Les techniques d'appariement entre arbres. Rapport Bibliographique

Anouar Ben Hassena, Laurent Miclet

Publication interne IRISA numéro 1911

Les techniques d'appariement entre arbres. Rapport Bibliographique.

Anouar BEN HASSENA, Laurent MICLET

Projet CORDIAL

Résumé : le problème de comparer deux arbres intervient dans divers domaines comme les documents structurés (XML), la bioinformatique (les structures secondaires d'ARN), etc. Les algorithmes reposent sur le principe de l'appariement, ou édition (*editing*), d'un arbre en un autre par la composition d'opérations élémentaires, en visant à minimiser leur coût cumulé (la distance d'édition). Dans ce cadre, nous étudions un certain nombre de travaux antérieurs sur l'appariement entre arbres, qui représentent un large éventail des méthodes existantes. Notre but est d'étendre ces méthodes pour définir une analogie entre quatre arbres.

Mots clés : Appariement d'arbres, distance d'édition, arbres ordonnés étiquetés.

(Abstract: pto)

IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex (France)

Téléphone : (33) 02 99 84 71 00 — Télécopie : (33) 02 99 84 71 71 — www.irisa.fr

Abstract: The comparison between two trees is an crucial problem in several domains: structured XML documents, biocomputing (ADN secondary structures), etc. The algorithme are based on the principle of matching the trees by the composition of editing operation. The distance between the trees is the matching of lowest cost, this cost being defined as the sum of the corresponding edit operation. We study several prior works on tree matching by editing, whoch are characteristic of these methods. Our goal is the extension of these methods to the matching of four trees, to define an analogy between trees.

Key-words: Tree matching, edit distance, ordered and labeled trees.

1 Introduction

Le but de notre travail de thèse est de concevoir une nouvelle approche pour la prédiction de la prosodie en parole synthétique, fondée sur la notion d'analogie entre arbres. Dans ce rapport, nous étudions plusieurs travaux antérieurs sur les comparaisons entre arbres par appariement (*editing*), représentant un large éventail des méthodes existantes, qui sont à la base de notre extension vers l'analogie.

Pour la prosodie, nous nous appuyons en particulier sur les travaux de L. Blin [2] où sont détaillés la modélisation prosodique du français en structure de performance ou en modèles accentuels. Dans les deux cas la prosodie est modélisée par une structure d'arbre.

Quant à l'analogie, nous pouvons la voir comme une généralisation de la transformation d'un arbre à un autre pour apparier un nombre fini d'arbres ; en particulier, nous définissons un certain type d'appariement de quatre arbres comme une *proportion analogique*.

Nous commençons par les notations et les définitions que nous allons utiliser tout au long de ce rapport.

1.1 Définitions et terminologie

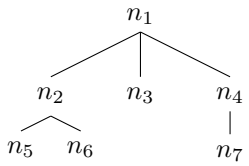
Soit un arbre T. Nous notons l'ensemble de ses nœuds (les feuilles sont inclues) par $V(T)$.

Arbre enraciné. Un *arbre enraciné* est un ensemble de nœuds et un ensemble d'arcs satisfaisant trois propriétés :

- Il existe un nœud particulier appelé *racine*.
- Tout nœud n, autre que la racine, est relié par un arc à un unique antécédent p appelé *père*. n est appelé *fils* de p.
- On peut atteindre la racine à partir de n'importe quel nœud de l'arbre, en se déplaçant de père en père.

On note aussi un arbre par $T(r)$, où r est la racine.

Arbre ordonné. Un arbre est *ordonné* s'il existe un ordre parmi les fils de chaque nœud. Par exemple soit l'arbre $T(n_1)$ ordonné de racine n_1 (dans cet exemple, l'ordre des indices des nœuds est compatible avec l'ordre des fils) :



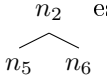
Ancêtre ou ascendant, descendant. Les *ascendants* ou *ancêtres* d'un nœud n sont les nœuds par lesquels on passe pour atteindre la racine (racine et nœud n inclus). Si n_1 est ascendant de n_2, par définition n_2 est descendant de n_1.

Ancêtre propre. Un *ancêtre propre* d'un nœud n est un ancêtre de n (n exclus). Par exemple, n_2 est un ancêtre propre de n_5, mais pas n_2.

Étiquette. Une *étiquette* d'un nœud n_i (étiquette(n_i)) est un élément d'un alphabet fini Σ étendu par le symbole *vide* λ que nous notons $\Sigma_\lambda = \Sigma \cup \lambda$.

Arbre étiqueté. Un arbre est *étiqueté* s'il existe une injection de l'ensemble des nœuds $V(T)$ dans un ensemble d'étiquettes. Ici, on assigne à chaque nœud une étiquette de Σ_λ.

Sous-arbre. Un *sous-arbre* est constitué d'un nœud n_i ainsi que de tous ses descendants. On l'appelle sous arbre de T de racine n_i.

Par exemple,  est un sous arbre de $T(n_1)$

Hauteur. La hauteur d'un nœud n_i est la longueur du chemin entre n_i et la feuille la plus distante dans le sous-arbre de de n_i. (par exemple, la hauteur de n_1 est 2).

Profondeur. La profondeur (ou le niveau) d'un nœud n_i est la longueur du chemin qui mène à n_i à partir de la racine. (par exemple, la profondeur de n_5 est 2).

Forêt. Une forêt est un ensemble fini ordonné d'arbres. Dans ce document nous associons une forêt à un nœud ou à un arbre. Nous notons $F(n_1)$ la forêt obtenue à partir de $T(n_1)$ en supprimant n_1:

$$F(n_1) = \quad n_2 \qquad n_3 \qquad n_4 \qquad\qquad n_5 \quad n_6 \qquad\qquad n_7$$

Cette forêt est constituée de tous les sous arbres enracinés par tous les fils de n_1, soit $T(n_2)$, $T(n_3)$ et $T(n_4)$. On la note aussi $F(n_2,n_4)$.

Forêt partielle. Une forêt partielle est une partie d'une forêt dont les arbres constituants sont adjacents. Par exemple, soit la forêt partielle de $F(n_1)$:

$$F\ (n_2,\ n_3) = \quad n_2 \qquad n_3 \qquad\qquad n_5 \quad n_6$$

Cette forêt partielle de $F(n_1)$ est constituée par les sous arbres $T(n_2)$ et $T(n_3)$.

Parcours préfixé. Le parcours préfixé consiste à traiter la racine, parcourir les sous arbres de gauche à droite. Exemple: Sur l'arbre $T(n_1$ les nœuds sont traités dans l'ordre: n_1,n_2,n_5,n_6,n_3,n_4,n_7.

Parcours postfixé. Le parcours postfixé consiste à parcourir les sous arbres de gauche à droite et puis traiter la racine. Exemple: Sur l'arbre $T(n_1)$ les nœuds sont traités dans un ordre: n_5,n_6,n_2,n_3,n_7,n_4,n_1.

Position d'un nœud. Les nœuds d'un arbre T sont identifiés par leur position, où une position est une séquence d'entiers. ϵ est la position du racine, 1 le premier enfant le plus à gauche (*leftmost*), etc (Voir FIG. 1). L'ensemble des positions des nœuds de $V(T)$ est noté $Pos(T)$.

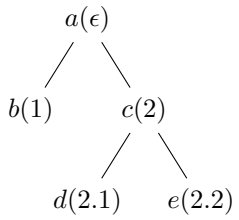


FIG. 1 – *Exemple d'un arbre T étiqueté. Les positions des nœuds sont entre parenthèses.*

Isomorphe. Deux arbres T_1 et T_2 sont dites isomorphes s'ils ont le même nombre n de nœuds, et si, pour tout $i \in Pos(T_1)$, le nœud de position i dans T_1 a le même nombre de sous-arbres que le nœud de position i dans l'arbre T_2.
Autrement dit, si deux arbres sont isomorphes, ils sont structurellement identiques.

1.2 Opérations d'édition

Étant T_1 et T_2 deux arbres étiquetés, nous présentons selon [15] une opération d'édition comme $(l_1 \rightarrow l_2)$, où $(l_1,l_2) \in (\Sigma_\lambda \times \Sigma_\lambda) \setminus (\lambda,\lambda)$. Cette opération s'appelle:

- *Insertion* si $l_1 = \lambda$.
- *Suppression* si $l_2 = \lambda$.
- *Substitution* si $l_1 \neq \lambda$ et $l_2 \neq \lambda$.

Cette opération porte sur les étiquettes, mais nous pouvons aussi la faire porter sur les nœuds en définissant $(v \rightarrow w)$ par (étiquette(v) $\rightarrow$ étiquette(w)).

Détaillons maintenant les propriétés de chaque opération.

1.2.1 Substitution

Cette opération consiste simplement à modifier l'étiquette d'un nœud sans changer la structure de l'arbre. La figure 2 illustre ce principe. L'étiquette a d'un nœud du premier arbre est substituée par une étiquette b pour former le deuxième arbre.

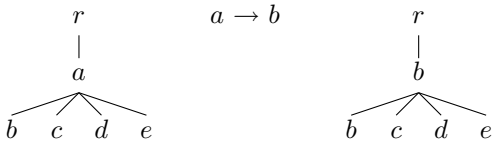


FIG. 2 – *Substitution d'un nœud a en un nœud b*

1.2.2 Suppression

Cette opération modifie la structure de l'arbre. Lors de la suppression d'un nœud v, les nœuds fils de v s'insère à la place que v occupait, devenant ainsi des fils du nœud père de v. La figure 3 présente un exemple de suppression.

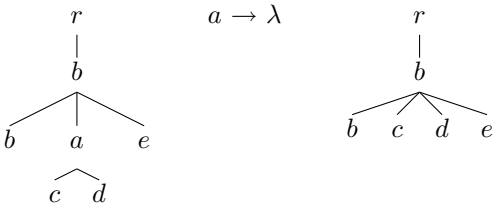


FIG. 3 – *Suppression d'un nœud a*

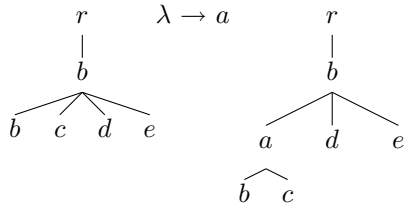
1.2.3 Insertion

L'insertion d'un nœud est l'opération complémentaire de la suppression. Insérer un nœud w en tant que fils d'un nœud v fait de w le père d'une séquence de nœuds consécutifs précédemment fils de v. Un exemple est fourni par la figure 4. Notons que contrairement aux opérations de substitution et de suppression, il n'existe pas une seule et unique façon d'insérer un nœud à un endroit donné en suivant cette définition. Il est envisageable que la séquence de nœuds dont w devient le père soit moins longue (figure 4(a)), ou bien qu'elle soit vide (figure 4(b)), ou qu'elle contienne tous les fils précédents de v (figure 4(c)). Ces alternatives dans l'application de l'opérateur offrent un degré de liberté supplémentaire que doivent gérer les algorithmes.

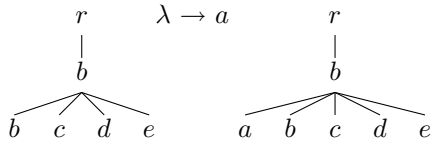
2 Distance entre arbres par édition

La comparaison de deux arbres étiquetés ordonnés peut se faire en utilisant les opérations d'édition de suppression, insertion, et substitution des nœuds. Ces opérations mènent à une *distance d'édition* ou un *alignement* des arbres, qui en est un cas très particulier. Dans ce chapitre, nous nous intéressons à la distance d'édition générale entre arbres, ainsi qu'à des variantes plus contraintes. Pour chaque méthode, nous étudions les travaux antérieurs et leurs résultats et nous présentons le ou les algorithmes proposés et leur complexité.

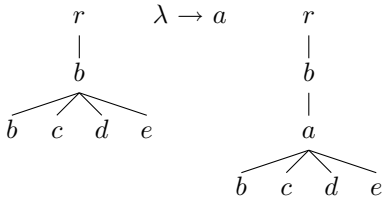
Le problème de la comparaison des arbres intervient dans divers domaines tel que les bases de données et les documents structurés (XML), la biologie et les structures secondaires d'ARN, l'optimisation des compilateurs, le traitement du langage naturel et l'analyse d'image [15, 18, 10, 6, 12, 19].



(a) le nouveau nœud prend comme fils deux des feuilles.



(b) le nouveau nœud ne prend aucune feuille comme fils



(c) le nouveau nœud prend toutes les feuilles comme fils

FIG. 4 – *Trois possibilités d'insertion d'un nœud a comme fils du nœud b, le fils de la racine*

Par exemple, en biologie, où la comparaison des structures secondaire d'ARN, représentés par des arbres, permet de déterminer les similitudes fonctionnelles entre ces molécules [18, 8].

La première section présente d'abord la notion d'appariement (*mapping*) [1] entre arbres, puis nous présentons la distance entre arbres qui en découle et les travaux récents autour de cette notion.

2.1 Mapping

2.1.1 Mapping entre arbres

Afin de représenter plus facilement la transformation d'un arbre en un autre, il est utile d'introduire le concept de *mapping*. Intuitivement, un mapping est une description de la façon dont une séquence d'opérations d'édition transforme un arbre en un autre, sans tenir compte de l'ordre dans lequel ces opérations sont effectuées.

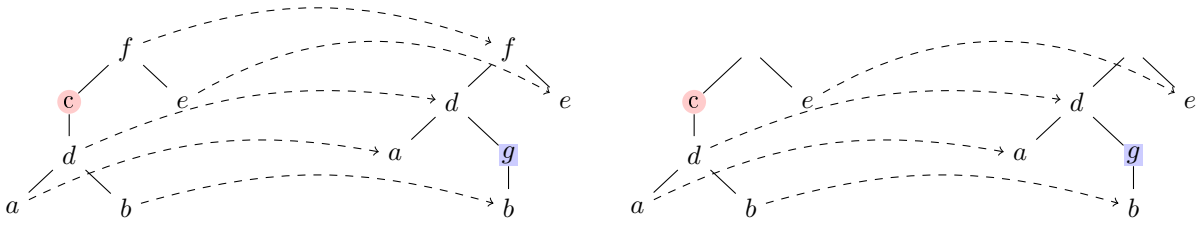
Formellement, un mapping peut être défini comme un ensemble M de couples de nœuds (i, j) respectivement des arbres T_1 et T_2 définissant les opérations élémentaires pour passer d'un arbre à un autre et satisfaisant les deux conditions suivantes:
Pour toutes les paires (i_1,j_1) et (i_2,j_2) dans M :

1. $i_1 = i_2 \iff j_1 = j_2$. (*one-to-one condition*)
2. i_1 est à gauche de $i_2 \iff j_1$ est est à gauche de j_2. (*sibling condition*)
3. i_1 est un ancêtre de $i_2 \iff j_1$ est un ancêtre de j_2. (*ancestor condition*)

1. Nous conservons dans ce texte certains termes techniques anglais sans les traduire.

2.1.2 Mapping entre forêts

En rappelant que nous nous limitons au forêt finie ordonnée issue d'un arbre en supprimant sa racine (voir p.4), nous définissons le mapping entre forêts comme un mapping entre deux arbres en ignorant leurs racines tout en conservant les conditions définies ci dessus.



(a) Exemple de mapping entre arbres

(b) Exemple de mapping entre forêts

FIG. 5 – *Exemple de mapping*

2.2 Distance d'édition entre arbres

Le principe de la distance entre arbres est similaire à celui de la distance entre séquences, définie par Wagner et Fisher [16]. A chacun des trois opérateurs de base on associe un coût $c(v,w)$ qui assigne une valeur réelle positive à l'opération d'édition correspondante ($v \rightarrow w$). La fonction c est ensuite étendue au coût d'une séquence d'opérations correspond à un mapping M par simple addition des opérations : $c(M) = \sum_{(v,w)\in M} c(v,w)$.

La distance d'édition entre deux arbres T_1 et T_2 est alors définie par :

$$dist(T_1,T_2) = min\ \{c(M) \mid M\ mapping\ de\ T_1\ vers\ T_2\}$$

L'étape suivante consiste à trouver ce mapping de coût minimal. Plusieurs algorithmes existent pour traiter cela. Historiquement, le premier algorithme pour la distance d'édition entre arbres a été proposé par Tai en 1979 [15]. C'est un dinosaure, qui a été remplacé avantageusement par des algorithmes épurés, plus rapides. Nous le présentons pour commencer.

L'algorithme de base [15] Nous présentons tout d'abord une simple récursion de [15] qui forme la base des algorithmes de programmation dynamique présentés dans les sections suivantes.

Soit une forêt F et v un nœud dans F. Nous notons $F - v$ la forêt obtenue par la suppression du nœud v de F, de même $F - T[v]$ la forêt obtenue par la suppression de v et de tous les descendants de v. Le lemme suivant montre une manière de calculer la distance d'édition dans le cas général des forêts.

Lemme 1 Soit F_1 et F_2 deux forêts ordonnées et c la fonction coût définie sur les nœuds. Soit v et w les racines des arbres les plus à droite (*rightmost*) dans F_1 et F_2 respectivement. On a,

$$\begin{aligned}
&c(\theta,\theta) = 0\\
&c(F_1,\theta) = c(F_1 - v,\theta) + c(v \rightarrow \lambda)\\
&c(\theta,F_2) = c(\theta,F_2 - w) + c(\lambda \rightarrow w)\\
&c(F_1,F_2) = min = \begin{cases} c(F_1 - v,F_2) + c(v \rightarrow \lambda)\\ c(F_1,F_2 - w) + c(\lambda \rightarrow w)\\ c(F_1[v],F_2[w]) + c(F_1 - T_1[v],F_2 - T_2[w]) + c(v \rightarrow w) \end{cases}
\end{aligned}$$

Preuve. Les trois premières équations sont triviales. Pour montrer la dernière équation, nous considérons un mapping M de coût minimal entre F_1 et F_2. Il y a trois possibilités pour v et w :

Cas 1. $(v,\lambda) \in M$, donc le premier cas de la dernière équation est appliqué.

Cas 2. $(\lambda,w) \in M$, donc le deuxième cas de la dernière équation est appliqué.

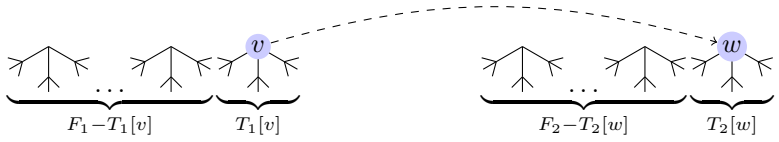


FIG. 6 – *Dernièrer équation du 3ème cas du Lemme 1*

Cas 3. Montrons que $(v,w) \in M$. Supposons que (v,h) et (k,w) sont dans M. Si v est à droite de k, h doit être à droite de w par *sibling-condition*.
Si v est un ancêtre propre de k donc h doit être un ancêtre propre de w par *ancestor-condition*. Ces deux configurations sont impossibles puisque que v et w sont des racines *rightmost* (voir figure 6), et par suite $(v,w) \in M$ et le troisième point de la dernière équation est établi.

Le lemme 1 se traduit par un algorithme de programmation dynamique. Nous remarquons bien que la distance $c(F_1,F_2)$ passe par des appels récursifs sur les *sous-problèmes* de F_1 et F_2. Chaque nouveau *sous-problème* peut être calculé dans un temps constant. Par conséquent, la complexité $O(c(F_1,F_2))$ est bornée par le nombre de ses appels récursifs multiplié par le temps mis pour chaque appel. Ces appels dépendent évidement du nombre de paires de *sous-problèmes* (s_{i_1},s_{j_2}) de F_1 et F_2.

Afin de calculer le nombre des *sous-problèmes*, nous définissons d'abord pour une forêt F la sous forêt *(i,j)-deleted* $(0 \leq i+j \leq |F|\ )$ comme une forêt obtenue par la suppression de sa racine *rightmost* j fois et puis similairement de sa racine *leftmost* i fois.

Nous avons bien la sous forêt *(0,j)-deleted* préfixe de F et *(i,0)-deleted* suffixe de F. Nous constatons que le nombre de *(i,j)-deleted* sous forêt est $O(|F|^2)$ (pour chaque i il y a $|F| - i$ choix pour j).

En analysant les appels récursifs du lemme 1, on montre que les *sous-problèmes* s_{i_1} et s_{j_2}de F_1 et F_2 associés à ces appels sont bien des *(i,j)-deleted* sous forêts pour chaque forêt. Par conséquent, on montre que la complexité de cette algorithme est largement majoré par $O(|F_1|^2\,|F_2|^2)$. Améliorer cette complexité par optimisation du nombre des sous problèmes paraît possible. D'où l'apparition de nouveaux algorithmes que nous allons les présenter dans les sections suivantes.

Optimisation 1 : Zhang et Shasha [18] Le principe est d'améliorer l'algorithme précédent en ne faisant pas porter le calcul sur tous les nœuds de l'arbre, mais uniquement sur certains nœuds stratégiques.

Pour cela, définissons pour un arbre enraciné ordonné T des nœuds particuliers (nous notons $root(T)$ le nœud racine de T.

$$Keyroots(T) = \{root(T)\} \cup \{v \in V(T) | v \text{ a un frère gauche}\}$$

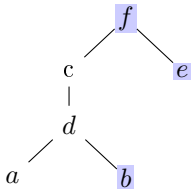


FIG. 7 – *Les* keyroot *d'un arbre T*

Les *sous forêts particulières* de T sont définies comme les $F(v)$, $v \in Keyroots(T)$. Les *sous-problèmes* de T respectant les *keyroots* sont les préfixes de tous les sous forêts particuliers $F(v)$, nous les notons dans cette section *relevant-subproblems*.

Zhang et Shasha [18] ont montré qu'il suffit d'appliquer le lemme 1 sur ces nœuds particuliers et donc qu'il suffit de calculer $c(s_{i_1},s_{j_2})$ pour chaque *relevant-subproblems* s_{i_1} et s_{j_2} de T_1 et T_2 respectivement. On obtient ainsi un temps de calcul :

$$\sum_{i\in Kr(T_1)}\sum_{j\in Kr(T_2)}|T_1(i)|\times|T_2(j)| = \sum_{i\in Kr(T_1)}|T_1(i)|\times\sum_{j\in Kr(T_2)}|T_2(j)|$$

Cette complexité se calcule comme suit. Pour chaque noeud $v\in V(T)$, nous définissons *cdepth(v)* comme le nombre des ancêtres *keyroots* de v. Ainsi, $cdepth(T)$ est le maximum des $cdepth(v)$.

Lemme 2 Pour un arbre T le nombre des *relevant-subproblems* est $O(|T|cdepth(T))$.

$$\sum_{v\in Kr(T)}|T(v)| \;=\; \sum_{v=1}^{|T|}cdepth(v) \;\leq\; |T|cdepth(T)$$

Preuve.

- *Nous considérons la formule à gauche,*
  - *un noeud i est compté a chaque fois qu'il appartient à un sous arbre $T(v)$*
  - *Seulement les sous arbres des keyroots sont considéré.*
- $\Rightarrow$ *Le nœud i est compté pour chaque keyroots.*
- *$cdepth(i)$ est le nombre des ancêtres keyroots de i (par définition).*

D'où la première égalité et la deuxième inégalité est évidente.

Nous pouvons écrire maintenant la formule de complexité suivante :

$$\sum_{i\in Kr(T_1)}|T_1(i)|\times\sum_{j\in Kr(T_2)}|T_2(j)| \;\leq\; |T_1||T_2|cdepth(T_1)cdepth(T_2)$$

Lemme 3 Pour un arbre T, cdepth$(T)\leq$ min {profondeur(T),feuille(T) }

Theorème 1. Pour un arbre T_1 et T_2, la distance d'édition entre eux est de complexité :

- $O(|T_1||T_2|$ min{profondeur(T_1),feuille(T_1)} min{profondeur(T_2),feuille(T_2)}) en temps.
- $O(|T_1||T_2|)$ en espace.

Optimisation 2 : Klein [9] L'algorithme de Klein est basé, aussi, sur une extension de la récursion du lemme 1 en évitant de visiter tous les nœuds. Klein définit une décomposition de T, en *relevant-subforest*, basée sur la technique *heavy path decomposition*[5]. Nous classifions chaque nœud de T soit *heavy* (ou *lourd*)soit *light* comme suit. La racine est *light* et pour chaque nœud v nous choisissons parmi ses fils celui dont la profondeur du sous-arbre dont il est la racine arbre est la plus grande. Nous le classifions comme *heavy* et les autres comme *light*. Nous définissons par la suite $ldepth(v)$ comme le nombre des nœuds *light* dans le chemin de v à la racine.

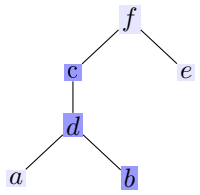


FIG. 8 – *Décomposition d'un arbre T en light et heavy nœuds*

Lemme 4. Pour un arbre T et $v\in V(T)$, $ldepth(v)\leq log|T|+O(1)$.

En supprimant les nœuds *light*, T est partitionné en *chemins lourds*.

Nous définissons les *relevant-subforest* de T relatifs aux nœuds *light* comme suit. Pour un nœud v de T, $F(v)$ est *relevant-subforest*. Si v n'est pas une feuille, soit u le fils le plus lourd de v, soit l et r les nombre des nœuds à gauche et à droite de u respectivement. Donc les sous forêts *(i,0)-deleted* de $F(v)$, $0 \leq i \leq l$, et les *(l,j)-deleted* sous forêts de $F(v)$, $0 \leq j \leq r$, sont des *relevant-subforest*. Récursivement pour tous les *relevant-subforest* de $F(u)$.

Les *relevant-subforest* de T relatifs aux nœuds *light* est l'union de tous les F_v ,*relevant-subforest* de $F(v)$, où $v \in V(T)$ est un nœud *light*.

Lemme 5. Le nombre total des *relevant-subforest* d'un arbre T relatif à la technique *heavy path decomposition* est au maximun :

$$\sum_{v \in light(F)} |F_v| \leq \sum_{v \in F} 1 + ldepth(v) \leq \sum_{v \in F} (log|F| + O(1)) = O(|F|log|F|).$$

La distance est ensuite calculé en confrontant les *relevant-subforest* du premier arbre (*directeur*) à l'ensemble des forêts du second arbre.

Théorème 2. Pour deux arbres ordonnés enracinés T_1 et T_2, le calcul de la distance d'édition entre eux est de complexité

- $O(|T_1|^2|T_2|log|T_2|)$ en temps.
- $O(|T_1||T_2|)$ en espace.

Optimisation générique : Dulucq et Touzet [3] Dans [3] Dulucq et Touzet ont introduit le concept de *stratégie de décomposition* en tant qu'un cadre d'analyse commun pour les deux algorithmes précédents [9, 18]. Ils montrent que ces deux approches, développées *a priori* de manière indépendante, sont en fait tout à fait analogues dans l'esprit.

Définition 1.(Stratégie de décomposition) .

Une *stratégie de décomposition* est succession de choix entre une décomposition à gauche ou une décomposition à droite. Plus formellement si nous définissons l'ensemble des sous-forêts de F, $SF(F)$ et l'ensemble des choix de décomposition {droite,gauche}, alors une stratégie S pour le calcul de $c(F_1,F_2)$ est définie comme l'application $SF(F_1) \times SF(F_2) \xrightarrow{S} \{droite,gauche\}$.

Dans cette généralisation, la stratégie de l'algorithme de Zhang et Shasha est : $S(F_{i_1},F_{j_2}) = droite$ pour tous F_{i_1} et F_{j_2}. Celle de l'algorithme de Klein est : $S(F_{i_1},F_{j_2}) = droite$ si le nœud le plus à gauche est *heavy* et $S(F_{i_1}, F_{j_2}) = gauche$ sinon.

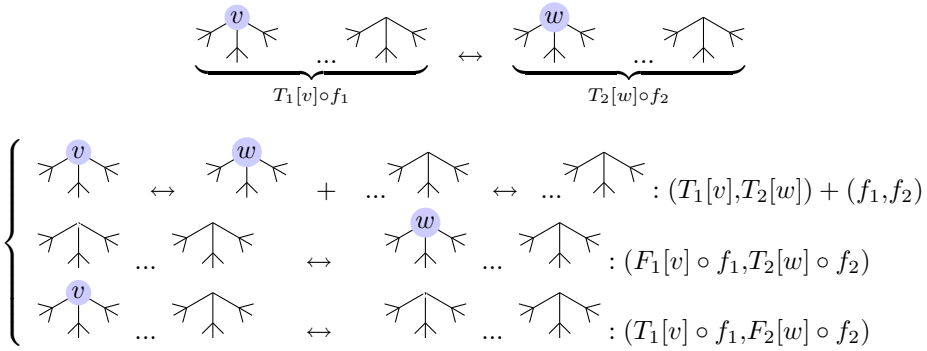


FIG. 9 – *Algorithme de décomposition basé sur la stratégie gauche*

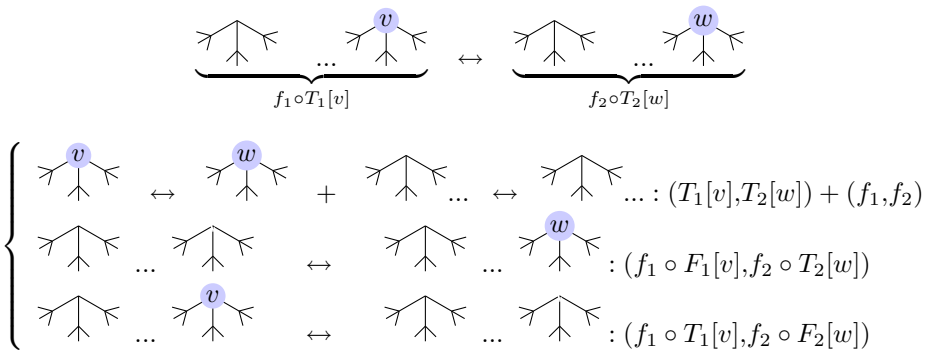
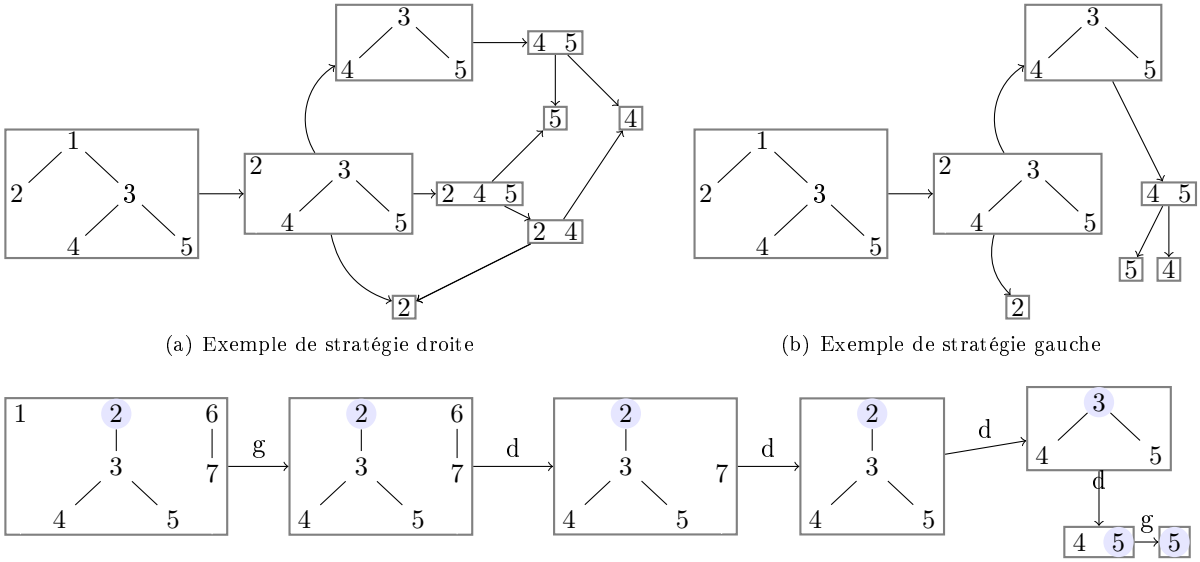


FIG. 10 – *Algorithme de décomposition basé sur la stratégie droite*



(a) Exemple de stratégie droite

(b) Exemple de stratégie gauche

(c) Exemple de la stratégie de Klein

FIG. 11 – *Exemples de stratégie*

Dulucq et Touzet prouvent une complexité de $O(|T_1||T_2|log(|T_1||T_2|))$ pour chaque stratégie de décomposition.

Lemme 6. Soit F_1 et F_2 deux forêts. Pour chaque stratégie S, pour chaque nœud i de F_1 et pour chaque j de F_2, $(F_1(i), F_2(j))$ est un élément de l'ensemble des *relevant forest* indépendamment de la stratégie.

Dans ce cadre, les auteurs ont définit une nouvelle stratégie (*cover strategies*) comme une extension des deux stratégies analysées (Zhang et klein). Ils ont proposé un algorithme qui minimise le nombre de *relevant-subforest*. Il est de complexité $O(|T|^3log(|T))$ dans le pire des cas et $O(|T|^3)$ en moyenne.

3 Distance d'édition sous contraintes

Le fait que la distance d'édition est en général un problème difficile à résoudre a conduit à étudier des versions restreintes du problème. Dans [17], Zhang a introduit la notion de *distance d'édition sous*

contraintes , δ_c, qui est définie comme une distance d'édition avec des restrictions sur les mappings possibles des sous arbres.

Formellement $\delta_c(T_1, T_2)$ est définie comme le coût minimum du mapping (M_c, T_1, T_2) satisfaisant la contrainte suivante:

Soit *nca* le plus petit ancêtre commun. Pour tous les (v_1, w_1), (v_2, w_2) et $(v_3, w_3) \in M_c$:

– $nca(v_1, v_2)$ est un ancêtre propre de v_3 si et seulement si $nca(w_1, w_2)$ est un ancêtre propre de w_3.

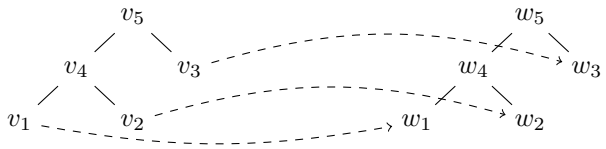
Dans le même cadre, Richter [13] a introduit la *distance d'édition respectant la structure*, δ_s définie comme le coût minimum du mapping (M_s, T_1, T_2) satisfaisant la contrainte additionnelle suivante :

– $nca(v_1, v_2) = nca(v_1, v_3)$ si et seulement si $nca(w_1, w_2) = nca(w_1, w_3)$.

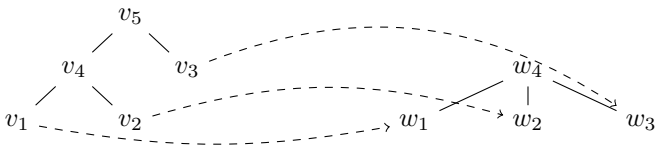
Ce n'est pas difficile de montrer que ces deux notions de distance sont équivalent. Nous les notons, simplement, dans ce qui suit, par *distance d'édition sous contraintes*. Par exemple, selon [1], nous considérons les mappings de la figure 12.

12(a) est un mapping sous contrainte parce que $nca(v_1, v_2) \neq nca(v_1, v_3)$ et $nca(w_1, w_2) \neq nca(w_1, w_3)$. 12(b) ne vérifie pas la contrainte, nous avons bien $nca(v_1, v_2) = v_4 \neq nca(v_1, v_3) = v_5$, tans que $nca(w_1, w_2) = w_4 = nca(w_1, w_3)$. Aussi pour 12(c) vu que $nca(v_1, v_3) = v_5 = nca(v_2, v_3)$, tans que $nca(w_1, w_3) = w_5 \neq nca(w_2, w_3) = w_4$.

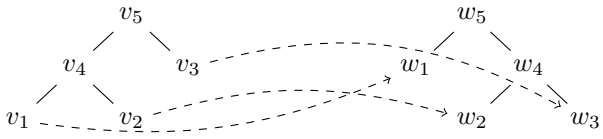
Dans [17] Zhang présente un algorithme pour calculer le coût minimum d'un mapping sous contraintes. Dans le cas où les arbres sont ordonnées, son algorithme atteint une complexité de $\theta(|T_1|\ |T_2|)$ en temps et en espace. L'idée est que la restriction sur le mapping engendre moins de sous problèmes considérés et par conséquent un algorithme de programmation dynamique plus rapide. De sa part, Richter [13]a présenté un algorithme de complexité $\theta(|T_1||T_2|\ \deg(T_1)\ \deg(T_2))$ en temps et $\theta(|T_1|\ \deg(T_2)$ profondeur$(T_2))$ en espace. Donc, pour un degré et une profondeur assez petits cet algorithme améliore la complexité en espace par rapport à celui de Zhang.



(a) *Mapping sous-contrainte et less-constrained*



(b) *Mapping non sous-contrainte mais less-constrained*



(c) *Mapping ni sous-contrainte ni less-constrained*

FIG. 12 – *Exemples de mapping sous contraintes*

Récemment, Lu et al. [11] présente une nouvelle notion de distance d'édition sous contrainte (*less-constrained edit distance*), δ_l, qui étend le mapping sous contrainte en satisfaisant la contrainte suivante :

pour tous (v_1, w_1), (v_2, w_2) et $(v_3, w_3) \in M_l$ sachant que ni v_1 ni v_2 et ni v_3 n'est pas un ancêtre des autres :

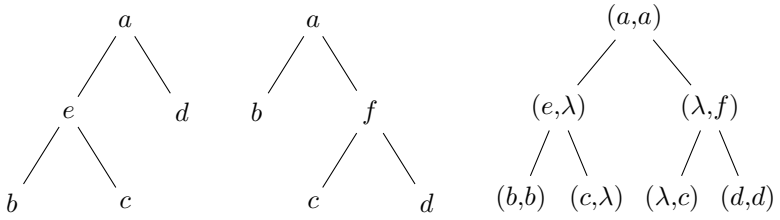
– profondeur$(nca(v_1, v_2)) \geq$ profondeur$(nca(v_1, v_3))$ et $nca(v_1, v_3) = nca(v_2, v_3)$ si et seulement si profondeur$(nca(w_1, w_2)) \geq$ profondeur$(nca(w_1, w_3))$ et $nca(w_1, w_3) = nca(w_2, w_3)$.

Par exemple, considérant les mappings de la figure 12. (a) est *less-constrained* parce que il est déjà *sous contrainte*. (b) est non *sous contrainte* mais *less-constrained* puisque profondeur($nca(v_1, v_2)$) $\geq$ profondeur($nca(v_1, v_3)$), $nca(v_1, v_3) = nca(v_2, v_3)$, $nca(w_1, w_2) = nca(w_1, w_3)$ et $nca(w_1, w_3) = nca(w_2, w_3)$. (c) est non *less-constrained* puisque profondeur($nca(v_1, v_2)$) $\geq$ profondeur($nca(v_1, v_3)$) et $nca(v_1, v_3) = nca(v_2, v_3)$ tant que $nca(w_1, w_3) = nca(w_2, w_3)$.

4 Alignement des arbres

Dans ce chapitre, nous traitons une autre manière de calculer une ressemblance entre arbres ordonnés, par un mapping sous contraintes particulières appelé *alignement*.

Soient T_1 et T_2 deux arbres enracinés étiquetés et c une fonction de coût sur les étiquettes. Un alignement A de T_1 T_2 est un mapping qui est obtenu tout d'abord, par l'insertion de nœuds d'étiquette λ dans T_1 et dans T_2 de telle façon qu'ils aient la même structure. Ensuite, les opérations d'éditions nécessaires pour passer d'un arbre à l'autre sont comptabilisées. Le coût d'un alignement A est la somme de tous les coûts des opérations dans A. Nous le notons par $\alpha(T_1, T_2)$. La figure 13 (de [8]) montre un exemple d'alignement entre deux arbres ordonnés.



(a) Arbre T_1 (b) Arbre T_2 (c) Alignement de T_1 et T_2

FIG. 13 – *Exemple d'un Alignement A*

Un alignement d'arbre est un cas particulier d'un mapping. Il est une restriction au cas où toutes les insertions doivent être faites avant toutes les suppressions[2]. De cela, nous avons bien la distance d'édition: $\delta(T_1, T_2) \leq \alpha\ (T_1, T_2)$. Par exemple, nous considérons la figure 13 où $c(x,y) = 1$ *si* $x \neq y$, 0 *sinon* . La distance d'édition optimale est obtenue par la suppression de e puis l'insertion de f, d'où $\delta(T_1, T_2) = 2$. Un alignement optimal est obtenu selon l'arbre de la fig.16(c), avec un coût de 4.

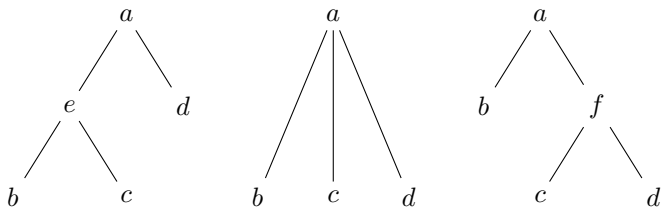
Remarques :

1. Inégalité triangulaire :
   Il est évident à partir de l'exemple suivant Fig.14 que l'alignement ne satisfait pas l'inégalité triangulaire et par la suite nous ne pouvons pas le considérer comme une distance métrique.
2. Alignement et distance d'édition dans les séquences :
   Il est prouvé que la distance d'édition et l'alignement des séquences sont deux notions équivalentes (voir [4]). Par contre, cela n'est pas vrai dans les arbres (voir exemple précédent).

4.1 Algorithme d'alignement de deux arbres ordonnés

L'alignement des arbres ordonnés a été introduit par Jiang et al. dans [8]. Cet algorithme présente une complexité de $O(|T_1||T_2|(\text{degré}(T_1) + \text{degré}(T_2))^2)$ en temps et $O(|T_1||T_2|(\text{degré}(T_1) + \text{degré}(T_2)))$ en espace. Nous le détaillons dans cette section.

2. Cette définition, donnée telle quelle par Jiang, ne permet pas de bien comprendre quel sous-ensemble des éditions entre arbres est ainsi délimité. Nous sommes en train de la reformuler en terme de contraintes sur le mapping, exprimées par des relations d'ordres sur les positions des nœuds appariés.



(a) Arbre T_1 (b) Arbre T_3 (c) Arbre T_2

FIG. 14 – $\alpha(T_1,T_3) + \alpha(T_3,T_2) = 2 < 4 = \alpha(T_1,T_2)$.

4.1.1 Algorithme de Jiang et al.

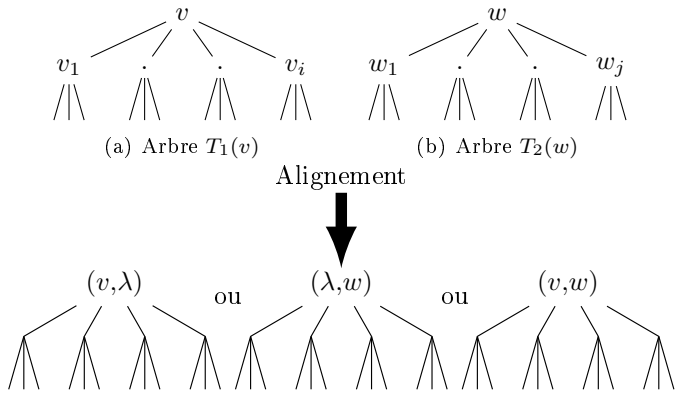
Nous présentons dans cette section l'algorithme de Jiang et al.[8]. Nous montrons comment calculer un alignement entre deux arbres. Soit c la fonction coût sur les étiquettes, nous notons par $c(v,w)$ la fonction coût sur les étiquettes des nœuds v et w, au lieu de c(étiquette(v),étiquette(w), $\alpha(T_1,T_2)$ est le coût d'alignement des arbres T_1, T_2, $\alpha(F_1,F_2)$ est le coût d'alignement des forêts[3] F_1, F_2.

Lemme 1. Soit v un nœud de T_1 et w un nœud de T_2 avec $v_1,...,v_i$ et $w_1,...,w_j$ leurs fils respectifs. Nous avons, en notant θ l'arbre vide :

$$\begin{array}{l}\alpha(\theta,\theta) = 0\\ \alpha(T_1(v),\theta) = \alpha(F_1(v),\theta) + c(v,\lambda)\\ \alpha(\theta,T_2(w)) = \alpha(\theta,F_2(w)) + c(\lambda,w)\\ \alpha(F_1(v),\theta) = \sum_{k=1}^{i} \alpha(T_1(v_k),\theta)\\ \alpha(\theta,F_2(w)) = \sum_{k=1}^{j} \alpha(T_2(\theta,w_k))\end{array}$$

Lemme 2. Soit v un nœud de T_1 et w un nœud de T_2 avec $v_1,...,v_i$ et $w_1,...,w_j$ leurs fils respectifs. Nous avons :

$$\alpha(T_1(v),T_2(w)) = min \left\{ \begin{array}{l} \alpha(T_1(v),\theta) + min_{1\leq r\leq i}\{\alpha(T_1(v_r),T_2(w)) - \alpha(T_1(v_r),\theta)\}\\ \alpha(\theta,T_2(w)) + min_{1\leq r\leq j}\{\alpha(T_1(v),T_2(w_r)) - \alpha(\theta,T_2(w_r))\}\\ \alpha(F_1(v),F_2(w)) + c(v,w)\end{array}\right.$$



(a) Arbre $T_1(v)$ (b) Arbre $T_2(w)$

(c) Les 3 cas de la racine de l'arbre d'alignement A

FIG. 15 – *Alignement A des arbres T_1 et T_2*

3. La définition d'alignement peut être étendu aux forêts ordonnées de la même façon que le mapping à la page 7

Preuve. (d'après [8]) Considérant un alignement optimal A de $T_1(v)$ et $T_2(w)$. Il y a trois cas possibles:

1. la racine de A est (v,λ) et (k,w) est un nœud de A pour $k \in V(T_1) \cup \{\lambda\}$. Dans ce cas:
$$\alpha(T_1(v),T_2(w)) = \alpha(T_1(v),\theta) + min_{1\leq r\leq i}\{\alpha(T_1(v_r),T_2(w)) - \alpha(T_1(v_r),\theta)\}$$
2. La racine de A est (λ,w) et (v,k) est un nœud de A pour $k \in V(T_2) \cup \{\lambda\}$. Dans ce cas:
$$\alpha(T_1(v),T_2(w)) = \alpha(\theta,T_2(w)) + min_{1\leq r\leq j}\{\alpha(T_1(v),T_2(w_r)) - \alpha(\theta,T_2(w_r))\}$$
3. la racine de A est (v,w). Dans ce cas:
$$\alpha(T_1(v),T_2(w)) = \alpha(F_1(v),F_2(w)) + c(v,w)$$

Lemme 3. Soit v un nœud de T_1 et w un nœud de T_2 avec $v_1,...,v_i$ et $w_1,...,w_j$ leurs fils respectivement. Pour tout $1 \leq s \leq i$ et $1 \leq t \leq j$, nous avons,

$$\alpha(F_1(v_1,v_s),F_2(w_1,w_t)) = min \begin{cases} \alpha(F_1(v_1,v_{s-1}),F_2(w_1,w_t)) + \alpha(T_1(v_s),\theta) \\ \alpha(F_1(v_1,v_s),F_2(w_1,w_{t-1})) + \alpha(\theta,T_2(w_t)) \\ \alpha(F_1(v_1,v_{s-1}),F_2(w_1,w_{t-1})) + \alpha(T_1(v_s),T_2(w_t)) \\ c(\lambda,w_t) + min_{1\leq k\leq s}\{\alpha(F_1(v_1,v_{k-1}),F_2(w_1,w_{t-1})) + \alpha(F_1(v_k,v_s),F_2(w_t))\} \\ c(v_s,\lambda) + min_{1\leq k\leq t}\{\alpha(F_1(v_1,v_{s-1}),F_2(w_1,w_{k-1})) + \alpha(F_1(v_s),F_2(w_k,w_t))\} \end{cases}$$

Preuve. (d'après[8]) Considérons un alignement optimal A entre les forêts $F_1(v_1,v_s)$ et $F_2(w_1,w_t)$. Il y a trois cas possibles pour la racine *rightmost* de la forêt résultante de l'alignement A.

1. Soit (v_s,w_t), donc l'arbre la plus à droite de A doit être l'alignement optimal de $T_1(v_s)$ et $T_2(w_t)$. Par conséquent,
$\alpha(F_1(v_1,v_{s-1}),F_2(w_1,w_{t-1})) + \alpha(T_1(v_s),T_2(w_t))$
2. Soit (v_s,λ), donc $T_1(v_s)$ est aligné avec la sous-forêt $F_2(w_{t-k+1},w_t)$, $0 \leq k \leq t$. Les sous-cas suivants peuvent se présenter:

   (a) $(k = 0)$. $T_1(v_s)$ est aligné avec $F_2(w_{t+1},w_t) = \theta$. Donc,
   $\alpha(F_1(v_1,v_{s-1}),F_2(w_1,w_t)) + \alpha(T_1(v_s),\theta)$
   (b) $(k = 1)$. $T_1(v_s)$ est aligné avec $F_2(w_t,w_t) = T_2(w_t)$(similaire avec le cas 1.

   (c) $(k \geq 2)$, c'est le cas générale où,
   $c(v_s,\lambda) + min_{1\leq k\leq t}\{\alpha(F_1(v_1,v_{s-1}),F_2(w_1,w_{k-1})) + \alpha(F_1(v_s),F_2(w_k,w_t))\}$

3. Soit (λ,w_t). Similaire au cas 2.

Cette récursion peut être à la base d'un algorithme de programmation dynamique *bottom-up*. Considérant deux nœuds v et w avec $v_1,...,v_i$ et $w_1,...,w_j$ leurs fils respectivement. Nous avons besoin de calculer $\alpha(F_1(v_h,v_k),F_2(w))$ pour tous $1 \leq h \leq k \leq i$, et $\alpha(F_1(v),F_2(w_h,w_k))$ pour tous les couples (h,k) tels que $1 \leq h \leq k \leq j$. Autrement dit, nous avons besoin de calculer l'alignement optimal de $F_1(v)$ avec chaque sous-forêt de $F_2(w)$ et respectivement l'alignement optimal de $F_2(w)$ avec chaque sous-forêt de $F_1(v)$. Ces alignements doivent passer par des calculs intermédiaire de $\alpha(F_1(v_s,v_p),F_2(w_t,w_q))$ tels que $1 \leq s \leq p \leq i$, et $1 \leq t \leq q \leq t$. Il est très facile de montrer que ces alignements entre sous-forêts partielles sont traités aussi par la lemme 3.

Complexité en temps : Pour une entrée de $F_1(v_s,v_i)$ et $F_2(w_t,w_j))$ tels que $1 \leq s \leq p \leq i$, et $1 \leq t \leq q \leq t$, $\alpha(F_1(v_s,v_i),F_2(w_t,w_j))$ peut se calculer en $O((i-s)(j-t)(i-s+j-t)) = O(ij(i+j))$. Par conséquent, le temps de calculer les (i+j) sous-problèmes, est bornée par $O(ij(i+j)^2)$. Pour un alignement optimal de T_1 et T_2, il faut :

$$\sum_{v\in V(T_1)} \sum_{w\in V(T_2)} O(deg(v)deg(w)(deg(v) + deg(w))^2)$$

$$\leq \sum_{v\in V(T_1)} \sum_{w\in V(T_2)} O(deg(v)deg(w)(deg(T_1) + deg(T_2))^2)$$

$$\leq O((deg(T_1)+deg(T_2))^2) \sum_{v\in V(T_1)} \sum_{w\in V(T_2)} O(deg(v)deg(w))$$

$$\leq O(|T_1||T_2|(deg(v)+deg(w))^2)$$

Théorème : Pour deux arbres ordonnées T_1 et T_2, leur alignement optimal est calculé en $O(|T_1||T_2|(deg(v)+deg(w))^2)$ en temps et $O(|T_1||T_2|(deg(v)+deg(w)))$ en espace.

4.1.2 Travaux connexes

Récemment, dans [7], un nouveau algorithme a été proposé pour les arbres similaires. Spécifiquement, si un alignement optimal de T_1 et T_2 utilise au maximum s nœud vide (λ), l'algorithme le calcule en $O((|T_1|+|T_2|)log(|T_1|+|T_2|)(\text{degré}(T_1)+\text{degré}(T_2))^4 s^2)$ en temps. Cet algorithme a été inspiré de la méthode d'alignement optimal entre séquences similaires (voir section3.3.4 dans [14]). L'idée est d'accélérer l'algorithme de Jiang et al. par que la considération des sous-arbres de T_1 et de T_2 qui diffèrent au plus $O(s)$.

4.2 Proposition d'une nouvelle version de l'algorithme Jiang et al

Nous essayons dans cette section d'étudier la possibilité de proposer un nouveau lemme qui fusionne à la fois les deux récursions du lemme 2 et 3 en une seule récursion. L'idée est que une forêt est un ensemble ordonné d'arbres et que cet ensemble peut avoir un seul élément.

Lemme :

Pour $F_1(v_s,v_p)$ et $F_2(w_t,w_q)$ tels que $1 \leq s \leq p \leq i$, et $1 \leq t \leq q \leq j$:

$$\alpha(F_1(v_s,v_p),F_2(w_t,w_q)) = \\ min \begin{cases} c(v_p,w_q)+\alpha(F_1(v_p),F_2(w_q))+\alpha(F_1(v_s,v_{p-1}),F_2(w_t,w_{q-1})) \\ c(\lambda,w_q)+min_{s\leq k\leq (p+1)}\{\alpha(F_1(v_s,v_{k-1}),F_2(w_t,w_{q-1}))+\alpha(F_1(v_k,v_p),F_2(w_q))\} \\ c(v_p,\lambda)+min_{t\leq k\leq (q+1)}\{\alpha(F_1(v_s,v_{p-1}),F_2(w_t,w_{k-1}))+\alpha(F_1(v_p),F_2(w_k,w_q))\} \end{cases}$$

(Preuve.) Considérant un alignement optimal de $F_1(v_s,v_p)$ et $F_2(w_t,w_q)$. Il y a trois cas possibles en raisonnant sur les deux racines *rightmost* des deux forêts F_1 et F_2 :

1. Substitution des deux racines (v_p,w_q). Dans ce cas, le premier cas du *min* est appliquée.
2. Insertion de la racine *rightmost* de $F_2(w_t,w_q)$, donc nous avons (λ,w_q) et il reste à aligner $F_1(v_s,v_p)$ avec $(F_2(w_t,w_q)-w_q)$, celui là nous le décomposons en deux alignements (1)$F_1(v_s,v_{k-1})$ et $F_2(w_t,w_{q-1})$, (2) $F_1(v_k,v_p)$ et $F_2(w_q)$, tout en cherchons le minimum sur k. Cette décomposition qui témoigne la restriction de l'alignement vs la distance d'édition.
3. Suppression de la racine *rightmost* de $F_1(v_s,v_p)$. Similaire au cas 2.

En appliquant ce Lemme aux arbres ($T(v)=F(v,v)$) nous aurions, pour $T_1(v_p)=F_1(v_p,v_p)$ et $T_2(w_q)=F_2(w_q,w_q)$:

$$\alpha(F_1(v_p,v_p),F_2(w_q,w_q)) = \\ min \begin{cases} c(v_p,w_q)+\alpha(F_1(v_p),F_2(w_q))+\alpha(F_1(v_p,v_{p-1}),F_2(w_q,w_{q-1})) \\ c(\lambda,w_q)+min_{p\leq k\leq (p+1)}\{\alpha(F_1(v_p,v_{k-1}),F_2(w_q,w_{q-1}))+\alpha(F_1(v_k,v_p),F_2(w_q))\} \\ c(v_p,\lambda)+min_{q\leq k\leq (q+1)}\{\alpha(F_1(v_p,v_{p-1}),F_2(w_q,w_{k-1}))+\alpha(F_1(v_p),F_2(w_k,w_q))\} \end{cases}$$

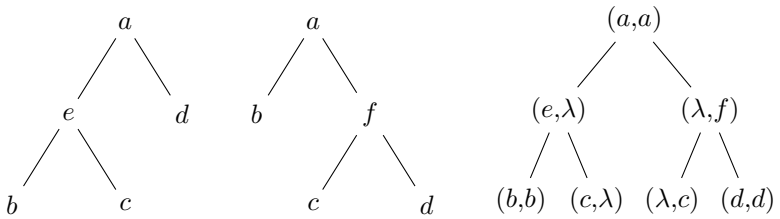
$$=^{F(v)=F(v_1,v_{n_v})}_{F(v_k,v_{k-1})=\Theta}$$

$$min \begin{cases} c(v_p,w_q) + \alpha(F_1(v_p),F_2(w_q)) + \alpha(\Theta,\Theta) \\ c(\lambda,w_q) + \alpha(\Theta,\Theta) + \alpha(F_1(v_p,v_p),F_2(w_q))\} \\ c(\lambda,w_q) + \alpha(F_1(v_p,v_p),\Theta) + \alpha(\Theta,F_2(w_q))\} \\ c(v_p,\lambda) + \alpha(\Theta,\Theta) + \alpha(F_1(v_p),F_2(w_q,w_q))\} \\ c(v_p,\lambda) + \alpha(\Theta,F_2(w_q,w_q)) + \alpha(F_1(v_p),\Theta)\} \end{cases}$$

Nous avons donc bien un lemme qui peut être à la base d'un algorithme de programmation dynamique, à la fois, pour l'alignement des forêts aussi bien que pour les arbres ($T(v) = F(v,v)$). Cette nouvelle formulation va nous permettre maintenant de nous intéresser aux alignements d'un nombre fini $n > 2$ d'arbres à la fois, puisque que ce lemme nous offre la possibilité d'aligner un arbre avec une forêt, ce qui n'était pas possible avec les deux anciens lemmes de Jiang.

4.3 Alignement multi-arbres ordonnés et analogie

Nous rappelons tout d'abord, d'une façon informelle, la définition de l'alignement entre deux arbres ordonnés et étiquetés. En effet, un alignement représente un appariement nœud à nœud de même position entre les deux arbres, dans lequel un ou plusieurs nœuds vides peuvent être insérés. L'appariement (λ, λ) n'est pas permis.



(a) Arbre t_1 (b) Arbre t_2 (c) Alignement de t_1 et t_2

FIG. 16 – *Exemple d'un Alignement entre arbres ordonnés*

Nous allons essayer d'étendre cette notion d'alignement à un plus grand nombre d'arbres et en particulier pour quatre arbres afin de définir la proportion analogique entre arbres comme suit.

Définition 4.1. .
Soit x, y, z et t quatre arbres d'étiquettes prises dans un ensemble fini Σ. On suppose qu'une analogie dans Σ_λ est définie. Nous disons que ces arbres sont en proportion analogique s'il existe quatre arbres x', y', z' et t' d'étiquettes $\in \Sigma_\lambda$, de même structure, dans lesquelles nous avons inséré des nœuds vides λ et vérifiant :

– $\forall\ i \in Pos$[4] *, les analogies $x'_i : y'_i :: z'_i : t'_i$ sont vraies.*

Par exemple, soit les analogies suivantes dans Σ_λ :
– $a : b :: A : B$
– $a : \alpha :: b : \beta$
– $A : \alpha :: B : \beta$
L'alignement suivant entre les quatre arbres présente une proportion analogique.

4. voir la définition de *Pos* à la page 4

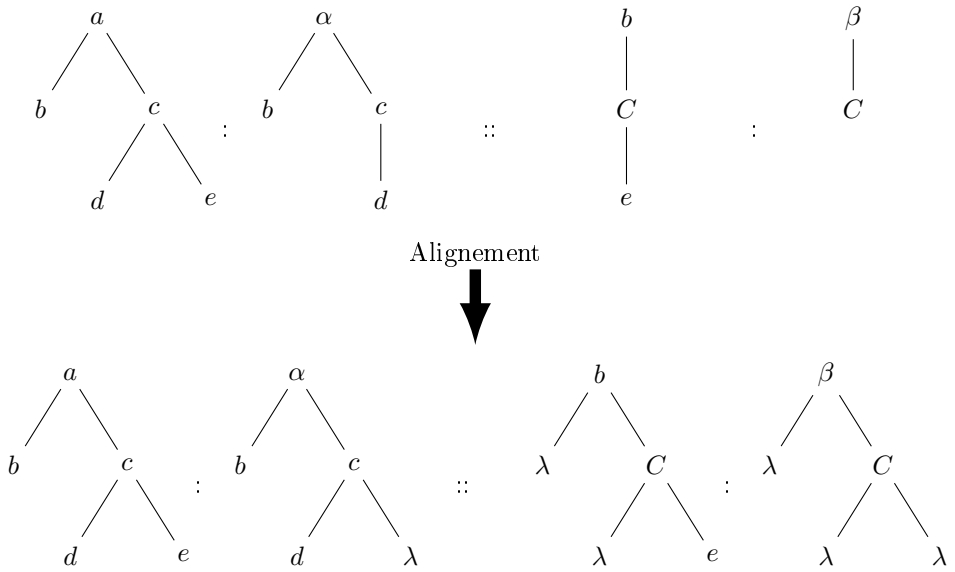


FIG. 17 – *Exemple de proportion analogique basée sur l'alignement*

5 Conclusion

Nous avons présenté dans ce rapport plusieurs travaux antérieurs qui ont soulevé le problème d'appariement entre arbres ordonnés. Plusieurs auteurs ont proposé des méthodes, au fur et à mesure, optimisées au sens de la complexité de leurs algorithmes. Mais le fait que ce problème reste difficile à résoudre, a conduit d'autres auteurs à étudier des versions restreintes du problème, par exemple Zhang qui a introduit la notion de distance d'édition sous contraintes. L'alignement, aussi bien, présente une autre manière de calculer une ressemblance entre arbres par un mapping sous contraintes particulières. Cette notion d'alignement, vu ses propriétés compatible avec l'analogie, est à la base de notre extension vers la définition de la proportion analogique entre arbres.

Références

[1] P. Bille. A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337(1-3):217–239, 2005.

[2] Laurent Blin. Apprentissage de structures d'arbres à partir d'exemples. *Thèse*, 2002.

[3] Serge Dulucq and Hélène Touzet. Analysis of tree edit distance algorithms. In *In Proceedings of the 14th annual symposium on Combinatorial Pattern Matching (CPM*, pages 83–95. Springer-Verlag, 2003.

[4] Dan Gusfield. Algorithms on strings, trees, and sequences. In *Computer Science and Computational Biology Cambridge University Press*, 1997.

[5] Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.

[6] Christoph M. Hoffmann and Michael J. O'Donnell. Pattern matching in trees. *J. ACM*, 29(1):68–95, 1982.

[7] Jesper Jansson and Andrzej Lingas. A fast algorithm for optimal alignment between similar ordered trees. *Fundam. Inform.*, 56(1-2):105–120, 2003.

[8] Tao Jiang, Lusheng Wang, and Kaizhong Zhang. Alignment of trees - an alternative to tree edit. In *CPM '94: Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching*, pages 75–86, London, UK, 1994. Springer-Verlag.

[9] Philip N. Klein. Computing the edit-distance between unrooted ordered trees. In *ESA*, pages 91–102, 1998.

[10] Philip N. Klein, Srikanta Tirthapura, Daniel Sharvit, and Benjamin B. Kimia. A tree-edit-distance algorithm for comparing simple, closed shapes. In *SODA*, pages 696–704, 2000.

[11] Chin Lung Lu, Zheng-Yao Su, and Chuan Yi Tang. A new measure of edit distance between labeled trees. In *COCOON*, pages 338–348, 2001.

[12] R. Ramesh and I. V. Ramakrishnan. Nonlinear pattern matching in trees. *J. ACM*, 39(2):295–316, 1992.

[13] Thorsten Richter. A new measure of the distance between ordered trees and its applications, technical report 85166-cs. *Technical report, Department of Computer Science, University of Bonn*, 1997.

[14] J. Setubal and J. Meidanis. Introduction to computational biology. *PWS Publishing Company*, 1997.

[15] Kuo-Chung Tai. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, 1979.

[16] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, 1974.

[17] Kaizhong Zhang. Algorithms for the constrained editing distance between ordered labeled trees and related problems. *Pattern Recognition*, 28(3):463–474, 1995.

[18] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262, 1989.

[19] Kaizhong Zhang, Dennis Shasha, and Jason Tsong-Li Wang. Approximate tree matching in the presence of variable length don't cares. *J. Algorithms*, 16(1):33–66, 1994.