de cet alogrithme est donc de $\mathbf{O(\ell^2 \times N^2)}$.

II.3.d Dans la vraie vie

Le programme que nous avons utilisé (et qui est largement répandu en bioinformatique) est `clustalw`. Il permet à l'utilisateur de choisir entre algorithme exact (programmation dynamique) et algorithme heuristique (alignement progressif). Ses algorithmes sont très proches de ceux que nous avons exposés dans cette section ; il utilise un modèle relativement parfectionné de conversion (score de similarité → distance), ainsi qu'un algorithme plus récent pour construire l'arbre guide.

III Construction d'arbres phylogénétiques

III.1 Des arbres, mais encore ?

Les phylogénéticiens ont légèrement adapté cet objet mathématique pour satisfaire à leurs besoins. En particulier :

- Dans un arbre phylogénétique, les feuilles sont des espèces ou, dans notre cas, les séquences analysées (qui sont normalement des gènes orthologues, ou parfois paralogues). Les nœuds représentent en général une spéciation (divergence d'une espèce unique en deux espèces distinctes) ou, plus rarement, une duplication (copie d'un même gène au sein d'une seule espèce).
- On fait clairement apparaître la racine, et on dessine même une branche au-dessus de la racine pour matérialiser l'hypothèse que toutes les espèces terrestres dérivent d'un ancêtre commun. Cependant, l'algorithme utilisé pour construire un arbre ne fait pas de proposition sur la racine ; dans ce cas, on peut représenter des arbres sans racine (*rootless*). Pour savoir où se trouve la racine d'un arbre phylogénétique correspondant à un gène donné, on y inclut

[6] Cette construction s'effectue à l'aide de l'algorithme de *clustering* de Fitch et Margoliash (1967), similaire à l'algorithme de contruction d'un arbre phylogénétique exposé dans la section III.

le même gène (s'il existe !) chez une espèce extrêmement éloignée des organismes étudiés (séquence « extérieure »).

- La longueur des branches est significative. Elle représente le nombre de mutations (modifications dans la séquence) qui sépare deux nœuds. On assimile parfois la dimension verticale des arbres phylogénétiques au temps qui s'écoule. Bien que cela ne soit pas exact car cela supposerait que la vitesse de mutation est constante et même uniforme pour toutes les protéines (ce qui est faux), la longueur des branches donne tout de même une idée grossière des périodes d'évolution.
- Tous les arbres phylogénétiques sont binaires, ce qui est logique étant donnés les ordres de grandeur des périodes d'évolution des espèces. Ce n'est d'ailleurs pas une grande limitation puisque l'on peut assimiler un arbre non binaire par un arbre binaire où certaines branches sont extrêmement courtes (et c'est probablement plus proche de la réalité de l'évolution des espèces).

III.2 Un arbre à partir des distances de paires : algorithme UPGMA

La manière la plus logique de construire un arbre, et la seule que nous aborderons ici, est de partir de la matrice des distances de paires.

III.2.a La distance

Il existe de nombreuses façons de définir la distance d_{ij} entre deux séquences i et j.

Par exemple, en supposant que les deux séquences soient déjà alignées, on peut dire que d_{ij} est la fraction f des colonnes où i et j présentent des symboles différents. Cette définition est satisfaisante pour de petites fractions f, mais pour des séquences éloignées, f tend vers le pourcentage de différences entre deux séquences aléatoires alors que l'on voudrait que d_ij continue d'augmenter lorsque f tend vers cette valeur.

Une meilleure (mais pas la seule) définition de d_{ij} est

$$d_{ij} = -\frac{3}{4} \log\left(1 - \frac{4}{3} f\right)$$

qui tend vers l'infini lorsque f tend vers la valeur d'équilibre de f (75% de symboles différents[7]).

III.2.b L'algorithme

UPGMA sont les initiales de *unweighted pair group method (using arithmetic) averages* (méthode de regroupement par paires sans pondération et utilisant des moyennes arithmétiques). L'algorithme est beaucoup plus simple que son nom ne le laisse supposer.

L'arbre est construit de bas en haut (des feuilles à la racine). L'idée est de regrouper les séquences : à chaque itération, on fusionne deux groupes et on ajoute un nœud à l'arbre. Les longueurs des branches sont déterminées à partir des distances entres les groupes.

Commençons donc par définir la distance entre deux groupes C_i et C_j (on utilise la lettre C pour « *cluster* »). Il s'agit de la moyenne des distances entre une séquence du premier groupe et un séquence du second :

$$d_{ij} = \frac{1}{|\mathrm{C}_i|\ |\mathrm{C}_j|} \sum_{\substack{p \in \mathrm{C}_i \\ q \in \mathrm{C}_j}} d_{pq}$$

où $|\mathrm{C}_\ell|$ est le nombre de séquences incluses dans le groupe C_ℓ. On en déduit facilement que si C_k est l'union des deux groupes C_i et C_j, et si C_ℓ est un autre groupe quelconque :

$$d_{kl} = \frac{d_{i\ell}\ |\mathrm{C}_i| + d_{j\ell}\ |\mathrm{C}_j|}{|\mathrm{C}_i| + |\mathrm{C}_j|} \qquad (\star)$$

[7] Cette valeur est empirique, et constitue une limite supérieure aux valeurs habituellement observées en pratique (séquences apparentées), ce qui explique également qu'il n'est pas gênant, en général, que d_{ij} ne soit pas définie pour des valeurs de f supérieures à $3/4$. Bien sûr, des définitions plus sophistiquées de la distance existent.

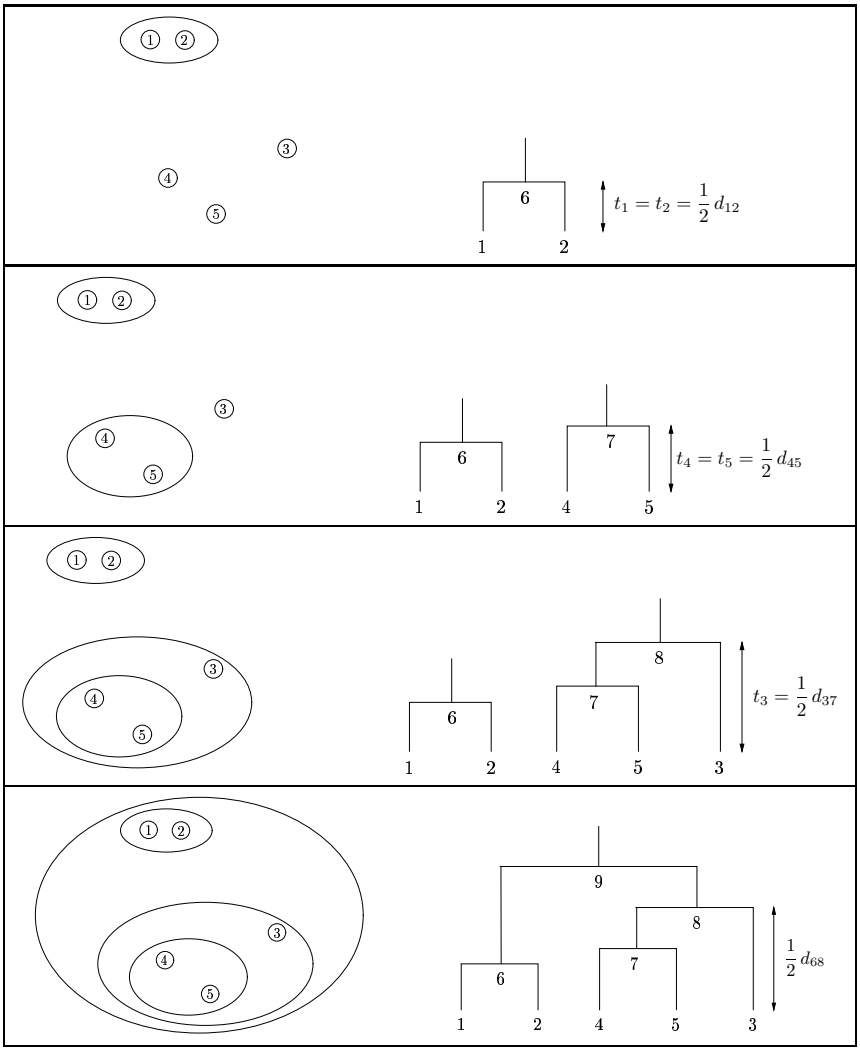
L'algorithme est le suivant :

1. Initialisation
   (a) Donner à chaque séquence i son propre groupe C_i.
   (b) Dessiner une feuille de l'arbre pour chaque séquence et la placer à hauteur nulle.
2. Itération
   (a) Déterminer le couple de groupes $(i,\ j)$ pour lequel d_{ij} est minimal. Faire un choix arbitraire en cas d'égalité.
   (b) Définir un nouveau groupe $\mathrm{C}_k = \mathrm{C}_i \cup \mathrm{C}_j$ et calculer les coefficients $d_{k\ell}$ pour tous les ℓ grâce à l'équation $(\star)$.
   (c) Dessiner un nœud k qui a pour fils i et j et le placer à la hauteur $d_{ij}/2$.
   (d) Ajouter k aux groupes courants et supprimer i et j.
3. Terminaison
   Lorsqu'il ne reste plus que deux groupes α et β, placer la racine à la hauteur $d_{\alpha\beta}/2$.

La procédure de choix des groupements (par distances croissantes) assure que les longueurs de branches sont bien définies, *ie* qu'un père est toujours au-dessus de ses fils.

III.3 Exemple

Voici un exemple de construction d'un arbre phylogénétique simple. Nous avons choisi une distribution de distances telle qu'on puisse la représenter dans un plan, mais ce n'est bien sûr pas le cas général.



III.4 Complexité

Ce qui demande du temps ici est :

- le calcul initial de toutes les valeurs d_{ij} : $\mathrm{N}(\mathrm{N}-1)/2 = \mathrm{O}(\mathrm{N}^2)$, où N est le nombre de séquences ;
- le calcul du minimum parmi ces N nombres : O(N) (fait à chaque itération) ;
- le nouveau calcul des distances $d_{k\ell}$ après un regroupement (fait à chaque itération).

Les autres étapes s'effectuent en temps constant. La complexité de cet algorithme est par conséquent $\mathbf{O(N^2)}$. On remarquera que le temps d'exécution est indépendant de la longueur des séquences, puisque la donnée de départ est ici réduite à la matrice des distances.

III.5 Dans la vraie vie

Un algorithme plus récent et plus utilisé que UPGMA est dit « *neighbour-joining* » (« rassemblement entre voisins »). Il diffère quelque peu de UPGMA dans sa manière de procéder, mais le principe en est tout à fait similaire. Une méthode pour évaluer la robustesse de l'arbre, très utile en pratique et que nous avons déjà évoquée, est le *bootstrap*. En choisissant aléatoirement un nombre fixé de colonne de l'alignement, en contruisant l'arbre correspondant et en répétant cette opération de nombreuses fois, on peu accéder à une estimation de la robustesse de chaque embranchement.

Deuxième partie

Recherches

La répétition rassure. Elle est preuve d'intimité. Elle en est la respiration même.

Daniel Pennac, *Comme un roman*, p. 64.

Chapitre 3

Démarche expérimentale

I Outils déjà présents

Faisons ici un bref inventaire des programmes déjà disponibles et dont nous nous sommes servi :

- **GetEntry** permet d'obtenir la séquence d'un gène particulier, dans un génome donné, soit grâce à son nom s'il a déjà été identifié (ou toute autre chaîne de caractères qui figure dans sa description), soit directement par son numéro d'identification.
- **Opscan** prend en argument un gène g et un génome entier $\mathcal{G}$, et indique le gène de $\mathcal{G}$ qui « ressemble » le plus à g (dit « *best match* »).
- **Clustalw** est un outil d'alignement multiple de séquences. On lui donne en entrée un ensemble de plusieurs séquences (qui représentent des gènes que l'on suppose orthologues) et `clustalw` aligne ces séquences au mieux pour faire correspondre, acide aminé par acide aminé (ou nucléotide par nucléotide si l'on travaille sur de l'ADN), toutes les séquences en introduisant des *gaps* là où c'est nécessaire (voir la section I du chapitre 2 sur l'alignement de séquences).
- **Seaview** permet de lire, de visualiser et éventuellement de corriger les alignements faits par `clustalw`. La figure 3.1 donne un aperçu de l'interface graphique de ce programme. À l'aide d'une représentation colorée des acides aminés (les acides aminés de même nature étant représentés par une même couleur), `seaview` est très pratique pour corriger les imperfections des alignements faits par `clustalw`.

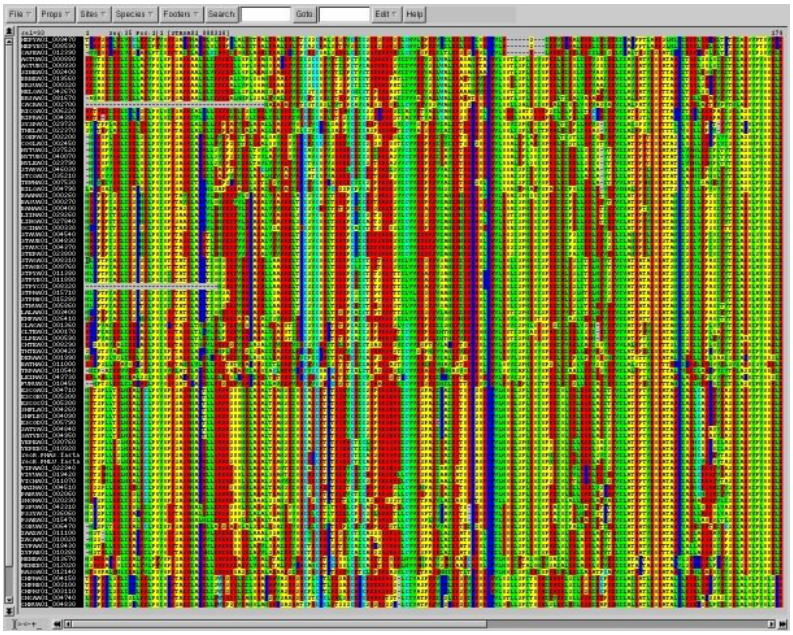


Fig. 3.1 – L'interface graphique du programme `seaview`.

- **phylip, bionj, puzzle, go_puzzleboot** : `phylip`, `bionj` et `puzzle` sont des programmes de phylogénie. Ils réalisent un arbre phylogénétique à partir d'un alignement de plusieurs séquences. Ils permettent, en outre, de calculer les valeurs de *bootstrap* (voir page 14) associées à chaque embranchement de l'arbre. `go_puzzleboot` est un script d'Eduardo Rocha qui appelle les différentes fonctions de `phylip` et de `bionj` au moment opportun, et avec les paramètres adéquats, de sorte que nous n'avons pas eu à nous en préoccuper : ce script fournit les résultats souhaités à partir d'un alignement finalisé.

II Programmes originaux

Malgré la multitude de petits programmes déjà fournis (voir section I), nous avons écrit quelques programmes originaux (en langage Perl), qui ont permis d'automatiser certaines tâches de façon plus astucieuse. Le code source de ces programmes est disponible entièrement dans l'annexe C, ou téléchargeable à l'adresse suivante :

`www.eleves.ens.fr/home/cornet/Stage/Programmes`

Ils ont tous été écrits en langage **Perl**, particulièrement adapté pour traiter des chaînes de caractères ou, de façon plus générale, des fichiers de type texte. Ils sont tous interactifs et ont été programmés pour atteindre une portabilité maximale : ils pourront resservir aux chercheurs du laboratoire d'accueil, y compris sous d'autres systèmes d'exploitation, sans modification nécessaire.

- **Parque** : vu le nombre très élevé de fois où l'on aurait eu besoin d'appeler le programme `opscan` au cours de notre étude, il semblait indispensable d'écrire un programme qui puisse l'appeler de façon répétée. Le programme `parque` est interactif ; il demande le fichier contenant le gène de départ et recherche pour chaque génome bactérien (c'est le comportement par défaut, mais on peut aussi lui proposer une liste de génomes différente) le gène qui correspond le mieux au gène donné. `parque` produit alors un fichier contenant la référence[1] de tous les gènes qu'il a trouvés.
- **Valkyrie** : le résultat donné par `parque` n'est pas suffisamment fiable en soi, puisque le programme a recherché des gènes orthologues à partir d'un gène appartenant à un seul organisme particulier. Il est donc plus judicieux d'appeler `parque` deux fois, en prenant pour base deux gènes (supposés orthologues) appartenant à deux organismes différents, aussi éloignés que possible d'un point de vue phylogénétique. C'est alors que `valkyrie` intervient : il se charge de comparer les résultats fournis par `parque` et de ne conserver que l'intersection de ces deux résultats.
- **GetEntries** : ce programme est une sorte de généralisation de `getentry` (décrit plus haut). Il produit, à partir d'une liste de gènes[2] fournie par `valkyrie` ou `parque`, un fichier au format « fasta » (format standard en bioinformatique) contenant la séquence de tous les gènes désirés. C'est à partir de ce dernier fichier que `clustalw` opère son alignement.
- **Distance** : probablement l'un des plus complexes, `distance` rassemble toutes les matrices de distances (voir en page 13) des gènes étudiés et rassemble toutes les valeurs en une matrice « linéarisée », d'une douzaine de milliers de lignes et de vingt-trois colonnes.
- **Build** : ce petit script n'a pas servi directement aux recherches, mais il a été extrêmement utile pour en embrasser les résultats avec plus d'aisance. En effet, les gènes découverts au fur et à mesure (ainsi que leurs numéros d'identification) étaient simplement consignés à la main dans un fichier texte, de format très simple. `build` rassemble toutes ces informations et formatte les tableaux qui figurent à partir de la page 33. En particulier, le premier d'entre eux (un tableau « booléen » : l'organisme i a, ou n'a pas, le gène g) permet d'avoir une vision extrêmement globale sur l'avancement et la signification des recherches effectuées.

> L'élaboration du programme `distance` s'est révélée délicate. Linéariser une matrice de distances implique d'attribuer une ligne à chaque couple (i, j) de génomes. Il existe une

[1] Tous les génomes disponibles ont été indexés selon une nomenclature propre au laboratoire d'accueil, chaque gène recevant une référence à six chiffres, qui lui est propre pour un génome donné.

[2] Une telle liste ne contient pas les séquences des gènes cités, mais uniquement leurs références.

telle matrice pour chacun des vingt-trois gènes étudiés : il faut toutes les linéariser puis les rassembler en une matrice unique, comportant environ 12000 lignes. En outre, les génomes ne sont classés suivant aucun ordre particulier dans ces vingt-trois matrices, qui sont stockées sous la forme de fichiers texte où une ligne du fichier contient sept coefficients, une ligne réelle de la matrice s'étalant donc sur une dizaine de lignes de texte. Tout ceci a nécessité une gestion complexe des indices des lignes, colonnes (dans la matrice finale et dans chacune des matrices de départ), lignes du fichier texte, position sur une ligne, etc.

III Démarche

Nous présentons ici, très schématiquement, la démarche suivie pour chacun des gènes g_i étudiés, où $i \in [\![1 ; 25]\!]$.

1. On commence par récupérer la séquence du gène g_i dans un génome relativement bien connu (*Escherichia coli*, la plupart du temps, ou *Bacillus subtilis* si *E. coli* ne le possède pas) à l'aide du programme **GetEntry**. Ce gène peut être recherché à partir de son nom, ou directement grâce à sa référence, lorsqu'elle est connue.
2. On recherche alors, à l'aide du programme **Parque**, tous les gènes qui ressemblent au maximum au gène g_i (les *best matches*). `parque` appelle `opscan` avec comme paramètres la matrice BLOSUM60 et un rapport maximal de $1,4$ entre les longueurs de séquences orthologues.
3. Les deux premières étapes sont répétées, en partant d'un autre génome, le plus éloigné possible du premier. On dispose ainsi, à la fin de cette étape, de deux listes de gènes, supposés orthologues à g_i. Précisons que `parque` (et donc `opscan`) ne fournit pas nécessairement un *best match* pour chaque génome $\mathcal{G}$ étudié. À cela, on peut proposer deux explications :
   - soit les paramètres d'`opscan` étaient trop restrictifs et, lorsque le programme a examiné le gène de $\mathcal{G}$ orthologue à g_i, il ne l'a pas trouvé suffisamment ressemblant pour le retenir – c'est pour cela que nous avons toujours appelé `opscan` avec des paramètres de tolérance maximale ;
   - soit le gène g_i n'existe pas dans le génome $\mathcal{G}$.
4. L'étape suivante consiste à comparer les deux listes dont on dispose, et à ne retenir que leur intersection, afin de diminuer l'incertitude des résultats. Ceci est réalisé automatiquement au moyen du programme **Valkyrie**.
5. Il est possible que certains gènes aient déjà été répertoriés comme orthologues de g_i (grâce à des recherches antérieures) ; dans ce cas le nom de g_i figure dans leur description. Un appel judicieux de la commande **grep** permet de retrouver tous les cas où un gène a déjà été identifié comme orthologue à g_i. Il s'agit ensuite de comparer ces résultats avec les recherches déjà effectuées, et d'intégrer les éventuelles lacunes pour les étapes suivantes. À ce stade, on dispose donc d'une liste complète des références des gènes orthologues à g_i, dans l'ensemble des génomes étudiés.
6. Le programme **Getentries** permet de passer d'une simple liste de références à un fichier complet, contenant toutes les séquences des gènes orthologues, au format fasta.
7. Il est, bien entendu, difficile de comparer deux séquences sans d'abord les aligner (voir l'exemple de la page 13). C'est **Clustalw** qui se charge d'effectuer les alignements multiples de tous les gènes orthologues à g_i.
8. On examine ensuite l'alignement produit par `clustalw` avec le programme **Seaview** et on fait les modifications qui s'imposent. Cette édition manuelle a deux buts distincts.
   - L'alignement effectué par `clustalw` n'est pas toujours parfait ; l'utilisation de couleurs adéquates (pour une capture d'écran montrant l'interface de `seaview`, voir en page 27) permet parfois de détecter de petites erreurs dans cet alignement.

- Certaines parties de l'alignement contiennent presque exclusivement des *gaps* (c'est le cas lorsque quelques séquences sont sensiblement plus longues que les autres), et d'autres montrent des régions de la protéine qui ont été très mal conservées[3], ce qui se traduit par des séquences très différentes. Ces deux types de régions apportent en réalité plus de « bruit » que de « signal » pour la construction de l'arbre phylogénétique ; c'est pourquoi elles sont supprimées.

9. La dernière étape consiste à lancer le script **Go_puzzleboot** qui, à partir de l'alignement final (épuré de ses régions bruitées), construit la matrice des distances, l'arbre phylogénétique et calcule les valeurs de *bootstrap* pour chaque embranchement.

À la fin de cette procédure, trois types de données nous intéressent et, une fois rassemblés, constitueront nos résultats expérimentaux : ① l'arbre phylogénétique lui-même ; ② la matrice des distances, qui contribuera à réaliser un graphe global des vitesses d'évolution des gènes grâce au programme `distance` ; ③ les références de tous les gènes orthologues à g_i, qui seront répertoriées dans un tableau récapitulatif.

> Nous souhaitons attirer l'attention du lecteur sur l'ordre de grandeur des temps d'exécution des différents programmes cités ci-dessus, afin de montrer le versant « pratique » des valeurs de complexité temporelle calculées plus haut sur une machine de 2004.
>
> En effet, si certaines étapes (par exemple l'étape notée 1 ci-dessus, ou encore la numéro 4) sont très rapides (quelques secondes), d'autres prennent un temps relativement long et demandent de s'organiser pour gérer correctement les capacités de calcul de la machine et optimiser l'ordre des tâches en fonction de leur temps d'exécution. Ainsi, là où `parque` (étape 2) a fini son travail au bout de quelques minutes, `clustalw` demande jusqu'à une heure pour terminer son alignement, et la création d'un arbre phylogénétique avec calcul des valeurs de *bootstrap* et de la matrice des distances peut prendre deux ou trois jours.

[3] Si l'on suppose que toutes ces protéines ont la même fonction au sein de leurs organismes respectifs, on peut dire que les zones qui sont très bien conservées correspondent probablement aux régions fonctionnelles de la protéine tandis que les autres ont souvent une importance moins significative, et ne contribuent probablement qu'à la géométrie globale de la protéine dans l'espace.

Le darwinisme postule la survivance du plus apte. Comment mesure-t-on l'aptitude ? Par la survie. Le darwinisme postule donc la survivance des survivants. N'est-ce pas une tautologie ?

Michel Rio, *Les jungles pensives*, p. 65[1].

Chapitre 4

Résultats

Il n'était pas certain, au début du stage, que nous puissions traiter l'ensemble des vingt-trois gènes proposés pour l'ensemble des cent dix-sept génomes disponibles. Cependant, l'écriture de programmes originaux a permis d'automatiser certains processus, et nous avons finalement pu traiter l'ensemble des données à notre disposition.

I Tableau des gènes

Les données présentées sont découpées en deux tableaux, à partir de la page 33. Le premier donne une information de type « booléen » (une bactérie possède un gène donné, ou pas) ; le second, étalé sur deux pages, présente les identificateurs (numérotation interne au laboratoire d'accueil) de tous les gènes reconnus. Dans le premier tableau, nous avons ajouté (à droite) quatre colonnes supplémentaires qui testaient la présence conjuguée de plusieurs gènes qui sont supposés faire partie d'un complexe ou du moins fonctionner de concert : de gauche à droite, recBCD, recFOR, ruvAB et addAB. Certaines particularités de cette distribution de gènes attirent notre attention.

- **Gènes essentiels** : on remarque que certains des gènes étudiés existent chez presque toutes les bactéries ; c'est le cas notamment de *recA*, *recR*, *ruvA* et *ruvB*.
- **Gènes « surestimés »** : en revanche, certains gènes, très étudiés, sont beaucoup moins représentés qu'attendu, comme *recB*, *recC* et *recD*.
- **Cohérence des complexes** : les gènes codant pour des bactéries qui sont supposées fonctionner en coopération apparaissent ensemble, en général.
  - Les gènes *ruvA* et *ruvB* apparaissent systématiquement en couple (pas d'exception), ce qui renforce l'hypothèse d'un fonctionnement totalement coopératif (complexe).
  - Malgré quelques exceptions, *recB*, *recC* et *recD* sont présents ensemble la plupart du temps. On peut imaginer que l'une ou l'autre des protéines correspondantes puissent remplir certaines fontions à elles seules, mais le plus probable, dans les situations de séparations, est qu'il s'agit simplement de vestiges qui ne sont plus fonctionnels.
  - *addA* et *addB* apparaissent eux aussi ensemble dans la majorité des cas. En observant les colonnes de droite (« BCD » et « aAB »), on remarque que les gènes correspondant aux complexes addAB et recBCD ne sont jamais présents simultanément (addAB est présent uniquement chez les firmicutes), ce qui conforte l'hypothèse que ces deux complexes remplissent des fonctions analogues, chez des organismes de nature différente.
- **Gènes écartés** : trois gènes qui apparaissaient très peu, *recE*, *recT* et *rusA*, ont été écartés des résultats expérimentaux. Leur présence n'est probablement pas critique pour le bon fonctionnement de la recombinaison.

[1] Une réfutation de la thèse selon laquelle la théorie de l'évolution serait tautologique figure dans Stephen J. Gould, *Darwin et les grandes énigmes de la vie – Réflexions sur l'histoire naturelle*, traduction de Daniel Lemoine, Éd. du Seuil, Paris, 1997.

- **Organismes marginaux** : certains organismes se démarquent par le faible nombre de gènes détectés chez eux. Par exemple, nous n'avons détecté aucun gène lié à la recombinaison, mis à part quelques restes de recA, chez *Onion yellows phytoplasma* (`onph`). Il s'agit d'une bactérie très particulière, qui n'a jamais pu être séparée de son hôte (une plante) ; le séquençage de son génôme a donc dû être effectué sur des extraits de plante, et il n'est donc peut-être pas extrêmement fiable. Trois γ-protéobactéries (`buap, busp, cabl`) possèdent très peu de gènes, et n'ont pas recA – ce sont les seules avec `onph`. Les mollicutes possèdent en moyenne moins de gènes que les autres groupes phylogénétiques, mais conservent tout de même *recA* et *ruvAB*. Toutes ces bactéries sont des organismes fortement dépendants de leur hôte. En outre, beaucoup sont intracellulaires, ce qui les isole des autres bactéries ainsi que des principaux facteurs mutagéniques. Dans ces conditions, la recombinaison leur est beaucoup moins nécessaire, ce qui pourrait expliquer le peu de gène retrouvés. Par ailleurs, ce sont des génomes qui évoluent très vite[2] (comme l'attestent la longueur des branches des *Mycoplasma* sur les arbres phylogénétiques, en particulier celui que nous présentons dans ce rapport) et les algorithmes utilisés n'ont peut-être pas reconnu tous leurs gènes.
- Globalement, on observe tout de même une certaine homogénéité dans le tableau : si certains gènes/complexes sont manifestement importants pour la plupart des génomes étudiés (*recA, ruvAB, recBCD/addAB*, etc.), d'autres sont plutôt propres à certains organismes particuliers.

II Synthèse

Un schéma synthétique des gènes identifiés dans chacune des principales branches de l'arbre des bactéries figure page 36. Il permet de présenter une vision globale des tableaux de données « brutes ».

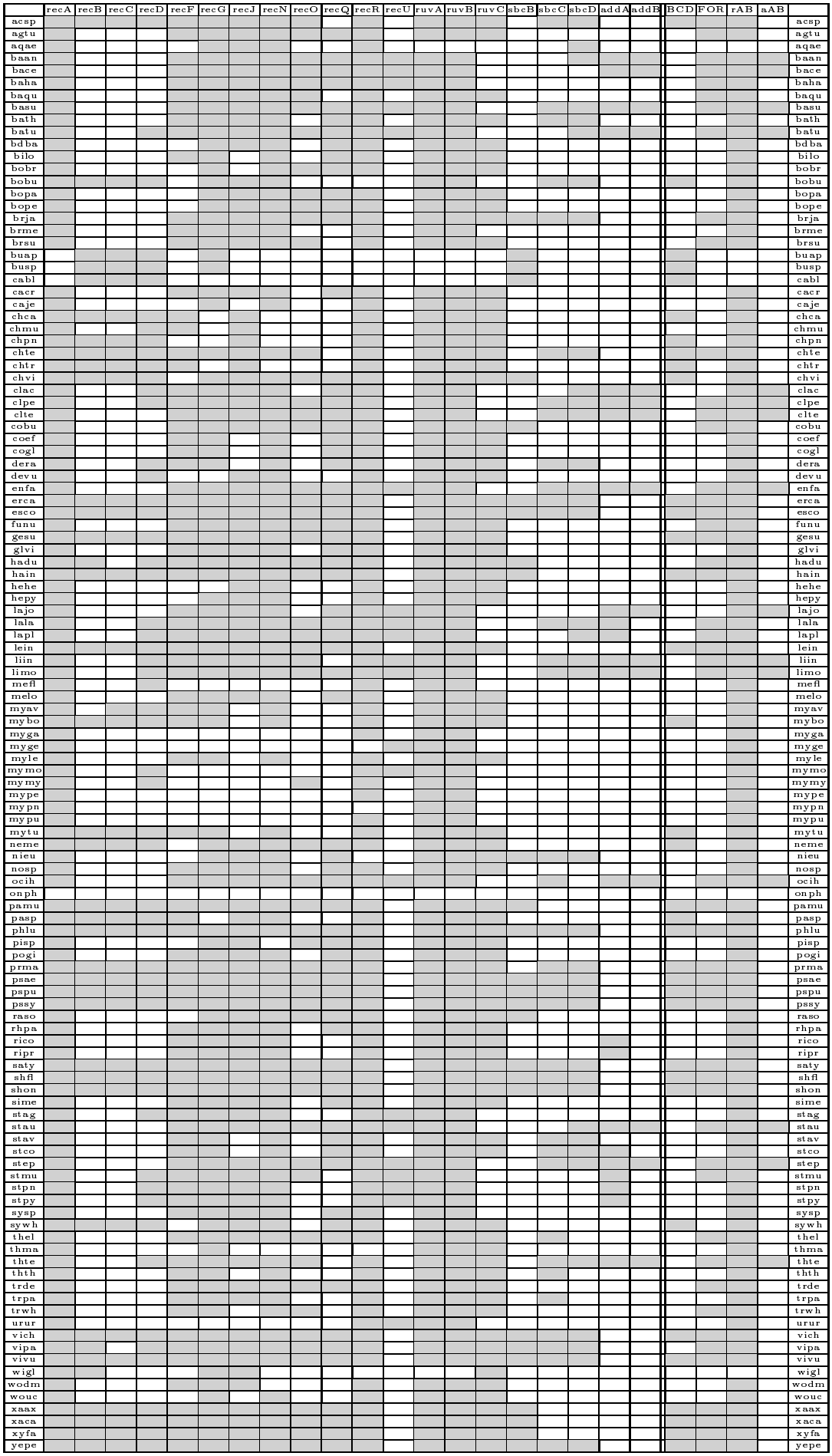
III Arbres phylogénétiques

Nous montrons page 37 l'un des vingt-trois arbres phylogénétiques qui ont été construits (celui de recA). Tous les arbres sont disponibles, aux formats PDF et PostScript, à l'adresse

`www.eleves.ens.fr/home/cornet/Stage/Arbres`

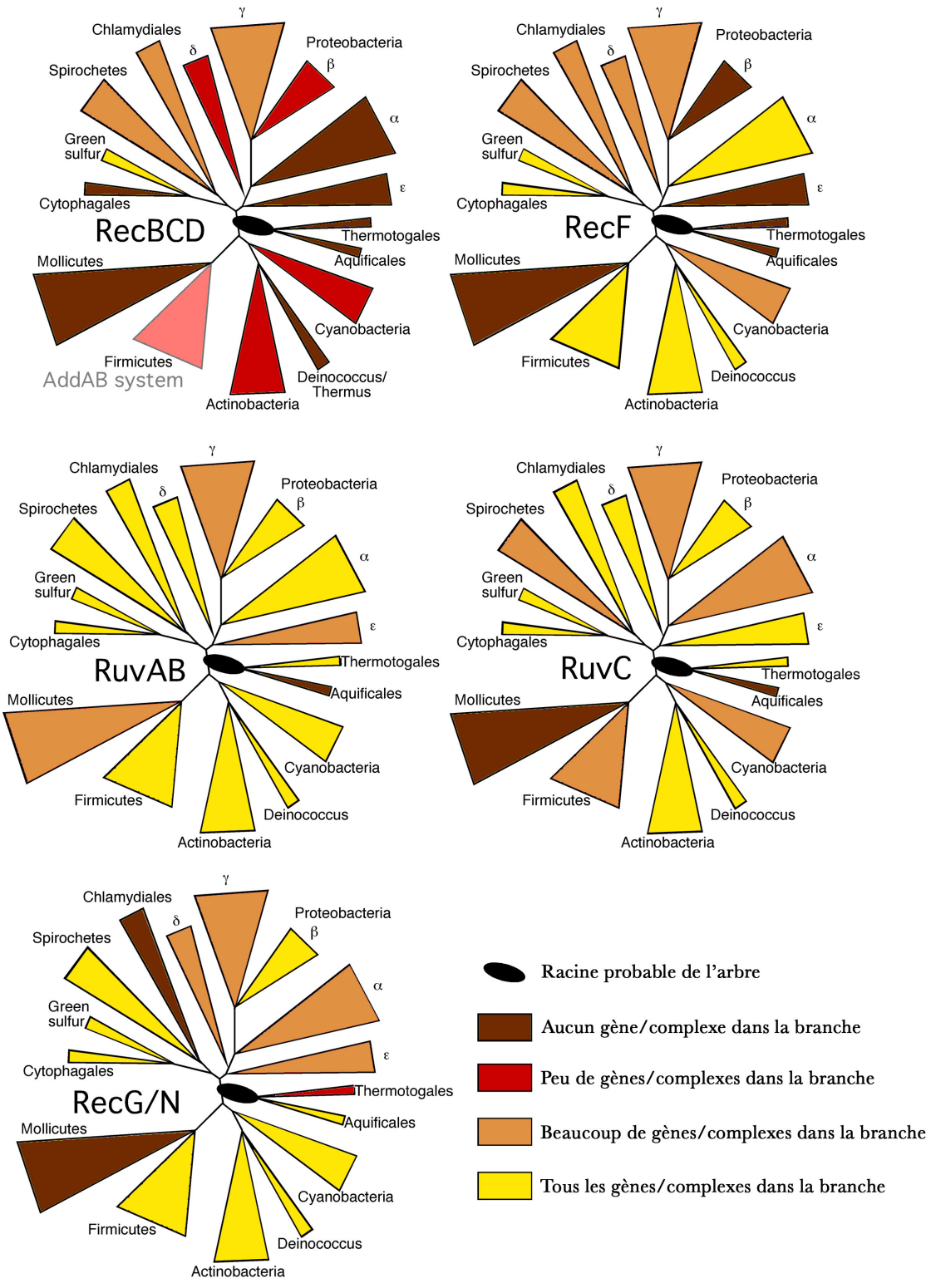
Ces arbres sont tout à fait en accord avec la phylogénie bactérienne communément admise de nos jours.

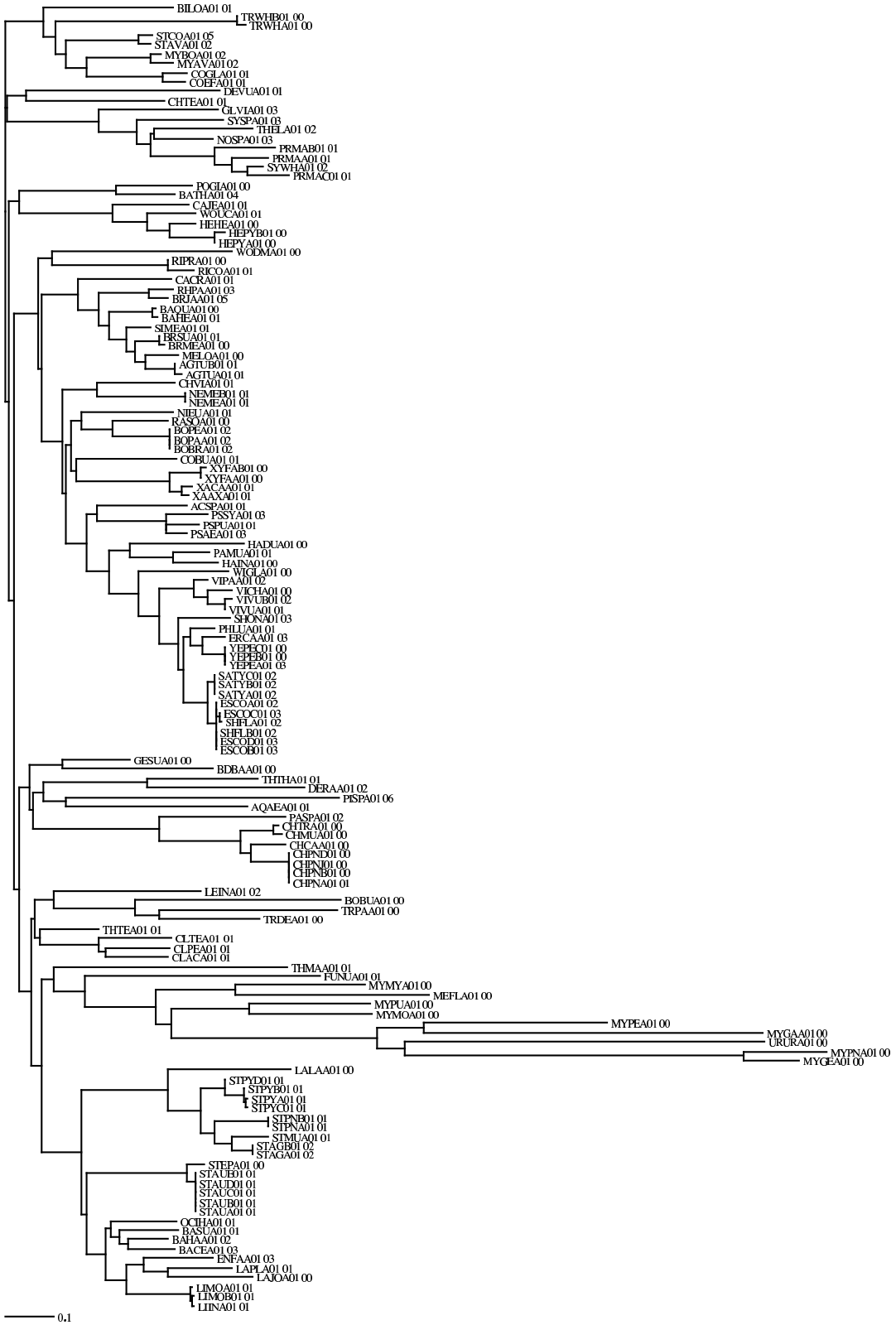
[2] Cette vitesse d'évolution élevée est probablement due leur ADN-polymérase (protéine qui réplique l'ADN) assez peu performante (génère beaucoup d'erreurs de réplication).


recA recB recC recD recF recG recJ recN recO recQ recR recU ruvA ruvB ruvC sbcB sbcC sbcD addA addB BCD FOR rAB aAB
acsp agtu aqae baan bace baha baqu basu bath batu bdba bilo bobr bobu bopa bope brja brme brsu buap busp cabl cacr caje chca chmu chpn chte chtr chvi clac clpe clte cobu coef cogl dera devu enfa erca esco funu gesu glvi hadu hain hehe hepy lajo lala lapl lein liin limo mefl melo myav mybo myga myge myle mymo mymy mype mypn mypu mytu neme nieu nosp ocih onph pamu pasp phlu pisp pogi prma psae pspu pssy raso rhpa rico ripr saty shfl shon sime stag stau stav stco step stmu stpn stpy sysp sywh thel thma thte thth trde trpa trwh urur vich vipa vivu wigl wodm wouc xaax xaca xyfa yepe

	recA	recB	recC	recD	recF	recG	recJ	recN	recO	recQ(S)
acsp	013150	003770	003760	003780	000030	030190	032530	003360	024160	
agtu	018190				000670	017290	015450	020150	010380	000560
aqae	015310					014590	015350	004120		
baan	036850				000040	037580	043800	041490	042690	026880
bace	037030				000040	037750	043170	043120	044450	029190
baha	024660				000040	025780	013190	028610	014480	016900
baqu	007940				001130	007620	006610	008760	004290	
basu	017770				000040	016700	028470	025080	026130	023860
bath	046920				043350	040000	040150	013880		018810
batu	035920,035930			041970	000040	036680	042120	039870	041070	014530,026530
bdba	004730					021490	020640	029400		029750
bilo	013900				006140	003090		010200		011310
bobr	021040					029660		039820	037830	040510
bobu	001310	006500	006510	006490		005930	002560	008530		
bopa	026670					030060	020730	035220	033210	036050
bope	025410					016020	010990	024970	024190	034170
brja	058010				008350	046340	044020	066400	050940	002430
brme	008140				008140	007030	007520	006100		
brsu	011830				000030	005610	012530	014020	006420	
buap		003940	004310	004330		002690				
busp		004460	004450	004470		002890				
cabl		002670	002650	002660						
cacr	010950				001590	014530	013980	019990		034870
caje	016460					004460		006300		
chca	009840	000070	000080	006390	004370		001770			
chmu	000180			002870	003290		006780			
chpn	010180	007640	007630	005930			001750			
chte	019480	010920	010910	010930	023080	015980	003760	016410	021140	
chtr	006900	006790	006800	000330	000770		004730			
chvi	016550	041810	041820	041780		009470	025320	023780	021250	004820
cjac	018410			028880	000040	017630	022500	020930		027240
cipe	017180			022280	000040	017760	021160	018600	020650	003800
clte	011830				000450	011280	016040	014460	018490	019170
cobu	010310				000030	003050	004990	012690	014690	004660
coef	018650				000040	014510		015640		009810
cogl	019100				000030	012940		013800		008690
dera	023450			018970	010960	019100		014830		012970
devu	011140			009170			009260	025070		
enfa	030290				000040	029770	016370	009390	023040	015030,023530
erca	034160	010110	010090	010120	045090	000380	007890	008570	033070	042330
esco	026950	028160	028180	028150	036940	036460	028880	026130	025620	038180
funu	011800				006320	001640	010070	009010	021250	012110
gesu	001500	015510	015500	015520	000020	013380	026420	020930	005810	009080
glvi	034570				024330	012900	000410	002740		040260
hadu	003710	009580		005520	007510	016810	011790	007250	014030	004310
hain	006080	013440	009690	013450	010210	017720	012400	000700	003410	007540
hehe	006540						013880	013320		
hepy	001600					015640	003580	014310		
lajo	007000				000040	013430	012880	013580		010390
lala	003540			017400	019750	022440	006320	008600	000580	018080
lapl	019980			018640	000040	014090	017970	013890	016930	008360,016220
lein	021990	009750	009740	009760	000030	039820	030230	023410	017000	009730
liin	014550			015640	000050	019720	015800	014250	015170	
limo	014200			015310	000050	018610	015470	013900	014820	028040
mefl	002140			003320						
melo	000240				043200	006550	007300	012100		031210
myav	028480		040940	040910	000030	030090		014030		
mybo	027930	006520	006530	006500	000030	030350		017450		
myga	005070									
myge	003860									
myle	010030				000030	017030		013880		
mymo	005450			000940,000970						
mymy	004040			004970,006350					004470	
mype	006670									
mypn	005260									
mypu	002630									
mytu	027690	003660	006340	006320	000030	030050		017150		
neme	013960	009770	019340	013620		006680	010290	009320	019980	021740
nieu	019700					018870	000130	015100		026100
nosp	033030				034050	048350	051790	050090		048890
ocih	017040				000040	016000	021120	019540	020360	018940
onph										
pamu	018170	005160	009610	005170	011590	009190	001920	003320	018670	014270
pasp	020380	000090	000080	000100	017420		002530	006930		
phlu	012690	006420	006400	006430	000030	002640	036180	034420	034060	047030
pisp	066100					005570	048920		025530	033630,036540,052340,058380,073520
pogi	007910				003560	003090	000490	016310		003700,016200
prma	017510	011160	011140	011170	017630	008490	010080	019220	004100	001970
psae	036560	043330	043340	043320	000030	054100	037630	048240	007790	033820
pspu	016310	046300	046310	046290	000120	052620	014800	046880	014380	044740
pssy	039990	007780	007770	007790	000030	000620	014800	044670	041800	016350
raso	005540					027560	011280	026960	010720	030770
rhpa	038860				000030	026860	025260	035540		048780
rico	012130				000340	009250	007810	002350		
ripr	007580				000290	005920	005270	001820		
saty	027860	029640	029660	029570	038020	037090	030120	026600	025550	039190
shfl	027910	029020	029040	029010	038480	037750	029500	027380	026850	039940
shon	034420	021530	021540	021520	000100	043630	009600	034700	013600	042390
sime	018390				001940	016370	017610	022060		023590
stag	021360			017690	021980	017280	012700	005800		
stau	011880				000040	011300	015360	014200	014660	007130,013830
stav	024990				043610	026820		065750		049020
stco	056440				037750	054460		017220		044690
step	009690				000030	009080	013240	012060	012590	005030
stmu	019640			017690	020290	017330	013810	005730	000420	
stpn	018510			003690	021490	015950	005760	011260		
stpy	016850			014800	019150	014360	007450	011980		
sysp	031970				010060	022710	009910	019130		008920
sywh	021030	009320	009290	009330		011090	012230	025750		019890
thel	021210				024870	007470	012610	008130	022370	008130
thma	019070					002180				
thte	013110			010090	000040	014250	020080	012410	009440	
thth	014870				017540	009140		011740		
trde	008650				002290	025600	021740	012900	004330	003230
trpa	007330				000030	007280	007470	004660		
trwh	006480				000030	005990		001080	003010	
urur	000820									
vich	005830	023830	023850	023820	000140	027770	024790	009010	025200	002100
vipa	026180	024310		024300	000130	001680	005230	006670	026380	031430
vivu	015440	017440	017430	017450	009730	008340,008350	006870	003640	015210	009240
wigl	005400	005050				000280	003220			
wodm	009660				011980	007600	002850			
wouc	013720					003290		014810		
xaax	017380	043340	043350	043330	000030	033890	018630	015170	013260	031220
xaca	017200	041970	041980	041960	000030	032360	018440	014690	012740	029450
xyfa	001300	004400	016240	004420	000030	003690	011370	023730	022770	014130
yepe	032660	010060	010040	010070	040630	000400	008760	010990	026910	038070

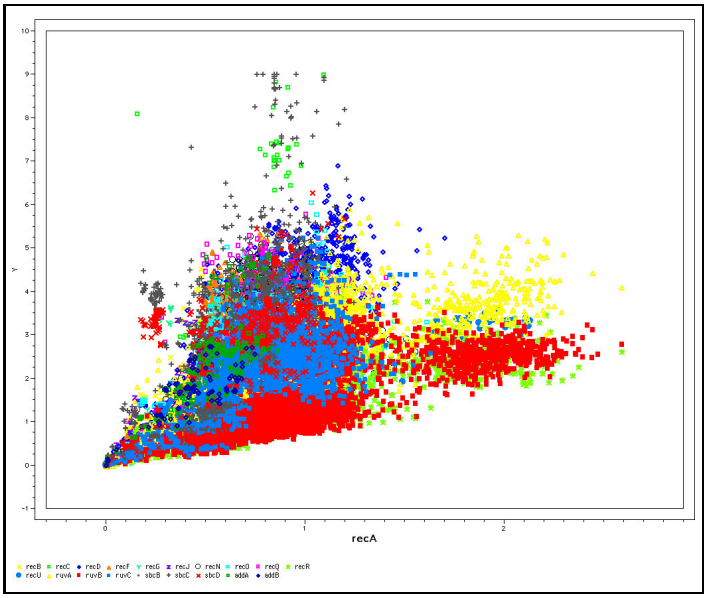
	recR	recU	ruvA	ruvB	ruvC	sbcB	sbcC	sbcD	addA	addB
acsp	021690		024480	024490	019320		008650	008660		
agtu	000880		011360	011370	007210					
aqae	010570							013430		
baan	000260	015460	043920	043910				022640	011480	011470
bace	000250	015320	043290	043280					011220	011210
baha	000400	017860	013030	013040						
baqu	002180		011860	011850	011870					
basu	000270	023150	028590	028580			011470	011460	011450	011440
bath	011000		020170	033500	016960		002460	002450		
batu	000260	015170	042240	042230				022140	011290	011280
bdba	034470		022940	022930	022950					
bilo	004790		007060	007070	007050					
bobr	014520		040470	040500	040460					
bobu			000230	000220			008530	008520		
bopa	012390		036010	036040	036000					
bope	015420		034130	034160	034120					
brja	081670		015540	015550	015530	000680	019390	019440		
brme	019560		003430	003440						
brsu	000320		016820	016810	016830					
buap						005340				
busp						005470				
cabl						004600				
cacr	002700		032590	032580	032600					
caje	012390		007750	013400	017060					
chca	004740		001190	003990	001180					
chmu	004830		007310	002960	007320					
chpn	004150		001210	003320	001200					
chte	008290		002740	016530	016870		011700	011710		
chtr	002530		005330	000420	005340					
chvi	016600		043400	043320	043420	013670				
ciac	001360		023030	023020				027730	022800	022810
cipe	000590		019980	019970			002450	002440	000300	000290
clte	000170		020280	020270			005220	005210	006430	006170
cobu	006470		015360	015380	015350	013330				
coef	002200		017900	017890	017910					
cogl	002450		016210	016200	016220					
dera	001990		012820	006030	004390		019160	019150		
devu	032650		022930	022920	022950					
enfa	026410	011150	000610	000620			025700	025710	010790	010780
erca	011960		025160	025150	025170	025890	011260	011270		
esco	004710		018590	018580	018610	020080	003970	003980		
funu	010450		017370	018500	008470					
gesu	001010		010830	010840	010820					
glvi	035400		029190	031560	033400					
hadu	002980		015380	015370	005230	012390				
hain	004510		003240	003230	003250	014010				
hehe	015560		005320	005240	011460					
hepy	009470		009030	003660	008980					
lajo	004480	010600	004910	004920					010670	010680
lala	003400	005440	022070	022060			013210	013220	000040	
lapl	006150	015150	019900	019890				013400	023120	
lein	043730		029280	008190	007310					
liin	029260	020520	015880	015870			017060	017070	024180	024190
limo	027840	019410	015550	015540			016670	016680	023230	023240
mefl	007060		004380	004370						
melo	042670		030110	030100	030140					
myav	003160		010370	010380	010360					
mybo	037830		026570	026560	026580					
myga	006360		006410	006400						
myge		004030	004090	004100						
myle	023790		030110	004950	004930					
mymo	000750	005070	002830	002840						
mymy	000470		005250	005260						
mype	008370		002910	002900						
mypn			005750	005760						
mypu	000510		006850	006860						
mytu	040070		026210	026200	026220					
neme	013670		021820	013730	015810					
nieu			002220	002230	002210	006920	014230	014210		
nosp	050250		003780	029220	023200					
ocih	000330	018350	021230	021220			023380		012610	012600
onph										
pamu	002060		009770	009760	009780	006110				
pasp	017600		000200	011360	000190					
phlu	039060		021720	021730	021710	029110	039800	039790		
pisp	036620		025650	039230	026040					
pogi	011120		007330	004350	011740					
prma	011220		007740	018130	011720		007200	007210		
psae	015470		009770	009780	009760	043650	043310	043300		
pspu	042310		012210	012220	012200	013690	020190	020200		
pssy	036060		039440	039430	039450	042770	037350	037340		
raso	012140		005030	005020	005050	031870				
rhpa	006260		011130	011150	011120					
rico	006220		005440	005460	001630				006920	
ripr	004380		003850	003860	001190				004920,007310	
saty	004840		018770	019540	018800	020450	004000	003940		
shfl	004260		019060	017380	019080	021210	003430	003440		
shon	020230		021300	021290	024310	027960	028430	028440		
sime	002400		028000	027990	028010					
stag	008310	003690	021380	000810						
stau	004540	013500	015420	015410				012460	008720	008710
stav	046030		069180	069190	069170		071430	071420		
stco	035210		014610	014600	014620		012460	012470	046790	
step	023800	011430	013310	013300			010350	010340	006670	006660
stmu	005860	004660	019670	000790					014050	
stpn	015710	003500	001730	002500					010780	
stpy	011380	013300	016880	000550					006180	
sysp	029720		010280	024170						
sywh	009430		010510	001370	007260					
thel	022370		005440	022910	011540		022080			
thma			001660	017750	005990					
thte	000420		011250	011260	011240		002580,019390	002570	002560	002550
thth	012510		017290	000380	007390		009340			
trde	015380		018500	019000	018510					
trpa	010540		005730	001720	005460		006630			
trwh	007530		002840	002850	002830					
urur	000870	003060	004700	004690						
vich	011070		018930	018920	018940	012890	005100	005090		
vipa	022340		010990	011000	010960	013500	010380	010390		
vivu	019420		020850	020867,00208770	020830	026760	000760	000750		
wigl					000120					
wodm	010890		010260	010250	001320					
wouc	006520		004090	016670	020440					
xaax	011100		031470	031450	031480	015970				
xaca	010020		030230	030210	030240	015490				
xyfa	018430		019380	019360	019390	020520				
yepe	010920		020360	020370	020350	015380	031620	010060		


RecBCD
AddAB system
RecF
RuvAB
RuvC
RecG/N
Chlamydiales
Spirochetes
Green sulfur
Cytophagales
Mollicutes
Firmicutes
Actinobacteria
Deinococcus/ Thermus
Deinococcus
Cyanobacteria
Aquificales
Thermotogales
Proteobacteria
α
β
γ
δ
ε
Racine probable de l'arbre
Aucun gène/complexe dans la branche
Peu de gènes/complexes dans la branche
Beaucoup de gènes/complexes dans la branche
Tous les gènes/complexes dans la branche

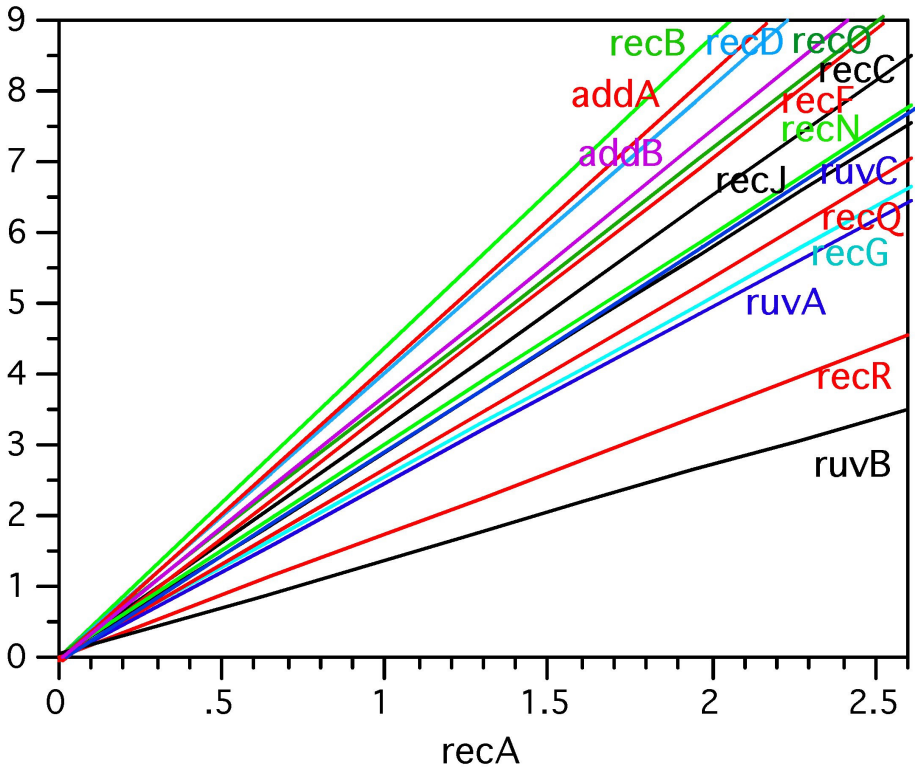

BILOA01 01
TRWHB01 00
TRWHA01 00
STCOA01 05
STAVA01 02
MYBOA01 02
MYAVA01 02
COGLA01 01
COEFA01 01
DEVUA01 01
CHTEA01 01
GLVIA01 03
SYSPA01 03
THELA01 02
NOSPA01 03
PRMAB01 01
PRMAA01 01
SYWHA01 02
PRMAC01 01
POGIA01 00
BATHA01 04
CAJEA01 01
WOUCA01 01
HEHEA01 00
HEPYB01 00
HEPYA01 00
WODMA01 00
RIPRA01 00
RICOA01 01
CACRA01 01
RHPAA01 03
BRJAA01 05
BAQUA01 00
BATEA01 01
SIMEA01 01
BRSUA01 01
BRMEA01 00
MELOA01 00
AGTUB01 01
AGTUA01 01
CHVIA01 01
NEMEB01 01
NEMEA01 01
NIEUA01 01
RASOA01 00
BOPEA01 02
BOPAA01 02
BOBRA01 02
COBUA01 01
XYFAB01 00
XYFAA01 00
XACAA01 01
XAAXA01 01
ACSPA01 01
PSSYA01 03
PSPUA01 01
PSAEA01 03
HADUA01 00
PAMUA01 01
HAINA01 00
WICLA01 00
VIPAA01 02
VICHA01 00
VIVUB01 02
VIVUA01 01
SHONA01 03
PHLUA01 01
ERCAA01 03
YEPEC01 00
YEPEB01 00
YEPEA01 03
SATYC01 02
SATYB01 02
SATYA01 02
ESCOA01 02
ESCOC01 03
SHFLA01 02
SHFLB01 02
ESCOD01 03
ESCOB01 03
GESUA01 00
BDBAA01 00
THTHA01 01
DERAA01 02
PISPA01 06
AQAEA01 01
PASPA01 02
CHTRA01 00
CHMUA01 00
CHCAA01 00
CHPND01 00
CHPNB01 00
CHPNB01 00
CHPNA01 01
LEINA01 02
BOBUA01 00
TRPAA01 00
TRDEA01 00
THTEA01 01
CLTEA01 01
CLPEA01 01
CLACA01 01
THMAA01 01
FUNUA01 01
MYMYA01 00
MEFLA01 00
MYPUA01 00
MYMOA01 00
MYPEA01 00
MYGAA01 00
URURA01 00
MYPNA01 00
MYGEA01 00
LALAA01 00
STPYD01 01
STPYB01 01
STPYA01 01
STPYC01 01
STPNB01 01
STPNA01 01
STMUA01 01
STAGB01 02
STAGA01 02
STEPA01 00
STAUE01 01
STAUD01 01
STAUC01 01
STAUB01 01
STAUA01 01
OCIHA01 01
BASUA01 01
BAHAA01 02
BACEA01 03
ENFAA01 03
LAPLA01 01
LAJOA01 00
LIMOA01 01
LIMOB01 01
LIINA01 01
0.1

IV Graphe des vitesses d'évolution des gènes

Le graphe ci-après est une représentation graphique brute des vitesses d'évolution des gènes, avec pour référence celle du gène *recA* (dont la vitesse d'évolution est la plus lente).



Le graphe représentant les régressions linéaires pour chaque gène est plus exploitable et beaucoup plus lisible ; les données sont là encore représentées en fonction de *recA*.



Formulons quelques remarques concernant ce graphique :

- La distribution de vitesses d'évolution est **plutôt large** : *recB* (le plus rapide) évolue un peu plus de quatre fois plus vite que *recA* (le plus lent).
- La principale observation que l'on peut émettre est la suivante : certains gènes dont les protéines agissent conjointement **restent relativement groupés** dans leur évolution. C'est

le cas de *addA* et *addB* ou de *recB*, *recC* et *recD* ou encore de *ruvA* et *ruvB*, qui se situent dans les vitesses les plus faibles. Le gène *recA*, choisi comme référence, est le plus conservé.

- On pourrait se poser la question suivante : les « trous » apparaissant dans notre tableau page 33 ne sont-ils pas dus à des vitesses d'évolution trop rapides de certains gènes, entraînant une non-reconnaissance en tant que gènes orthologues ? Pourtant, le gène *recB*, celui qui évolue le plus vite, reste notablement représenté dans le tableau ; de même, *recO* et *recF*, dont les vitesses sont au-dessus de la moyenne, sont présents dans presque tous les génomes. Ces constatations viennent ainsi renforcer la validité de nos résultats. Pour la renforcer davantage, nous avons utilisé le programme *Evolver*[3] : il s'est avéré que les gènes qu'`opscan` n'a pas repérés sont probablement très limités, et principalement localisés chez les mycoplasmes.
- L'allure générale des vitesses d'évolution tend à faire dire que *les gènes sont d'autant plus conservés qu'ils sont répandus*, mais après avoir tracé le graphe $n = f(a)$, où n est le nombre de génomes où apparaît un gène donné et a sa vitesse d'évolution (pente de la droite sur le graphe ci-dessus), on peut dire que si la tendance est nette, la courbe est irrégulière. Toutefois, notre vision de ce problème est peut-être faussée par le fait que notre échantillon de bactéries comprend beaucoup de γ-protéobactéries, ce qui a tendance à augmenter n pour certains gènes typiques de cette branche.

[3] En substance, *Evolver* utilise une méthode d'« évolution artificielle » (mutations aléatoires) et se sert des arbres phylogénétiques produits pour donner une idée du nombre de gènes qui ont pu ne pas être détectés.

Conclusion et perspectives

Nos travaux ont permis d'évaluer la distribution de l'ensemble des gènes (connus) impliqués dans la recombinaison homologue parmi les génomes bactériens actuellement disponibles ; ils ont également permis d'observer d'une façon très globale la vitesse d'évolution de ces gènes, et de voir comment la coopération entre plusieurs gènes les maintenait groupés, à travers les génomes et à travers les âges.

Cette recherche gagnera bien sûr à être complétée à mesure que de nouveaux génomes seront publiés. Mais la perspective la plus fascinante serait de pousser l'étude comparative jusqu'à l'explication, au niveau moléculaire, des phénomènes que nous avons mis en évidence. Certains gènes disparaissent chez plusieurs génomes ; pourquoi ? Comment remplir la fonction désormais manquante ? Une question plus passionnante encore serait : à quelle époque des gènes comme *recA* sont-ils apparus, et vue leur importance apparemment fondamentale pour le métabolisme des bactéries, comment les organismes antérieurs arrivaient-ils à s'en passer, ou à contourner la difficulté ? Une dernière proposition pourrait être d'étudier le rôle du transfert horizontal dans l'évolution des gènes étudiés. En effet, si la bonne adéquation de nos arbres phylogénétiques avec la phylogénie communément admise laisse supposer que le transfert horizontal n'a pas eu une importance capitale pour l'évolution de ces gènes, il apparaît pourtant que les seuls organismes qui sont privés du gène *recA* vivent totalement coupés du reste du monde bactérien (ce sont des bactéries intracellulaires absolument isolées). Il y a fort à parier que l'importance des phénomènes de transfert horizontal se révéleront dans le futur d'une importance bien plus grande que ce que nous avons soupçonné jusqu'à présent.

Annexes

Il semble que la perfection soit atteinte
non quand il n'y a plus rien à ajouter
mais quand il n'y a plus rien à retrancher.

Antoine de Saint-Exupéry, *Terre des hommes*, p. 98.

Annexe A : Génomes étudiés

Le tableau complet des noms des bactéries étudiées figure page suivante.

Annexe B : Le code des acides aminés

Code à une lettre	Code à trois lettres	Nom de l'acide aminé
A	Ala	alanine
C	Cys	cystéine
D	Asp	acide aspartique
E	Glu	acide glutamique
F	Phe	phénylalanine
G	Gly	glycine
H	His	histidine
I	Ile	isoleucine
K	Lys	lysine
L	Leu	leucine
M	Met	méthionine
N	Asn	asparagine
P	Pro	proline
Q	Gln	glutamine
R	Arg	arginine
S	Ser	sérine
T	Thr	thréonine
V	Val	valine
W	Trp	tryptophane
Y	Tyr	tyrosine

Code	Nom complet
acsp	*Acinetobacter sp.*
agtu	*Agrobacterium tumefaciens*
aqae	*Aquifex aeolicus*
baan	*Bacillus anthracis*
bace	*Bacillus cereus*
baha	*Bacillus halodurans*
baqu	*Bartonella quintana*
basu	*Bacillus subtilis*
bath	*Bacteroides thetaiotaomicron*
batu	*Bacillus thuringiensis*
bdba	*Bdellovibrio bacteriovorus*
bilo	*Bifidobacterium longus*
bobr	*Bordetella bronchiseptica*
bobu	*Borrelia burgdorferi*
brja	*Bradyrhizobium japonicum*
brme	*Brucella melitensis*
brsu	*Brucella suis*
buap	*Buchnera aphidicola*
busp	*Buchnera sp.*
cabl	*Candidatus blochmannia*
cacr	*Caulobacter crescentus*
caje	*Campylobacter jejuni*
chca	*Chlamydophila caviae*
chmu	*Chlamydia muridarum*
chpn	*Chlamydia pneumoniae*
chte	*Chlorobium tepidium*
chtr	*Chlamydia trachomatis*
chvi	*Chromobacterium violaceum*
clac	*Clostridium acetobutylicum*
clpe	*Clostridium perfringens*
clte	*Clostridium tetani*
cobu	*Coxiella burnetti*
coef	*Corynebacterium efficiens*
cogl	*Corynebacterium glutamicum*
dera	*Deinococcus radiodurans*
devu	*Desulfovibrio vulgaris*
enfa	*Enterococcus faecalis*
erca	*Erwinia carotovora*
esco	*Escherichia coli*
funu	*Fusobacteriumi nucleatum*
gesu	*Geobacter sulfurreducens*
glvi	*Gloeobacter violaceus*
hadu	*Haemophilius ducreyi*
hain	*Haemophilus influenzae*
hehe	*Helicobacter hepaticus*
hepy	*Helicobacter pylori*
lajo	*Lactobacillus johnsonii*
lala	*Lactococcus lactis*
lapl	*Lactobacillus planctarum*
lein	*Leptospira interrogans*
liin	*Listeria innocua*
limo	*Listeria monocytogenes*
mefl	*Mesoplasma florum*
melo	*Mesorhizobium loti*
myav	*Mycobacterium avium*
mybo	*Mycobacterium bovis*
myga	*Mycoplasma gallisepticum*
myge	*Mycoplasma genitalium*
myle	*Mycobacterium leprae*
mymo	*Mycoplasma mobile*
mymy	*Mycoplasma mycoides*
mype	*Mycoplasma penetrans*
mypn	*Mycoplasma pneumoniae*
mypu	*Mycoplasma pulmonis*
mytu	*Mycoplasma tuberculosis*
neme	*Neisseria meningitidis*
nieu	*Nitrosomonas europaea*
nosp	*Nostoc sp.*
ocih	*Oceanobacillus iheyensis*
onph	*Onion yellows phytoplasma*
pamu	*Pasteurella multocida*
pasp	*Parachlamydia sp.*
phlu	*Photorabdus luminescens*
pisp	*Pirellula sp.*
pogi	*Porphyromonas gingivalis*
prma	*Prochlorococcus marinus*
psae	*Pseudonomas aeruginosa*
pspu	*Pseudomonas putida*
pssy	*Pseudomonas syringae*
raso	*Ralstonia solanacearum*
rhpa	*Rhodopseudomonas palustris*
rico	*Rickettsia conorii*
ripr	*Rickettsia prowazekii*
saty	*Salmonella typhimurium*
shfl	*Shigella flexneri*
shon	*Shewanella oneidensis*
sime	*Sinorhizobium meliloti*
stag	*Streptococcus agalactiae*
stau	*Staphylococcus aureus*
stav	*Streptomyces avermitilis*
stco	*Streptomyces coelicolor*
step	*Staphylococcus epidermidis*
stmu	*Streptococcus mutans*
stpn	*Streptococcus pneumoniae*
stpy	*Stretococcus pyogenes*
sysp	*Synechocystis sp.*
sywh	*Synechococcus sp.*
thel	*Thermosynechococcus elongatus*
thma	*Thermotoga maritima*
thte	*Thermoanaerobacter tengcongensis*
thth	*Thermus thermophilus*
trde	*Treponema denticola*
trpa	*Treponema pallidum*
trwh	*Tropherima whipplei*
urur	*Ureaplasma urealyticum*
vich	*Vibrio cholerae*
vipa	*Vibrio parahaemolyticus*
vivu	*Vibrio vulnificus*
wigl	*Wigglesworthia glossinidia*
wodm	*Wolbachia drosophila melanogaster*
wouc	*Wolinella succinogenes*
xaax	*Xanthomonas axonopodis*
xyfa	*Xylella fastidiosa*
xaca	*Xanthomonas campestris*
yepe	*Yersinia pestis*

Annexe C : Programmes originaux

Nous montrons ici le code source (en langage Perl) des programmes originaux que nous avons été amené à écrire au cours du stage. Leur code source peut également être téléchargé à l'adresse suivante :

`www.eleves.ens.fr/home/cornet/Stage/Programmes`

1 Programme « Parque »

```
#!/usr/bin/perl

=head1
Programme "Parque", qui appelle opscan de façon récursive.
                    © Manu Cornet, 16 juin 2004
=cut

##############################################################################
################################ Description ################################
##############################################################################

print "Bonjour ! Je suis le programme Parque. Je sais éxécuter la
commande opscan sur plusieurs génomes à la fois.";

##############################################################################
############################## Chemins d'accès ##############################
##############################################################################

my $bacteriesd = "/home/manu/Stage/Bacteries";
my $matricesd = "/home/manu/Stage/Matrices";

##############################################################################
################################ Séquence ################################
##############################################################################

print
"\nJe vais tout d'abord vous demander de m'indiquer le fichier contenant la
séquence (ou opéron) à rechercher, puis le fichier dans lequel vous avez placé
le nom des génomes à analyser.\n" ;

# On commence par prendre le fichier avec la séquence à chercher.
print "\nVeuillez maintenant m'indiquer le chemin du fichier
contenant la séquence à rechercher.\n\n";

my $sequence=<STDIN>;

# Si la séquence ne contient qu'un retour chariot, on insulte l'utilisateur.
chomp($sequence);
if ($sequence eq "") {
    die "\n Vous ne m'avez pas donné de fichier de départ ! Je suis vexé... "
  . "À bientôt :o)"
}
else {print "\nMerci. "};

# On affiche le contenu pour éviter les mauvaises surprises.
print "Voici ce que contient le fichier $sequence :\n\n";
system "cat $sequence";

# Il faut à présent demander les génomes

##############################################################################
################################ Génomes ################################
##############################################################################


print "\n\nVeuillez à présent m'indiquer le chemin de fichier contenant
la liste des génomes à analyser. Cette liste doit contenir des noms de
génomes comprenant quatre lettre minuscules plus éventuellement une
lettre majuscule (souche), à raison d'un nom par ligne. Si la souche
n'est pas précisée dans le nom, je prendrai par défaut la souche notée
A.  Le fichier par défaut est /home/manu/Stage/Genomes\n\n";

my $genomes=<STDIN>;
chomp($genomes);

if ($genomes eq "") {
    $genomes="/home/manu/Stage/Genomes";

    print"\nVous ne m'avez pas fourni de chemin ; je prends
    donc le fichier $genomes.\n";
}

print "\nVoici les génomes que je vais analyser :\n";

my $compteurtab = 0;

open(FILEREAD,"<$genomes") or die $!;
while(<FILEREAD>) {
    chomp($_);
    print "$_\t";
    $compteurtab++;

# On veut passer à la ligne lorsqu'une ligne est remplie (calculé sur 80
# colonne)
    if ($compteurtab % 10 == 0) {
        print "\n"
    }
}

close(FILEREAD);

##############################################################################
################################ Matrice ################################
##############################################################################

print "\n\nIl faut maintenant que tout soit clair entre nous sur la
matrice de scoring que je vais utiliser. Vous pouvez soit me préciser le
nom de cette matrice (je saurai trouver le chemin tout seul), soit appuyer
simplement sur entrée, et j'utiliserai alors la matrice BLOSUM60.\n\n";

my $matrice = <STDIN>;
chomp($matrice);

# On garde une variable avec juste le nom de la matrice pour la synthèse
# des paramètres.
my $nom_matrice;

 if ($matrice eq "") {
    print "Vous n'avez pas précisé de matrice. Je vais utiliser BLOSUM60\n";

    $nom_matrice = "BLOSUM60";
    $matrice = "$matricesd/BLOSUM60"
} else {
    $nom_matrice = $matrice;
    $matrice = "$matricesd/$matrice";
}

print "Voici donc la bonne petite bouille de la matrice dont je vais me
servir : \n";
system "cat $matrice";

##############################################################################
################################ Sortie ################################
##############################################################################

print "\nPouvez-vous m'indiquer le nom du fichier dans lequel je dois
écrire mes résultats ? Entrez ce nom maintenant s'il vous plaît.\n
Le nom par défaut est Res.\n\n";

my $res = <STDIN>;
chomp($res);

if ($res eq "") {
    print "Vous n'avez rien écrit. Je prends donc 'Res'\n\n";
    $res = Res;
}

# On veut supprimer le fichier s'il existe déjà, mais on ne veut pas voir
# l'erreur s'il n'existe pas.

system "rm -rf $res 2> /dev/null";

if ($res eq "") {
    die "Désolé, mais j'ai besoin d'un nom de fichier sortie.\nÀ bientôt !\n"
}

print "\nJe suis maintenant prêt à faire l'analyse avec les paramètres
suivants :\n
Séquence : $sequence
Fichier génomes : $genomes
Matrice : $nom_matrice
Fichier de sortie : $res
Les informations annexes et les erreurs éventuelles d'opscan seront stockées
dans le fichier $res.errors\n
Appuyez sur la touche entrée pour continuer.\n";

my $suite = <STDIN>;
chomp($suite);

if ($suite ne "")
    {
    die "Pourquoi me contrarier ? Je voulais simplement un appui sur la "
      . "touche entrée...\nÀ bientôt ! :o)";
}

##############################################################################
################################ Opscan ################################
##############################################################################

# On a le nom du génome sur chaque ligne, mais il n'est pas complet par
# rapport à la base de données.

my $nomcomplet;

# La variable recherche va nous servir à chercher le vrai nom du génome
# voulu.
```

```
my $recherche;
print "\n\nChaque point sur la ligne suivante correspond à un "
 ."génome traité.\n\n";

open (FILEREAD, "<$genomes")  or die $!;
while(<FILEREAD>) {
    chomp($_);

# Le nom doit avoir 4 caractères s'il n'y qu'une souche, et cinq si la
# souche est précisée.

    if (length($_) > 5) {
        die "\n Je suis désolé, mais votre fichier de génômes est invalide : "
      . "aucune ligne ne devrait dépasser cinq caractéres.\n À bientôt ! :o)"
    } elsif (length($_)==4) {
        $recherche = "$_\A.*.prt";
    } elsif (length($_)==5) {
        $recherche = "$_.*.prt";
    }

    system "ls $bacteriesd/$recherche > .nomcomplet";
    open(NOM, "<.nomcomplet") or die $!;
    while (<NOM>) {
        chomp($_);
        $nomcomplet=$_;
    }

# C'est maintenant qu'on appelle opscan.

    system "opscan -M $matrice  -r 1.4 -O $sequence $nomcomplet "
     . "> .ResultatTemporaire 2>>$res.errors ";

# Il faut traiter le fichier de sortie d'opscan pour avoir un fichier
# tout propre.

    open(FILEWRITE, ">>$res") or die $!;
    open(READRESULT, "<.ResultatTemporaire") or die $!;
    while(<READRESULT>) {
# On ne veut, pour l'instant que la ligne qui commence par CL

    my $identifiant = substr($_,0,2);
        if ($identifiant eq "CL") {

# On repére les positions des infos qui nous intéressent sur un fichier de
# sortie standard de opscan.

            $seqdedepart=substr($_, 9, 14);
            $genequimatche=substr($_, 34, 14);
            $pourcentage=substr($_,54,5);
            $differencedetaille=substr($_,68,4);
print FILEWRITE "$seqdedepart\t$genequimatche\t$pourcentage\t"
                ."$differencedetaille\n";
        } else {}
    }


# Un petit indicateur d'avancement.
    print ".";
}

close(FILEREAD);
close(FILEWRITE);
system "rm -rf .nomcomplet .ResultatTemporaire";
print "\n\nVoilà, j'ai terminé. À bientôt :o) !\n";
```

2 Programme « Valkyrie »

```
#!/usr/bin/perl

system "clear";

################################################################################
############################## Description #####################################
################################################################################

=head1

Programme "Valkyrie", qui analyse les résultats fournis par deux
appels différents de "Parque"et ne garde que les résultats qui coïcident.
                    © Manu Cornet, 21 juin 2004

Cette nouvelle version accepte aussi les fichiers résultats qui ont un
nombre de lignes différents.
                    © Manu Cornet, 5 juillet 2004
=cut

print
"Bonjour ! Je suis le programme Valkyrie. Je peux comparer deux fichiers de
résultats produit par le programme Parque, les comparer, et ne garder que les
résultats qui coïncident.\n\n";

################################################################################
############################## Noms des fichiers ###############################
################################################################################

# On commence par demander le chemin des deux fichiers à comparer.

print
"J'ai besoin, tout d'abord, que vous me précisiez le chemin des deux fichiers
que je vais devoir comparer. Veuillez m'indiquer le chemin du premier fichier
maintenant, s'il vous plaît.\n\n";

my $fich1 = <STDIN>;
chomp($fich1);

if ($fich1 eq "") {die "Désolé, mais j'ai besoin d'un nom de fichier."};

print "Veuillez m'indiquer à présent le chemin du second fichier.\n\n";

my $fich2 = <STDIN>;
chomp $fich2;
if ($fich2 eq "") {die "Désolé, mais j'ai besoin d'un nom de fichier."};

print
"J'ai également besoin que vous m'indiquiez dans quel fichier je dois inscrire
mon résultat.\n\n";

my $res = <STDIN>;
chomp($res);

if ($res eq "") {die "Désolé, mais j'ai besoin d'un nom de fichier."};


################################################################################
######################### Stockage dans deux tableaux ##########################
################################################################################

# tab1 et tab2 contiendront les listes des best fits de chaque fichier
# résultat
my @tab1;
my @tab2;

# On veut aussi la similarité et la différence de longueur : 4 tableaux
# sim1 sim2, long1 long2.
my @sim1;
my @sim2;
my @long1;
my @long2;

# $jeteveux 1, 2 et 3 contiendront les parties de la ligne qu'on veut garder
# (le nom du gêne best fit), la similarité et la diff de longueur
my $jeteveux1;
my $jeteveux2;
my $jeteveux3;

open (FILEREAD, "<$fich1") or die $!;
my $compteur = 0;
while (<FILEREAD>)
    {
    $jeteveux1 = substr($_,15,14);
    $jeteveux2 = substr($_,30,5);
    $jeteveux3 = substr($_,36,4);
    $tab1[$compteur] = $jeteveux1;
    $sim1[$compteur] = $jeteveux2;
    $long1[$compteur] = $jeteveux3;
    $compteur++;
    }
close (FILEREAD);

open (FILEREAD, "<$fich2") or die $!;
my $compteur = 0;
while (<FILEREAD>)
    {
    $jeteveux1 = substr($_,15,14);
    $jeteveux2 = substr($_,30,5);
    $jeteveux3 = substr($_,36,4);
    $tab2[$compteur] = $jeteveux1;
    $sim2[$compteur] = $jeteveux2;
    $long2[$compteur] = $jeteveux3;
    $compteur++;
    }
close (FILEREAD);


################################################################################
############################## Comparaison #####################################
################################################################################

# i et j sont des compteurs qui vont permettre de parcourir les deux
# tableaux

my $i = 0;
my $j = 0;

# test permet de voir si un élémént d'un tableau n'a pas trouvé chaussure
# á son pied. On fait un tableau test pour 1 et 2

my @test1;
my @test2;

# card1 et card2 sont le nombre d'éléments de chaque tableau

my $card1 = @tab1;
my $card2 = @tab2;

# On voudra savoir à la fin combien on a des gênes identiques

my $combienok;

# On initialise les tableaux de test à zéro.

for ($i = 0 ; $i < $card1 ; $i++)
    {
    $test1[$i] = 0;
    }

for ($i = 0 ; $i < $card2 ; $i++)
    {
    $test2[$i] = 0;
    }

open (FILEWRITE, ">$res") or die $!;

print FILEWRITE "<Test>\t<Gène>\t\t<1>\t<2>\t<1>\t<2>\n";

$i = 0; $j = 0; $combienok = 0;

for ($i = 0; $i < $card1; $i++)
    {
    for ($j = 0; $j < $card2; $j++)
        {
        if ($tab1[$i] eq $tab2[$j])
            {
            print FILEWRITE "OK : \t$tab1[$i]\t$sim1[$i]\t$sim2[$i]\t"
        . "$long1[$i]\t$long2[$i]\n";
            $test1[$i] = 1;
            $test2[$j] = 1;
            $combienok++;
            }

        }
    }


$i = 0;

if ($card1 != $card2)
    {
    print FILEWRITE "\nGênes n'ayant pas trouvé chaussure à leur pied :\n"
    ."Pour le premier fichier proposé :\n";
    for ($i = 0;$i < $card1; $i++)
        {
        if ($test1[i] == 0)
            {
            print FILEWRITE "1 only : \t$tab1[$i]\t$sim1[$i]\t$long1[$i]\n";
            }
        }



    $i = 0;

    print FILEWRITE "\nPour le second fichier proposé :\n";
    for ($i = 0;$i < $card2; $i++)
        {
        if ($test2[i] == 0)
            {
            print FILEWRITE "2 only : \t$tab2[$i]\t$sim2[$i]\t$long2[$i]\n";
            }
        }

    }
close FILEWRITE;

print "Voilà, j'ai terminé !\n";

print "J'ai trouvé ---$combienok--- gênes identiques !\n";

print "La structure du fichier résultat est la suivante :\n
<Ok ou Différent>\t<BestMatch1>\n\n puis\n
<Similarité avec 1>\t<avec 2>\t<Différence de longueur avec1>\t<avec 2>\n\n";

system "mv $res $res.$combienok";

print "Le fichier de sortie s'appelle $res.$combienok\n";

print "À bientôt :o)";
```

3 Programme « GetEntries »

```
#!/usr/bin/perl

my $bacteriesd = "/home/manu/Stage/Bacteries";


################################################################################
############################## Présentation ####################################
################################################################################

system "clear";

print "Bonjour ! Je suis le programme GenEntrieS. Je sais écrire un
fichier au format fasta qui contient toutes les séquences des gènes
mentionnés dans un fichier de sortie du programme valkyrie. Je ne
prendrai que les lignes qui commencent par Ok. J'ai besoin de la
commande getentry pour fonctionner correctement.";

################################################################################
############################## E & S ###########################################
################################################################################


print"\n\nQuel est le fichier que je dois prendre en entrée ?\n\n";

my $read = <STDIN>;
chomp($read);

print "\n\nQuel nom voulez-vous que je donne au ficher de sortie ?\n\n";

my $write =  <STDIN>;
chomp($write);


################################################################################
############################## Traitement ######################################
################################################################################


open (FILEREAD, "<$read") or die $!;
# open (FILEWRITE, ">$write") or die $!;

print "Je calcule... Chaque point sur la ligne suivante représente un génome
traité.\n\n";


while (<FILEREAD>)
    {
    if ($_ =~ /^OK/)
        {

# On veut d'abord le nom du génome : quatre lettres, que l'on met en
# minuscules

        my $genome = lc(substr($_, 6, 4));

# Puis on veut le nom de la souche : la cinquième lettre, que l'on
# concatène avec le nom du génôme.

        $genome = $genome.substr($_, 10, 1);

# Puis l'identifiant du gène

        my $identif = substr($_, 6,14);

# On veut le nom complet du fichier genome

        my $recherche = "$genome.*.prt";
        system "ls $bacteriesd/$recherche > .nomcomplet";
        open (NOM, "<.nomcomplet") or die $!;
        while (<NOM>)
            {
            chomp($_);
            $nomcomplet = $_;
            }

# Enfin, on appelle getentry
        my $requete = "getentry -i $identif $nomcomplet >> $write";
        system "$requete\n";
        }
print ".";
    }


close (FILEREAD);
# close (FILEWRITE);


#system "rm -rf .nomcomplet";

print "\nVoilà, j'ai terminé ! À bientôt pour de nouvelles "
. "aventures génomiques !\n";
```

4 Programme « Distance »

```
#!/usr/bin/perl

=head1

Ce programme nécessite un fichier contenant l'ensemble des génomes, en
noms abrégés et à raison d'un par ligne ; il nécessite aussi l'ensemble
des fichiers des matrices distances utiles. Il sert à calculer une matrice
globale des distances concernant tous les gènes cherchés, pour chaque
couple possible.

                    © Manu Cornet, 17 juillet 2004
=cut

my $Gen = "/home/manu/Stage/Genomes" ;
my $Gens = "/home/manu/Stage/Genes";
my $NBLIGNES = 15576;

################################################################################
################## Chargement des infos nécessaires ############################
################################################################################

# On a besoin du fichier contenant les génomes

print "Bonjour ! Je dois être appelé à partir du répertoire Etudes.
Je me sers du fichier $Gen\n";

my @genomes ;
my $i = 0;

open (FILEREAD, "<$Gen") or die $!;
while (<FILEREAD>) {
    chomp($_);
    $genomes[$i] = $_;
    $i++ ;
}

my $cardgenomes = $i;

# On a aussi besoin du fichier contenant les gènes
print "\nJe me sers également du fichier $Gens\n";

my @genes ;
$i = 0;

open (FILEREAD, "<$Gens") or die $!;

while (<FILEREAD>) {
    chomp($_);
    $genes[$i] = $_;
    $i++ ;
}

my $cardgenes = $i;

################################################################################
################################################################################
######################### Construction de la matrice ###########################
################################################################################
################################################################################

my $matrice;

################################################################################
########### Génération des couples de génomes - 1ère colonne ###################
################################################################################

my $k;

$i = 0; $j = 0; $k = 1 ;

# i, j, k sont des compteurs. On initialise k à 1 pour laisser vide la case
# [0][0] de "matrice".

for ($i = 0 ; $i < $cardgenomes ; $i++) {
    for ($j = $i ; $j < $cardgenomes  ; $j++) {
        $matrice[$k]->[0] = "$genomes[$i]-$genomes[$j]";
        $k++;
    }
}

################################################################################
############## Génération des noms des gènes - 1ère ligne ######################
################################################################################

for (0..$cardgenes) {
    $matrice[0]->[$_ + 1] = $genes[$_];
}

################################################################################
############################## La grande Boucle ! ##############################
################################################################################

# Le principe est le suivant : soit un fichier de matrice distance donné.
# Comme les génomes ne sont pas ordonnés suivant un ordre particulier, il
# faut d'abord savoir la position de chacun ; ceci sera stocké dans un
# tableau @pos. Il faut ensuite ouvrir le fichier et parcourir chaque ligne
# pour placer les valeurs là où il faut dans la matrice finale. Chaque
# ligne des fichiers dist contient sept valeurs ; il faut donc compter ce
# qu'il faut pour tenir compte de ces lignes, savoir quand une "vraie"
# nouvelle ligne de la matrice commence et quel genome elle concerne : ceci
# sera stocké dans $genomedebutligne. Enfin, je n'ai pas trouvé de
# meilleure solution que de faire, à chaque fois, une recherche dans
# l'en-tête des lignes de la matrice, à la recherche du couple qui nous
# intéresse. On n'a, en effet aucun moyen de savoir a priori où se situe le
# couple cherché parmi les milliers de lignes de la matrice. L'exécution
# est donc assez longue.

 $i = 0; $j = 0; $k = 0; $iplus = 0;

# k sert au début de la boucle pour attribuer les valeurs de @pos (voir
# ci-desous

my $compt; my $posligne; my $lignes;

# Compt va servir à parcourir la grande matrice à la recherche de la ligne
# à modifier

# Posligne est un compteur de la position dans la ligne de la matrice (de 0
# à 6) et lignes permet de se souvenir combien de lignes on a parcourues


my @pos;

# Pos est un tableau, propre à chaque matrice, qui se souvient de la position
# de chaque génome dans cette matrice

my $genomedebutligne;

for ($i = 0 ; $i < $cardgenes ; $i++) {
print "\nJe m'attaque au gène $genes[$i]\n";
    $k = 0;
    open (DIST,"<./$genes[$i]/$genes[$i].aln.phy.dist") or die
    "Pas de fichier ./$genes[$i]/$genes[$i].aln.phy.dist\n";
    while (<DIST>) {
        if (substr($_,0,2) ne "  ") {
            $pos[$k] = lc(substr($_,0,4)).substr($_,4,1);

            # On met le nom du genome en minuscules, mais
            # on garde la souche en maj.

            $k++;
        }
    }
    open (DIST,"<./$genes[$i]/$genes[$i].aln.phy.dist") or die $!;
    while (<DIST>) {
    if (substr($_,4,1) ne "\n") {
        if (substr($_,0,2) ne "  ") {
            $genomedebutligne = lc(substr($_,0,4)).substr($_,4,1);
            print "J'analyse la ligne $genomedebutligne\n";
            $lignes = 0;
        }
        $posligne = 0;
        until($posligne == 7) {
            $compt = 1;
            my $fini =0;
            my $somme = $posligne + $lignes;
            if ($pos[$somme] ne "") {
                until($fini == 1 or $compt > $NBLIGNES + 1) {

                    if ($matrice[$compt]->[0]
                    eq "$genomedebutligne-$pos[$somme]"
                    or $matrice[$compt]->[0]
                    eq "$pos[$somme]-$genomedebutligne") {

                    # On ne sait pas dans quel ordre
                    # le couple figure dans la matrice
                        my $resu = 12 + 9 * $posligne;
                        $iplus = $i + 1;
                        $matrice[$compt]->[$iplus]
                        = substr($_,$resu,7);

                        $fini = 1;
                    }
                    $compt++;
                }
            }
            $posligne++;
        }
        $lignes = $lignes + 7;
    }}
}


################################################################################
############################## Affichage de la matrice #########################
################################################################################

open (RESULTAT, ">Resultat");

$i = 0; $j = 0;

for ($i = 0 ; $i < $NBLIGNES ; $i ++) {
    for ($j = 0 ; $j < 25 ; $j++) {
        print RESULTAT "$matrice[$i]->[$j]\t";
    }
    print RESULTAT "\n";
}
```

5 Programme « Build »

```
#!/usr/bin/perl

=head1

Ce script permet de constuire de beaux tableaux LaTeX (dont un
"booléen") à partir d'un fichier "Data" au format texte, très simple.
                                        © Manu, 24 juillet 2004
=cut


# Les fichiers qu'on va traiter
$fich = $ARGV[0] or die "Usage :    Recup nom_du_tableau_latex";
$latex = "Tableau.tex";
print "Je prends les données dans le fichier $fich\n\n";

open (DATA, $fich) or die "Pas réussi à ouvrir $fich : $!\n";
open (LATEX, ">$latex") or die "Impossible d'écrire dans $new : $!\n";

########################                          ######################
########################   Lecture des données    ######################
########################                          ######################

$i=0;

while (<DATA>) {
    chomp;
    @ligne = split (";", $_);

    $j=0;
    foreach $egalite (@ligne) {
        if ($j == 0) {$data[$i][$j] = $egalite;} else {
            @tempo = split("=",$egalite);
            $gene = $tempo[1];
            $data[$i][$j] = $gene;
        }
        $j++;
    }
    $i++;
}

print LATEX "\\input{ManuTableau.sty}\n";
print LATEX
"\\usepackage{color}
\\newcommand{\\couleur}{%
\\colorbox[cmyk]{0.18,0.18,0.18,0.}{\\makebox[5.5ex][l]{\\rule{0pt}{1ex}}}
}\n\n";

print LATEX "\\begin{document}\n\\tiny\n";
print LATEX
"\\setlength{\\tabcolsep}{0.2pt}
% Pour qu'il n'y ait pas d'espaces sur les côtés.

\\setlength{\\fboxsep}{0.8ex}
% Pour qu'il n'y ait pas d'espace en haut et en bas entre la
% couleur et le bord de la case

\\setlength{\\fboxrule}{0pt}
% Pour dire que le trait de la \\fbox a une épaisseur nulle.
";

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%                      %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%   Tableau booléen    %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%                      %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

## Il s'agit d'un tableau avec uniquement des cercles : un cercle si le
## gène est présent, rien sinon

# Début du tableau

print LATEX
"\\noindent\n\\centers[0]{
\\begin{tabular}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c||c|c|c|c|c|}";
print LATEX "\n\\hline\n";

# Les noms de colonnes ont plein de \! pour gagner de l'espace horizontal
print LATEX
"&recA&recB&recC&recD&recF&recG&recJ&recN&recO&recQ
&recR&recU&ruvA&ruvB&ruvC&sbcB&sbcC&sbcD&addA\\!&addB\\!&BCD&FOR&rAB&aAB&\\\\
\\hline\n";

# Boucle

for ($i = 0 ; $i < 117 ; $i++) {

    # On a d'abord besoin de l'identifiant du génome de cette ligne
    print LATEX "\\,$data[$i][0]\\, & ";

    for ($j = 1 ; $j < 24 ; $j++) {
        # On ne veut plus recE, recT, rusA
        if ($j == 5 || $j == 13 || $j == 15) {next;}

        if ($data[$i][$j] =~ /\d{6}/) {
            print LATEX "\\couleur";
        }
        print LATEX "& ";
    }

    # La colonne "recBCD" : si recB, C et D sont présents
    if ($data[$i][2] =~ /\d{6}/
     && $data[$i][3] =~ /\d{6}/
     && $data[$i][4] =~ /\d{6}/) {
        print LATEX "\\couleur";
    }

    print LATEX " & ";

    # La colonne "FOR" : si recF, O et R sont présents
    if ($data[$i][6] =~ /\d{6}/
     && $data[$i][10] =~ /\d{6}/
     && $data[$i][12] =~ /\d{6}/) {
        print LATEX "\\couleur";
    }

    print LATEX " & ";

    # La colonne "ruvAB" : si ruvA et B sont présents
    if ($data[$i][16] =~ /\d{6}/
     && $data[$i][17] =~ /\d{6}/) {
        print LATEX "\\couleur";
    }

    print LATEX " & ";

    # La colonne "addAB" : si ruvA et B sont présents
    if ($data[$i][22] =~ /\d{6}/
     && $data[$i][23] =~ /\d{6}/) {
        print LATEX "\\couleur";
    }

print LATEX " & \\,$data[$i][0]\\,\\\\  \\hline \n";
}

# Fin du tableau

print LATEX "\n\\end{tabular}}\\newpage\n";

print LATEX "\\setlength{\\tabcolsep}{5pt}\n";
# On peut remettre ces longueurs à des valeurs "normales"

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%                      %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%   Premier Tableau    %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%                      %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

# Comme toutes les données ne tiennent pas sur une page, on sépare en deux.

# Début du tableau

print LATEX "\\noindent
\\centers{\n\\begin{tabular}{|c|c|c|c|c|c|c|c|c|c|c|c|c|}";
print LATEX "\n\\hline\n";
print LATEX
"&recA&recB&recC&recD&recF&recG&recJ&recN&recO&recQ(S)\\\\ \\hline\n";

# Boucle

for ($i = 0 ; $i < 117 ; $i++) {
    for ($j = 0 ; $j < 12 ; $j++) {
        # On ne veut plus recE
        if ($j == 5) {next;}

        print LATEX "$data[$i][$j] ";
        if ($j != 11) {print LATEX "& ";}
    }
print LATEX "\\\\  \\hline \n";
}

# Fin du tableau

print LATEX "\n\\end{tabular}}\\newpage";

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%                      %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%   Second tableau     %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%                      %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

# Début du tableau

print LATEX
"\\noindent\n\\centers{\n\\begin{tabular}{|c|c|c|c|c|c|c|c|c|c|c|c|c|}";
print LATEX "\n\\hline\n";
print LATEX
"&recR&recU&ruvA&ruvB&ruvC&sbcB&sbcC&sbcD&addA&addB\\\\ \\hline\n";

# Boucle

for ($i = 0 ; $i < 117 ; $i++) {
    print LATEX "$data[$i][0] & ";
    for ($j = 12 ; $j < 24 ; $j++) {
        # On ne veut plus recT et rusA
        if ($j == 13 || $j == 15) {next;}

        print LATEX "$data[$i][$j] ";
        if ($j != 23) {print LATEX "& ";}
    }
    print LATEX "\\\\  \\hline \n";
}

# Fin du tableau

print LATEX "\n\\end{tabular}}\n\\end{document}";


# On ferme tout et on dit au revoir
close DATA; close LATEX;
print "\nVoilou, j'ai fini !\n";
```

Bibliogaphie

- Simon Cozens, *Beginning Perl*, 2001, disponible à l'adresse suivante :

  `http ://learn.perl.org/library/beginning_perl`

- Sébastien Desreux, *Perl pour l'impatient*, Éditions H&K, Paris, 2004.
- Richard Durbin, Sean R. Eddy, Anders Krogh et Graeme Mitchison, *Biological sequence analysis – Probabilistic models of proteins and nucleic acids*, Cambridge University Press, Cambridge, 1998.
- Cynthia Gibas et Peter Jambeck, *Introduction à la bioinformatique*, traduction d'Hélène Dauchel, Isabelle Milazzo et Laurent Mouchard, Éd. O'Reilly, Paris, 2002.
- Des Higgins et Willie Taylor, *Bioinformatics – Sequence, structure and databanks*, Oxford University Press, New York, 2000.
- Bradley M. Kuhn, *Picking Up Perl*, 2001, disponible à l'adresse suivante :

  `http ://www.ebb.org/PickingUpPerl`

- Joseph W. Lengeler, Gerhart Drews et Hans G. Schlegel, *Biology of the Prokaryotes*, Georg Thieme Verlag, Suttgart, 1999.
- Masatoshi Nei et Sudhir Kumar, *Molecular Evolution and Phylogenetics*, Oxford University Press, New York, 2000.
- Kenro Oshima, Shigeyuki Kakizawa, Hisashi Nishigawa, Hee-Young Jung, Wei Wei, Shiho Suzuki, Ryo Arashida, Daisuke Nakata, Shin-ichi Miyata, Masashi Ugaki, Shigetou Namba, *Reductive evolution suggested from the complete genome of a plant-pathogenic phytoplasma*, Nature Genetics, Vol. 36, n° 1, 2004.
- Ahmad Oulmouden, Didier Delourme, Abderrahman Maftah, Jean-Michel Petit, Raymond Julien, *Génétique*, Éd. Dunod, Paris, 1999.
- Larry Wall et Randal L. Schwartz, *Programming Perl*, O'Reilly and associates, Usa, 1991.