



Algorithmique des chaînes – M2 ATAL Recherche approchée de motifs

Irena.Rusu@univ-nantes.fr LS2N, bât. 34, bureau 303 tél. 02.51.12.58.16



Plan

- Les distances
- Recherche selon la distance de Hamming algorithme Kangourou
- Recherche selon la distance de Levenshtein





Algorithmique des chaînes – M2 ATAL Recherche approchée de motifs

Les distances





Terminologie (rappels ?)

Alignement de deux mots x et y sur un alphabet Σ
mise en correspondance de chaque caractère de x et y avec soit un
caractère de l'autre mot soit le caractère « - » (n'appartenant pas à Σ) de
sorte que la précédence des caractères dans chaque mot (x et y) soit
gardée par la correspondance.

Exemple:
$$x = abcbabc$$
 un alignement $-abc-babc$ $y = cabbc$ c $ab-b-c$ -

Opérations : insertion, suppression (délétion), substitution (aussi appelées différences, ou mismatches)





« Distances » entre deux mots x et y

- Distance de Hamming : nombre de substitutions dans l'alignement sans « - » de x et y, qui ont la de la même longueur
- Nombre de différences : le nombre minimum d'insertions, suppressions et substitutions dans un alignement de x et y.
- Distance de Levenshtein, ou d'édition : en supposant que chaque opération a un coût spécifique, le coût minimum d'un alignement de x et y.





« Distances » entre deux mots x et y mathématiquement parlant

- Distance de Hamming : C'est une distance.
- Nombre de différences: C'est une distance.
- Distance de Levenshtein, ou d'édition : c'est une distance si et seulement si :
 - Le coût Sub(a,b) d'une substitution est une distance
 - Les coûts Del(a) et Ins(a) de la délétion et insertion d'un élément sont égaux.





Algorithmique des chaînes – M2 ATAL Recherche approchée de motifs

Recherche de motifs selon la distance de Hamming



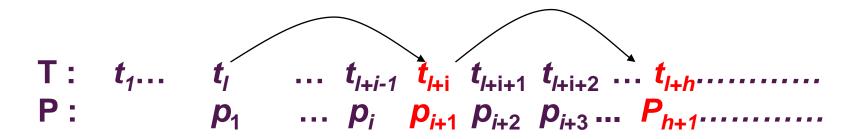
Le problème

Problème

Entrée : texte T de taille n, motif P de taille m, seuil k

Sortie : toutes les sous-séquences de T à distance de Hamming au plus k par rapport à P

Idée: l'algorithme Kangourou (Landau, Vishkin, 1986)

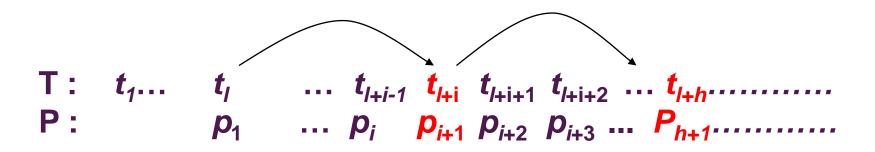






Recherche de motif avec différences

Idée: l'algorithme Kangourou (Landau, Vishkin, 1986)



Si P[1..i] =T[1..l+i-1] mais P[i+1] \neq T[l+i] alors

- * compter une différence, et si nb. différences ≤ k
- * chercher à aligner P[i+2..m] et T[l+i+1..n], qui sont tous les deux des suffixes → utiliser l'arbre des suffixes



Algorithme Kangourou (P, T)

Algorithme Kangourou

Entrée : texte T, motif P, entier k

Sortie : occurrences de P dans T à k erreurs près

```
pour I ← 1 à n-k+1 faire
  q \leftarrow 0; i \leftarrow -1;
  tant que (q≤k) et (i<m) faire
      s← plus long préfixe commun de T[l+i+1..n] et P[i+2..m]
                                             Complexité:
      j← |s|;
                                                    O(|T|.k.f(n,m))
      si i+j+1 < m alors q←q+1 fin si
      si i+j+1=m alors Afficher I fin si
                                             où f(n,m) est la complexité
      i← i+j+1;
                                             du calcul du plus long
  fin tant que
                                             préfixe
fin pour
```





Arbre des suffixes et LCA (least – or lowest - common ancestor)

- A arbre des suffixes (généralisé) pour P# et T\$
- Chercher le plus long préfixe de P[a..m] et T[b..n] :
 - Trouver l'ancêtre commun le plus proche (LCA) des deux suffixes dans A
 - Récupérer la longueur de la séquence lui correspondant dans A

Lemme

- Peut se faire en O(1), pour tous P, T, a, b
- ... avec un pré-traitement en O(m+n) de A

Donc complexité de l'algorithme Kangourou : O(|T|k+|P|)





Algorithmique des chaînes – M2 ATAL Recherche approchée de motifs

Recherche selon la distance de Levenshtein

- Rappels (?) sur l'alignement et la programmation dynamique
- Algorithme pour la recherche de motifs





Alignement et distance de Levenshtein

Alignement global

Entrée : deux textes T1, T2

Sortie: un alignement de T1 et T2 qui minimise la distance de

Levenshtein.

Alignement local

Entrée : texte T (taille n), motif P (taille m), entier k

Sortie: les occurrences P' de P dans T telles que la distance de

Levenshtein de P à P' est ≤ k.

Convention pour ce cours : P'= occurrence approchée





Qu'est-ce qui change par rapport à recherche exacte/distance de Hamming?

- Tailles de P et T « extensibles », donc beaucoup de possibilités d'aligner P à une position dans T
- L'approche par fenêtre glissante (KMP,BM) ne permet pas de faire des sauts lorsqu'il y a des erreurs
- Les erreurs sont difficilement gérables avec l'arbre des suffixes
- → Programmation dynamique
 - * guidée mathématiquement (par une formule)
 - * « extension » de P ou de T à chaque étape





Programmation dynamique – principes

- En général, problèmes d'optimisation (min/max d'un coût)
- « Diviser pour régner », mais :
 - Séparer la recherche du coût min/max de celle de l'« objet » produisant le min/max (c.à.d. de la solution elle-même)
 - Exprimer le min/max du problème en combinant des solutions de sousproblèmes, sous forme récursive
 - Résoudre les sous-problèmes les plus faciles d'abord, et garder leurs solutions dans un tableau
 - Garder de l'information permettant de construire à la fin la solution.
 - Programmation itérative, pas récursive





Le problème que nous allons résoudre

Recherche de motif approché, selon dist. Levenshtein

Entrée : texte T (taille n), motif P (taille m), entier k

Sortie: les principales occurrences P' de P dans T telles

que la distance de Levenshtein de P à P' est ≤ k.

Remarques:

- Chercher toutes les occurrences n'est pas forcément intéressant (à chaque position, garder plutôt la/les meilleure(s))
- En plus, elles peuvent être nombreuses → algorithmes forcément plus lourds





Relation de récurrence (1)

Soient

- Σ'=Σ∪{ } l'alphabet étendu avec l'espace « »
- c(x,y) le coût de l'alignement de x avec y
- V(i,j) = le coût de l'alignement de P[1..i] et T[1..j]
 (dans cet ordre, càd T sur la ligne en-dessous la ligne de P)

Alors

V(0,j) = 0 (à la différence de l'alignement global)

$$V(i,0) = \sum_{h=1,...,i} c(P[h],-)$$

(ça ne coûte pas de glisser P le long de T, mais ça coute d'ajouter des espaces à T)



Relation de récurrence (2)

En général (i,j>0)

$$V(i,j) = \min \begin{cases} V(i-1,j) + c(P[i],-) & \text{// d\'el\'etion (de P par rapport \`a T)} \\ V(i,j-1) + c(-,T[j]) & \text{// insertion (dans T par rapport \`a P)} \\ V(i-1,j-1) + c(P[i],T[j]) & \text{// substitution} \end{cases}$$

(trois possibilités d'aligner P[1..i] et T[1..j], soit l'une des dernières positions avec -, soit les dernières positions ensemble ; la partie « avant » étant calculée de manière optimale par récursivité)

Note. formule sur la valeur optimale; l'alignement lui-même ne guide pas la recherche



Etapes suivantes

- Avec la formule, on remplit la table V(i,j)
 - 1ère ligne et colonne d'abord (formules avec i=0 et j=0)
 - Ensuite, ligne par ligne, de gauche à droite
- On garde la trace des choix effectués selon le min
- On retrouve l'alignement utilisant ces traces
- pas d'algorithme récursif

Exemple. P=noel, T= cannelle, k=1 coût(insertion/délétion)=2, coût(substitution)=1



Remplissage de la table V(i,j)

Algorithme CalculV (P,T)

```
V(0,0) \leftarrow 0
 pour j \leftarrow 1 à n faire V(0,j) \leftarrow 0 fin pour
 pour i←1 à m faire
     V(i,0) \leftarrow V(i-1,0) + c(P[i],-); dir(i,0) \leftarrow \ll \uparrow \gg;
      pour j ← 1 à n faire
          V(i,j) \leftarrow \min\{V(i-1,j)+c(P[i],-), V(i-1,j-1)+c(P[i],T[j]), V(i,j-1)+c(-,T[j])\}
          si V(i,j)= V(i-1,j)+c(P[i],-) alors marquer (i,j) avec «↑»
          sinon si V(i,j)= V(i-1,j-1)+c(P[i],T[j]) alors marquer (i,j) avec «√»
                 sinon si V(i,j)=V(i,j-1)+c(-,T[j]) alors
                             marguer (i,i) avec \leftarrow »
                        fin si fin si fin si
        fin pour
fin pour
```





Calcul des occurrences approchées

Lemme.

Il existe une occurrence approchée de P qui finit à la position p si et seulement si V(m,p)≤k. Dans ce cas, il existe un chemin (m,p), ... (1,h), (0, k') dans la table, indiquant une occurrence approchée T[h..p] de P, de coût V(m,p).

Remarques

- plusieurs occurrences peuvent finir à la même position dans T
- on ne les trouve pas toutes (certaines sont masquées par des occurrences plus proches)
- → trouver seulement les occurrences les plus courtes





Calcul des occurrences approchées les plus courtes

Pour tout p t.q. V(m,p)≤k faire

Tracer un chemin de la cellule (m,p) à une cellule (0,k') en préférant les pointeurs « \uparrow » aux « \nwarrow » et aux « \leftarrow » (la préférence est déjà inscrite dans l'algorithme) :

- Pointeur «↑» de (i,j) vers (i-1,j) : « » dans T face à P[i] (délétion)
- Pointeur «[►] » de (i,j) vers (i-1,j-1) : T[j] aligné avec P[i] (substitution)
- Pointeur « ← » de (i,j) vers (i,j-1): T[j] face à « » dans P (insertion)



Cas particuliers et complexité

- Distance dite « nombre de différences » : rien ne change
- Distance de Hamming (ou « k mismatches »):

```
coût insertion/délétion : + ∞ (ou juste n+1)
```

coût substitution: 1

Complexité:

O(|T||P|) (à comparer avec O(|T|k+|P|) pour Hamming)





Conclusions sur la recherche approchée

Complexités logiquement croissantes avec la difficulté

- Hamming: O(|T|k+|P|)
- Nombre de différences : O(|T|k) en moyenne, O(|T||P|) au pire
- Levenshtein: O(|T||P|)

Méthodes:

- Arbre des suffixes seulement pour la distance de Hamming
- Programmation dynamique quasi-incontournable