Alignements optimaux

Julie Parreaux

2018-2019

Référence du développement : Crochemore [2, p.224]

Leçons où on présente le développement : 907 (Algorithme du texte) ; 931 (Schémas algorithmiques).
Leçons où on peut l'évoquer : 921 (Recherche).

1 Introduction

Les distances d'éditions permettent de donner la similitude entre deux mots. Connaître cette distance permet aux correcteurs orthographiques de proposer des corrections en cas de mots mal orthographier (on va plus loin que juste savoir si le mot existe ou non) ou en bio-informatique elle permet de mettre en évidence des mutations probables dans un génome sur deux individus distincts.

Calculer une distance d'édition revient souvent à calculer l'alignement optimal. En effet, une distance d'édition est le coût minimal d'une transformation d'un mot à l'autre. Le calcul de l'alignement optimal peut également chercher un motif avec à une distance d'au plus k.

Remarques sur le développement

Ce développement permet de présenter la construction d'un algorithme de programmation dynamique : il est donc important d'en faire ressortir les apports d'un tel paradigme ainsi que sa structure.

1. Présentation du problème et limites du graphe d'édition (à l'oral).
2. Présentation de la structure du problème.
3. Preuve de l'équation récursive solution.
4. Écriture de l'algorithme final.

2 Calcul des alignements optimaux

Cadre : Soient $x, y \in \Sigma^*$ de longueurs respectives m et n.
Objectif : calculer un alignement optimal (qui n'est pas unique) afin de calculer $Dist(x, y)$.

Le graphe d'édition Le graphe d'édition traduit l'ensemble des choix qui décrit l'alignement (Figure 1, faire le dessin).

Définition ((On la donne à l'oral))**.** Le graphe d'édition est composé des sommets correspondant à l'ensembles des paires de préfixes des deux mots dont on calcul l'alignement optimal. On passe alors d'un état vers un autre à l'aide d'une opération d'édition et son coût pondère le graphe.

(Un chemin de l'origine jusqu'à un nœud traduit l'alignement entre les deux préfixes.) Tout chemin de $(-1, -1)$ vers $(m-1, n-1)$ dans ce graphe est un alignement entre x et y. Le calcul d'un alignement optimal ou de $Dist$ se ramène au calcul d'un chemin de coût optimal sur le graphe d'édition G. Comme G est acyclique, on a besoin d'un seul passage pour le calculer (on met un ordre topologique). Cependant le calcul de ce graphe et son parcours peuvent être long, on utilise alors la programmation dynamique afin de réduire ces complexités (limites).

Utilisation de la programmation dynamique afin de calculer ce graphe On va alors donner un algorithme sous le paradigme de la programmation afin de calculer le graphe et le plus court chemin à partir du sommet origine correspondant à la paire des mots vide.
Notation : On note la manipulation des chaînes de caractères comme suit

— x_i est le préfixe de taille $i+1$ du mot x ;
— x_{-1} est le mot vide (correspondant au préfixe de taille 0) ;
— $x[i]$ est le i$^{\text{ième}}$ caractère de x ;
— $\Sigma_{u,v}$ est l'ensemble des suites d'opérations permettant de passer de u à v avec $u, v \in \Sigma^*$.

Sous-structure optimale Ce problème possède une sous-structure optimale : l'alignement pour les paires des préfixes (ce sont les sommets du graphe). Si σ est une suite optimale d'alignements pour les préfixes x_n, y_m. Alors, on peut écrire $\sigma = \sigma'.a$ où $a\{Ins, Del, Sub\}$. De plus, σ' est optimale. On peut alors écrire l'équation récursive du problème.

Équation récursive On pose T un tableau représentant le coût d'un chemin minimal de $(-1,-1)$ à (i,j) dans le graphe d'édition à l'aide d'un tableau de taille $(m+1)\times(n+1)$. On définie T tel que pour tout $i \in [\![0, m-1]\!]$ et pour tout $j \in [\![0, n-1]\!]$, on a $T[i,j] = Dist(x_i, y_j)$. **Remarque** : Les indices de T commence à -1 pour encoder le mot vide.
Remarque : On peut montrer que le calcul de $T[i,j]$ ne dépend que de trois valeurs : $T[i-1, j-1]$, $T[i-1, j]$ et $T[i, j-1]$.

Proposition. *Pour tout $i \in [\![0, m-1]\!]$ et pour tout $j \in [\![0, n-1]\!]$, on a :*

1. $T[-1,-1] = 0$
2. $$T[i,j] = \min \begin{cases} T[i-1, j-1] + Sub(x[i], y[j]) \\ T[i-1, j] + Del(x[i]) \\ T[i, j-1] + Ins(y[j]) \end{cases}$$

Démonstration. Soit $i \in [\![0, m-1]\!]$ et pour tout $j \in [\![0, n-1]\!]$, on a :

1. Par définition (sur le mot vide, on ne fait aucune opération donc on a un coût nul).
2. (Dans le cas du développement, on n'a pas le temps de présenté cette égalité) Par définition, $T[i,-1] = Dist(x_i, \epsilon)$. La suite minimale qui transforme le mot $x_i x[i]$ en mot vide se termine nécessairement par la suppression de $x[i]$. Le reste de la suite transforme x_{i-1} en mot vide. On a :
$$\begin{aligned} Dist(x_i, \epsilon) &= \min\{\text{coût } \sigma \ ; \ \sigma \in \Sigma_{x_i, \epsilon}\} \\ &= \min\{\text{coût } \sigma'.(x[i], \epsilon) \ ; \ \sigma' \in \Sigma_{x_{i-1}, \epsilon}\} \\ &= \min\{\text{coût } \sigma' \ ; \ \sigma' \in \Sigma_{x_{i-1}, \epsilon}\} + Del(x[i]) \\ &= Dist(x_{i-1}, \epsilon) + Del(x[i]) \end{aligned}$$
3. On raisonne de même.
4. On distingue les cas selon la dernière opération de la suite minimale σ.

Cas 1 La dernière opération de la suite minimale σ est une substitution entre $x[i]$ et $y[j]$. Dans ce cas, on peut écrire $\sigma = \sigma'.(x[i], y[j])$ avec σ' qui reste minimale (sinon on obtient une contradiction car σ n'est pas minimale, argument de l'optimalité de la sous-structure).

$$\begin{aligned} Dist(x_i, y_j) &= \min\{\text{coût } \sigma \ ; \ \sigma \in \Sigma_{x_i, y_j}\} \\ &= \min\{\text{coût } \sigma'.(x[i], y[j]) \ ; \ \sigma' \in \Sigma_{x_{i-1}, y_{j-1}}\} \\ &= \min\{\text{coût } \sigma' \ ; \ \sigma' \in \Sigma_{x_{i-1}, y_{j-1}}\} + Sub(x[i], y[j]) \\ &= Dist(x_{i-1}, y_{j-1}) + Sub(x[i], y[j]) \end{aligned}$$

Cas 2 et 3 On raisonne de manière analogue si la dernière opération est une insertion ou une suppression.

□

Remarque : Stratification des problèmes du plus petit au plus grand.

Algorithm 1 Algorithme calculant un alignement optimal.

```
1:  function CALCUL-ALIGNEMENT(x, y)
2:      m ← longueur(x) ; n ← longueur(m)
3:      T un tableau de taille m × n + 1                    ▷ Graphe
4:      T[−1, −1] ← 0
5:      for i = 0 à m − 1 do                ▷ Calcul la passage de x à ϵ
6:          T[i, −1] ← T[i − 1, −1] + Del(x[i])
7:      end for
8:      for j = 0 à n − 1 do
9:          T[−1, j] ← T[−1, j − 1] + Ins(y[j]) ▷ Calcul de ϵ à y
10:         for i = 0 à m − 1 do                ▷ Traitement de x à y
11:             T[i, j] ← T[i − 1, j − 1] + Sub(x[i], y[j])
12:             if T[i, j] > T[i, j − 1] + Ins(y[j]) then
13:                 T[i, j] ← T[−1, j − 1] + Ins(y[j])
14:             end if
15:             if T[i, j] > T[i − 1, j] + Del(x[i]) then
16:                 T[i, j] ← T[i − 1, j] + Del(x[i])
17:             end if
18:         end for
19:     end for
20:     Renvoie T
21: end function
```

Calcul d'une valeur de la solution optimale L'algorithme 1 suivant nous permet de calculer une valeur de la solution optimale par méthode ascendante.

L'algorithme est correct par la proposition précédente.

Complexité : en temps et en espace $O(mn)$. On peut faire mieux en conservant juste les trois dernières valeurs pour T (les seules nécessaires à calculer).

Construire une solution optimale On peut vouloir calculer l'alignement optimale en exhibant le chemin optimal dans le graphe. Pour cela, on peut utiliser un tableau *Direction* qui contient le nœud précédent de notre nœud courant. En remontant ce tableau, on obtient le chemin souhaité. On n'est pas toujours obligé de stocker tout le tableau, on peut le faire en $O(m+n)$.

3 Autours des principes de la programmation dynamique

Nous allons présenter la programmation dynamique et ses principes [1, p.333]. La programmation dynamique s'inscrit dans le but de résoudre les limites du paradigme diviser pour régner dans le cas d'une structure "infinie". En effet, elle s'applique à des problèmes possédant des sous-structures optimales dont les sous-problèmes qui lui sont liés se chevauche. On a une belle application du principe compromis temps-espace.

Définition (Paradigme de la programmation dynamique)**.** Écrire un algorithme sous le paradigme de la programmation dynamique se réalise en quatre étapes.

1. Caractériser la structure optimale dans notre problème. Un problème exhibe une sous-structure optimale si une solution optimale au problème contient en elle des solutions optimales de sous-problèmes. Un alignement optimale pour un préfixe de taille n contient un alignement optimal du préfixe de taille $n-1$. Pour exhiber cette sous-structure optimale on suit la méthode suivante :
   (a) Exhiber le choix nécessaire pour trouver la solution du problème (*Ins*, *Del* et *Sub*).
   (b) On suppose connaître une solution optimale que nous possédons.
   (c) On donne alors les sous-problèmes qui en découlent et on décrit comment caractériser au mieux ce sous-espace (les alignements optimaux pour les préfixes de taille inférieure).
   (d) Montrer que les sous-solutions sont optimales (raisonnement par l'absurde).

2. Définir récursivement la valeur de cette structure optimale. Pour cela, on calcul le graphe des sous-problème qui nous permet d'établir l'ordre du calcul des différents sous-problèmes.

   Définition. Le graphe des sous-problèmes est un graphe orienté contenant chacun des sous-problèmes et leur dépendance.

3. Calcul de la valeur d'une solution optimale, généralement de manière ascendante. On a deux approches possibles.

   Mémoïsation procédure écrite de manière récursive classique dont le résultat de chaque sous-problème est stocké dans un tableau. On commence par vérifier si on a déjà résolut un problème.

   Ascendante on résout les problèmes du plus petit au plus gros en utilisant les résultats précédemment calculés.

4. Construction d'une solution optimale (si nécessaire).

Une solution exponentielle d'un problème peut alors devenir polynomiale si et seulement si les sous-problèmes distincts impliqués sont en nombre polynomial et peuvent être résolus en temps polynomial.

4 Autours des distances d'éditions

Les distances d'éditions permettent de donner la similitude entre deux mots. On définit ici ce que sont ces distances d'éditions et quelques une de leur propriétés.

Distance d'édition On peut définir plusieurs distances sur les mots (plus ou moins pratique) et plusieurs distances d'éditions. Nous allons en étudier quelques unes. Nous allons commencer par définir des distances théoriques (elles n'ont aucune utilité en pratique mais peuvent servir pour des raisonnements).

— La distance préfixe : $\forall u, v \in \Sigma^*, d_{pref} = |u| + |v| - 2|lcp(u, v)|$ où $lcp(u, v)$ est le plus long préfixe commun de u et v.

— La distance suffixe : $\forall u, v \in \Sigma^*, d_{suff} = |u| + |v| - 2|lcs(u, v)|$ où $lcs(u, v)$ est le plus long suffixe commun de u et v.

— La distance facteur : $\forall u, v \in \Sigma^*, d_{fact} = |u| + |v| - 2|lcp(u, v)|$ où $lcf(u, v)$ est le plus long facteur commun de u et v.

Une distance plus importante et surtout utilisé en pratique est la distance de Hamming. Elle n'est pas très expressive : il y a une hypothèse forte sur la longueur des mots et elle nous donne juste un nombre et non quelles sont les modifications pour arriver à ce nombre. Cependant, elle reste la plus facile à calculer (linéaire en la taille du mot), c'est probablement pour cela qu'elle reste si utilisée.

Définition. La distance de Hamming sur deux mots de la même longueur donne le nombre de différences de lettres entre les deux mots.

Exemple : Si $u = aab$ et $v = abb$, la distance de Hamming appliquée à u et à v est deux.

La distance d'édition est une distance définie sur toutes les paires de mots. De plus, contrairement à la distance de Hamming, on est capable de donner une explication du passage de x à y. Pour cela, nous commençons par définir trois opérations élémentaires munies de leurs coûts.

— La substitution qui positionne a à la place de b dans x à une position donnée. On note sa fonction coût $Sub(a, b)$.

— L'insertion qui insère a dans x à une position donnée. On note sa fonction coût $Ins(a)$.

— La suppression qui supprime a d'une position donnée dans x. On note sa fonction coût $Del(a)$.

Définition. La distance d'édition de Leveinstein est une fonction $Dist$ définie telle que :

$$Dist(x, y) = \min\{\text{coût de } \sigma : \sigma \in \Sigma_{x,y}\}$$

où $\Sigma_{x,y}$ est une suite d'opérations élémentaires permettant de transformer x en y où son coût est la somme des coûts de ces opérations élémentaires.

Remarque. La distance de Hamming est une distance d'édition où $Del(a) = Ins(a) = +\infty$.

Proposition. *Dist est une distance définie sur* Σ^* *(au sens mathématiques) si et seulement si Sub est une distance sur* Σ *et si* $Del(a) = Ins(a) > 0, \forall a \in \Sigma$.

Démonstration. $\Rightarrow$ Supposons que *Dist* est une distance.

- $\forall a, b \in \Sigma, Dist(a, b) = Sub(a, b)$: *Sub* est une distance.
- $Del(a) = Dist(a, \epsilon) = Dist(\epsilon, a) = Ins(a)$: $Del(a) = Ins(a) > 0$.

$\Leftarrow$ Montrons que *Dist* est une distance : on montre qu'elle possède les quatre propriétés.

- Positivité : $Sub \geq 0$ (*Sub* distance), $Del, Ins > 0$ (hypothèse) : leur somme ≥ 0.
- Séparation : si $u = v$, $Dist(u, v) = 0$ (*Sub* distance). Réciproquement, si $Dist(u, v) = 0$, $u = v$ (*Sub* est la seule fonction qui s'annule).
- Symétrie : la fonction *Sub* est symétrique (*Sub* distance) et les fonctions *Ins* et *Del* sont équivalentes : *Dist* est symétrique (suite minimale est donnée par lecture inverse).
- Inégalité triangulaire : raisonnement par l'absurde, on suppose que $\exists w \in \Sigma^*$ tel que $Dist(u, w) + Dist(w, v) < Dist(u, v)$. On a donc une suite minimale de u à w puis une de w à v de coût inférieur à une suite minimale de u à v.

□

Problèmes pour le calcul : Il n'existe pas d'unicité du calcul et cela reste un problème d'optimisation.

Alignement L'alignement est une façon de visualiser leur similitudes. Lors du calcul de la distance d'édition (*Dist*), nous calculons un alignement optimal. Un alignement entre deux mots x et y est un mot $z \in (\Sigma \cup \{\epsilon\}) \times (\Sigma \cup \{\epsilon\}) \setminus \{(\epsilon, \epsilon)\}$ dont la projection sur la première composante est x et la projection sur la seconde est y.

Exemple : Voici un alignement (qui est même optimal).

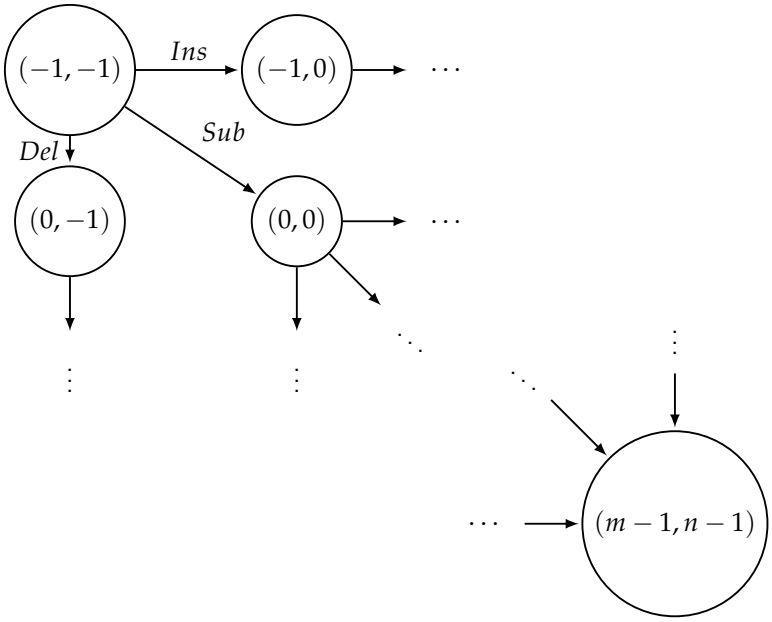
$$\begin{pmatrix} A & C & G & - & - & A \\ A & C & G & C & T & A \end{pmatrix}$$

Le nombre d'alignements entre deux mots est exponentiel. On donne maintenant une borne sur ce nombre.

Proposition. *Soient* $x, y \in \Sigma^*$ *de longueurs respectives m et n avec* $m \leq n$. *Le nombre d'alignement entre x et y ne contenant pas de suppression consécutives de lettres de x est* $\binom{2n+1}{m}$.

Démonstration. Un alignement est caractérisé par les substitutions aux n positions sur y ainsi que les suppressions ($n + 1$ emplacements possibles (en comptant un emplacement avant $y[0]$ et un après $y[n-1]$)) qui le compose. Un alignement est donc un choix de m substitutions ou suppressions parmi $2n + 1$ emplacements possibles. □

Graphe d'édition Le graphe d'édition traduit l'ensemble des choix qui forment l'alignement (Figure 1).



Le graphe d'édition est composé des sommets correspondant à l'ensembles des paires de préfixes des deux mots (un chemin de l'origine jusqu'à un nœud traduit la transformation d'un préfixe vers le second). On passe alors d'un état vers un autre à l'aide d'une opération. Tout chemin de $(-1,-1)$ vers $(m-1,n-1)$ dans ce graphe est un alignement entre x et y.

FIGURE 1 – Un exemple partiel de graphe d'édition.

Le calcul d'un alignement optimal ou de *Dist* se ramène au calcul d'un chemin de coût optimal sur le graphe d'édition G. Comme G est acyclique, on a besoin d'un seul passage pour le calculer (on met un ordre topologique). On utilise alors la programmation dynamique afin de calculer le graphe et d'en faire le parcours.

Références

[1] Rivest R. Stein C. Cormen T., Leiserson C. *Algorithmique, 3ème édition*. Dunod, 2010.

[2] L. Lecroq M. Crochemore, C. Hancart. *Algorithmique du texte*. Vuibert, 2001.