Comparaison de paires par alignement de séquences

Sommaire

Insérer ou supprimer du code entre la version originale et la copie d'un morceau de code source induit un impact sur la représentation de celui-ci, que ce soit sous une forme lexemisée ou par un arbre de syntaxe. Ainsi la recherche de facteurs exacts ou sous-arbres exacts, que nous allons aborder ultérieurement, se révèle inadéquate car découpant des correspondances pour cause de fossés non correspondants. Nous rappelons ainsi dans ce chapitre des techniques d'alignement de chaînes et d'arbres par programmation dynamique permettant de récupérer de telles correspondances approchées. Si celles-ci sont de complexité générale prohibitive pour la comparaison globale de projets, elles peuvent s'avérer intéressantes afin de raffiner la nature de zones de forte similarité pressenties par d'autres méthodes de filtrage de complexité temporelle plus avantageuse.

5.1 Introduction à l'alignement de séquences de lexèmes

Présentation La recherche de similarité entre deux projets exprimés sous la forme de chaînes de lexèmes peut bénéficier de méthodes d'alignement de séquences. Ces méthodes utilisant des techniques de programmation dynamique sont couramment utilisées en bioinformatique afin de déterminer des zones de similarité entre séquences biologiques (nucléotides d'ADN, ARN ou acides aminés) et quantifier leur niveau de différence ou de ressemblance afin de déterminer des phylogénies. Il s'agit ainsi schématiquement de déterminer les chaînes ajoutées, supprimées ou modifiées entre les deux séquences comparées. Dans cet objectif soit une métrique de similarité, soit une métrique de distance entre deux séquences t_1 et t_2 d'éléments d'un alphabet Σ est calculée à l'aide d'opérations élémentaires que sont :

1. La correspondance entre deux éléments (*match*).
2. La suppression d'un élément (*del*).
3. L'ajout d'un élément (*add*).
4. Le remplacement d'un élément (*sub*).

À chacune de ces opérations est associée un coût $\mathcal{C}$: l'objectif étant de trouver la séquence d'opérations d'édition maximisant la métrique de similarité ou minimisant la métrique de distance. Dans le premier cas les coûts des opérations d'édition ($\mathcal{C}_{\text{del}}$, $\mathcal{C}_{\text{add}}$, $\mathcal{C}_{\text{sub}}$) sont strictement inférieurs au coût de correspondance ($\mathcal{C}_{\text{match}}$), dans le second cas il s'agit du contraire. On note que le jeu d'opérations élémentaires présenté n'est pas minimal (dans la mesure où une substitution peut être remplacée par une suppression suivie d'un ajout) ; il pourrait également être étendu avec de nouvelles opérations si cela présente un intérêt pratique (substitutions d'une chaîne de longueur non-unitaire d'éléments, coûts personnalisés utilisant des matrices de coût...).

Plus longue sous-séquence commune (LCS : *Longest Common Subsequence*) et suite d'opérations d'édition à deux chaînes d'éléments En cherchant à minimiser la distance d'édition avec $\mathcal{C}_{\text{match}} = 0 < \mathcal{C}_{\text{del}} = \mathcal{C}_{\text{add}} = \mathcal{C}_{\text{sub}}$ et en collectant la suite d'opérations élémentaires *match*, nous pouvons déterminer une plus longue sous-séquence commune à deux séquences de lexèmes u et v. Nous rappelons (voir page 11) qu'une sous-séquence u' de la chaîne u est une suite extraite de u et se présente sous la forme $u' = u'_1 u'_2 \cdots u'_{n'}$ avec pour tout $i \in [1..n'-1]$ l'existence d'indices $\alpha < \beta$ tels que $u'_i = u_\alpha$ et $u'_{i+1} = u_\beta$. La détermination de la LCS de deux suites finies de lexèmes u et v dérivés d'unités de code source nous permet de relever des chaînes discontinues de lexèmes similaires entre u et v liées à la présence de lexèmes insérés, supprimés ou modifiés par des opérations d'édition de code. Alors que la connaissance des opérations de *match* permet l'établissement de la LCS, celle complémentaire des opérations d'édition permet d'établir la séquence d'opérations élémentaires d'édition de coût minimal transformant u en v. Ainsi par exemple, le couple de chaînes $u = abc$ et $v = acd$ possède pour LCS la sous-séquence ac (obtenue par les opérations de *match* sur les lexèmes concordants a et c) et pour séquence d'opérations d'édition la suppression de b puis l'ajout de d.

Facteurs correspondants L'objectif d'une méthode d'alignement de lexèmes ne consiste pas à déterminer la LCS de deux unités lexemisées mais plutôt un ensemble de couples de facteurs exactement ou approximativement correspondants. Un couple de facteurs exactement

correspondants présente une égalité lexème par lexème contrairement aux couples approximativement correspondants possédant une distance d'édition non nulle. Le seul critère d'une distance d'édition seuil apparaît néanmoins limitatif pour juger de la pertinence d'un couple de facteurs correspondants. Celle-ci doit être mise en relation avec la longueur (voire le volume) des facteurs. La présence d'une zone de non-correspondance (fossé) importante peut également motiver la réductibilité d'un couple de correspondances, i.e. son découpage en plusieurs couples de plus petites longueurs n'intégrant pas le fossé.

5.2 Programmation dynamique

Le calcul de la métrique de similarité (ou de distance) $s(u, v)$ basée sur une des séquences d'opérations élémentaires optimales $e(u, v)$ sur une paire de séquences de lexèmes ($u = u_1 \cdots u_{m-1}u_m, v = v_1 \cdots v_{n-1}v_n$) est un problème fondamentalement récursif[1] nécessitant la connaissance de :

$\{s, e\}(u_1 \cdots u_{m-1}, v_1 \cdots v_{n-1})$	$\{s, e\}(u_1 \cdots u_{m-1}, v)$
$\{s, e\}(u, v_1 \cdots v_{n-1})$	

Il s'agit ensuite de sélectionner l'opération d'édition optimisant s à partir d'une des trois paires de sous-chaînes de u et v. Le coût de cette opération est alors ajouté pour s et celle-ci est ajoutée à la séquence d'opération e. Nous constatons que le calcul récursif des valeurs de s et e pour les trois paires de sous-chaînes occasionne des calculs redondants avec jusqu'à $3^{\max(m,n)}$ appels récursifs. Cela explique l'utilisation d'une méthode de programmation dynamique par stockage des valeurs de métrique dans une matrice où la cellule (i, j) contient $s(u_1 \cdots u_i, v_1 \cdots v_j)$: la matrice est alors calculée par balayage par ligne, colonne, diagonale ou anti-diagonale. Finalement la séquence d'opérations d'édition optimale peut être obtenue par la détermination du chemin de la matrice partant de la cellule (m, n) (chaînes (u, v)) jusqu'à la cellule $(0, 0)$ (chaînes (ϵ, ϵ)) en déduisant pour chaque cellule (i, j) l'opération d'édition optimale ainsi que la cellule prédécesseur parmi les trois cellules $(i - 1, j)$, $(i, j - 1)$ et $(i - 1, j - 1)$. Le chemin représentant la séquence d'opérations sur la matrice n'est pas nécessairement unique.

Nous constatons ainsi que la détermination de la métrique de similarité $s(u, v)$ nécessite mn opérations pour le remplissage de la matrice avec la nécessité de conserver en mémoire uniquement une ligne (calcul par balayage par ligne) ou une colonne (calcul par balayage par colonne) de la matrice et la cellule précédemment calculée. La reconstitution de la séquence d'opérations élémentaires d'édition optimale nécessite en revanche la conservation complète de la matrice, soit une mémoire de $mn \log_2(|E|)$ bits (avec E le jeu d'opérations élémentaires d'édition), les opérations pouvant être conservées à la place des valeurs de similarité.

On notera que lorsque le calcul de la matrice est réalisé par balayage par anti-diagonale, seules les deux précédentes anti-diagonales sont nécessaires au calcul des valeurs de l'anti-diagonale courante : l'absence de dépendance à une cellule de l'anti-diagonale courante autorise le calcul simultané en parallèle de toutes les valeurs des cellules de cette anti-diagonale ce qui est un moyen avantageux d'exploiter certaines capacités d'unités de traitement graphiques (GPU) [46, 73], dans la limite de la taille de leurs tampons de calcul.

[1] Pour l'initialisation, nous convenons de $s(\epsilon, \epsilon) = 0$ et $e(\epsilon, \epsilon) = \emptyset$.

Méthode mémoriellement linéaire de Hirschberg Lorsque les ressources mémorielles sont restreintes et que la séquence des opérations élémentaires d'édition est requise, la méthode de Hirschberg [44] par division récursive de l'espace matriciel peut être utilisée. Elle consiste à déterminer la cellule $(x_0, \lfloor n/2 \rfloor)$ du chemin suivi par les opérations élémentaires : x_0 peut être calculé par la conservation de pointeurs vers les cellules de la colonne $\lfloor n/2 \rfloor$ pour la colonne précédente de calcul et la cellule précédemment calculée. Finalement, le procédé est répété sur les sous-matrices correspondant aux chaînes $(u_1 \cdots u_{x_0}, v_1 \cdots v_{\lfloor n/2 \rfloor})$ et $(u_{x_0+1} \cdots u_m, v_{\lfloor n/2 \rfloor} \cdots v_n)$ pour déterminer x_{11} et x_{12} et ainsi de suite récursivement jusqu'à l'obtention de matrices de dimensions directement manipulables en mémoire centrale, voire de matrices de taille unitaire dans le cas extrême, le traitement de ces matrices pouvant être réparti sur plusieurs machines. Ce procédé ne nécessite que la conservation d'une colonne et une cellule de pointeurs et de valeur de similarité, soit $m+1$ cellules mais augmente le temps d'exécution car nécessitant le calcul d'une matrice de dimensions (m, n) à l'itération 0 (on suppose que n est une puissance de 2), de 2 matrices de dimensions $(x_{11}, n/2)$ et $(x_{12}, n/2)$ à l'itération 1, ..., de 2^i matrices de dimensions $(x_{i1}, n/2^i), \cdots, (x_{i2^i}, n/2^i)$, ... Par ailleurs, $\sum_{1 \leq k \leq 2^i} x_{ik} = m$ d'où $\frac{mn}{2^i}$ cellules à calculer à la récursion i, soit $2mn + o(mn)$ cellules pour l'ensemble des récursions : le temps d'exécution est donc multiplié par un facteur 2.

5.3 Alignement global

Les méthodes d'alignement global sur deux chaînes de lexèmes ont pour objectif de calculer une métrique de similarité ou de distance entre la *globalité* de ces chaînes ainsi que les opérations élémentaires d'édition associées. Ces méthodes sont particulièrement adaptées à la recherche de différences (séquence d'opérations d'édition) entre codes sources relativement similaires plutôt qu'à la recherche de petites similitudes enfouies dans des codes sources différents. Elles sont donc généralement utilisées par les gestionnaires de version afin de déterminer une séquence d'opérations élémentaires d'édition ainsi que la plus longue sous-séquence commune entre deux versions successives de fichier (les lexèmes élémentaires considérés étant souvent les lignes du fichier).

5.3.1 Alignement global avec code non-transposable

À titre d'exemple, nous cherchons à déterminer la plus longue sous-séquence commune et les opérations élémentaires d'édition entre les deux implantations de fonction de calcul du n^{e} nombre de Fibonacci (cf figure 5.1). Pour cela, dans un premier temps, nous identifions les instructions similaires. Ensuite nous utilisons une méthode d'alignement global sur la séquence linéaire des instructions des deux implantations :

$$f_1 = aaabbcdeec'd \quad f_2 = aaaefgcdeec'dh$$

La matrice de programmation dynamique obtenue par l'alignement global des deux chaînes est représentée en figure 5.1c avec les valeurs de métrique de distance et les flèches symbolisant les chemins de retour nécessaires afin de déterminer la séquence des opérations d'édition. Nous constatons que la distance d'édition est de 4 et que trois séquences d'opérations d'édition permettent d'obtenir cette distance minimale :

– la première comprend l'insertion de e (déclaration inutile) puis la substitution de bb par fg (réécriture des instructions *return*),

```
1  int f1(int n)
   {
3   int k = 1; /* a */
    int l = 1; /* a */
    int m = 0; /* a */
    if (n == 1) return k; /* b */
    if (n == 2) return l; /* b */
8   for (int i = 3; i < n; i++) { /* c */
     m = k + l; /* d */
     k = l; /* e */
     l = m; /* e */
    } /* c' */
13  return m; /* d */
   }
```

(a) Fonction originale f_1

```
1  int f2(int n)
   {
    int k = 1; /* a */
    int l = 1; /* a */
    int m = 0; /* a */
6   byte inutile = 0; /* e */
    if (n == 1 || n == 2) return k; /* f */
    if (n < 1) return 0; /* g */
    for (int i = 3; i < n; i++) /* c */
    {
11   m = k + l; /* d */
     k = l; /* e */
     l = m; /* e */
    } /* c' */
    return m; /* d */
16  assert(false); /* h */
   }
```

(b) Fonction copiée f_2

	ϵ	a	a	a	e	f	g	c	d	e	e	c'	d	h
ϵ	0	1	2	3	4	5	6	7	8	9	10	11	12	13
a	1	↖ 0	1	2	3	4	5	6	7	8	9	10	11	12
a	2	1	↖ 0	1	2	3	4	5	6	7	8	9	10	11
a	3	2	1	↖ 0	← 1	2	3	4	5	6	7	8	9	10
b	4	3	2	1	↖ 1	←↖ 2	3	4	5	6	7	8	9	10
b	5	4	3	2	2	↖ 2	↖← 3	4	5	6	7	8	9	10
c	6	5	4	3	3	3	3	↖ 3	4	5	6	7	8	9
d	7	6	5	4	4	4	4	4	↖ 3	4	5	6	7	8
e	8	7	6	5	4	5	5	5	4	↖ 3	4	5	6	7
e	9	8	7	6	5	5	6	6	5	4	↖ 3	4	5	6
c'	10	9	8	7	6	6	6	7	6	5	4	↖ 3	4	5
d	11	10	9	8	7	7	7	7	7	6	5	4	↖ 3	← 4

(c) Matrice de distance d'édition avec information de retour

FIG. 5.1 – Copie de fonction avec instructions insérées, supprimées et remplacées avec la matrice de distance d'édition (par lignes) correspondante

– la seconde considère le remplacement de b par e, l'insertion de f et le remplacement de b par g,
– la troisième alternative consiste à remplacer bb par ef puis à insérer g.

Dans une optique de recherche de similarité, la première alternative est préférable. Il aurait été possible de supprimer la seconde et la troisième alternative en utilisant des coûts de remplacement personnalisés selon la similarité des instructions (avec ici $\mathcal{C}_{\text{sub}}(b,f), \mathcal{C}_{\text{sub}}(b,g) < \mathcal{C}_{\text{sub}}(b,e)$).

5.3.2 Alignement global avec code transposable

Les méthodes d'alignement global montrent leur limitation lorsque des opérations de transposition de volume important de code sont réalisées. Nous étudions ici deux cas caractéristiques. Le premier concerne une unité composée de deux zones de code qui auraient été échangées (telles que des fonctions) : seule la plus grande zone (en termes de lexèmes) est reportée comme LCS. Pour le second cas, des petits facteurs communs conduisent au report d'une LCS rendant invisible l'existence d'un facteur commun plus grand.

La plus longue sous-séquence commune est un facteur (ou un quasi-facteur)

Nous considérons deux fonctions f et g implantées dans l'ordre fg pour l'unité de compilation 1 et gf pour l'unité de compilation 2. f et g sont représentées par leurs séquences d'instructions. La détermination de la plus longue sous-séquence commune par alignement global retourne la sous-séquence correspondant à f si $|f| > |g|$: aucune similitude n'est alors relevée pour g. Plus généralement si une paire de séquence de lexèmes u et v possèdent les facteurs communs maximaux (i.e. extensibles ni sur la gauche, ni sur la droite) $\alpha_1, \alpha_2, \cdots, \alpha_k$, les facteurs étant deux à deux distincts, alors la plus longue sous-séquence commune de lexèmes est la plus longue sous-séquence commune de facteurs communs maximaux. Ainsi si u est de la forme $\alpha_1 \cdots \alpha_2 \cdots\cdots \alpha_k$ et v de la forme $\alpha_k \cdots \alpha_{k-1} \cdots\cdots \alpha_1$, la plus longue sous-séquence commune de lexèmes sera le plus grand facteur commun parmi l'ensemble des facteurs communs $\{\alpha_i\}_{1 \leq i \leq k}$.

La remarque précédente peut également être étendue pour des couples de facteurs approximativement correspondants, extension floue de la notion de facteur commun intégrant des fossés de non-correspondance.

La sous-séquence commune la plus longue étant une sous-chaîne, une solution pourrait alors être de réexécuter un alignement global sur les séquences u et v après avoir supprimé le plus grand facteur commun sélectionné. La complexité temporelle globale du procédé itéré est alors cubique en la longueur $\max(|u|, |v|)$.

La plus longue sous-séquence commune n'est pas un facteur

Nous envisageons maintenant le cas où la plus longue sous-séquence commune n'est pas un facteur (ou un quasi-facteur). C'est notamment le cas si $u = \cdots\alpha\cdots\beta_1\cdots\beta_2\cdots\cdots\beta_k$ et $v = \cdots\beta_1\cdots\beta_2\cdots\cdots\beta_k\cdots\alpha\cdots$ avec α le plus grand facteur commun et $\beta_1, \beta_2, \cdots, \beta_k$ des petits facteurs communs tels que $\sum_{1 \leq i \leq k} |\beta_i| > |\alpha|$: la plus longue sous-séquence commune est $\beta_1\beta_2\cdots\beta_k$. L'existence du facteur commun α n'est donc pas reportée.

5.4 Alignement local

Comme vu précédemment, les méthodes d'alignement global s'avèrent difficilement utilisables lorsque du code transposé est présent entre les deux unités comparées. Une idée est donc de privilégier non pas la recherche de la plus longue sous-séquence commune mais des quasi-facteurs communs les plus intéressants entre les deux sous-séquences de lexèmes.

5.4.1 Algorithme de Smith-Waterman

Nous souhaitons introduire une métrique de similarité pour les sous-séquences locales intéressantes. Dans cette optique, nous utilisons la méthode de Smith-Waterman [47]. Nous utilisons un coût positif ($\mathcal{C}_{\text{match}}$) pour l'opération de correspondance (extension de correspondance) tandis que des coûts négatifs sont adoptés pour l'ajout, la suppression ($\mathcal{C}_{\text{add}} = \mathcal{C}_{\text{del}}$) et le remplacement ($\mathcal{C}_{\text{sub}}$).

Afin de ne pas prolonger inutilement des sous-séquences qui présenteraient une similarité inintéressante, $\omega \leq 0$ est fixé comme borne minimale de la métrique de similarité. Nous obtenons alors les relations de récurrences suivantes pour le calcul de la matrice de programmation dynamique :

$$s[i][j] = s(u[1..i], v[1..j]) = \begin{cases} \omega & \text{si } (i,j) = (0,0) \\ s(u[1..i-1], v[1..j-1]) + \mathcal{C}_{\text{match}} & \text{si } u[i] = u[j] \\ \max \left(\begin{array}{l} \omega \\ s(u[1..i-1], v[1..j]) + \mathcal{C}_{\text{add}} \\ s(u[1..i], v[1..j-1]) + \mathcal{C}_{\text{del}} \\ s(u[1..i-1], v[1..j-1]) + \mathcal{C}_{\text{sub}} \end{array} \right) & \text{sinon} \end{cases}$$

La recherche de sous-séquences locales utiles est ensuite réalisée en localisant les cellules (i, j) présentant la métrique de similarité la plus élevée non-nulle : une sous-séquence correspondante peut alors être trouvée par détermination du chemin des opérations d'édition par remontée dans la matrice, jusqu'à l'atteinte d'une cellule de similarité ω (ω correspondant au coût d'ouverture d'une sous-séquence).

La principale difficulté réside alors dans la détermination des valeurs de similarité localement maximales $s(u[1..i], v[1..j])$ dans la matrice telles que ces cellules ne soient pas dominées. Il y a domination d'une cellule lorsque celle-ci appartient à un chemin d'opérations élémentaires comportant une cellule de valeur de similarité supérieure. Deux types de domination sont à distinguer :

- La post-domination intervient lorsque la cellule (i, j) est suivie par une cellule (i', j') avec $i' > i$ et $j' > j$: dans ce cas, la sous-séquence peut être étendue jusqu'à (i', j') pour obtenir une meilleure valeur de similarité.
- La pré-domination survient lorsque la cellule (i, j) est précédée par une cellule (i', j') avec $i' < i$ et $j' < j$: la sous-séquence doit alors être réduite à (i', j').

Une méthode peut alors consister à récupérer les chemins d'opérations élémentaires des cellules non-dominées de plus forte similarité à la cellule de plus faible similarité. Ceci ne nous garantit cependant pas que les sous-séquences trouvées ne se chevauchent pas.

Réitération Un moyen de localiser les sous-séquences de similarité maximale non-chevauchantes peut consister à réitérer à chaque fois l'application de l'algorithme de recherche sur les zones ne participant pas à la sous-séquence trouvée. Ainsi si nous recherchons les sous-séquences de $u[1..m]$ et $v[1..n]$, nous déterminons en premier lieu la cellule (i, j) de similarité maximale M : ceci peut être réalisé en espace linéaire. Ensuite avec la méthode de Hirschberg, nous déterminons un chemin d'opération d'éditions (parmi tous les existants) menant de (i, j) à une cellule de similarité ω (notée (i_0, j_0)) : ce chemin ne doit pas comporter de similarité M ; si cela est le cas, la sous-séquence est close à cette cellule. Nous obtenons finalement une sous-séquence extraite des facteurs $u[i_0..i]$ et $v[j_0..j]$. Il reste alors à réitérer la recherche de la sous-séquence de similarité maximale sur les facteurs $u[1..i_0 - 1], u[i + 1..m]$ et $v[1..j_0 - 1]$, $v[j + 1..n]$, soit 4 paires à comparer. Globalement, cette méthode se révèle de complexité temporelle en $O(\max(m, n)^3)$. Une méthode [49], de coût temporel plus avantageux mais nécessitant le stockage intégral de la matrice, consiste à remplacer les valeurs des cellules du chemin de la sous-séquence trouvée par ω et à recalculer uniquement les $O((i - i_0)^2)$ cellules — en bas et à droite du début du chemin (i_0, j_0) — affectées par ce changement. Pour K alignements locaux intéressants de longueur moyenne L relevés, la complexité temporelle s'élève alors en $O(nm + KL^2)$.

5.4.2 Algorithme amélioré avec coupure

Les sous-séquences obtenues par la méthode précédemment présentée sont certes de similarité maximale, mais pour des besoins pratiques de recherche de similarité sur des séquences de lexèmes, nous tolérons la présence d'ajout, de suppression et de remplacement de lexèmes isolés mais pas de longues séquences. Pour répondre à cette problématique, [67] propose de casser le suivi d'une sous-séquence dans la matrice de similarité lorsque nous constatons que la cellule en cours est largement pré-dominée par une autre cellule ; la sous-séquence est alors close par cette cellule pré-dominante. En pratique, il s'agit de déterminer pour chaque cellule (i, j) la similarité de la cellule pré-dominante en utilisant une matrice auxiliaire m définie ainsi :

$$m[i][j] = \begin{cases} \omega & \text{si } s[i][j] = 0 \\ \max(m[i-1][j-1], s[i-1][j-1]) & \text{si } u[i] = u[j] \\ \max_{(p,q)\in\mathcal{P}((i,j))}(m[p][q], s[p][q]) & \text{sinon} \end{cases}$$

avec $\mathcal{P}((i, j))$ les cellules prédécesseures de (i, j). Cette matrice est utilisée pour calculer une valeur de similarité révisée s' réinitialisée à ω pour couper la sous-séquence suivie lorsque la cellule courante est trop largement pré-dominée, i.e. $m - s \geq t_c$ où t_c est le seuil de coupure fixé. Ce seuil peut éventuellement être variable en étant une fonction croissante de s : nous sommes alors plus tolérants pour la coupure lorsque la sous-séquence suivie est déjà de similarité importante. Une coupure systématique peut également être imposée lors de la fin d'une unité syntaxique (telle qu'une fonction) afin d'éviter l'obtention de sous-séquences chevauchantes entre unités.

5.4.3 Raccordement de facteurs par matrice *dotplot*

La matrice *dotplot* d'une paire de séquences de lexèmes (u, v) de longueurs respectives m et n est définie comme la matrice de booléens de dimensions m et n dont la cellule de coordonnées

(i, j) spécifie si $u[i] = v[j]$. Elle est couramment utilisée pour la représentation visuelle de similarités (comme décrit en 14.1.2).

Plutôt que de calculer la globalité de la matrice de similarité (dans la plupart des cas assez creuse) pour déterminer ensuite les sous-séquences en correspondance les plus intéressantes, il peut être plus judicieux de déterminer dans un premier temps l'ensemble des facteurs répétés maximaux entre les deux séquences de lexèmes analysées u et v puis de rechercher ensuite des raccordements (représentant des *fossés* de non-correspondance). Cette approche utilisée par [79] et [81] se révèle moins coûteuse en présence d'un petit nombre de facteurs correspondants. La recherche des k occurrences de facteurs répétés maximaux peut être menée en temps $O(m + n + k)$ en utilisant une structure d'indexation de suffixes[2] (voir chapitre 6).

Il est nécessaire ensuite de raccorder des paires de clones exacts (visualisés par une diagonale sur la matrice *dotplot*) qui sont proches entre-eux. À cet effet, nous pouvons décider d'imposer un critère de distance d'édition pour le fossé de raccordement inférieur à un seuil t_g fixé. On détermine ensuite pour chaque paire de clones exacts (c_1, c_2) les paires de clones exacts les plus proches (dont la tête est à une distance inférieure à t_g de la queue de (c_1, c_2)) : ceci peut être réalisé par l'utilisation d'un index sur les têtes de clones. Nous obtenons ainsi finalement un graphe acyclique de clones exacts reliés entre-eux par des fossés matérialisés par des arêtes. Les sous-séquences communes peuvent ensuite être extraites par le parcours de ce graphe avec filtrage de sous-séquences de similarité trop faible (comportant des fossés trop importants) et éventuelle détermination de sous-séquences non-chevauchantes.

Nous présentons en figure 5.2 un exemple de raccordement de clones exacts : ici les facteurs a, b, c et d représentent des correspondances exactes sur u et v. Alors que $u = abcd$, v présente une transposition entre b et c. Deux possibilités de raccordement sont proposées : en pratique, puisque $|c| < |b|$, le raccordement permettant l'obtention de la correspondance approchée utilisant la sous-séquence abd sera utilisée car minimisant les distances de raccordement.

5.5 Extension aux alignements sur les arbres

5.5.1 Approches algorithmiques

Les méthodes d'alignement étudiées précédemment peuvent être généralisables pour des séquences à n dimensions, i.e. comportant n types de relations. En particulier, nous nous intéressons aux séquences à 2 dimensions que sont les arbres à nœuds étiquetés et ordonnés (dotés de deux types de relation que sont les relations de fratrie et de parenté). Nous pouvons ainsi définir des coûts pour l'ajout, la suppression et le remplacement d'un frère ou alors d'un enfant : nous recherchons ensuite une métrique de similarité ou une métrique de distance quantifiant l'ensemble des opérations d'édition. En conservant la matrice de programmation dynamique (de dimension 4 pour la comparaison d'arbres), il est possible de reconstituer une séquence optimale d'opérations d'édition pour transformer un arbre en l'autre. La figure 5.3 présente sur une petit exemple une séquence d'édition optimale pour transformer un arbre en un autre.

[2] [81] propose une méthode moins efficace où les paires de clones sont déterminés par les diagonales de la matrice *dotplot*, ce qui nécessite mn comparaisons de lexèmes. [79] utilise CCFinder [71] pour la recherche de clones exacts par arbre de suffixes.

$$u = \cdots (a_1 a_2 \cdots a_w)(b_1 b_2 \cdots b_x)(c_1 c_2 \cdots c_y)(d_1 d_2 \cdots d_z) \cdots$$
$$v = \cdots (a_1 a_2 \cdots a_w)(c_1 c_2 \cdots c_y)(b_1 b_2 \cdots b_x)(d_1 d_2 \cdots d_z) \cdots$$

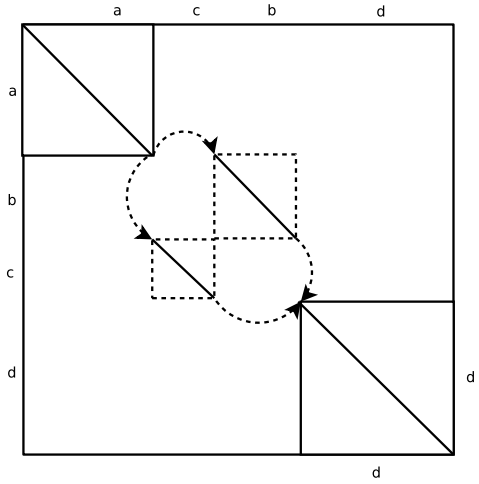
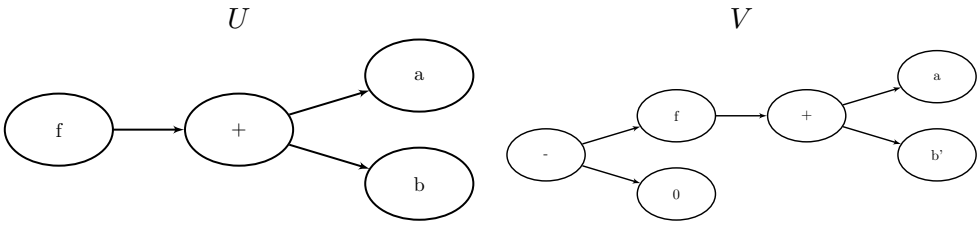


FIG. 5.2 – Raccordement de clones exacts proches sur matrice *dotplot* avec transposition d'un petit morceau de code



Distance d'édition : $s(U, V) = \mathcal{C}_{\text{add}}(-) + \mathcal{C}_{\text{add}}(0) + \mathcal{C}_{\text{sub}}(b, b')$

FIG. 5.3 – Distance d'édition sur deux arbres syntaxiques d'expression

Nous notons que la comparaison d'arbres dont les enfants d'un nœud ne sont pas ordonnés — ce qui peut être utile pour des structures à enfants commutatifs (opérateurs commutatifs, classes avec fonctions d'ordre indifférent) — est un problème NP-complet dans le cas général nécessitant la comparaison avec tous les ordonnancements possibles.

Si nous souhaitons comparer deux forêts d'arbres (chaînes d'arbres ordonnés) $\mathcal{F}_1$ et $\mathcal{F}_2$, nous pouvons décomposer ces forêts de deux manières en isolant l'arbre de gauche ou l'arbre de droite :

$$\mathcal{F}_i = \underbrace{\gamma_i(A_i) \cdot F_i}_{\text{décomposition gauche}} \quad \text{ou} \quad \underbrace{F_i \cdot \gamma_i(A_i)}_{\text{décomposition droite}}$$

À la suite du choix d'une décomposition, la formule de récurrence suivante est utilisée pour calculer la valeur de similarité entre les forêts $\mathcal{F}_1$ et $\mathcal{F}_2$ (nous notons $F_i - \beta$ la forêt F_i dont le nœud β a été supprimé et remplacé par ses enfants directs et $F_i - \beta(A)$ la forêt dont l'arbre $\beta(A)$ a été supprimé) :

$$s(\mathcal{F}_1, \mathcal{F}_2) = \min \begin{cases} s(\gamma_1(A_1), \gamma_2(A_2)) + s(\mathcal{F}_1 - \gamma_1(A_1), \mathcal{F}_2 - \gamma_2(A_2)) \\ s(\mathcal{F}_1 - \gamma_1, \mathcal{F}_2) + \mathcal{C}_{add} \\ s(\mathcal{F}_1, \mathcal{F}_2 - \gamma_2) + \mathcal{C}_{del} \end{cases}$$

Lorsque les deux forêts comparés se limitent à deux arbres (à une racine) $\gamma_1(A_1)$ et $\gamma_2(A_2)$, nous utilisons la récurrence suivante :

$$s\left(\gamma_1(A_1), \gamma_2(A_2)\right) = \min \begin{cases} s(A_1, A_2) + \mathcal{C}_{match/sub}(\gamma_1, \gamma_2) \\ s(\gamma_1(A_1), A_2) + \mathcal{C}_{del}(\gamma_1) \\ s(A_1, \gamma_2(A_2)) + \mathcal{C}_{add}(\gamma_2) \end{cases}$$

Décomposer les forêts sur la gauche ou la droite entraîne un nombre d'appels récursifs variable pour le calcul des valeurs de similarité. Ainsi en utilisant une décomposition systématique sur la gauche ou sur la droite, nous obtenons l'algorithme utilisé par Zang et Shasha [50] dont la complexité temporelle est en :
$\Theta\left(\prod_{i\in\{1,2\}} |F_i| \min\left(\text{hauteur}(F_i), \text{feuilles}(F_i)\right)\right)$.
Une amélioration introduite par Klein [45] consiste à comparer la taille des arbres à gauche (L_1) et à droite (R_1) afin de réaliser une décomposition sur la gauche si $|L_1| < |R_1|$, sur la droite dans le cas contraire. Cette stratégie de décomposition permet de réduire la complexité temporelle en $\Theta(|F_1| \log |F_1| + |F_2|^2)$ avec une complexité mémorielle en $\Theta(|F_1||F_2|)$. Des stratégies de décomposition encore plus efficaces [48, 43] peuvent être employées afin d'améliorer la complexité temporelle.

5.5.2 Applications

La détermination de la distance d'édition entre deux arbres, de par son importante complexité, ne peut raisonnablement pas être utilisée directement pour la comparaison systématisée de grands arbres de syntaxe (comportant typiquement plusieurs milliers de nœuds). En revanche, pour la comparaison de structures syntaxiques de taille limitée, celle-ci peut comporter un intérêt.

Le chapitre 9 est consacré aux méthodes de hachage sur des arbres de syntaxes. Nous y examinons différentes vues abstraites des arbres de syntaxe conduisant à l'obtention de valeurs de hachage dégradées. Parmi les abstractions envisageables figure le remplacement de tous les petits sous-arbres par un nœud unique. Cette abstraction permet d'attribuer une valeur de hachage unique à des sous-arbres différant par le contenu de leurs petits sous-arbres. Des couples de sous-arbres de même abstraction peuvent ensuite être comparés de façon plus approfondie par la détermination de la séquence d'éditions élémentaires et de la similarité afférente permettant de transformer l'un en l'autre. Cette technique est notamment utilisée par l'outil CloneDr [63, 91].