

# Quelques éléments d'algorithmique du texte

---

[felipe@iro.umontreal.ca](mailto:felipe@iro.umontreal.ca)

**RALI**

Dept. Informatique et Recherche Opérationnelle  
Université de **Montréal**



V0.5

Last compiled: 24 janvier 2018



- ▶ Pour une très bonne exposition aux algorithmes du texte, lire :  
*[Crochemore et al., 2001]*.

# Plan

Distances classiques entre formes

Applications

Fonctions de Hachage



# Plan

Distances classiques entre formes

Applications

Fonctions de Hachage



# Distance entre deux mots

- ▶ Une fonction  $d : \Sigma^* \times \Sigma^* \rightarrow \mathcal{R}$  est une **distance** sur  $\Sigma^*$  si pour tout  $u, v \in \Sigma^*$  on a :
  - ▶  $d(u, v) \geq 0$
  - ▶  $d(u, v) = 0 \iff u = v$
  - ▶  $d(u, v) = d(v, u)$
  - ▶  $d(u, v) \leq d(u, w) + d(w, v) \quad \forall w \in \Sigma^*$
  
- ▶ Si l'une de ces propriétés n'est pas vérifiée, on parle plutôt de **mesure de dissimilarité**

# Distance de Hamming

- ▶ Définie pour deux mots  $u$  et  $v$  de même longueur par le nombre de positions  $p$  où  $u_p \neq v_p$
  
- ▶ Exemples :
  - ▶  $d_h(\text{ahuri,aluni}) = 2$
  - ▶  $d_h(\text{aluni,ahuri}) = 2$
  - ▶  $d_h(\text{salade,salace}) = 1$
  - ▶  $d_h(\text{salace,sagace}) = 1$
  - ▶  $d_h(\text{salade,sagace}) = 2$

# Distance de Jaro-Winkler

- ▶ Adaptation par William E. Winkler (1999) d'une distance proposée par Matthew A. Jaro (1989)
  - ▶ Adaptée pour les chaînes courtes (noms propres, mots de passe, etc.)
  - ▶ Pas une distance (pas symétrique)
- ↪ Mesure de similarité entre 0 (absence de similarité) et 1
- ▶ Exemples :
    - ▶  $d_{jw}(\text{intimité, inimité}) = 0.96$
    - ▶  $d_{jw}(\text{opinion, opignon}) = 0.95$
    - ▶  $d_{jw}(\text{bernard, bertrand}) = 0.92$
    - ▶  $d_{jw}(\text{lagacé, leveillé}) = 0.57$

# Distance de Jaro

$$d_j(x, y) = \frac{1}{3} \left( \frac{m}{|x|} + \frac{m}{|y|} + \frac{m - t}{m} \right)$$

Où :

- ▶  $m$  est le nombre de caractères qui se **correspondent** dans  $x$  et  $y$  :
  - ▶ **Def** :  $x_i$  et  $y_j$  se correspondent **ssi** :
    - $x_i = y_j$
    - $|i - j| \leq \rho \equiv \left\lfloor \frac{\max(|x|, |y|)}{2} \right\rfloor - 1$
- ▶  $t$  est le nombre de **transpositions**, cad : le nombre de caractères qui diffèrent lorsqu'on dresse la **séquence** des caractères de chaque mot qui ont un correspondant dans l'autre chaîne, divisé par 2



## Distance de Jaro

		0	1	2	3	4	5	6
		B	E	R	N	A	R	D
0	B	1						
1	E		1					
2	R			1				
3	T							
4	R			1				
5	A					1		
6	N				1			
7	D							1

BERTRAND → B E R R A N D  
 BERNARD → B E R N A D

- ▶  $\rho = 3$ , matchs = 7, transpositions = 3 ( $m = 7$ ,  $t = 3/2 = 1.5$ )
- ▶  $d_j(\text{BERTRAND}, \text{BERNARD}) = \frac{1}{3} \left( \frac{7}{7} + \frac{7}{8} + \frac{5.5}{7} \right) = 0.89$

# Distance de Jaro-Winkler

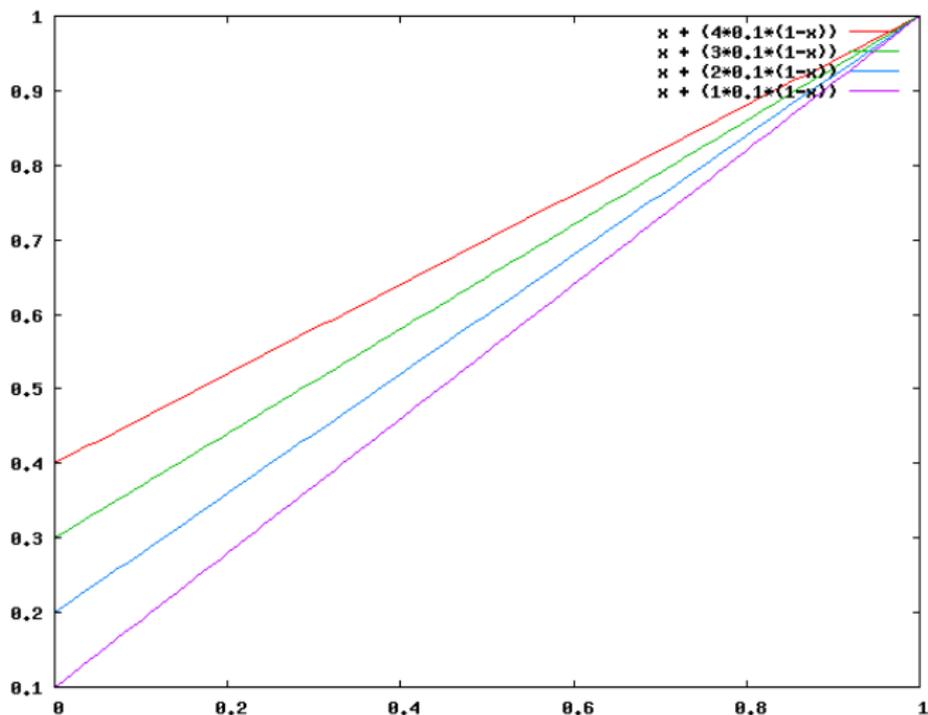
- ▶ Soit  $l$  la taille du plus grand **préfixe commun** entre les deux chaînes ( $l \leq 4$ )
- ▶ Soit  $p$  un coefficient ( $p = 0.1$ ) :

$$d_{jw}(x, y) = d_j(x, y) + (l \times p \times (1 - d_j(x, y)))$$

**intuition** : favoriser les mots qui partagent un "grand" préfixe

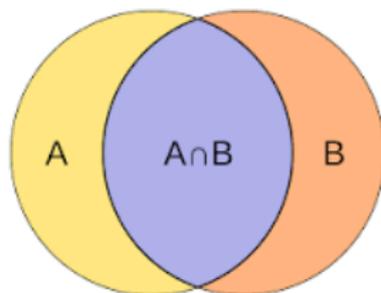
- ▶  $l = 3$  dans notre exemple (BERNARD, BERTRAND)
- ▶  $d_{jw}(\text{BERTRAND}, \text{BERNARD}) = 0.92$

## Distance de Jaro-Winkler



- ▶ importance de la taille du préfixe entre les deux chaînes
- ▶  $x$  est ici  $d_j$

# Dissimilarité de Jaccard



- ▶ indice de Jaccard :  $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$
- ▶ dissimilarité de Jaccard :  $1 - J(A, b)$
- ▶ appliqué aux chaînes en considérant A et B comme des multiset (sacs de caractères), où dans son format binaire :

ch	hi	ie	en	ni	ic	he
1	1	1	1			
1				1	1	1

$$J(\text{chien}, \text{niche}) = 4 / 7$$

# Distance cosinus

- ▶  $\cos(A, B) = \frac{A \cdot B}{\|A\|_2 \cdot \|B\|_2}$
- ▶ on peut utiliser une représentation binaire  
 $a_i = 1$  si le symbole associé dans l'alphabet est présent dans A
- ▶ ou utiliser une mesure comme **tf-idf** :
  - ▶  $tf_{\alpha, A}$  : fréquence du symbole  $\alpha$  dans la forme A  
 $\log(tf_{\alpha, A} + 1)$
  - ▶  $idf_{\alpha}$  : inverse du nombre de formes ayant le symbole  $\alpha$   
 $\log(idf_{\alpha})$
  - ▶ **tf.idf** = produit des deux

## Distance d'édition entre $x$ et $y$

substitution substitution d'une lettre ( $l_1$ ) de  $x$  par une lettre de ( $l_2$ )  $y$ ; soit  $\text{sub}(l_1, l_2)$  son coût

**Ex :**  $x = \text{narine} \Rightarrow y = \text{marine}$  en substituant  $n$  à  $m$  dans  $x$  en position 1 dans  $x$

insertion insertion d'une lettre ( $l$ ) de  $y$  dans  $x$ ; soit  $\text{ins}(l)$  son coût

**Ex :**  $x = \text{marine} \Rightarrow y = \text{martine}$  en insérant  $t$  en position 4 dans  $x$

suppression opération inverse à la précédente,  $\text{del}(l)$  son coût

- ▶ Le nombre **minimum d'opérations** à appliquer pour transformer une chaîne en une autre est appelée la **distance de Levenshtein** [Levenshtein, 1966], ou encore la distance d'édition (**edit distance**).

$d_L(\text{voiture}, \text{moteur}) ?$ 

- ▶ Séquence de 4 opérations qui transforme la chaîne **voiture** en la chaîne **moteur**.

$x$	opération	coût
VOITURE	del(E)	1
VOITUR	sub(T,E)	1
VOIEUR	sub(I,T)	1
VOTEUR	sub(V,M)	1
MOTEUR		

Chaque opération, autre que  $sub(a,a)$  a un coût unitaire.

- ▶ Le coût de cette séquence de transformations est 4.
- ▶ Peut-on faire mieux ?

$$d_L(\text{voiture}, \text{moteur}) ?$$

- ▶ Séquence de 4 opérations qui transforme la chaîne **voiture** en la chaîne **moteur**.

$x$	opération	coût
VOITURE	del(E)	1
VOITUR	sub(T,E)	1
VOIEUR	sub(I,T)	1
VOTEUR	sub(V,M)	1
MOTEUR		

Chaque opération, autre que  $sub(a,a)$  a un coût unitaire.

- ▶ Le coût de cette séquence de transformations est 4.
- ▶ Peut-on faire mieux ?
- ▶ Réponse en calculant une **table d'édition**

$d_L(\text{voiture}, \text{moteur}) ?$ 

- $T[i, j] = d_L(x[1 \dots i], y[1 \dots j])$

	(6)	M	O	T	E	U	R
(7)		(1,d)	(2,d)	(3,d)	(4,d)	(5,d)	(6,d)
V	(1,i)	(1,s)	(2,d)	(3,d)	(4,d)	(5,d)	(6,d)
O	(2,i)	(2,i)	(1,=)	(2,d)	(3,d)	(4,d)	(5,d)
I	(3,i)	(3,i)	(2,i)	(2,s)	(3,d)	(4,d)	(5,d)
T	(4,i)	(4,i)	(3,i)	(2,=)	(3,d)	(4,d)	(5,d)
U	(5,i)	(5,i)	(4,i)	(3,i)	(3,s)	(3,=)	(4,d)
R	(6,i)	(6,i)	(5,i)	(4,i)	(4,i)	(4,i)	(3,=)
E	(7,i)	(7,i)	(6,i)	(5,i)	(4,=)	(5,d)	(4,i)

- $d$ (éléction)<sup>1</sup>,  $i$ (nsertion),  $s$ (ubstitution),  $=$  (égalité)

1. Je sais c'est une horreur...

# $d_L(\text{voiture}, \text{moteur}) ?$

- ▶ On peut démontrer que :

$$T[i, j] = \min \begin{cases} T[i-1, j-1] + \text{sub}(x[i], y[j]) \\ T[i-1, j] + \text{ins}(x[i]) \\ T[i, j-1] + \text{del}(y[j]) \end{cases}$$

- ▶ Cas particuliers aux bornes :

$$\begin{aligned} T[0, 0] &= 0 \\ T[i, 0] &= T[i-1, 0] + \text{ins}(x[i]) \\ T[0, j] &= T[0, j-1] + \text{del}(y[j]) \end{aligned}$$

$$d_L(\text{voiture}, \text{moteur}) ?$$

**Require:**  $X = x_1, \dots, x_N$  et  $Y = y_1, \dots, y_M$

**function**  $Solve(i, j)$

**if**  $i = 0$  **then**

**if**  $j = 0$  **then**

**return** retourne 0

**else**

**return**  $Solve(0, j - 1) + del(Y[j])$

**else**

**if**  $j = 0$  **then**

**return**  $Solve(i - 1, 0) + ins(X[i])$

**else**

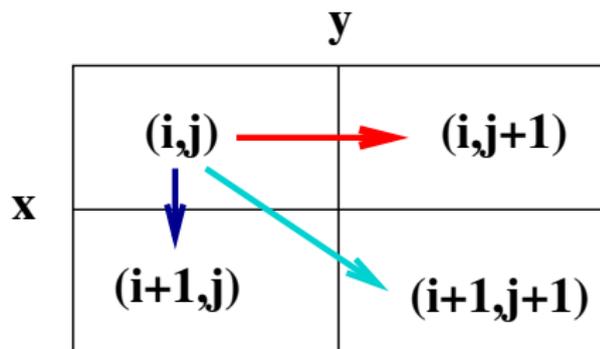
$i_1 \leftarrow Solve(i - 1, j - 1) + sub(X[i], Y[j])$

$i_2 \leftarrow Solve(i - 1, j) + ins(X[i])$

$i_3 \leftarrow Solve(i, j - 1) + del(Y[j])$

**return**  $min(i_1, i_2, i_3)$



$d_L(\text{voiture}, \text{moteur}) ?$ 


**suppression de  $y[j+1]$**

**substitution de  $x[i+1]$  par  $y[j+1]$**

**insertion de  $x[i+1]$**

$d_L(\text{voiture}, \text{moteur}) ?$

// init au bornes

for  $i \leftarrow 0$  à  $I$  do

$T[i][0] \leftarrow i$

for  $j \leftarrow 1$  à  $J$  do

$T[0][j] \leftarrow j$

// boucle principale

for  $i \leftarrow 1$  à  $I$  do

for  $j \leftarrow 1$  à  $J$  do

$$T[i][j] = \min \begin{cases} T[i-1, j-1] & + \text{sub}(x[i], y[j]) \\ T[i-1, j] & + \text{ins}(x[i]) \\ T[i, j-1] & + \text{del}(y[j]) \end{cases}$$

//Le résultat est dans  $T[I][J]$



$$d_L(\textit{voiture}, \textit{moteur}) ?$$

$$i \leftarrow I, j \leftarrow J, a \leftarrow \{\}$$

```

while (i > 0) || (j > 0) do
  if T[i][j-1]+del(y[j]) == T[i][j] then
    a ← a ∪ del(y[j])
    j ← j-1
  else if T[i-1][j]+ins(x[i]) == T[i][j] then
    a ← a ∪ ins(x[i])
    i ← i-1
  else
    a ← a ∪ sub(x[i],y[j])
    i ← i-1, j ← j-1

```

- ▶ L'ordre des tests influe sur le meilleur alignement retourné lorsqu'il y en a plusieurs
- ▶ On peut éviter cet algorithme en mémorisant dans chaque cellule de la table l'opération (2 bits sont nécessaires).

$d_L(\text{voiture}, \text{moteur}) ?$ 

		-1	0	1	2	3	4	5
			M	O	T	E	U	R
-1		0	1	2	3	4	5	6
0	V	1	1	2	3	4	5	6
1	O	2	2	1	2	3	4	5
2	I	3	3	2	2	3	4	5
3	T	4	4	3	2	3	4	5
4	U	5	5	4	3	3	3	4
5	R	6	6	5	4	4	4	3
6	E	7	7	6	5	4	5	4

$d_L(\text{voiture}, \text{moteur}) ?$ 

- Les deux alignements de la figure précédente (chacun de coût minimal 4) :

V	O	I	T	U	R	E
M	O	T	E	U	R	
S	=	S	S	=	=	I

V	O	I	T	U	R	E
M	O		T	E	U	R
S	=	D	=	I	=	I

- Le second alignement correspond à la table de l'acétate 16
- On peut représenter l'ensemble des alignements par un automate

## Raffinements

[Ukkonen, 1985]

- ▶ Il existe un algo qui étant donné  $x_1^m$ , et  $y = y_1^n$  et  $t$  un seuil, teste si  $D(x, y) \leq t$  en un temps  $O(t \cdot \min(m, n))$  et en mémoire :  $O(\min(t, n, m))$ .
- ▶ en énumérant en ordre croissant les valeurs de  $t$ , on a un algo qui calcule la distance d'édition  $e$  en un temps  $O(e \cdot \min(m, n))$  et une mémoire  $O(\min(e, m, n))$  (si l'alignement n'est pas requis)
- ▶ si les coûts sont unitaires, il existe des raffinements supplémentaires



# Damerau-Levenshtein

// init au bornes

for i ← 0 à I do

$T[i][0] \leftarrow i$

for j ← 1 à J do

$T[0][j] \leftarrow j$

// boucle principale

for i ← 1 à I do

  for j ← 1 à J do

$$T[i][j] = \min \begin{cases} T[i-1, j-1] & + \text{sub}(x[i], y[j]) \\ T[i-1, j] & + \text{ins}(x[i]) \\ T[i, j-1] & + \text{del}(y[j]) \end{cases}$$

if  $i \geq 2$  and  $j \geq 2$  and  $x[i] == y[j-1]$  and  $x[i-1] == y[j]$  then

$$t[i][j] = \min \begin{cases} t[i][j] \\ t[i-2][j-2] + \text{cost\_transposition} \end{cases}$$

//Le résultat est dans  $T[I][J]$

# Distance de Needleman-Wunsch

- ▶ plus de 6000 citations à **[Needleman and Wunsch, 1970]** !
- ▶ matrice de similarité *sim* de taille  $\Sigma \times \Sigma$
- ▶ recherche les alignements de score maximal de manière analogue à la distance d'édition, en donnant une **pénalité de trou** identique  $d$  à chaque insertion ou suppression.
- ▶ la récurrence :

$$F_{ij} = \max \begin{cases} F_{i-1,j-1} + \text{sim}(x_i, y_j) \\ F_{i,j-1} + d \\ F_{i-1,j} + d \end{cases}$$

$$F_{0j} = F_{i0}, \quad \forall i, j$$

# Distance de Needleman-Wunsch

- avec  $d = -5$  et  $sim$  :

	A	G	C	T
A	10	-1	-3	-4
G	-1	7	-5	-3
C	-3	-5	9	0
T	-4	-3	0	8

- l'alignement suivant de `AGACTAGTTAC` avec `CGAGACGT` a un coût de 1 :

A	G	A	C	T	A	G	T	T	A	C
C	G	A	-	-	-	G	A	C	G	T
-3	7	10	-5	-5	-5	7	-4	0	-1	0

- Est-ce un alignement optimal (cad de similarité maximale) ?

# Plus longue sous-chaine commune à $X$ et $Y$

- ▶ Soit  $X = x_1 \dots x_N$  et  $Y = y_1 \dots y_M$ 
  - ▶  $Z = z_1 \dots z_K$  est une sous-chaine de  $X$ , notée  $Z = SC(X)$ , ssi il existe  $\rho : [1, K] \rightarrow [1, N]$  tel que :  
 $Z = x_{\rho(1)} \dots x_{\rho(K)}$  avec  $\rho(i) > \rho(j), \forall i > j$
  - ▶  $Z$  est une sous-chaine commune à  $X$  et  $Y$ , notée  $Z = SCC(X, Y)$  ssi  $Z=SC(Y)$  et  $Z=SC(X)$
  
- ▶ **Exemple :**  $X = ACGATCCACGT, Y = AGCTACGT$ 
  - ▶  $AAA, CT, ACGT$  sont des sous-chaines de  $X$ ,
  - ▶  $AA, AGACT$  sont des sous-chaines communes à  $X$  et  $Y$

**Idée :**  $X$  et  $Y$  sont d'autant plus proches qu'elle possèdent une sous-chaine commune qui est longue.



# Plus longue sous-chaine commune à $X$ et $Y$

- ▶ Soit  $X = x_1 \dots x_N$   $Y = y_1 \dots y_M$  et  $Z = z_1 \dots z_K$
- ▶ si  $Z$  est **une** plus longue sous-chaine de  $X$  et  $Y$ ,  
ce que l'on note  $Z = PLSCC(X, Y)$
- ▶ **alors**
  - ▶ **si**  $x_N = y_M$  **alors**  $z_K = x_N$  et  $Z_1^{K-1} = PLSCC(X_1^{N-1}, Y_1^{M-1})$
  - ▶ **sinon**
    - **si**  $z_K \neq x_N$  **alors**  $Z_1^K = PLSCC(X_1^{N-1}, Y)$
    - **si**  $z_K \neq y_M$  **alors**  $Z_1^K = PLSCC(X, Y_1^{M-1})$

# Récurrance

- Soit  $T[i, j]$  la longueur de la plus longue sous-chaine commune à  $X[1...i]$  et  $Y[1...j]$  :

$$T[i, j] = \begin{cases} 0 & \text{si } i * j = 0 \\ T[i - 1, j - 1] + 1 & \text{si } i, j > 0 \text{ et } x_i = y_j \\ \max \begin{cases} T[i - 1, j] \\ T[i, j - 1] \end{cases} & \text{si } i, j > 0 \text{ et } x_i \neq y_j \end{cases}$$

- **Exemple :**

$$\begin{array}{c}
 \overbrace{\hspace{15em}}^{PLSCC(AABB, AAB)=3} \\
 \overbrace{\hspace{10em}}^{PLSCC(AAB, AA)=2} \\
 \overbrace{\hspace{5em}}^{PLSCC(AA, AA)=2} \quad \overbrace{\hspace{5em}}^{PLSCC(AAB, A)=1} \\
 \overbrace{\hspace{3em}}^{PLSCC(A, A)=1} \quad \overbrace{\hspace{3em}}^{PLSCC(AA, A)=1} \\
 1 + \max(1 + 1 + \underbrace{PLSCC(\epsilon, \epsilon)}_0, \max(1 + \underbrace{PLSCC(A, \epsilon)}_0, \underbrace{PLSCC(AAB, \epsilon)}_0))
 \end{array}$$

# Plus longue sous-chaine commune à $X$ et $Y$

**Require:**  $T$  une table  $N \times M$  dont un élément est  $T[i,j] = \langle \text{score}, \text{back pointeur} \rangle$

**Ensure:**  $T[N,M]$  contient la longueur d'une PLSSC de  $X$  et  $Y$

**for**  $i : 0 \rightarrow N$  **do**  $T[i,0] = \langle 0, \downarrow \rangle$

**for**  $j : 1 \rightarrow M$  **do**  $T[0,j] = \langle 0, \leftarrow \rangle$

**for**  $i : 1 \rightarrow N$  **do**

**for**  $j : 1 \rightarrow M$  **do**

**if**  $x_i = y_j$  **then**

$T[i,j] = \langle T[i-1, j-1].\text{score} + 1, \swarrow \rangle$

**else**

$T[i,j] = \langle T[i-1, j].\text{score}, \downarrow \rangle$

**if**  $T[i, j-1].\text{score} > T[i,j].\text{score}$  **then**

$T[i,j] = \langle T[i, j-1].\text{score}, \leftarrow \rangle$



Plus longue sous-chaine commune à  $X$  et  $Y$ 

T	0	↓	1	↓	2	↓	3	↓	4	↓	5	↓	6	↓	7	↓	8	↙
G	0	↓	1	↓	2	↓	3	↓	4	↓	5	↓	6	↓	7	↙	7	←
C	0	↓	1	↓	2	↙	3	↓	4	↙	5	↙	6	↓	6	←	6	←
A	0	↓	1	↙	2	↓	3	↙	4	↓	5	↓	6	↙	6	←	6	←
G	0	↓	1	↓	2	↓	3	↓	4	↓	5	↓	5	←	6	↙	6	←
C	0	↓	1	↓	2	↙	3	↓	4	↙	5	↙	5	←	5	←	5	←
A	0	↓	1	↙	2	↓	3	↙	4	↓	4	←	5	↙	5	←	5	←
C	0	↓	1	↓	2	↙	3	↓	4	↙	4	↙	4	←	4	←	4	←
G	0	↓	1	↓	2	↓	3	↓	3	←	3	←	3	←	4	↙	4	←
G	0	↓	1	↓	2	↓	3	↓	3	←	3	←	3	←	4	↙	4	←
A	0	↓	1	↙	2	↓	3	↙	3	←	3	←	3	↙	3	←	3	←
G	0	↓	1	↓	2	↓	2	←	2	←	2	←	2	←	3	↙	3	←
C	0	↓	1	↓	2	↙	2	←	2	↙	2	↙	2	←	2	←	2	←
A	0	↓	1	↙	1	←	1	←	1	←	1	←	1	↙	1	←	1	←
X	0	↓	0	←	0	←	0	←	0	←	0	←	0	←	0	←	0	←
	Y		A		C		A		C		C		A		G		T	

► Y est **une** PLSCC(X,Y)

# Plan

Distances classiques entre formes

Applications

Fonctions de Hachage

## Regroupement de noms propres

*[Cohen et al., 2003]*

- ▶ identifier des doublons dans des noms propres (auteurs, restaurants, etc.)

philippe's the original, "1001 n. alameda st.", "los angeles", "213/628-3  
philippe the original, "1001 n. alameda st.", "chinatown", "213-628-378

- ▶ datasets : <https://www.cs.utexas.edu/users/ml/riddle/data.html>





# Mesure de la qualité d'un système de traduction

SRC	cependant , il y a ici deux problèmes qui apparaissent
REF	however , there are two problems here .
SMT	however , there are two problems emerging here . (11%)
SRC	les limites des circonscriptions électorales
REF	electoral boundaries
SMT	the electoral boundaries (33%)
SRC	nous sommes fiers de ces habitants de london et d' autres canadiens qui consacrent leur temps et leur énergie à bâtir un monde meilleur .
REF	we are proud of these londoners and of other canadians who devote their time and energies to improving our world .
SMT	we are proud of these people of london and other people spend their time and energy to build a better world (50%)
SRC	quelle plus belle image peut on donner du canada ?
REF	this is canada at its best .
SMT	what more can be nice to canada ? (100%)

## WER

	(7)	this	is	canada	at	its	best	.
(8)	(0,i)	(1,d)	(2,d)	(3,d)	(4,d)	(5,d)	(6,d)	(7,d)
what	(1,i)	(1,s)	(2,d)	(3,d)	(4,d)	(5,d)	(6,d)	(7,d)
more	(2,i)	(2,i)	(2,s)	(3,d)	(4,d)	(5,d)	(6,d)	(7,d)
can	(3,i)	(3,i)	(3,i)	(3,s)	(4,d)	(5,d)	(6,d)	(7,d)
be	(4,i)	(4,i)	(4,i)	(4,i)	(4,s)	(5,d)	(6,d)	(7,d)
nice	(5,i)	(5,i)	(5,i)	(5,i)	(5,i)	(5,s)	(6,d)	(7,d)
to	(6,i)	(6,i)	(6,i)	(6,i)	(6,i)	(6,i)	(6,s)	(7,d)
canada	(7,i)	(7,i)	(7,i)	(6,=)	(7,d)	(7,i)	(7,i)	(7,s)
?	(8,i)	(8,i)	(8,i)	(7,i)	(7,s)	(8,d)	(8,i)	(8,i)

this	is	canada	at	its	best	.	
what	more	can	be	nice	to	canada	?
sub	sub	sub	sub	sub	sub	sub	

# Application à la correction orthographique

- ▶ De nombreux types d'erreurs (qui peuvent être cumulés)
  - ▶ fautes d'orthographe : **éphémaire**, **opignon**, etc.
  - ▶ fautes de frappe : **qviron**, **errueurs**
  - ▶ impropriété (malapropism) : il est **âpre** / Il est **râpe**
  - ▶ mots d'emprunts : tu n'as qu'à **pi-per** la sortie de ta commande avant de faire ton **debugging**
  - ▶ fautes de grammaire : la femme que j'ai **vu**.
  - ▶ erreurs de césure : il est **con fondant**/**confondant**
  - ▶ formes extra-lexicales : **U2**, **OQP**, etc.
  
- ▶ Les transparents qui suivent sont des idées simples qui illustrent comment on peut utiliser un modèle de langue pour faire un mini-correcteur orthographique.

# Application à la correction orthographique

Soit  $\mathcal{L}$  un lexique fini contenant de nombreux mots d'une langue.

- ▶ Identifier dans un texte  $T_1^n$  les mots inconnus de  $\mathcal{L}$  (on écarte ici les formes extra-lexicales chiffrées)
- ▶ Pour chaque mot inconnu  $u$ , réunir les mots de  $\mathcal{L}$  qui lui sont proches au sens d'une métrique à définir.

soit  $v_u$  l'ensemble de ces mots proches de la forme  $u$

- ▶ Si  $h(u)$  désigne l'historique de  $u$  dans le texte, alors corriger  $u$  en  $\hat{u}$  :

$$\hat{u} = \operatorname{argmax}_{v \in v_u} p(v|h(u))$$

j'aime la **aperture**  $\Rightarrow$  j'aime la **friture**  
 j'aime la **aperture**  $\Rightarrow$  j'aime la **aperture**



# Distance d'édition

- ▶ les mots les plus proches de **asfalte** :

asphalte	2	spalter	3	state	3
svelte	3	falce	3	fate	3
faute	3	faîte	3	halte	3

- ▶ Les mots les plus proches de **ammateur** :

armateur	1	amateur	1	aviateur	2
amateurs	2	animateur	2	armateurs	2
orateur	3	radiateur	3	sénateur	3

- ▶ Les mots les plus proches de **courier** :

<b>courier</b>	0	courrier	1	courtier	1
courir	1	courber	1	sourir	2
tourner	2	usurier	2	écourter	2

# Plan

Distances classiques entre formes

Applications

Fonctions de Hachage



# L'algorithme du Soundex

- ▶ Algorithme simple capable d'associer à chaque mot (suite de caractères) un "*code phonétique*"
- ▶ Il existe des soundex pour de nombreuses langues
- ▶ **Idée** : les mots proches phonétiquement ont le même code soundex  
 $\text{soundex}(\text{computer}) = \text{soundex}(\text{camputter}) = \text{C513}$
- ▶ brevet déposé par Robert, R. Russel en 1918
- ▶ nombreuses variantes (ex : metaphone, NYYSIIS)

# Soundex Anglais

- 1 mettre le mot en majuscule, éliminer les ponctuations
- 2 garder la première lettre du mot
- 3 supprimer les occurrences de : A E I O U H W Y
- 4 faire les changements suivants :
  - ▶ B F P V → 1 bilabiales
  - ▶ C G J K Q S X Z → 2 labiodentales
  - ▶ D T → 3 dentales
  - ▶ L → 4 alvéolaires
  - ▶ M N → 5 vélares
  - ▶ R → 6 laryngales
- 5 si deux lettres adjacentes dans la chaîne de départ ont le même code, ne garder que la première de ces lettres (élimination des doublons)
- 6 retourner les 4 premiers caractères (compléter par des zéros le cas échéant)

# Soundex Français

- ▶ Règles de conversion pour le français :

B P	→	1		L	→	4		G J	→	7
C K Q	→	2		M N	→	5		X Z S	→	8
D T	→	3		R	→	6		F V	→	9

- ▶ On peut calculer le soundex du dictionnaire et encoder cela dans une table de hachage multiple de manière à retrouver en temps constant l'ensemble des mots proches d'un mot inconnu.



**Cohen, W. W., Ravikumar, P., and Fienberg, S. E. (2003).**

A comparison of string distance metrics for name-matching tasks.

*In Proceedings of the 2003 International Conference on Information Integration on the Web*, pages 73–78.



**Crochemore, M., Hancart, C., and Lecroq, T. (2001).**  
*Algorithmique du texte.*

Number ISBN-2-7117-8628-5. Vuibert, vuibert informatique edition.



**Levenshtein, V. I. (1966).**

Binary codes capable of correcting deletions, insertions and reversals.

*Sov. Phys. Dokl.*, 6 :707–710.



**Needleman, S. B. and Wunsch, C. D. (1970).**

A general method applicable to the search for similarities in the amino acid sequence of two proteins.

*Journal of molecular biology*, 48(3) :443–453.



**Ukkonen, E. (1985).**

Algorithms for approximate string matching.

*Inf. Control*, 64(1-3) :100–118.