IFT2015 automne 2013 — Devoir 3

Miklós Csűrös

6 octobre 2013 — remise mardi le 15 octobre à 20 :15

Remettez un rapport écrit par email (à csuros@iro. . .) en format PDF. La taille de votre message ne doit pas dépasser 2^{20} octets. Travaillez seul sur tous les problèmes. Le devoir vaut 50 points ; vous pouvez avoir 10 points de boni pour la solution de 3.2.3c.

3.1 MMORPG (20 points)

Dans un certain jeu de rôle en ligne massivement multijoueur, on peut faire progresser son personnage dans des combats joueur-contre-joueur ou joueur-contre-monstre. Chaque joueur et monstre possède des **points de vie** $v(x)$ et un **niveau**/classement $c(x)$, tous les deux des entiers non-négatifs. Un joueur commence avec $v(x) = 1$ et $c(x) = 0$; les monstres commencent avec $v(x) > 0$ quelconque et $c(x) = \lfloor \lg v(x) \rfloor$. (Les rapports $v(x) : v(y)$ et $c(x) : c(y)$ déterminent les chances de gagner le combat entre x et y.) Les règles de combat s'appliquent aux joueurs et aux monstres identiquement comme suit : si x (joueur ou monstre) gagne contre y (joueur ou monstre), alors

(a) le vainqueur reçoit les points de vie de son adversaire, et le perdant s'élimine du jeu (et ne peut plus jamais combattre) : $v(x) \leftarrow v(x) + v(y), v(y) \leftarrow 0$;
(b) si x gagne contre un personnage de niveau supérieur, il s'élève au même niveau : $c(x) \leftarrow c(y)$ si $c(x) < c(y)$;
(c) si x gagne contre un adversaire de niveau égal, il avance un niveau : $c(x) \leftarrow c(x)+1$ si $c(x) = c(y)$;
(d) si x gagne contre un personnage de niveau inférieur, son classement ne change pas.

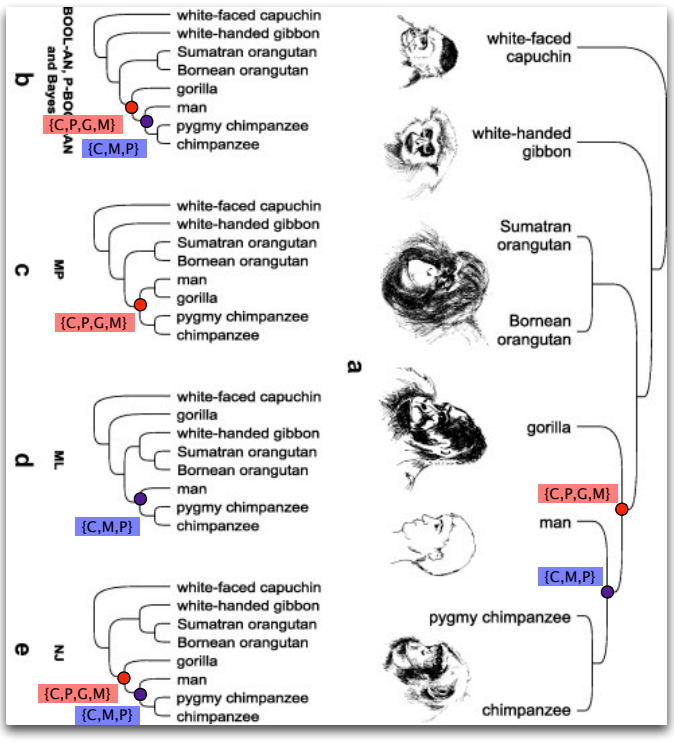
Classement et vitalité. (20 points) ▶ Démontrer formellement que $c(x) \leq \lfloor \lg v(x) \rfloor$ à tout temps pour tout personnage et monstre vivant x. («Vivant» : $v(x) > 0$; lg dénote logarithme binaire.)

Indice: *utiliser de l'induction dans le nombre de combats*

3.2 Comparaison d'arbres non-ordonnés (30+10 points)

3.2.1 Différence entre deux arbres

Supposons qu'on travaille avec des arbres non-ordonnés et on veut comparer si deux arbres ont la même structure (exemple : comparer deux regroupements hiérarchiques sur le même ensemble d'objets). Afin de représenter un arbre enraciné (où l'ordre des enfants n'est pas important), la structure la plus convenable est un arbre ordonné : on stocke les enfants de tout nœud interne x dans un tableau $x.\mathsf{children}[0..d-1]$. Le but de cet exercice est de développer un interface pour la comparaison d'arbres enracinés. Le défi est de détecter des arbres équivalents avec l'ordonnance arbitraire des enfants imposée par l'implantation.



Une application des arbres non-ordonnés est en biologie : une **phylogénie** d'espèces est un arbre enraciné où les nœuds externes correspondent aux espèces et les nœuds internes ont degré > 1. Tout nœud interne x correspond à un ancêtre hypothétique pour un sous-ensemble d'organismes existants. On définit le **clade** $C[x]$ pour chaque nœud x comme l'ensemble de nœuds externes dans son sous-arbre :

$$C[x] = \begin{cases} \{x\} & \text{si } x \text{ est externe ;} \\ \bigcup_{y \in x.\mathsf{children}} C[y] & \text{si } x \text{ est interne} \end{cases}$$

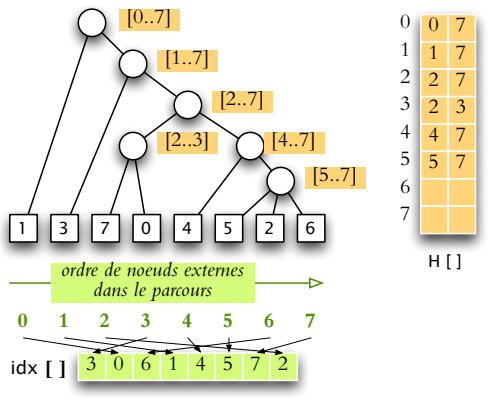
Clairement, la collection de ses clades $\mathcal{C}(T) = \{C[x] : x \in T\}$ détermine sans ambiguïté l'arbre T. (Créer un nœud interne pour tout $C \in \mathcal{C}(T)$, et les lier entre eux — C est la descendante de C' si $C \subset C'$.)

Est-ce que les arbres à la gauche montrent les mêmes relations entre les primates ? Non, parce que sur **c** on n'a pas d'ancêtre commun exclusif aux chimpanzés et l'humain (clade $\{C, M, P\}$). Par contre, **c** contient un ancêtre commun aux chimpanzés, l'humain et le gorille (clade $\{C, P, G, M\}$) mais **d** n'a pas de tel nœud interne.

Ari et al. *Mol Phylogenet Evol* **63** :193–202, 2012

3.2.2 Représentation de la hiérarchie (30 points)

Supposons que les nœuds externes sont étiquetés par $0, 1, 2, \ldots, n-1$ et que tout nœud interne a au moins deux enfants.



On détermine d'abord l'ordre des nœuds externes dans un parcours postfixe, en remplissant un tableau $\mathsf{idx}[0..n-1]$. La cellule $\mathsf{idx}[x]$ contient l'**indice** du nœud externe x ; le premier nœud visité obtient indice 0, le deuxième obtient 1, etc.

Par construction, chaque clade $C[x]$ contient des nœuds avec indices consécutives. En particulier, si $C[x] = \{y_1, y_2, \ldots, y_k\}$, alors $\{\mathsf{idx}[y_1], \mathsf{idx}[y_2], \ldots, \mathsf{idx}[y_k]\} = \{L[x], L[x]+1, \ldots, R[x]\}$ avec $L[x] = \min_i \mathsf{idx}[y_i]$ et $R[x] = \max_i \mathsf{idx}[y_i]$. Soit $N[x]$ le nombre de nœuds externes dans le sous-arbre de x. On a les récurrences suivantes :

$$L[x] = \begin{cases} \mathsf{idx}[x] & \text{si } x \text{ est externe ;} \\ \min_{y \in x.\mathsf{children}} L[y] & \text{si } x \text{ ext interne} \end{cases} \tag{3.2.1a}$$

$$R[x] = \begin{cases} \mathsf{idx}[x] & \text{si } x \text{ est externe ;} \\ \max_{y \in x.\mathsf{children}} R[y] & \text{si } x \text{ ext interne} \end{cases} \tag{3.2.1b}$$

$$N[x] = \begin{cases} 1 & \text{si } x \text{ est externe ;} \\ \sum_{y \in x.\mathsf{children}} N[y] & \text{si } x \text{ ext interne} \end{cases} \tag{3.2.1c}$$

a. (15 points) ▶ Donner un algorithme récursif qui remplit le tableau idx en parcours postfixe.

Indice: *Passer l'indice courant comme argument, et retourner le nombre de nœuds externes dans le sous-arbre.*

b. (15 points) ▶ Donner un algorithme qui calcule $L[x], R[x]$ (en utilisant idx[]), ainsi que $N[x]$ à chaque nœud x par les récurrences de (3.2.1).

Indice: *retourner le triple (L, R, N) dans un parcours postfixe.*

3.2.3 Comparaison de deux arbres (10 points boni)

Maintenant, on peut comparer deux arbres T_1, T_2 sur le même ensemble de nœuds externes $\{0, 1, 2, \ldots, n-1\}$. D'abord, il faut déterminer et stocker idx$[x]$ et $L[y]..R[y]$ dans un parcours postfixe du premier arbre T_1. Ensuite, on calcule (mais ne stocke pas) $L[x], R[x], N[x]$ dans un parcours du deuxième arbre T_2 en utilisant le tableau d'indices idx[] *du premier arbre* T_1. À chaque nœud interne de T_2 (sauf la racine), après avoir déterminé $L[x]$, $R[x]$ et $N[x]$, on fait le test suivant :

(i) si $N[x] \neq R[x] - L[x] + 1$, le clade de x n'est pas présent dans T_1 ;

(ii) si $N[x] = R[x] - L[x] + 1$, le clade de x est présent dans T_1 si et seulement si $L[x]..R[x]$ est l'intervalle d'indices pour un clade quelconque dans T_1.

Pour stocker les intervalles de T_1, on remplit un tableau $H[0..n-1]$ pendant le parcours de T_1. Dès qu'on détermine l'intervalle $L..R = L[x]..R[x]$ au nœud interne x de T_1, on met $L..R$ soit dans la cellule $H[R]$, soit dans la cellule $H[L]$. Si x est le premier enfant (gauche) de son parent, on place $H[R] \leftarrow L..R$; sinon $H[L] \leftarrow L..R$. Ainsi le test en (ii) prend $\Theta(1)$ — il suffit d'examiner $H[L[x]]$ et $H[R[x]]$—, et on peut performer les deux parcours en $\Theta(n)$ temps au total.

c. (10 points boni) Montrer que le remplissage de H est correct : quand on enregistre le clade pour nœud $x \in T_1$, soit $H[R[x]]$ est vide (quand x est le premier enfant), soit $H[L[x]]$ est vide (si x n'est pas le premier enfant).

REMARQUE. Une application de l'algorithme est pour mesurer la **distance** entre deux arbres non-ordonnés T_1, T_2 par le nombre de clades présent dans l'un mais non pas dans l'autre. Dans d'autres mots, on considère la différence symétrique entre les collections de clades $\mathcal{D}(T_1, T_2) = \{C : C \in \mathcal{C}(T_1), C \notin \mathcal{C}(T_2)\} \cup \{C' : C' \in \mathcal{C}(T_2), C' \notin \mathcal{C}(T_1)\}$, et on définit la distance $d(T_1, T_2)$ comme la taille de $\mathcal{D}(T_1, T_2)$. On compte $d(T_1, T_2)$ en (i) et (ii) par l'algorithme de §3.2.3. Calculer la distance entre deux arbres de taille n prend donc $\Theta(n)$ temps.

La même démarche s'applique au calcul d'un arbre de **consensus strict** à partir de plusieurs phylogénies alternatives (p.e., arbres inférés de gènes différents) $T_1, T_2, \ldots, T_k$. Dans cette aplication, on considère que chaque T_i «vote» pour des clades, et on cherche les clades supportés par chacune. On peut suivre l'algorithme de comparaison pour les paires (T_1, T_2), (T_1, T_3), …, (T_1, T_k), et enregistrer les votes en (ii). À la fin, il suffit d'examiner les clades de T_1 et retenir ceux avec assez de votes. L'algorithme finit en $\Theta(nk)$ temps, ce qui est optimal.