Varias sur les modèles de langue

- Applications classiques des modèles n-grammes: classification de texte, identification de la langue, correction orthographique
- Introduction (très incomplète) à la représentation de lexiques
- Introduction à la programmation dynamique

Pour une très bonne introduction aux algorithmes du texte, lire **?**.

Classification de documents

But: classer un texte selon plusieurs catégories (sport, religion, politique, spam, etc.).

Soit $\mathcal{C}$ l'ensemble des classes possibles et c_i, $i \in [1, |\mathcal{C}|]$ l'une de ces classes et soit T un texte dont on veut connaître la classe c_T, alors

$$\begin{aligned} c_T &= \underset{i \in [1,|\mathcal{C}|]}{\operatorname{argmax}} p(c_i|T) \\ &= \underset{i \in [1,|\mathcal{C}|]}{\operatorname{argmax}} \frac{p(T|c_i) \times p(c_i)}{p(T)} \\ &= \underset{i \in [1,|\mathcal{C}|]}{\operatorname{argmax}} \underbrace{p(T|c_i)}_{\text{langue}} \times \underbrace{p(c_i)}_{\text{a priori}} \end{aligned}$$

↪ **Note:** En pratique, un modèle unigramme donne des performances (étonnamment) raisonnables.

Identification de la langue

But: Découvrir la langue d'un texte T (T_i le i-ème caractère)

- une instance du problème précédant (les classes sont les langues), soit $\mathcal{L}$ l'ensemble des langues, l_i une langue de cet ensemble.
- un modèle n-gramme au niveau des caractères (T_c représente le c-ième caractère de T, T_a^b la séquence $T_a, T_{a+1}, \ldots, T_b$):

$$
\begin{aligned}
l_T &= \operatorname*{argmax}_{i \in [1,|\mathcal{L}|]} p(l_i|T) \\
&= \operatorname*{argmax}_{i \in [1,|\mathcal{L}|]} \underbrace{\prod_{c=1}^{|T|} p(T_c|T_{c-n+1}^{c-1}, l_i)}_{\text{n-car}} \times \underbrace{p(l_i)}_{\text{a priori}}
\end{aligned}
$$

- En pratique, il faut modéliser des paires (langue, encodage).

Jag talar en litten svenska

suédois, cp1252

suédois	cp1252	0.076
suédois	cp850	0.076
suédois	macintosh	0.076
norvégien	cp1252	0.056
norvégien	cp850	0.056
norvégien	macintosh	0.056
danois	cp1252	0.039
danois	cp850	0.039
danois	macintosh	0.039
néerlandais	cp1252	0.034
néerlandais	cp850	0.034
néerlandais	macintosh	0.034
allemand	cp1252	0.023
allemand	cp850	0.023
allemand	macintosh	0.023

estonien	iso-8859-4	0.016
hongrois	cp1250	0.015
hongrois	cp852	0.015
anglais	cp1252	0.014
français	cp1252	0.011
français	cp850	0.011
français	macintosh	0.011
espagnol	cp1252	0.010
espagnol	cp850	0.010
espagnol	macintosh	0.010
albanais	cp1252	0.010
albanais	cp850	0.010
albanais	macintosh	0.010
finnois	cp1252	0.010
finnois	cp850	0.010

Voir la démo http://www-rali.iro.umontreal.ca/SILC/

Mon char est parké au garage

français	cp1252	0.099
allemand	cp1252	0.064
français	cp850	0.036
français	macintosh	0.036

Je parke mon char

néerlandais	cp1252	0.046
néerlandais	cp850	0.046
néerlandais	macintosh	0.046
anglais	cp1252	0.046
allemand	cp1252	0.038

Réaccentuation automatique de textes

Le systeme m'accentue $\longrightarrow$ Le système m'accentue.
Le systeme m'a accentue $\longrightarrow$ Le système m'a accentué.

- Peut être vue comme un cas particulier de la correction orthographique.
- Soit w_1^n la phrase de n mots à réaccentuer, une implantation possible est la suivante:
  - désaccentuer tous les mots w_i,
  - sélectionner pour tout mot w_i les versions accentuées possibles. Soit a_i l'ensemble de ces "variantes" (lorqu'un mot n'est pas accentuable, $a_i = \{w_i\}$),
  - considérer toutes les phrases que l'on peut construire à partir des a_i (en prenant un mot par a_i) et sélectionner celle de plus forte probabilité selon le modèle de langue.
- Voir www.iro-rali.umontreal.ca pour une démonstration de cette méthode.

Correction orthographique

Il existe de nombreux types d'erreurs (qui peuvent êtres cumulés):

- fautes d'orthographe: éphémaire, opignon, etc.
- fautes de frappe: qviron, errueurs
- malaproprisme[1]: il est âpre / Il est râpe
- mots d'emprunts: tu n'as qu'à piper 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.

[1] terme emprunté à Graem Hirst

Correction orthographique

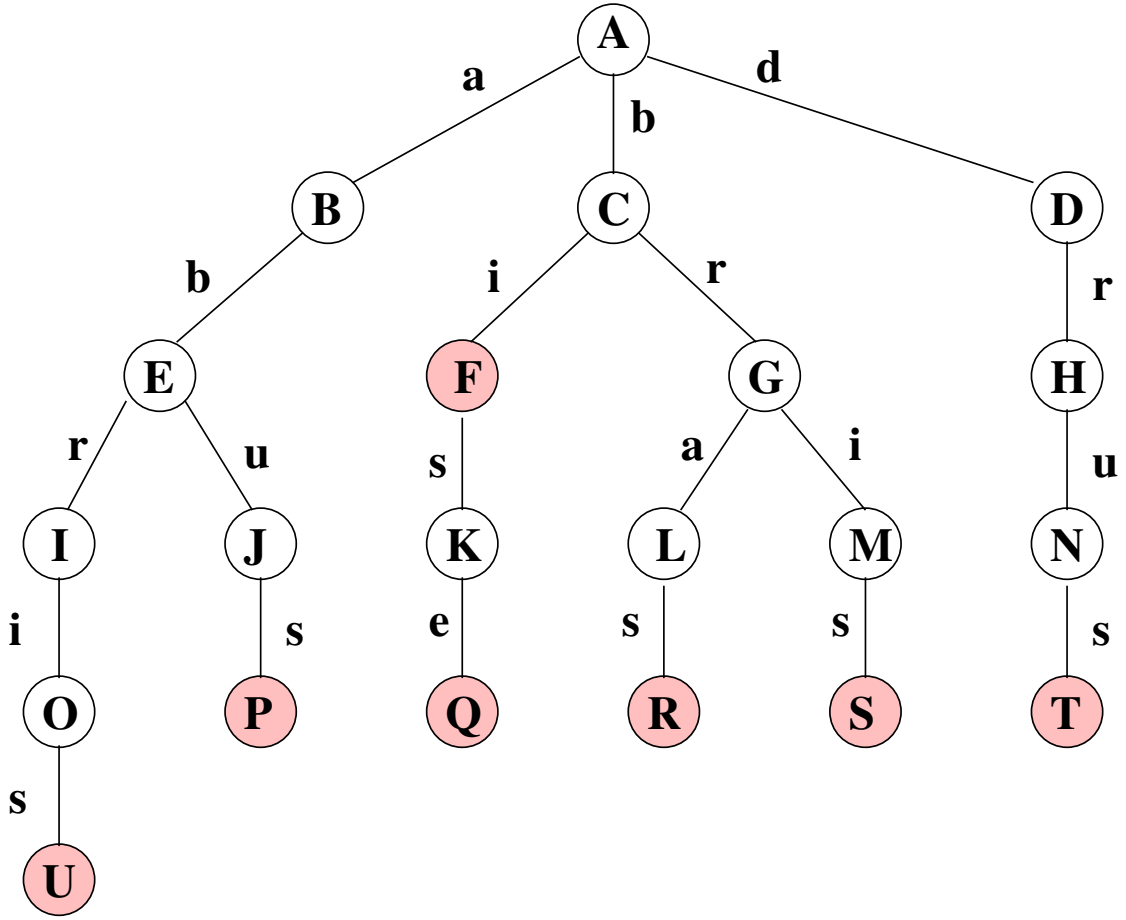
Soit $\mathcal{L}$ un lexique fini contenant de nombreux mots d'une langue.

- 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 L qui lui sont proches au sens d'une métrique à définir (soit v_u l'ensemble de ces mots proches)
- Si $h(u)$ désigne l'historique de u dans le texte, alors corriger u en $\hat{u}$:

$$\hat{u} = \underset{v_u}{\operatorname{argmax}}\, p(v_u|h(u))$$

j'aime la apriture $\Rightarrow$ j'aime la
- aperture
- friture
- aptitude

Représentation du lexique: abris, abus, bi, bise, bras, bris, drus



abris, abus, bi, bise, bras, bris, drus

Que l'on peut représenter sous forme tabulaire:

	1	2	3	4	5	6	7	8	9	10	. . .	21
PF	2	5	6	8	9	11	12	14	15	16	. . .	-
NF	3	1	2	1	2	1	2	1	1	1	. . .	0
S		a	b	d	b	i	r	r	r	u	. . .	s
	A	B	C	D	E	F	G	H	I	J	. . .	U

- $PF[i]$ indique l'indice (colonne) du premier fils du nœud décrit par i,
- $NF[i]$ indique le nombre de fils du nœud décrit par la colonne i,
- $S[i]$ indique le symbole à lire pour arriver sur le nœud i,
- Une convention particulière peut être prise pour coder les états terminaux qui ne sont pas des feuilles ($i.e.$ état F),
- La dernière ligne n'est là que pour faciliter la compréhension de la représentation

Passage d'un arbre à un tableau

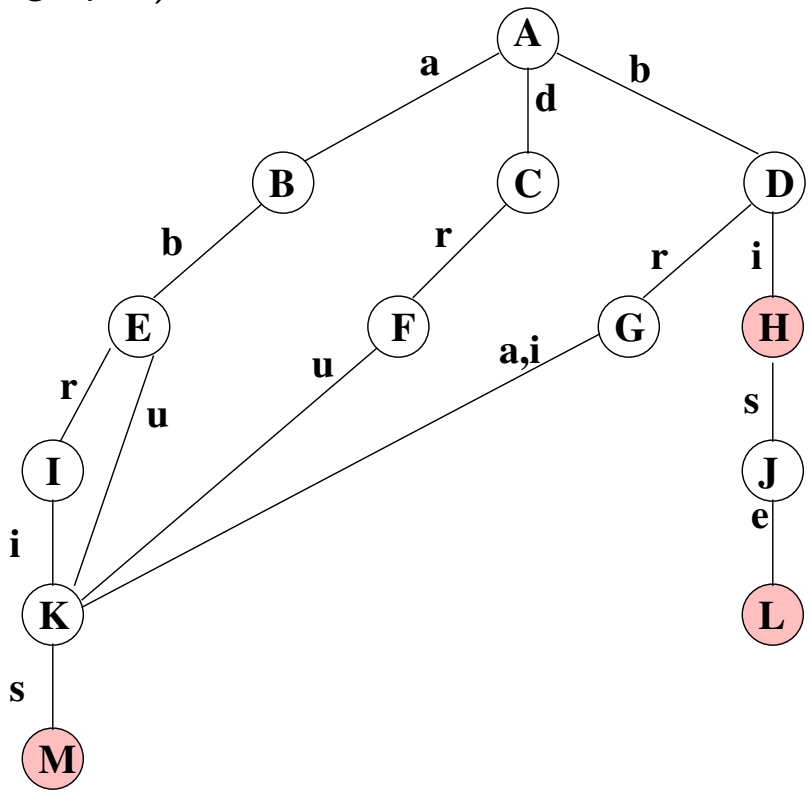
current ← 1, libre ← 2, file ← {}
ENFILE(file,racine(Arbre))

while NOT-EMPTY(file) **do**
  n ← DEFILE(file)
  PF[current] ← libre
  NF[current] ← NOMBRE_FILS(n)
  for i: 0 → NF[current] (exclus) **do**
    S[libre+i] = SYMBOLE(n,i)
    ENFILE(file,FILS(n,i))
  libre ← libre + NF[current]
  current ← current + 1

où `ENFILE` et `DEFILE` sont les fontions classiques d'ajout et de retrait dans une file, et `NOMBRE_FILS(n)` `SYMBOLE(n,i)`, et `FILS(n,i)` sont des fontions d'un nœud n qui retournent respectivement le nombre de fils de n, le ième symbole qui part de n et le ième fils de n.

abris, abus, bi, bise, bras, bris, drus

On peut coder encore plus efficacement un lexique en factorisant également les suffixes communs (on a alors un graphe):



Distance entre deux mots

Définition: une fonction $d : \Sigma^\star \times \Sigma^\star \rightarrow \mathcal{R}$ est une **distance** sur $\Sigma^\star$ si pour tout $u, v \in \Sigma^\star$ on a:

- $d(u, v) \geq 0$
- $d(u, v) = 0 \iff u = v$
- $d(u, v) = d(v, u)$
- $d(u, v) \leq d(u, w) + d(w, v) \quad \forall w \in \Sigma^\star$

Exemple: la distance de Hamming définie pour deux mots u et v de même longueur par le nombre de positions p où $u_p \neq v_p$

d_h(ahuri,aluni) = 2, d_h(aluni,ahuri) = 2
d_h(salade,salace) = 1 , d_h(salace,sagace) = 1
d_h(salade,sagace) = 2

Distance d'édition entre x et y

$\hookrightarrow$ Définie à partir d'opérations d'édition qui permettent de transformer un mot x en un mot y.

subsitution substitution d'une lettre (l_1) de x par une lettre de (l_2) y; soit sub(l_1,l_2) son coût

Ex: x =narine, y =marine, substituer n à m dans x en position 1 dans x transforme x en y

insertion insertion d'une lettre (l) de y dans x; soit ins(l) son coût

Ex: x =marine, y =martine, insérer t en position 4 dans x transforme x en y

suppression opération inverse à la précédente, del(l) son coût

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

$d_L(voiture, moteur)$?

Voici une séquence de 4 opérations qui transforment la chaîne voiture en la chaîne moteur. Chaque opération (autre que $sub(a,a)$) a un coût unitaire.

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

$\hookrightarrow$ Le coût de cette séquence de transformations est 4. Peut-on faire mieux ?

Réponse en calculant la **table d'édition**

Passer de MOTEUR à VOITURE: table d'édition

	(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)

Soit x de longueur m (de 0 à m-1) et y de longueur n (de 0 à n-1)

$T[i,j] = d_L(x[0 \ldots i], y[0 \ldots j])$ pout tout $i \in [-1, m[, j \in [-1, n[$

d (délétion[2]), i (insertion), s (substitution), $=$ (égalité)

[2] Je sais c'est une horreur...

Récurrence

On peut démontrer que:

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

Cas particuliers aux bornes:

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

Solution 1 (descendante)

Require: $X = x_1, \ldots, x_N Y = y_1, \ldots, y_M$

function Solve(n,m)

if $i = 0$ **then**

if $j = 0$ **then**

retourne 0

else

return Solve(0,m-1)

else

if $j = 0$ **then**

return Solve(n-1,0)

else

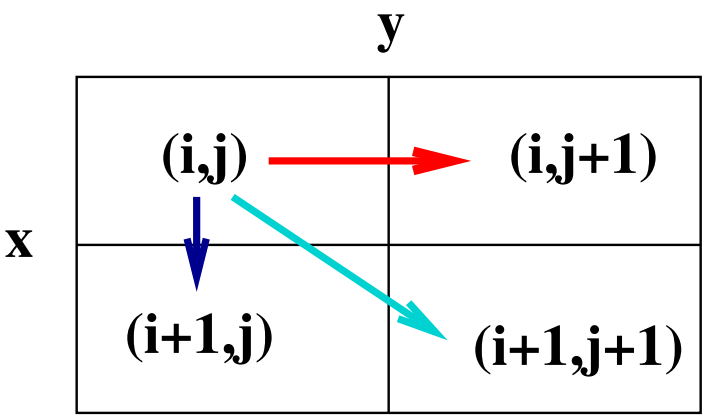
$i_1 \leftarrow Solve(i-1, j-1) + sub(X[i], Y[j])$

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

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

return $min(i_1, i_2, i_3)$

Solution 2 (ascendante): Passer de y à x



suppression de y[j+1]

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

insertion de x[i+1]

Remplissage de la table $\mathcal{O}(n \times m)$

// init au bornes

for i $\leftarrow 0$ à I-1 **do**
  T[i][0] $\leftarrow i$
for j $\leftarrow 0$ à J-1 **do**
  T[0][j] $\leftarrow j$

// boucle principale

for i $\leftarrow 0$ à I-1 **do**
  for j $\leftarrow 0$ à J-1 **do**

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

//Le résultat est dans T[I-1][J-1]

Algorithme de retour d'un meilleur alignement

$i \leftarrow I-1$, $j \leftarrow J-1$, $a \leftarrow \{\}$

while $(i > 0)$ || $(j > 0)$ **do**
  if T[i][j-1]+del(x[i]) == T[i][j] **then**
    a ← a + del(x[i])
    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
- En pratique, on peut éviter cet algorithme si l'on mémorise également dans chaque cellule de T l'opération (2 bits sont nécessaires).

http://www-igm.univ-mlv.fr/~lecroq/seqcomp

		-1	0	1	2	3	4	5
			M	O	T	E	U	R
-1		0	1	2	3	4	5	6
0	Y	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

Énumérer les meilleurs alignements

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 la page
- Si on souhaite faire des traitements sur l'ensemble des alignements, le mieux est de les coder sous forme d'automate, plutôt que de les énumérer tous.

Dictionnaire de 100 000 formes

- 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:

courier	0	courrier	1	courtier	1
courir	1	courber	1	sourir	2
tourner	2	usurier	2	écourter	2

Plus longue sous-chaine commune à X et Y

Soit X = $x_1 \ldots x_N$ et $Y = y_1 \ldots y_M$

- $Z = z_1 \ldots z_K$ est une sous-chaine de X (noté $Z = SC(X)$) ssi il existe $\rho : [1, K] \rightarrow [1, N]$ tel que:
  $Z = x_{\rho(1)} \ldots x_{\rho(K)}$ avec $\rho(i) > \rho(j), \forall i > j$
- Z est une sous-chaine commune à X et Y (noté $Z = SCC(X, Y)$) 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

Théorème:

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** $Z_1^K = PLSC(X_1^{N-1}, Y)$
  - **si** $z_K \neq y_M$ **alors** $Z_1^K = PLSC(X, Y_1^{M-1})$

Récurrence pour le calcul de la longueur d'une PLSCC de deux chaines

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

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

Exemple:

$$\overbrace{\overbrace{\overbrace{1 + \max(1 + \overbrace{\overbrace{1 + \underbrace{PLSCC(\epsilon,\epsilon)}_{0}}^{PLSCC(A,A)=1}}^{PLSCC(AA,AA)=2}, \max(\overbrace{1 + \underbrace{PLSCC(A,\epsilon)}_{0}}^{PLSCC(AA,A)=1}, \underbrace{PLSCC(AA,\epsilon)}_{0}))}^{PLSCC(AAB,A)=1}}^{PLSCC(AAB,AA)=2}}^{PLSCC(AABB,AAB)=3}$$

Programmation dynamique

Require: T une table $N \times M$ dont un élément est T[i,j] = <score,back pointeur>
Ensure: T[N,M] contient la longueur d'une PLSSC de X et Y
  for $i : 0 \rightarrow N$ **do** T[i,0] $= < 0, \downarrow >$
  for $j : 1 \rightarrow M$ **do** T[0,j] $= < 0, \leftarrow >$
  for $i : 1 \rightarrow N$ **do**
    for $j : 1 \rightarrow M$ **do**
      if $x_i = y_j$ **then**
        T[i,j] $= < T[i-1, j-1] + 1, \swarrow >$
      else
        T[i,j] $= < T[i-1, j], \downarrow >$
        if T[i,j-1].score $>$ T[i,j].score **then**
          T[i,j] $= < T[i, j-1], \leftarrow >$

Programmation dynamique

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	←
	0	↓	0	←	0	←	0	←	0	←	0	←	0	←	0	←	0	←
			A		C		A		C		C		A		G		T	

Mesure de la qualité d'un engin de traduction: WER

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%)

Note sur 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	I

Références