

Rapport de stage M1 au ZBIT de Tübingen : La représentation des réseaux phylogénétiques

Philippe Gambette

15 septembre 2005

1 Déroulement du stage

En fin de stage de licence au LIRMM, j'avais demandé à Olivier Gascuel qu'il m'indique des équipes allemandes de recherche en bioinformatique (je tenais à raviver quelques souvenirs de cette langue étudiée de moins en moins depuis le lycée...). J'ai donc été agréablement surpris quand je me suis aperçu que parmi les groupes conseillés se trouvait celui du professeur Daniel Huson, à l'origine du logiciel SplitsTree¹ [HB05]. J'avais entendu parler de ce programme lors d'une conférence sur la variabilité du VIH et les conséquences sur son traitement donnée par Simon Wain-Hobson à l'ENS Cachan en 2004. Il y citait SplitsTree comme une innovation pour représenter non pas de simples arbres phylogénétiques² comme c'est le cas habituellement, mais des réseaux phylogénétiques qui pouvaient représenter les recombinaisons.

Un des sujets de travail proposés par Daniel Huson se rapportait justement à la représentation de réseaux phylogénétiques dans SplitsTree. Je l'ai donc choisi avec enthousiasme, et après la découverte du thème avec quelques articles [DH04, HDKS04, HKLS05], et quelques débuts d'idées, le codage a très vite commencé, en Java. Ceci m'a permis de participer à la modification de ce gros programme, de découvrir CVS et l'environnement de programmation IntelliJ, nettement plus performant que le NetBeans que j'utilisais auparavant pour Java. La programmation s'est déroulée à vitesse variable, notamment pour le premier sujet (la représentation des *splitsgraphs*, présenté en section 3), qui nécessitait une approche graphique, et un douloureux débogage pour corriger les erreurs liées aux angles ou aux points confondus. En juin, Tobias Klöpper, un doctorant de l'équipe, m'a proposé de travailler sur un second sujet, lié à un autre type de réseaux phylogénétiques, les *réseaux réticulés*, ce qui sera détaillé en section 4.

Ce stage a aussi été l'occasion d'élargir mes connaissances en bioinformatique. J'ai suivi le cours magistral en anglais d'*Algorithmique pour la bioinformatique*³ de Daniel Huson, qui a détaillé et prolongé le cours de bioinformatique suivi à l'ENS en première année. Les présentations occasionnelles d'étudiants ou

¹SplitsTree4 webpage : <http://www-ab.informatik.uni-tuebingen.de/software/jsplits/welcome.html>

²Arbre qui montre les relations de parentés entre les espèces ou d'autres entités supposées avoir un ancêtre commun, voir figure 1.

³Algorithms in Bioinformatics II : <http://www-ab.informatik.uni-tuebingen.de/teaching/ss05/abi2/welcome.html>

de thésards m'ont aussi fait découvrir de nouveaux thèmes en bioinformatique, comme les microARN. Daniel Huson m'a aussi invité à l'accompagner avec un thésard et un étudiant au cycle de conférences *Mathematics of Evolution and Phylogeny*⁴ à Paris qui a été riche en découvertes. J'ai intégré ces nouvelles connaissances dans mes notes de bioinformatique [Gam05] afin d'en garder une trace. J'ai d'autre part regroupé toutes les informations sur ce stage sur la page <http://philippe.gambette.free.fr/Tuebingen/>.

2 Les réseaux phylogénétiques

Pour représenter l'évolution des espèces, ou des gènes, le modèle classique est l'*arbre phylogénétique*, comme celui présenté en figure 1⁵. Mais il s'avère insuffisant pour représenter les *transferts horizontaux de gènes*, les *recombinaisons* ou les *hybridations*.

Les *transferts horizontaux de gènes* correspondent au passage de matériel génétique d'une espèce à une autre, par l'intermédiaire d'un virus par exemple. Ils sont aussi très fréquents entre les bactéries, qui possèdent des systèmes d'acquisition de gènes, les *intégrons*⁶, qui leur permettent de se transmettre des gènes de résistance aux antibiotiques.

Les *recombinaisons* correspondent à un échange de matériel génétique entre deux portions d'un même génome. Par exemple au cours de la méiose, les *crossings-over* permettent de créer une cellule contenant des chromosomes provenant partiellement de la mère et du père, comme montré sur la figure 2.

Les *hybridations* ont lieu le plus souvent pour les plantes ou les poissons. Suite aux échanges de pollen entre deux espèces différentes par exemple, on peut créer artificiellement des espèces hybrides. Elles peuvent aussi avoir lieu naturellement.

De tels cas de transferts de gènes créent des irrégularités dans les phylogénies : deux gènes d'une même espèce n'auront pas la même provenance, et auront donc un arbre phylogénétique différent... et la phylogénie d'un groupe d'espèces ne sera pas la même que la phylogénie d'un certain gène chez ces espèces. Représenter ces deux phylogénies peut être fait sur un même dessin en utilisant non pas un arbre, mais un réseau (c'est à dire un graphe avec des cycles). Il n'existe pas encore de dénominations précises et universelles pour tous les types de réseaux phylogénétiques, ce champ de recherche étant assez récent. Nous décrirons en section 3 puis 4 deux catégories de *réseaux phylogénétiques* : ce que nous appellerons les *splitsgraphs* d'une part (parfois appelés *split networks* en anglais) et les *réseaux réticulés* d'autre part (*reticulate networks* en anglais).

Le logiciel SplitsTree permet de représenter et d'exploiter les conflits en construisant pour un ensemble de taxons leur réseau phylogénétique. SplitsTree propose aussi d'utiliser le *splitsgraph*, comme simple outil de visualisation de la robustesse d'une phylogénie.

⁴<http://www.lirmm.fr/MEP05/>

⁵Chaque feuille de cet arbre est le plus souvent un *taxon* (un groupe à un niveau quelconque de catégorie dans la classification hiérarchique des êtres vivants), c'est donc le terme que l'on utilisera de façon générique pour désigner les entités représentées par les feuilles de tout arbre phylogénétique. Toutefois, les feuilles d'un arbre phylogénétique peuvent aussi être des gènes paralogues par exemple, c'est à dire des gènes d'une même espèce qui partagent un ancêtre commun.

⁶Pour en savoir plus : <http://www.microbes-edu.org/etudiant/gene4.html>

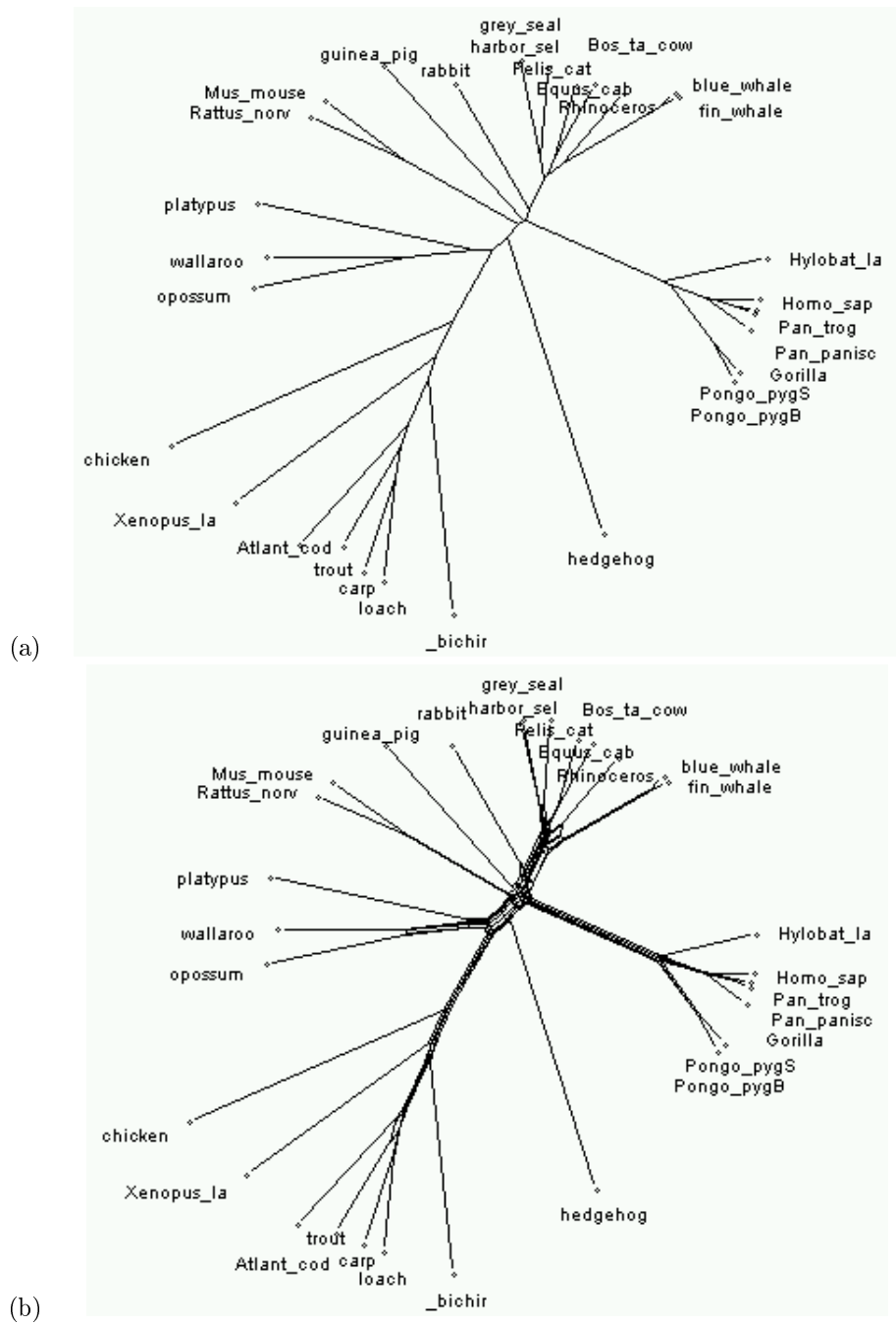


FIG. 1 – (a) Un arbre phylogénétique. (b) Un réseau phylogénétique, ou plus précisément un splitsgraph.

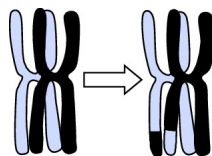


FIG. 2 – Une recombinaison pendant la prophase de la méiose

3 Les splitsgraphs et leur représentation

3.1 Quelques définitions

Soit X un ensemble de taxons. Un *arbre phylogénétique pour X* , ou *X -arbre*, est un arbre $T = (V, E)$ dans lequel tout noeud v est soit une *feuille* de degré 1, soit un noeud *interne* de degré ≥ 3 , associé à un étiquetage des feuilles $\nu : X \rightarrow V$ tel que toute feuille de T a une étiquette unique [SS03]. De plus, on désignera éventuellement le taxon $o \in X$ comme *outgroup*, pour considérer que l'arbre est alors *enraciné* au milieu ρ de l'arête qui mène à $\nu(o)$.

Un *split* (ou, plus précisément, un *X -split*) est une partition de X en deux ensembles non vides A and B , notée : $S = \frac{A}{B} (= \frac{B}{A})$.

Pour un X -arbre T , la suppression de toute arête e produit un graphe avec exactement deux composantes connexes. Ceci définit un split $\sigma_T(e) = \frac{A}{B}$, donné par les deux ensembles d'étiquettes de taxons des deux composantes [BD92]. L'ensemble de splits quand on applique ce processus sur toutes les arêtes de l'arbre est appelé l'*encodage en splits* $\Sigma(T)$ de T .

Deux X -splits distincts $S = \frac{A}{B}$ et $S' = \frac{A'}{B'}$ sont dits *compatibles*, si l'un est un raffinement de l'autre, comme montré en figure 3, c'est à dire qu'on a une des quatre inclusions suivantes : $A \subset A', A \subset B', B \subset A'$ ou $B \subset B'$.

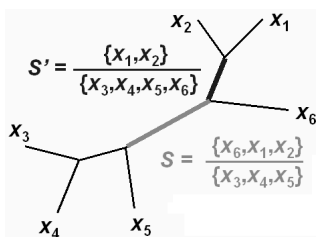


FIG. 3 – Un arbre phylogénétique et deux splits compatibles S et S'

Si S et S' ne sont pas compatibles, on les dit *incompatibles*, et on le note $S \not\parallel S'$. Un ensemble de X -splits Σ est dit *compatible*, si toutes les paires de splits de Σ sont compatibles.

Ce qui suit est un résultat classique de phylogénie [SS03] : un ensemble de X -splits Σ est compatible, si et seulement si il existe un unique X -arbre T avec $\Sigma = \Sigma(T)$. Dans ce cas, on dit que T *représente* Σ . De plus, un ensemble arbitraire de splits Σ , non nécessairement compatible, peut aussi être représenté par un graphe, appelé ici *splitsgraph* (en anglais *splits graph* ou *split network*). $SN(\Sigma)$ est le graphe connexe (V, E) associé à un étiquetage des noeuds $\nu : X \rightarrow V$ et une coloration des arêtes $\sigma : \Sigma \rightarrow E$, dont la propriété essentielle est que supprimer toutes les arêtes colorées par un split donné $S = \frac{A}{B} \in \Sigma$

produira exactement 2 composantes connexes, une étiquetée par les taxons de A , et l'autre par les taxons de B [DH04].

On peut voir cette coloration associée à un split, et donc plusieurs arêtes parallèles, dans la figure 4. Celle-ci représente deux splits incompatibles, ce qui crée donc, avec les deux paires d'arêtes de couleur différente, un parallélogramme, que l'on appelle *boîte* (*box* en anglais).

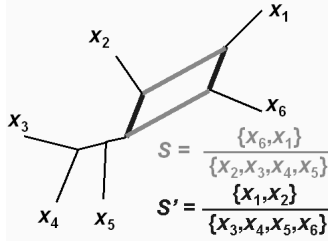


FIG. 4 – Un splitsgraph et deux splits incompatibles S et S' . À S (ainsi qu'à S') sont associées deux arêtes colorées qui séparent le graphe en deux composantes connexes.

Sur l'ensemble de taxons X , on peut définir un *ordre cyclique* (x_1, \dots, x_n) . On dit qu'un X -split S est *circulaire* (selon cet ordre cyclique), s'il existe des nombres p et q avec $1 < p \leq q \leq n$ tels que $S = \frac{\{x_p, \dots, x_q\}}{X \setminus \{x_p, \dots, x_q\}}$. Si tous les splits d'un splitsgraph sont circulaires, alors celui-ci peut-être représenté de manière planaire en utilisant l'algorithme des angles égaux présenté en section 3.3. De plus, l'algorithme NeighborNet [BM02], inspiré du Neighbor-Joining [SN87], permet de créer de tels splitsgraphs à partir de données de distance (par exemple entre des séquences protéiques de plusieurs espèces).

Il est aussi possible de créer des splitsgraphs non planaires, contenant donc des splits non circulaires, qui se traduisent par l'apparition de boîtes de dimension supérieure à 2, à partir des données de distance, en utilisant la méthode de décomposition en splits. Celle-ci commence par la création d'un ordre sur les taxons pour obtenir le plus de splits circulaires. Elle se termine par un algorithme *Convex Hull* [DH04] pour ajouter les splits non circulaires en créant donc des parties non planaires dans le splitsgraph.

3.2 Intérêt des splitsgraphs

Un split partage l'ensemble des taxons en deux. Ainsi, un ensemble de deux splits incompatibles, qui se traduit sur le splitsgraph par l'apparition d'une boîte, permet de visualiser un conflit phylogénétique : comme il y a un cycle, il existe plus d'un chemin possible pour la phylogénie.

Un splitsgraph peut donc servir comme consensus de plusieurs arbres phylogénétiques. Ce problème est connu en bioinformatique comme celui du *super-arbre* (ou encore du *consensus network*) [Bry03]. Le cas le plus simple est celui de plusieurs arbres phylogénétiques portant sur le même ensemble de taxons (par exemple les arbres obtenus par différents algorithmes de reconstruction) que l'on veut regrouper : on peut le faire en choisissant par exemple de ne garder que les splits qui apparaissent dans la majorité des arbres. On est certain avec cette méthode d'obtenir finalement uniquement des splits compatibles, et donc d'obtenir vraiment un arbre. Si on choisit de garder les splits qui apparaissent

pour plus d'1/3 des arbres construits, il est possible au contraire d'obtenir des splits incompatibles et donc un graphe avec des cycles. SplitsTree propose dans ce cas de régler l'épaisseur des arêtes du splitsgraph en fonction du pourcentage de présence du split associé parmi le groupe d'arbres de départ. De plus, grâce à la méthode de *Z-closure* [HDKS04], il est possible d'effectuer le consensus par cette méthode de splits même sur un ensemble d'arbres n'ayant pas le même ensemble de taxons.

Cette méthode de splits pour le superarbre peut-être couplée avec un bootstrap, méthode classique d'évaluation d'un arbre phylogénétique : on génère plusieurs arbres en effectuant de légères modifications aléatoires dans les données de départ (les distances entre les taxons par exemple). Puis on effectue le consensus des splits, pour obtenir par exemple un superarbre avec l'épaisseur des branches qui reflète la confiance qu'on peut leur accorder.

Enfin, le splitsgraph peut éventuellement représenter vraiment la superposition de plusieurs arbres phylogénétiques dans le cas d'un transfert de gènes ou d'une recombinaison. Mais cette distinction de l'événement biologique à l'origine du conflit phylogénétique ne peut être faite directement en regardant le splitsgraph et demande des analyses plus poussées [HKLS05], qui aboutissent par exemple à d'autres types de réseaux, les *réseaux réticulés* de la section 4.

3.3 Représentation d'un splitsgraph

Ainsi le splitsgraph est principalement un outil de visualisation. Encore faut-il arriver à le dessiner de façon lisible. Le problème est donc de dessiner un graphe connaissant les longueurs des arêtes (elles sont en effet fixées par les données biologiques de départ), et sachant que certaines arêtes (celles de même coloration, c'est à dire correspondant à un même split) sont parallèles.

On peut aussi remarquer le résultat suivant, qui permet de se contenter de stocker les angles et longueurs des arêtes d'un graphe pour pouvoir le représenter, et ceci de manière unique en fixant la position d'un noeud :

Lemme 1

Soient G un graphe dirigé et v un de ses noeuds. Soient ω et ω' deux tableaux de coordonnées associées aux noeuds de G . Si $\omega(v) = \omega'(v)$ et si toutes les arêtes ont même angle et longueur avec les deux tableaux de coordonnées, alors $\omega = \omega'$.

L'algorithme basique de dessin de splitsgraph [DH04] est l'algorithme *Equal Angle*, qui est un classique du dessin des arbres phylogénétiques [Fel04]. Il prend comme paramètre un ensemble de X -splits $\Sigma = \{S_1, \dots, S_k\}$ et un ordre cyclique (x_1, x_2, \dots, x_n) de l'ensemble de taxons X , et produit en sortie le splitsgraph représentant tous les splits de Σ circulaires selon l'ordre fixé.

Algorithme 1 (Equal Angle)

Si l'on dirige toutes les arêtes du graphe vers la direction opposée du noeud étiqueté x_1 , alors l'angle de toute arête $e \in E_i$ de S_i est donné par $\alpha_i = \frac{z_p + z_q}{2}$, où $z_p = \frac{p-1}{n} \times 360^\circ$ est l'angle associé au taxon x_p .

Pour visualiser ceci, on peut imaginer l'ensemble des taxons uniformément localisés sur le cercle unité dans l'ordre x_1, \dots, x_n , en commençant par x_1 à $z_1 = 0^\circ$, dans la direction positive. C'est ce qu'on appelle le *cercle des taxons*,

illustré en figure 5. L'angle de S est alors l'angle moyen assigné aux taxons x_p, \dots, x_q . Le splitsgraph obtenu dans ce cas est bien planaire.

Toutefois, pour que le splitsgraph reste planaire, il n'est pas nécessaire d'utiliser des angles égaux pour l'espace entre tous les taxons, tant que l'ordre sur les taxons est conservé.

3.4 L'optimisation Optimized Angle

David Bryant m'a suggéré l'idée de ne pas utiliser des angles égaux mais au contraire de choisir les meilleurs possibles pour améliorer la représentation du splitsgraph. Cette méthode avait l'avantage de ne pas nécessiter de vérifier la planarité du résultat obtenu après modifications, puisqu'elle est assurée si l'on garde le même ordre sur les taxons.

Pour assurer la rapidité de l'algorithme, on a choisi de fixer les angles des splits en fonction des angles des taxons : pour un split circulaire $S = \frac{\{x_p, \dots, x_q\}}{X \setminus \{x_p, \dots, x_q\}}$, on fixe l'angle du split comme l'angle médian des angles des taxons x_p et x_q .

Il faut désormais choisir un critère de score pour déterminer que la représentation d'un splitsgraph était meilleure qu'une autre. Comme on l'a vu, c'est voir les boîtes clairement qui est le plus intéressant avec l'outil de visualisation qu'est le splitsgraph, le critère que nous chercherons donc à optimiser par la suite est la somme des aires des boîtes du splitsgraph.

On effectue cette optimisation par l'algorithme *Optimized Angle* suivant :

Algorithme 2 (Optimized Angle)

Calculer les coordonnées initiales par l'algorithme Equal Angle, stocker les meilleures positions dans un tableau $bestP$, déterminer les angles de tous les splits et l'aire totale couverte, et la stocker dans $bestA$.

Répéter la boucle suivante l fois :

Pour chaque taxon, essayer de le déplacer à mi-distance vers l'un de ses deux voisins. Dans les deux cas, calculer les angles de tous les splits et l'aire totale trouvée.

Si l'aire totale est améliorée, sauver les nouvelles positions, et si elle est strictement supérieure à $bestA$, enregistrer les positions dans $bestP$ et l'aire dans $bestA$.

Sinon, enregistrer les positions avec une probabilité p .

L'algorithme est donc en $O(lkn)$ ⁷. En pratique, $p = 0.8$ et $l = 500$ donnent de bons résultats pour de petits splitsgraphs. Toutefois, l'algorithme n'est pas assez efficace sur de plus grand réseaux. En effet, la condition de préservation de l'ordre circulaire des taxons est trop stricte : il n'y a pas assez d'espace pour bouger les angles des taxons, et donc ceux des splits.

Remarquons aussi que l'algorithme ajuste, en bougeant les taxons, les angles des splits compatibles avec tous les autres, en particulier ceux menant aux feuilles, sans chercher à les optimiser. Pour corriger cela, l'algorithme Optimized Angle sera suivi par une application de l'algorithme Equal Daylight [GH05], adapté par Daniel Huson de l'algorithme similaire pour les arbres phylogénétiques [Fel04], qui permet de régler ce problème en équilibrant, autour de tout noeud, la portion libre d'autres noeuds du graphe, comme suggéré en figure 7.

⁷Rappelons que k est le nombre de splits et n le nombre de taxons

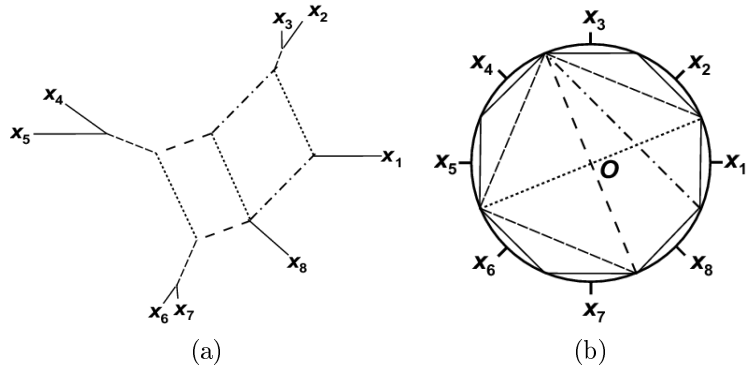


FIG. 5 – (a) Splitsgraph construit avec l’algorithme des angles égaux, où chaque split est représenté par une seule arête si le split est compatible avec tous les autres, un ensemble d’arêtes parallèles sinon. (b) Le même ensemble de splits est représenté par des cordes sur le cercle des taxons, chacune d’entre elles partitionnant les taxons de même que le split. Les angles utilisés en (b) sont orthogonaux à ceux utilisés en (a).

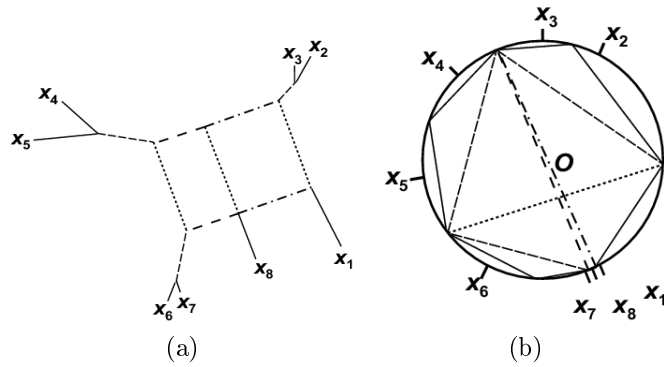


FIG. 6 – (a) Splitsgraph obtenu par l’algorithme Optimized Angle appliqué sur l’exemple de la figure 5. (b) Le cercle des taxons correspondant, pour lequel l’ordre circulaire des taxons, mais pas leur espacement angulaire, est préservé.

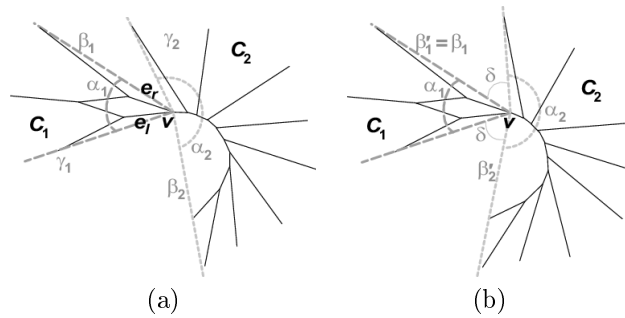


FIG. 7 – (a) Le noeud v sépare le splitsgraph G en deux parties C_1 et C_2 qui couvrent respectivement l’angle α_1 et α_2 . (b) On transforme ce graphe en équilibrant les deux angles entre C_1 et C_2 , qui sont appelés en anglais les *daylight angles* et prennent donc la valeur $\delta = \pi - \frac{\alpha_1 - \alpha_2}{2}$.

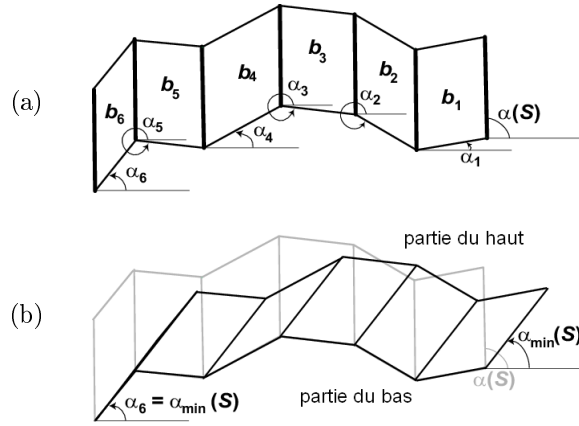


FIG. 8 – (a) Les arêtes correspondant au split S sont marquées en gras, d’autres arêtes représentent les splits S_1, S_2, \dots qui sont incompatibles avec S . L’angle α_i est associé avec le split S_i . (b) L’angle $\alpha(S)$ a pris la valeur critique $\alpha_{min}(S) = \alpha_6$, pour laquelle la boîte la plus à gauche s’écrase.

3.5 L’optimisation Box-Opening

Une alternative à l’algorithme précédent consiste à modifier le dessin du graphe, une fois celui-ci réalisé par l’algorithme Equal Angle. Remarquons qu’il est alors planaire, et que cette étape a lieu avant application de l’algorithme Convex Hull. C’est ce qui est habituellement fait à la main par l’utilisateur de SplitsTree. Comme c’est fait ici de façon automatique, il faudra faire attention aux collisions : les mouvements qui font perdre au splitsgraph son caractère planaire. Celles-ci sont de deux types : locales quand elles concernent uniquement le split dont on essaie de modifier l’angle, globales quand ce sont d’autres arêtes.

3.5.1 Eviter les collisions locales

Une *collision locale* arrive quand une des boîtes b_i est écrasée par le changement d’angle $\alpha(S)$ du split S . Pour éviter une telle collision, on peut noter qu’il y a deux angles critiques $\alpha_{min}(S)$ et $\alpha_{max}(S)$ qui limitent les valeurs possibles pour $\alpha(S)$:

$$\begin{aligned} \alpha_{max}(S) &= \alpha(S) + \min_{\text{boîte } b_i} \{(\alpha_i - \alpha(S) - \pi) \bmod 2\pi\} \\ \alpha_{min}(S) &= \alpha(S) - \min_{\text{boîte } b_i} \{(\alpha(S) - \alpha_i) \bmod 2\pi\}, \end{aligned}$$

où α_i est l’angle original du split S_i ($i = 1, \dots, k$), comme montré sur la figure 8. Tout choix d’angle compris entre α_{min} et α_{max} donnera un dessin du splitsgraph sans collision.

3.5.2 Eviter les collisions globales

On considérera, quand on modifie l’angle $\alpha(S)$ d’un split S , qu’une partie du splitsgraph, celle “du bas”, reste fixe, alors que la partie “du haut” translate, comme montré sur l’image 8. Cette translation de la partie haute peut créer une collision avec la partie basse.

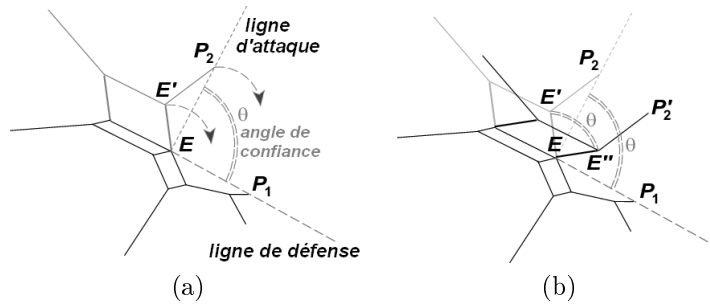


FIG. 9 – Exemple d’une situation critique où on tente d’éviter une collision globale à droite du split : avant (a) et après (b) la modification de l’angle du split.

Nous allons donc restreindre l’intervalle d’angles imposé par les contraintes de collisions locales, pour tenir compte de ces collisions globales. Celles-ci peuvent apparaître “sur la gauche” ou “sur la droite” du split S , indépendamment du fait que l’on accroisse ou décroisse l’angle du split. Ainsi, nous effectuerons des calculs similaires sur 4 points qui sont les *noeuds extrêmes* du split.

On considère donc un des cas, celui de l’arête du split S la plus à droite sur la figure 8(a), qu’on assimile au segment $[E, E']$ de la figure 9. On focalisera l’analyse sur le noeud E , en cherchant des collisions à sa droite quand on décroît l’angle du split.

On va alors déterminer deux points particuliers qui sont des localisations de noeuds : l’attaquant P_1 et le défenseur P_2 . Pour déterminer P_1 , on visite tous les noeuds de la partie du haut du splitsgraph (la partie mobile), et on choisit celui qui minimise l’angle $(\overrightarrow{EP_1}, \overrightarrow{EE'})$. De même, P_2 est obtenu en cherchant la localisation du noeud de la partie du bas (partie fixe) qui maximise l’angle $(\overrightarrow{EP_2}, \overrightarrow{EE'})$, comme montré en figure 9(a).

Une fois ces deux points obtenus, on peut remarquer que si l’on diminue l’angle $\alpha(S)$ de moins de $\theta = (\overrightarrow{EP_1}, \overrightarrow{EP_2})$ alors il n’y aura pas de collision globale en général, c’est pourquoi on l’appelle *angle de confiance* (on a choisi le terme *safe angle* en anglais). Une fois les 4 angles de confiance connus, on vérifie s’ils imposent de réduire l’intervalle permis par les angles critiques $\alpha_{min}(S)$ et $\alpha_{max}(S)$.

Cette méthode d’angle de confiance appelle quelques remarques. Tout d’abord, choisir de faire varier l’angle du split par une valeur inférieure à cet angle de confiance semble peu adaptée à un mouvement de translation. En effet, pour un mouvement de rotation de la partie du haut du splitsgraph ce serait parfait, mais là ce n’est pas le cas.

Et ceci produit deux effets. L’effet positif est qu’en modifiant l’angle $\alpha(S)$, il est possible qu’alors le noeud défenseur, ou l’attaquant, change. Par exemple, dans la figure 9, l’attaquant est P_2 en (a), mais E'' , et pas P_2' , en (b). Ainsi, après optimisation des angles de tous les splits du graphe, une nouvelle boucle d’optimisation reste possible.

L’effet négatif est que même si nous respectons l’angle de confiance lors du changement d’angle du split, il est possible que des collisions apparaissent tout de même. Ceci arrive peu en pratique, et nous l’avons expliqué en exhibant

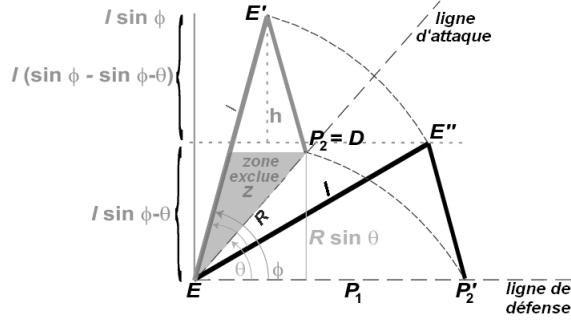


FIG. 10 – Localisation du noeud D tel que retrancher à l'angle du split l'angle de confiance θ placera D exactement sur la ligne de défense. Notons que $h = R \sin \theta$ implique $R \sin \theta = l \sin \phi - l \sin(\phi - \theta) = l(\sin \phi - \sin(\phi - \theta))$.

quatre *zones exclues*, une pour chaque noeud extrême du split, telles que, si elles ne contiennent pas de noeud du splitsgraph nous avons la garantie qu'en suivant les contraintes de l'angle de confiance, nous garderons un graphe planaire.

Le théorème suivant donne la localisation de la zone exclue associée avec le noeud en bas à droite du split S :

Théorème 1 (Zone exclue)

Soit P_1 le défenseur, P_2 l'attaquant, ϕ l'angle entre la ligne de défense (EP_1) et l'arête la plus à droite de S (EE'), $\theta' = -\min(\theta, \alpha(S) - \alpha_{\min}(S))$, et D le point tel que $ED = \frac{l(\sin \phi - \sin \phi - \theta)}{\sin \theta}$ et $(\overrightarrow{ED}, \overrightarrow{EE'}) = \theta$.

La zone d'exclusion Z est le triangle formé par la droite L parallèle à la ligne de défense et contenant D , l'arête EE' et la ligne d'attaque (E, P_2) (voir figure 10).

Si Z est vide, alors aucun noeud n'entrera en collision avec la ligne de défense à droite si l'on retranche à l'angle $\alpha(S)$ l'angle θ' .

Des exemples de ces zones exclues sont donnés dans la figure 11. On y remarque que ces zones sont fines et situées de telle façon qu'on n'y trouve généralement pas de noeud du splitsgraph.

Ces considérations débouchent sur l'algorithme suivant :

Algorithme 3 (Box-Opening)

Faire la boucle suivante l fois :

Pour chaque split $S \in \Sigma$:

- Identifier les angles critiques $\alpha_{\min}(S)$ et $\alpha_{\max}(S)$, pour éviter les collisions locales
- Pour chacun des 4 noeuds extrêmes du split, identifier le défenseur et l'attaquant et calculer l'angle de confiance.
- Si la zone d'exclusion est vide, utiliser les 4 angles de confiance et les contraintes locales pour déterminer l'intervalle I des nouveaux angles possibles pour S . Sinon définir $I = \emptyset$.
- Si I est non vide, alors calculer un nouvel angle dans I qui maximise l'aire totale couverte par les boîtes associées avec le split S .

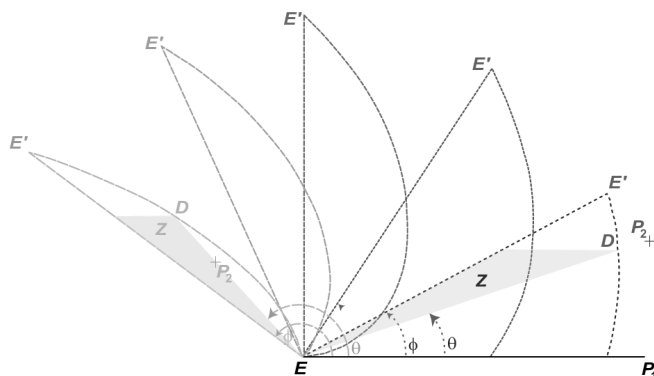


FIG. 11 – Pour différentes valeurs de l’angle ϕ entre l’arête extrême du split et la ligne de défense, les arcs indiquent les frontières de la région qui ne doit pas contenir l’attaquant pour éviter une collision de celui-ci avec la ligne de défense. Puis, connaissant la position de l’attaquant, le triangle de la zone exclue Z peut être dessinée : elle ne doit contenir aucun noeud de la partie du haut si on veut retrancher à l’angle du split la valeur de l’angle de confiance θ en étant sûr de ne pas créer de collision.

On peut remarquer que pour un split S_0 , l’angle $\alpha(S_0)$ qui maximise l’aire totale de l’ensemble de boîtes $\{b_1, \dots, b_t\}$ associées avec l’ensemble de splits $\{S_1, \dots, S_t\}$ qui sont incompatibles avec S_0 , peut être calculé directement. En effet, après un peu de trigonométrie, on arrive à écrire l’aire recherchée, qui est tout d’abord une somme de sinus d’angles différents, comme un cosinus :

$$\text{Area} = A \sin \alpha(S_0) + B \cos \alpha(S_0) = C \cos(\alpha(S_0) - D),$$

avec : $A = w_0 \sum_{S_i \parallel S_0} w_i \cos \beta_i$, $B = -w_0 \sum_{S_i \parallel S_0} w_i \sin \beta_i$, $C = \sqrt{A^2 + B^2}$ et $\tan D = \frac{B}{A}$, où w_i est le poids du split S_i , et $\beta_i = \alpha(S_i)$, si $\sin(\alpha(S_0) - \alpha(S_i)) > 0$, et sinon $\beta_i = \alpha(S_i) + \pi$. Il suffit alors de maximiser $\cos(\alpha(S_0) - D)$ sur l’intervalle I .

La complexité d’une itération de l’algorithme est alors en $O(\text{nombre de splits} \times \text{nombre de noeuds})$, c’est à dire $O(nk + k^3)$ [DH04]. On effectue une amélioration locale (de l’aire de chaque boîte, en considérant les splits à tour de rôle) pour arriver à une amélioration globale de l’aire des boîtes du graphe, et il arrive que l’aire totale ne soit pas améliorée, voire diminuée légèrement, à certaines itérations. Ces cas apparaissent le plus souvent quand le dessin du splitsgraph a déjà été beaucoup amélioré, et ne sont pas visibles. L’algorithme est donc efficace et rapide en pratique, comme illustré sur la figure 12.

4 La représentation de réseaux réticulés

On a vu que les splitsgraphs étaient principalement un outil de visualisation dont on ne pouvait tirer directement des interprétations biologiques. Au contraire, les réseaux réticulés permettent de représenter le phénomène biologique de recombinaison.

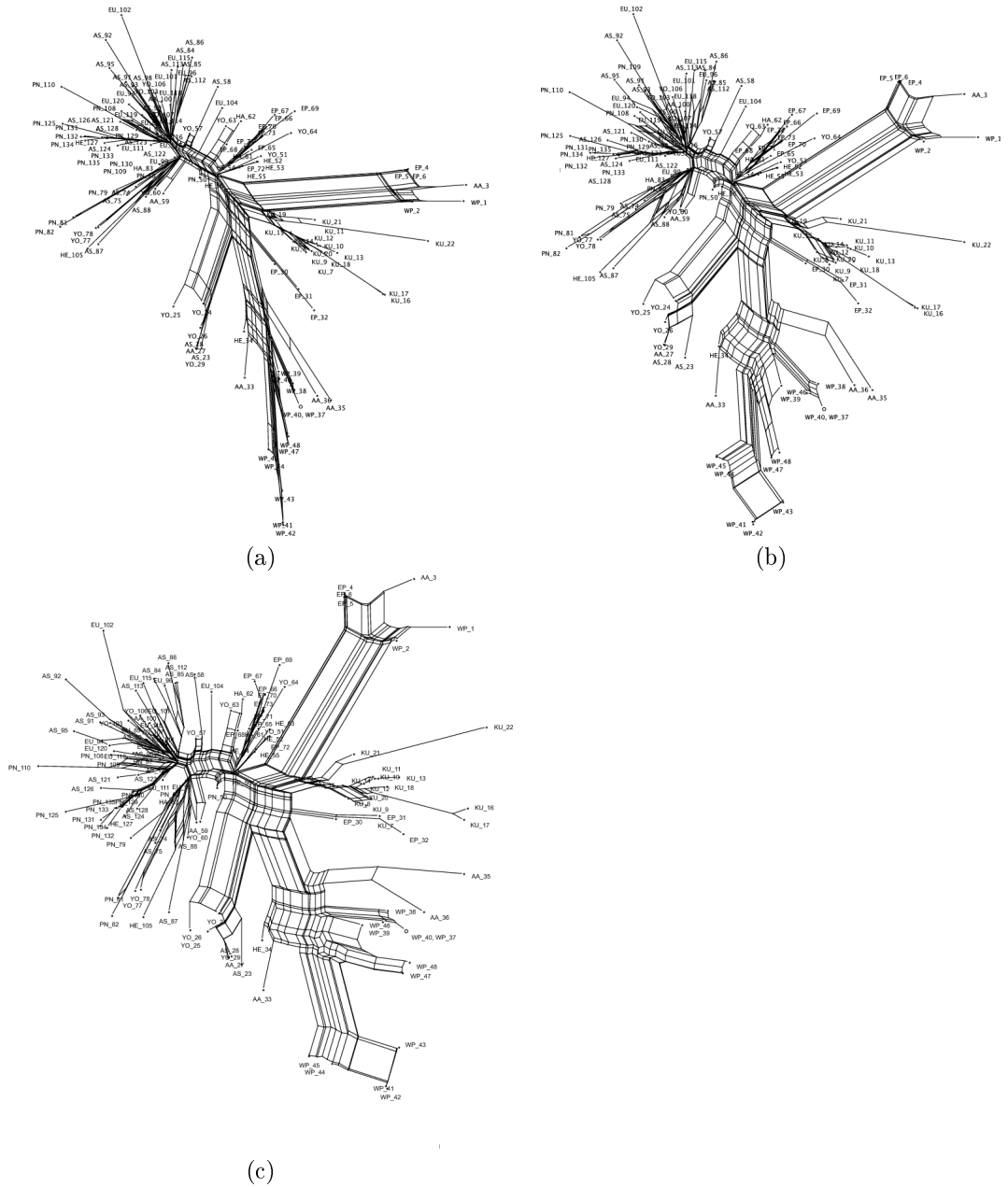


FIG. 12 – Résultats obtenus avec l’algorithme Box-Opening. (a) Splitsgraph original. (b) Après $l = 3$ itérations de la boucle d’optimisation (approximativement 40 secondes sur un Pentium 4 2.6GHz). (c) Après 30 itérations de la boucle d’optimisation, et une de l’algorithme Equal Daylight pour améliorer le dessin des arêtes menant aux feuilles (approx. 7 minutes).

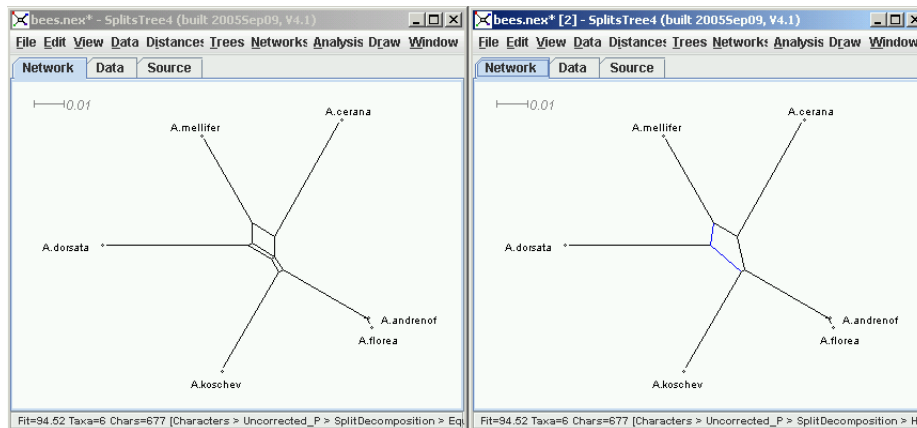


FIG. 13 – Un splitsgraph et le réseau réticulé qui en est déduit dans SplitsTree4.

SplitsTree propose d'utiliser la structure de splitsgraph pour en déduire un réseau réticulé, comme montré en figure 13, et détaillé en [HKLS05].

Cette approche ne propose aucune optimisation pour rapprocher les arêtes que l'on s'apprête à joindre en une *réticulation*, puisque l'on utilise simplement le splitsgraph comme base du dessin. Ainsi, nous avons commencé à travailler sur une méthode alternative qui consisterait à partir d'un arbre phylogénétique de base (*backbone tree* en anglais) pour y ajouter ensuite les réticulations, et ce en changeant l'ordre des taxons afin de minimiser la distance pour chaque paire d'arêtes à rejoindre. On se restreint aux réticulations qui s'appuient sur des branches existantes de l'arbre phylogénétique :

On pourrait utiliser pour ce faire les algorithmes existants pour le problème de *sériation de dendrogramme*, qui consiste à réordonner les taxons d'un dendrogramme ou arbre phylogénétique afin de les regrouper en clusters (s'ils partagent une caractéristique commune, par exemple plusieurs sous-espèces d'une même espèce), en gardant évidemment la structure planaire de l'arbre. Pour l'appliquer à notre problème, il suffit, pour chaque paire d'arêtes à joindre, de regrouper dans un même cluster les taxons qui se trouvent "en dessous" de ces arêtes, puis appliquer un l'algorithme de sériation de dendrogramme avec un tel clustering.

Nous cherchons donc à faire mieux, en utilisant les spécificités de notre problème : nous n'avons pas besoin de rassembler des clusters contenant plusieurs taxons, mais seulement des paires d'arêtes.

4.1 Le graphe dual

Soit un arbre phylogénétique T , avec son ensemble de taxons ordonnés $X = \{x_1, \dots, x_n\}$. Soit $R = (e_i, f_i)$, $1 \leq i \leq r$ l'ensemble des r réticulations à ajouter à l'arbre phylogénétique, les e_i et f_i étant des arêtes de cet arbre qui seront jointes pour former ces réticulations.

Notre méthode ne sera pas fondée sur le dessin de l'arbre, mais seulement sur sa topologie, donc on ne peut compter directement le nombre d'intersections créées par chaque réticulation pour tenter de le minimiser, mais on peut choisir une distance entre les deux arêtes d'une réticulation qui le reflètera : le nombre

de taxons qui “sépare” ces deux arêtes.

Plus formellement, on peut noter $\{x_p, \dots, x_q\}$, avec $p < q$ l’ensemble des taxons du sous-arbre de T issu de e_i , et $\{x_{p'}, \dots, x_{q'}\}$, avec $p' < q'$ celui du sous-arbre issu de f_i . On suppose que $q < p'$, l’autre cas étant symétrique. On a alors la distance

$$d(e_i, f_i) = \min(\delta(\text{prec}(p'), q), \delta(\text{prec}(p), q')),$$

où δ est la plus petite distance entre deux entiers modulo n , et $\text{prec}(p) = 1 + (p - 2 \bmod n)$.

On définit alors la *distance totale de réticulation* comme : $D = \sum_{1 \leq i \leq r} d(e_i, f_i)$

Pour calculer cette distance facilement, spécialement pour les arêtes qui ne mènent pas à une feuille, on représente l’arbre T avec une structure utile : son graphe dual.

Pour construire le graphe dual G^* d’un arbre phylogénétique T , comme celui de la figure 14(a), on commence par joindre toutes les feuilles de T en un noeud unique, comme montré en figure 14(b). On obtient alors un graphe G avec plusieurs faces, l’extérieur du graphe étant aussi considéré comme une face. Dans chaque face, on place un noeud P_i , et pour toute arête e_k de G qui sépare P_k et P_l , on ajoute à G^* une arête qui l’intersecte $e_i^* = (P_k, P_l)$. Par exemple, pour toute arête menant à une feuille x_k , son arête duale sera $(P_{\text{prec}(k)}, P_k)$. On peut de plus choisir de placer les noeuds de G^* de façon régulière sur un cercle pour obtenir le cercle des taxons défini en section 3.3. Ceci fournit le graphe dual planaire G^* illustré en figure 14(d).

Calculer la distance $d(e_i, f_i)$ peut alors être fait en temps constant en utilisant G^* , puisque les noeuds connectés par une arête e_i^* dans G^* indiquent la position du noeud le plus à gauche, et le plus à droite dans le sous-arbre de T issu de e_i :

$$\begin{aligned} d(e_i, f_i) = d^*(e_i^*, f_i^*) &= d^*((P_{\text{prec}(p)}, P_q), (P_{\text{prec}(p')}, P_{q'})) \\ &= \min(\delta(\text{prec}(p'), q), \delta(\text{prec}(p), q')). \end{aligned}$$

En plus de fournir un moyen de calcul aisé des distances, la structure de graphe dual est aussi un bon moyen de visualisation pour le problème, étant donné qu’il est plus simple de voir une distance entre deux arêtes qu’un nombre de taxons intercalés entre les deux dans un arbre, en particulier quand les arêtes sont éloignées des feuilles.

4.2 L’heuristique des images miroir

L’heuristique des images miroir des arêtes permet de minimiser la distance totale de réticulation en changeant l’ordre des taxons, et en gardant la planarité de l’arbre. En effet, si nous considérons toute permutation de l’ordre des taxons, alors certains de ces ordres ne fournissent pas un arbre planaire. On ne considère ici que des opérations de miroir sur les $n - 3$ arêtes (et même moins, comme on le verra plus loin), ce qui donne 2^{n-3} configurations possibles, au lieu de considérer les $n!$ permutations possibles sur n taxons.

L’opération clé est donc le miroir d’une arête e_i qui consiste à inverser le fils gauche et le fils droit de tous les noeuds du sous-arbre de T issu de e_i (dont ceux de e_i). Sur le graphe dual, cela revient à détacher les arêtes adjacentes à

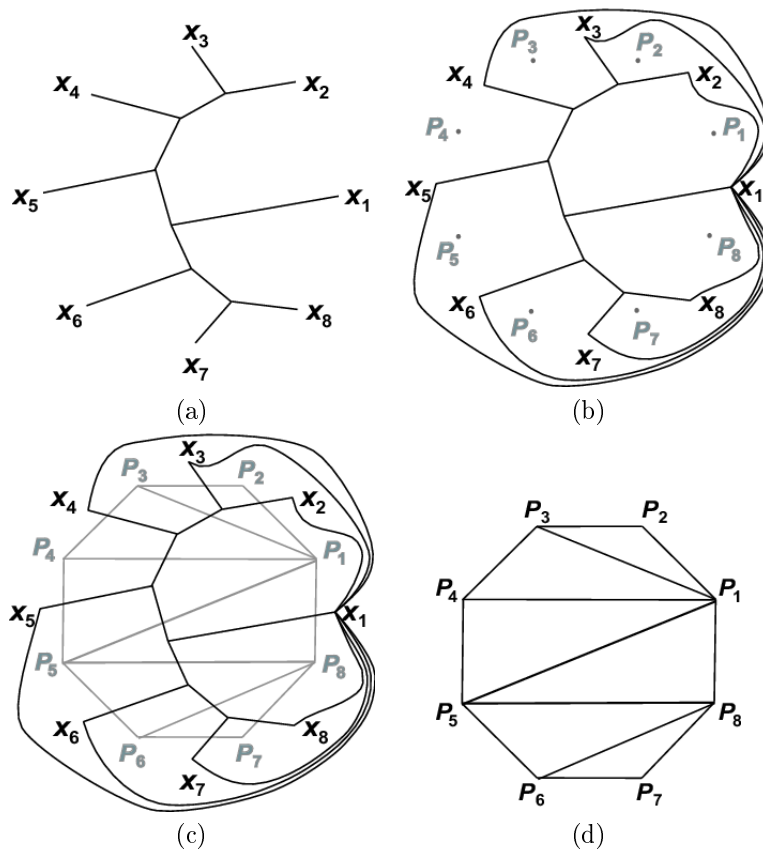


FIG. 14 – Construction du graphe dual d'un arbre T . (a) Joindre les feuilles de T . (b) Placer les noeuds du dual dans les faces (c) Dessiner les arêtes duales (d) Le graphe dual G^* .

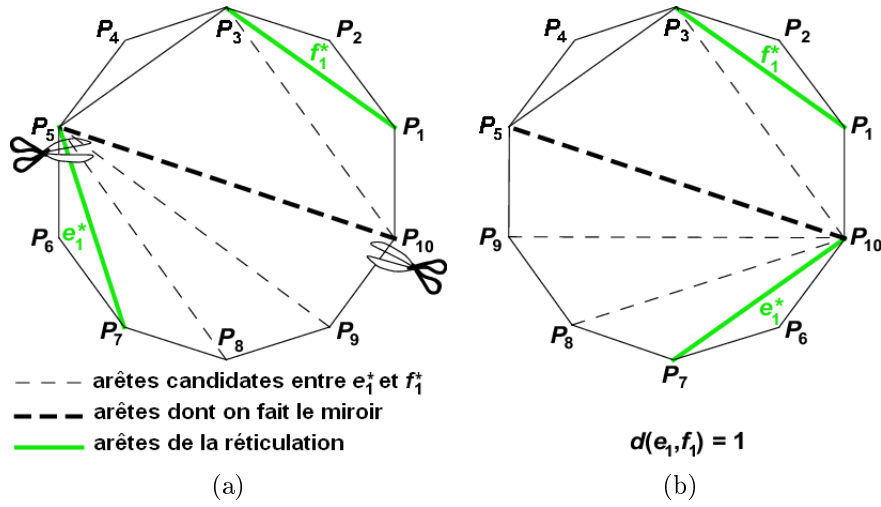


FIG. 15 – Miroir d’une arête. (a) Détacher les arêtes entre P_5 et P_{10} . (b) Les réattacher respectivement à P_{10} et P_5 , et faire le miroir de l’ordre P_6, P_7, P_8, P_9 .

$P_{\text{prec}(p)}$ et P_q , qui ont comme noeud opposé P_k où $p \leq k < q$, et les réattacher respectivement à P_q et $P_{\text{prec}(p)}$, puis effectuer l’image miroir de l’ordre des taxons entre $P_{\text{prec}(p)}$ et P_q . Cette opération de détachement/réattachement sous e_i^* est illustrée en figure 15. On l’appelle le *miroir de l’arête* e_i^* .

On remarque qu’une opération miroir de e^* change la distance $d(e_i^*, f_i^*)$ si et seulement si e^* est située entre e_i^* et f_i^* (e est sur le chemin entre e_i et f_i dans T). Ainsi, on va commencer par identifier l’ensemble C^* de ces *arêtes candidates* qui sont susceptibles de changer la distance totale quand on en effectue le miroir.

Une opération plus intéressante que le miroir d’une arête, pour améliorer la distance totale, serait la *rotation d’un sous-arbre gênant* qui serait placé entre deux arêtes d’une réticulation. On peut le faire avec deux miroirs d’arêtes. Pour bouger le sous-arbre issu de e_i , soient e_j et e_k les deux arêtes attachées à e_i et qui ne sont pas dans ce sous-arbre. Il faut alors effectuer le miroir de e_j^* et e_k^* .

On va donc utiliser ces deux opérations pour améliorer la distance totale de réticulation pour tenter d’en trouver un minimum global. Voici un algorithme avec choix aléatoire des sous-arbres dont on effectue la rotation ou dont on effectue le miroir, et avec un recuit simulé pour décider si l’on garde le réarrangement, seulement dans le cas de la rotation de sous-arbre :

Algorithme 4 (Algorithme des images miroir)

Soit un arbre T et un ordre sur ses taxons, construire son graphe dual G^* :

- Créer un noeud pour chaque taxon.
- Pour chaque arête, identifier la partition de l’ensemble des taxons induite par l’arête : $X = \{x_p, \dots, x_q\} \cap \{x_{\text{prec}(q)}\}$

Poser $C^* = \emptyset$. Pour toute paire d’arêtes $(e_i, f_i) \in R$, pour toute arête e^* de G^* , si e^* se situe entre e_i^* et f_i^* , alors mettre e^* dans C^* .

Calculer la distance de réticulation totale, et le stocker dans D .

Pour i de 1 à l , répéter :

- Régler la température du recuit simulé θ à $1/i$.

- Choisir au hasard une arête candidate $e \in C^*$. En faire le miroir, et calculer la distance de réticulation totale $TempD$. Si elle est supérieure strictement à D , refaire le miroir de e , sinon, poser $D = TempD$.
- Choisir aléatoirement deux arêtes candidates e_1 et e_2 dans C^* . En faire le miroir, calculer la distance de réticulation totale $TempD$. Si elle est inférieure ou égale à D , poser $D = TempD$. Sinon, choisir un décimal au hasard entre 0 et 1 : s'il est inférieur à $e^{\frac{D-tempD}{\theta}}$ alors poser $D = TempD$, sinon refaire le miroir de e_1 et e_2 .

Une opération de miroir peut être faite en $O(n)$, donc notre algorithme a une complexité en $O(ln)$. Il reste à déterminer comment choisir l , qui dépend en fait du nombre d'arêtes dans l'ensemble des candidats. Il s'agira donc effectuer un petit travail d'énumération pour connaître la taille moyenne de l'ensemble des arêtes candidates en fonction de n et r . Puis, après le choix d'une valeur de l en fonction de ces résultats, on pourra obtenir une complexité à comparer avec celles des algorithmes de sériation de dendrogrammes.

Toutefois, des tests avec des valeurs correctes pour l ont donné de bons résultats pour cet algorithme qui trouve visiblement l'ordre optimal sur de petits arbres et qui semble bien fonctionner sur de plus grands arbres avec une dizaine de réticulations.

5 Conclusion

La première partie de ce stage a débouché sur une bonne amélioration du dessin des splitsgraphs par SplitsTree. Obtenir des résultats concrets et satisfaisants était vraiment gratifiant, après la phase de tâtonnements sur différentes stratégies d'optimisation. En outre, Daniel Huson et moi avons préparé un article à ce sujet [GH05], dont une partie est largement reprise dans la section 3.3 et qui regroupe aussi d'autres optimisations à propos des réseaux phylogénétiques sur lesquelles Daniel a travaillé : l'algorithme *Daylight Angle*, l'algorithme *Spring Embedder*, ainsi que la représentation de splitsgraphs enracinés. Le code de l'algorithme Box-Opening a été intégré à la première version non beta de SplitsTree4, sortie le 9 septembre 2005.

En ce qui concerne la seconde partie, je reste en contact avec Tobias Klöpper pour y travailler. J'ai codé la partie de construction du graphe dual, et l'optimisation de l'ordre de ses taxons, mais il reste à l'intégrer dans le code de création de réseaux réticulés de Tobias, ainsi qu'à terminer la discussion sur la complexité de l'algorithme.

Références

- [BD92] H.-J. Bandelt and A. W. M. Dress. A canonical decomposition theory for metrics on a finite set. *Advances in Mathematics*, 92 :47–105, 1992.
- [BM02] D. Bryant and V. Moulton. NeighborNet : An agglomerative method for the construction of planar phylogenetic networks. In R. Guigó and D. Gusfield, editors, *Algorithms in Bioinformatics, WABI 2002*, volume LNCS 2452, pages 375–391, 2002.
- [Bry03] D. Bryant. A classification of consensus methods for phylogenies. In M. Janowitz, F.-J. Lapointe, F.R. McMorris, B. Mirkin, and F.S. Roberts, editors, *BioConsensus*, pages 55–66. DIMACS, 2003.
- [DH04] A. W. M. Dress and D. H. Huson. Constructing splits graphs. *IEEE/ACM Transactions in Computational Biology and Bioinformatics*, 1(3) :109–115, 2004.
- [Fel04] J. Felsenstein. *Inferring Phylogenies*. Sinauer Associates, Inc., 2004.
- [Gam05] P. Gambette. Notes de bioinformatique. Manuscript, <http://philippe.gambette.free.fr/SCOL/NotesBioinfo.pdf>, 2005.
- [GH05] P. Gambette and D.H. Huson. Improved layout of phylogenetic networks. Manuscript, 2005.
- [HB05] D. H. Huson and D. Bryant. Estimating phylogenetic trees and networks using SplitsTree 4. Manuscript in preparation, software available from <http://www.splitstree.org>, 2005.
- [HDKS04] D. H. Huson, T. Dezulian, T. Klopper, and M. A. Steel. Phylogenetic super-networks from partial trees. *IEEE/ACM Transactions in Computational Biology and Bioinformatics*, 1(4) :151–158, 2004.
- [HKLS05] D.H. Huson, T. Klopper, P.J. Lockhart, and M.A. Steel. Reconstruction of reticulate networks from gene trees. In *Proceedings of the Ninth International Conference on Research in Computational Molecular Biology (RECOMB)*, 2005.
- [SN87] N. Saitou and M. Nei. The Neighbor-Joining method : a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4 :406–425, 1987.
- [SS03] C. Semple and M. A. Steel. *Phylogenetics*. Oxford University Press, 2003.

Index

algorithme Box-Opening, 11
algorithme Convex Hull, 5
algorithme Equal Angle, 6
algorithme Equal Daylight, 7
algorithme Optimized Angle, 7
angle de confiance, 10

bootstrap, 6
boîte, 5

candidat, 17
cercle des taxons, 6
collision locale, 9
conflit phylogénétique, 2
crossing-over, 2

sériation de dendrogramme, 14

hybridation, 2

intégron, 2

opération miroir, 17

gènes paralogues, 2
arbre phylogénétique, 2, 4

recombinaison, 2
réseau réticulé, 1, 2
arbre phylogénétique enraciné, 4
rotation d'un sous-arbre, 17

score, 7
split, 4
split circulaire, 5
split network, 2
splits compatibles, 4
splits incompatibles, 4
splitsgraph, 1, 2, 4
superarbre, 5

taxon, 2
distance totale de réticulation, 15
transfert horizontal, 2

Z-closure, 6
zone exclue, 11