Recherche approximative d'un mot dans un texte

Problème: **trouver toutes les** *occurrences approximatives* **d'un mot** $P = p_1 p_2 \ldots p_m$ **dans un texte** $T = t_1 t_2 \ldots t_n$**.**

Exemple:

Mot: TATA

Texte: ACG TAATA G CATA GGGTGTTGG TATA ...

Formellement...

On définit

$$D(i,j) = \min_g D(p_1 \ldots p_i, t_g \ldots t_j),$$

où g varie de 1 à j, comme étant la distance d'édition minimale entre le préfixe $p_1 p_2 \ldots p_i$ de longueur i de P et les suffixes du texte T se terminant en position j.

Avec cette définition, si $D(m,j) \leq t$ alors il y a une ou des occurrences approximatives de P, ayant au plus t erreurs, se terminant en position j du texte.

Utilisations de la recherche approximative en biologie:

1. Recherche de similitudes entre 2 séquences biologiques.
2. Recherche d'une molécule nouvellement séquencée dans les banques de données existantes.
3. Recherche d'un gène dans un génome.
4. ...

Solution Classique: Calculer la matrice des distances $D[i,j]$

		A	C	G	T	A	A	T	A	G	C	...
	0	0	0	0	0	0	0	0	0	0	0	...
T	1	1	1	1	0	1	1	0	1	1	1	...
A	2	1	2	2	1	0	1	1	0	1	...	...
T	3	2	2	3	2	1	1	1	1	1	...	...
A	4	3	3	3	3	2	**1**	2	**1**	2	...	...
							↑		↑			

où

$$D[i,j] = \min \begin{cases} D[i-1,j-1] + \delta(p_i, t_j) \\ D[i,j-1] + 1 \\ D[i-1,j] + 1 \end{cases} \quad (1)$$

avec les conditions initiales $D[0,j] = 0$ et $D[i,0] = i$.

Plan de l'exposé

1. Ne calculer qu'une partie de la table des distances
2. Prétraitement du mot P cherché
3. Approche hybride (calcul de diagonales et prétraitement)
4. Approches vectorielles

Ne calculer qu'une partie de la table des distances [a]

Lemme: **Dans la matrice des distances D, on a toujours**

$$D(i,j) = D(i-1,j-1)$$

ou

$$D(i,j) = D(i-1,j-1)+1$$

$\Rightarrow$ si on garde en mémoire pour une colonne donnée j, la dernière ligne i pour laquelle la valeur de la cellule $(i,j) \leq t$ alors, lors du calcul des cellules de la colonne $j+1$, seulement les cellules des lignes 0 à $i+1$ contiendront des entrées $\leq t$.

[a] E. Ukkonen, *Finding Approximate Patterns in Strings*, Journal of Algorithms, 6, pp.132-137, 1985.

Exemple: Soit $T = GTTGCAGGAACG$, $P = AACG$ et $t = 1$, le nombre d'erreurs permis dans l'alignement:

	λ	G	T	T	G	C	A	G	G	A	A	C	G
λ	0	0	0	0	0	0	0	0	0	0	0	0	0
A	1	1	1	1	1	1	0	1	1	0	0	1	1
A	2	2	2	2	2	2	1	1	2	1	0	1	2
C	3	X	X	X	X	X	X	2	2	X	1	0	1
G	4	X	X	X	X	X	X	X	X	X	X	**1**	**0**

Deuxième exemple: Soit $T = AACGAGGAAC$,
$P = AACG$ et $t = 1$, le nombre d'erreurs permis dans l'alignement:

	λ	A	A	C	G	A	G	G	A	A	C
λ	0	0	0	0	0	0	0	0	0	0	0
A	1	0	0	1	1	0	1	1	0	0	1
A	2	1	0	1	2	1	1	2	1	0	1
C	3	X	1	0	1	2	2	2	2	1	0
G	4	X	X	**1**	**0**	**1**	2	X	3	X	**1**

Complexité: $\mathcal{O}(nm)$

Plan de l'exposé

1. Ne calculer qu'une partie de la table des distances
2. Prétraitement du mot P cherché
3. Approche hybride (calcul de diagonales et prétraitement)
4. Approches vectorielles

Prétraitement du mot P cherché[a]

- On va construire à partir de P, un automate fini déterministe, dénoté M_P
- Si on arrive dans un état final de l'automate, lors de la lecture de t_j, alors on a une occurrence approximative de P, ayant $e \leq t$ erreurs, se terminant à la position j du texte
- Intuitivement, étant donné un mot P fixé, chaque état de M_P correspond à une colonne pouvant apparaître dans la matrice des distances D quand le texte T varie.
- Un état de M_P est final si l'élément en position m dans la colonne correspondante de cet état est $\leq t$.

[a] E. Ukkonen, *Finding Approximate Patterns in Strings*, Journal of Algorithms, 6, pp.132-137, 1985.

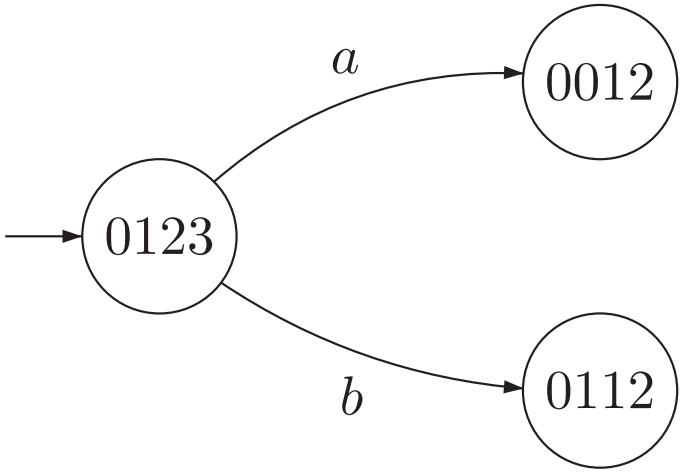
Exemple: Soit $P = aba$, $t = 1$ et T un texte sur l'alphabet $\{a, b\}$. On veut construire l'automate M_P.

- État initial = 0123 (Première colonne de D)
- Maintenant, le texte T commence soit par un a, soit par un b. On calcule donc, pour ces deux cas, la prochaine colonne de la table D:

	λ	a
λ	0	0
a	1	0
b	2	1
a	3	2

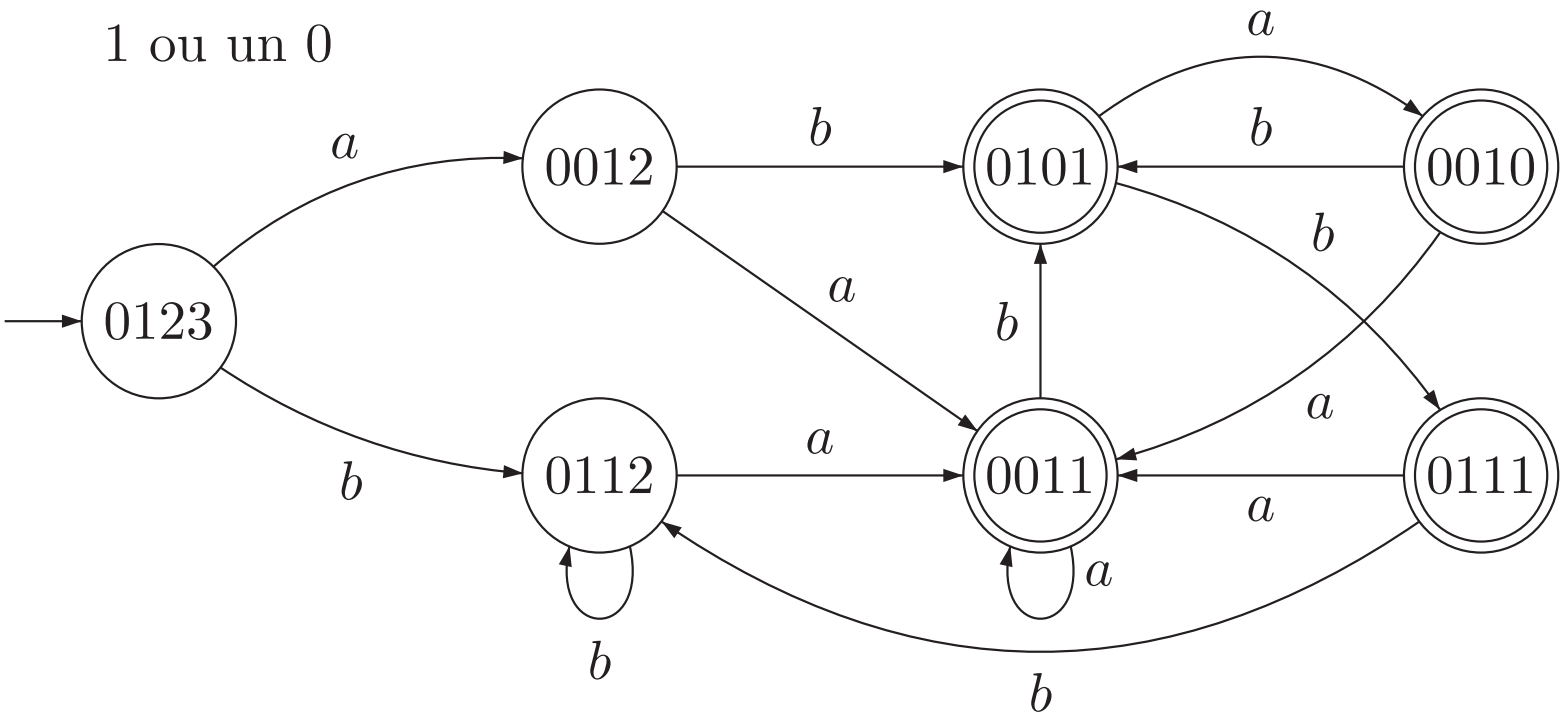
	λ	b
λ	0	0
a	1	1
b	2	1
a	3	2

- Donc, de l'état initial 0123, on se rend à l'état 0012 avec la transition a et à l'état 0112 avec la transition b.



- Pour ces deux nouveaux états, on calcule les colonnes de D correspondant à la lecture d'un a ou de b et on obtient encore de nouveaux états.
- On arrête l'algorithme de construction lorsque tous les nouveaux états (ou nouvelles colonnes de D) font déjà partie de l'automate.

- Comme le nombre d'erreurs permis est $t = 1$ dans cet exemple, les états finaux de l'automate sont les états se terminant par un 1 ou un 0



Complexité: $\mathcal{O}(m \cdot |\Sigma| \cdot 3^m)$ (prétraitement)
$\mathcal{O}(n)$ (recherche d'occurrences)

Plan de l'exposé

1. Ne calculer qu'une partie de la table des distances
2. Prétraitement du mot P cherché
3. Approche hybride (calcul de diagonales et prétraitement)
4. Approches vectorielles

Approche hybride [a] [b] [c]

- L'approche hybride utilise la table des distances D (programmation dynamique) par un calcul de diagonales et un prétraitement (du mot P ou du mot P ET du texte T) pour trouver la position des occurrences approximatives de P dans T ayant moins de t erreurs.
- La différence entre les algorithmes a,b et c est le prétraitement choisi. Tous ces algorithmes ont la même complexité.

---

[a] G.M. Landau et U. Vishkin, *Fast parallel and serial approximate string matching*, Journal of Algorithms, 10, pp.157-169, 1989.

[b] Z. Galil et K. Park, *An Improved Algorithm for Approximate String Matching*, SIAM J. Comput. , 19-6, pp.989-999, 1990.

[c] E. Ukkonen et D. Wood, *Approximate String Matching with Suffix Automata*, Algorithmica, 10, pp.353-364, 1993.

Quelques Définitions:

1) Un chemin-d, $d \leq t$, est un chemin commençant à la ligne 0 et décrivant un alignement contenant exactement d erreurs.

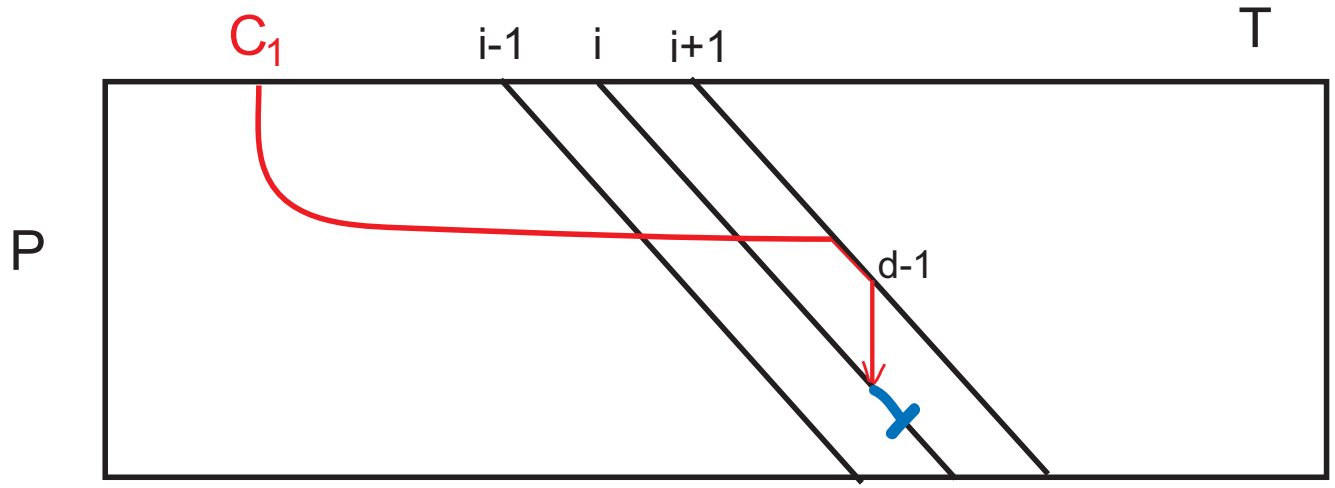
2) Un chemin-d est dit extrémal à la diagonale i s'il est le chemin-d atteignant la case la plus éloignée sur cette diagonale.

Idée générale:

- Procéder en t itérations, où t est le nombre d'erreurs permis.
- Pour chaque $d \leq t$ et pour chaque diagonale i de la table, trouver la case la plus éloignée sur la diagonale i contenant la valeur d
- Le chemin-d extrémal sur la diagonale i est déduit à partir des chemins-$(d-1)$ extrémaux sur les diagonales $i-1$, i et $i+1$.
- Un chemin-d se terminant sur la ligne m, représente une occurrence de P dans T contenant d erreurs.

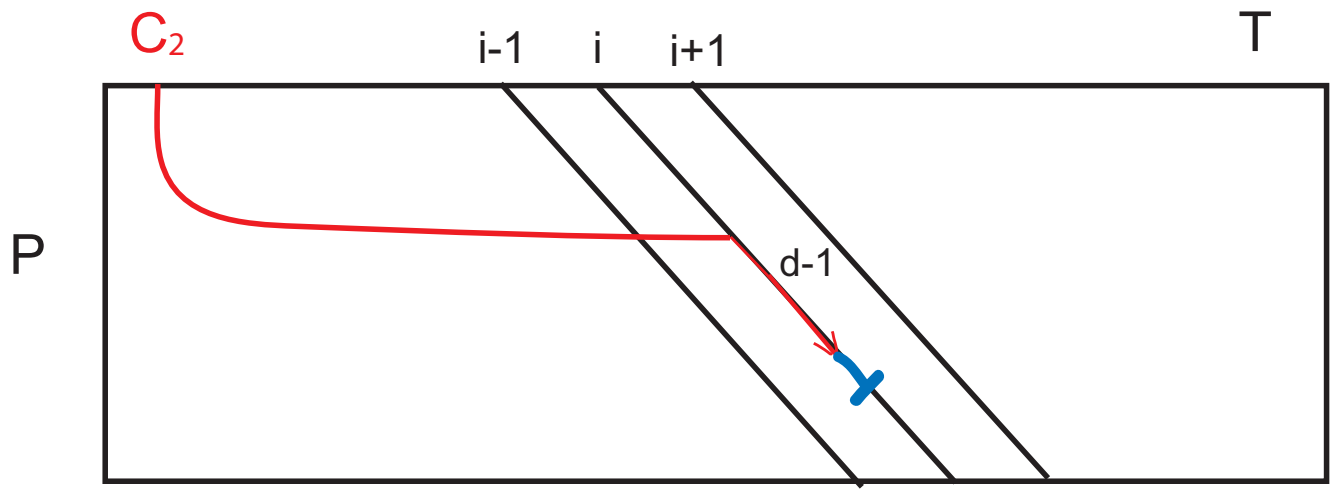
Détails:

- Pour $d = 0$, le chemin-0 extrémal pour une diagonale i est déterminé par le plus long facteur commun entre P et $T[i..n]$.
- Pour $d > 0$, le chemin-d extrémal à la diagonale i est obtenu à partir des trois chemins suivants:
  * C_1: chemin-$(d-1)$ extrémal à la diagonale $i+1$, suivi d'une suppression, suivi de l'extension maximale sur la diagonale i.



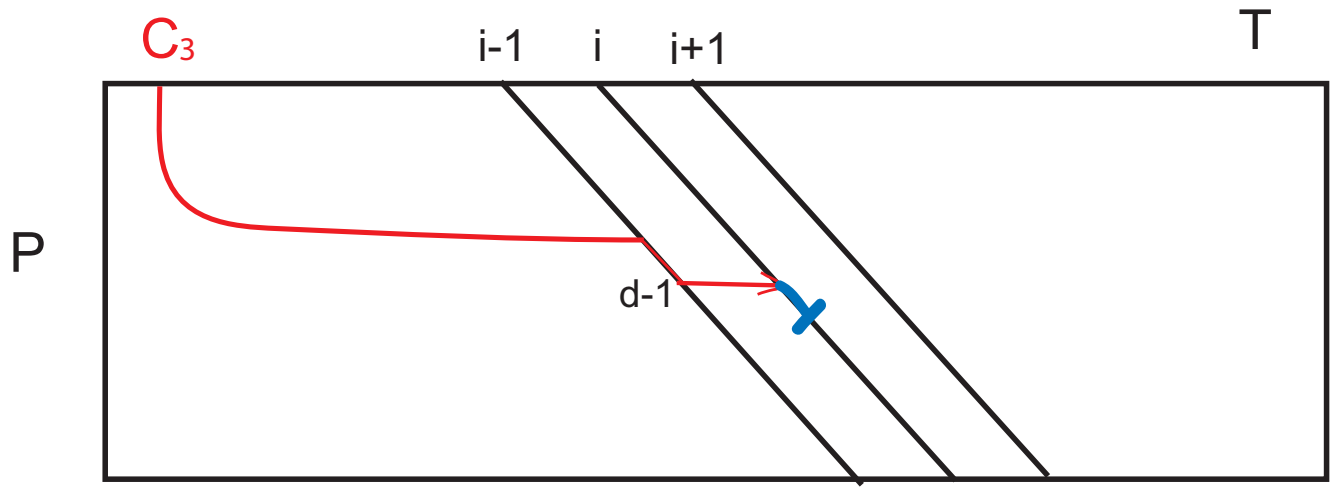
Détails (suite):

* C_2: chemin-$d-1$ extrémal sur la diagonale i, suivi d'une substitution, suivi de l'extension maximale sur la diagonale i correspondant à un plus long facteur identique entre P et T.



Détails (suite):

* C_3: chemin-$d-1$ extrémal sur la diagonale $i-1$, suivi d'une insertion, suivi de l'extension maximale sur la diagonale i correspondant à un plus long facteur identique entre P et T.



Théorème: Le chemin-d extrémal sur la diagonale i est celui des trois chemins C_1, C_2 ou C_3 qui atteint la case la plus éloignée sur la diagonale i.

Exemple:

Soit $T = AACGAGGAAC$, $P = AACGGG$ et $t = 2$, le nombre d'erreurs permis dans l'alignement:

	λ	A	A	C	G	A	G	G	A	A	C
λ	0	0	0	0	0	0	0	0	0	0	0
A	1	0									
A	2		0								
C	3			0							
G	4				0						
G	5										
G	6										

Exemple:

Soit $T = AACGAGGAAC$, $P = AACGGG$ et $t = 2$, le nombre d'erreurs permis dans l'alignement:

	λ	A	A	C	G	A	G	G	A	A	C
λ	0	0	0	0	0	0	0	0	0	0	0
A	1		0								
A	2										
C	3										
G	4				0						
G	5										
G	6										

Exemple:

Soit $T = AACGAGGAAC$, $P = AACGGG$ et $t = 2$, le nombre d'erreurs permis dans l'alignement:

	λ	A	A	C	G	A	G	G	A	A	C
λ	0	0	0	0	0	0	0	0	0	0	0
A	1		0								
A	2										
C	3										
G	4				0						
G	5										
G	6										

Exemple:

Soit $T = AACGAGGAAC$, $P = AACGGG$ et $t = 2$, le nombre d'erreurs permis dans l'alignement:

	λ	A	A	C	G	A	G	G	A	A	C
λ	0	0	0	0	0	0	0	0	0	0	0
A	1		0								
A	2										
C	3										
G	4				0						
G	5				1						
G	6										

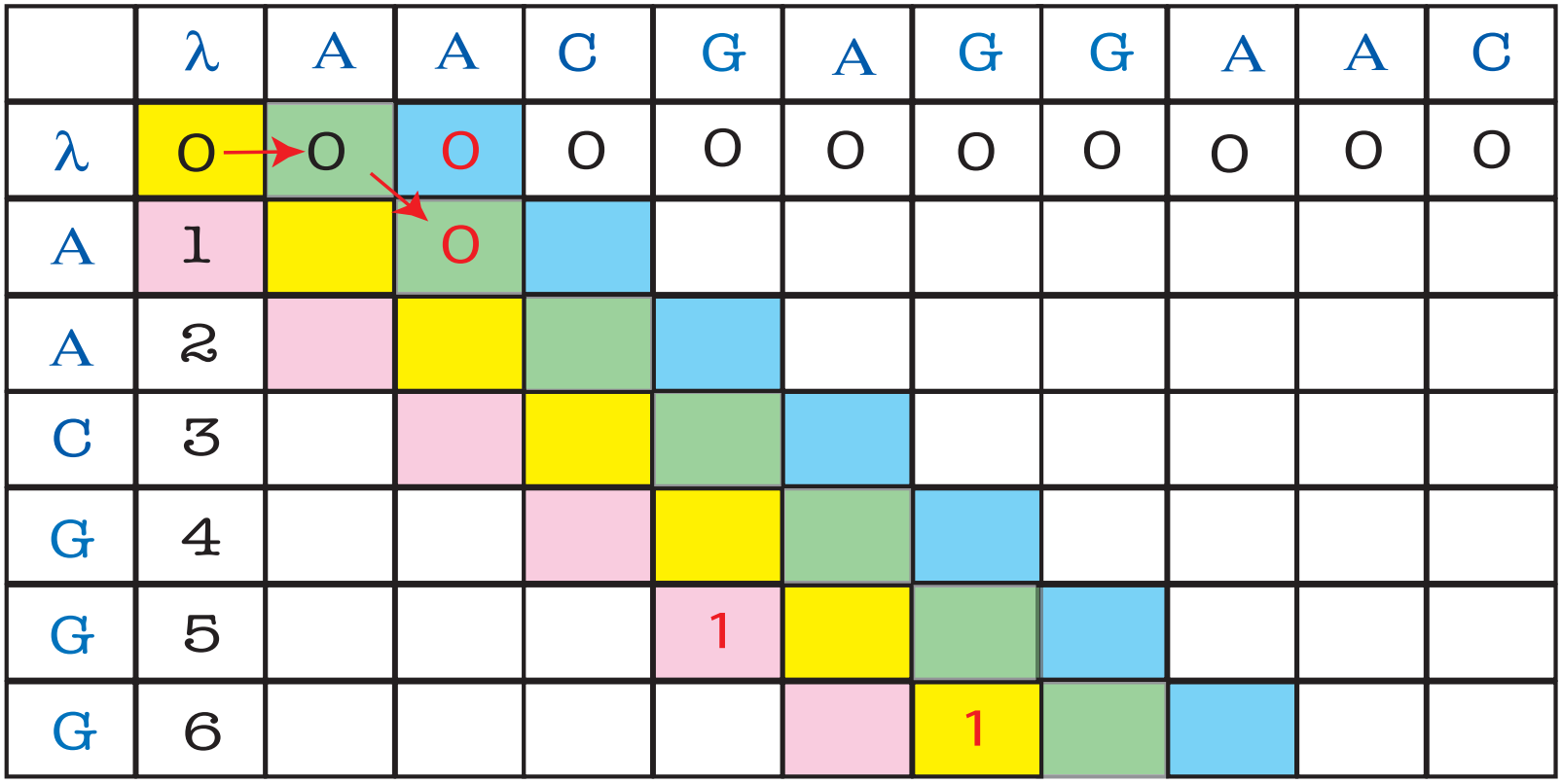
Exemple:

Soit $T = AACGAGGAAC$, $P = AACGGG$ et $t = 2$, le nombre d'erreurs permis dans l'alignement:

	λ	A	A	C	G	A	G	G	A	A	C
λ	0	0	0	0	0	0	0	0	0	0	0
A	1		0								
A	2										
C	3										
G	4				0						
G	5				1	1					
G	6						1				

Exemple:

Soit $T = AACGAGGAAC$, $P = AACGGG$ et $t = 2$, le nombre d'erreurs permis dans l'alignement:



Exemple:

Soit $T = AACGAGGAAC$, $P = AACGGG$ et $t = 2$, le nombre d'erreurs permis dans l'alignement:

	λ	A	A	C	G	A	G	G	A	A	C
λ	0	0	0	0	0	0	0	0	0	0	0
A	1		0								
A	2			1							
C	3										
G	4										
G	5				1						
G	6						1				

Complexité:

- **En temps:** Pour chaque d et chaque i, retrouver la plus longue extension le long de la diagonale i. Correspond à trouver un préfixe commun entre un suffixe de P et un suffixe de T commençant à des positions connues (peut se faire en temps constant);
  - **Galil-Park** Prétraitement de P + utilisation de triplets de référence. $\mathcal{O}(m^2)$
  - **Landau-Vishkin** Arbres des suffixes de $t_1 t_2 \dots t_n p_1 p_2 \dots p_m$. $\mathcal{O}(m+n)$
  - **Ukkonen-Wood** Automate des suffixes $\mathcal{O}(m)$

$$\Longrightarrow \mathcal{O}(t(n+m)) + \text{prétraitement} = \mathcal{O}(tn)$$

Plan de l'exposé

1. Ne calculer qu'une partie de la table des distances
2. Prétraitement du mot P cherché
3. Approche hybride (calcul de diagonales et prétraitement)
4. Approches vectorielles

Définition:

Un algorithme vectoriel est un algorithme qui trouve un vecteur de sortie en appliquant un nombre d'opérations sur le vecteur d'entrée, indépendant de la longueur du vecteur.

L'exploitation du parallélisme des opérations sur les vecteurs de bits améliore grandement le temps de calcul de ces algorithmes par rapport aux algorithmes présentés précédemment.

Un vecteur de bits

0110100100001000101

Opérations sur les vecteurs de bits

Soit $\boldsymbol{x}$ et $\boldsymbol{y}$ des vecteurs booléens - ou vecteurs de bits. On permet les opérations suivantes sur ces vecteurs:

1. disjonction $\boldsymbol{x} \vee \boldsymbol{y}$
2. conjonction $\boldsymbol{x} \wedge \boldsymbol{y}$
3. négation $\neg \boldsymbol{x}$
4. addition binaire avec retenue $\boldsymbol{x} +_{\boldsymbol{b}} \boldsymbol{y}$
5. déplacement vers la droite $\uparrow_a \boldsymbol{x}$
6. vecteur caractéristique d'une lettre a $\mathbf{a}$

Vecteurs de bits et recherche exacte [a]

Idée:

- Travailler avec des vecteurs de bits de longueur m, où m est la longueur de P.
- Pour chaque position j dans le texte T, on définit un vecteur de bits $\boldsymbol{R_j}$ qui contient l'information sur tous les alignements exacts de préfixes du mot P se terminant en position j du texte. Plus formellement, on a pour $1 \leq i \leq m$:

$$\boldsymbol{R_j}[i] = \begin{cases} 1 & \text{si le préfixe de longueur } i \text{ de } P \text{ est identique} \\ & \text{au suffixe de longueur } i \text{ de } t_1 \ldots t_j \\ 0 & \text{sinon} \end{cases}$$

[a] R.A. Baeza-Yates et G.H. Gonnet, A new approach to text searching, communications of the ACM, 35, pp. 74-82, 1992.

Idée (suite):

- À chaque position j du texte, si $\boldsymbol{R_j}[m] = 1$ alors il y a une occurrence exacte de P se terminant en position j du texte, et donc commençant en position $j - m + 1$ du texte.
- Soit $\boldsymbol{R_0}$ le vecteur de bits initial défini par $\boldsymbol{R_0}[i] = 0$, $\forall i,\ 1 \leq i \leq m$. Si on suppose que $\boldsymbol{R_j}[0] = 1$, $\forall j,\ 0 \leq j \leq n$, alors on a la transition suivante entre $\boldsymbol{R_j}$ et $\boldsymbol{R_{j+1}}$:

$$\boldsymbol{R_{j+1}}[i] = \begin{cases} 1 & \text{si } \boldsymbol{R_j}[i-1] = 1 \text{ et } p_i = t_{j+1} \\ 0 & \text{sinon} \end{cases}$$

Lemme: On a que $\boldsymbol{R_{j+1}} = \uparrow_1 \boldsymbol{R_j} \wedge \boldsymbol{t_{j+1}}$, où $\boldsymbol{t_{j+1}}$ représente le vecteur caractéristique de la lettre t_{j+1} dans le mot cherché P.

Remarque: Si la longueur m du mot P est plus petite que la longueur d'un mot machine, alors la calcul de $\boldsymbol{R_{j+1}}$ à partir de $\boldsymbol{R_j}$ requière seulement deux opérations sur les vecteurs de bits (un déplacement vers la droite et une conjonction) et se fait donc en temps constant.

Complexité: Si ω est la taille d'un mot machine (32 ou 64 bits), alors, la complexité en temps de l'algorithme est en $\mathcal{O}(\frac{m}{\omega}n)$

Exemple: Calculons les occurrences exactes du mot $P = TTA$ dans le texte $T = ACGTTACGTAAT$. Pour cela, on doit commencer par calculer les vecteurs caractéristiques de chacune des lettres, apparaissant dans le texte T, par rapport au mot P:

$$\boldsymbol{A} = 001, \qquad \boldsymbol{C} = 000, \qquad \boldsymbol{G} = 000, \qquad \boldsymbol{T} = 110.$$

Le vecteur $\boldsymbol{R_0}$= 000, par définition. Maintenant, on se sert du lemme $\boldsymbol{R_{j+1}} = \uparrow_1 \boldsymbol{R_j} \wedge \boldsymbol{t_{j+1}}$ pour calculer les valeurs des vecteurs $\boldsymbol{R_j}$, $1 \leq j \leq n = 12$:

$$\boldsymbol{R_1} = \uparrow_1 \boldsymbol{R_0} \wedge \boldsymbol{A} = 100 \wedge 001 = 000,$$

$$\boldsymbol{R_2} = \uparrow_1 \boldsymbol{R_1} \wedge \boldsymbol{C} = 100 \wedge 000 = 000,$$

$$\vdots$$

On obtient la table de vecteurs suivantes, où $\boldsymbol{R_j}$ est sous le caractère t_j de T:

	A	C	G	T	T	A	C	G	T	A	A	T
T	0	0	0	1	1	0	0	0	1	0	0	1
T	0	0	0	0	1	0	0	0	0	0	0	0
A	0	0	0	0	0	**1**	0	0	0	0	0	0

Il y a une occurrence exacte de $P = p_1 \ldots p_m$ dans T en position j si et seulement si $\boldsymbol{R_j}[m] = 1$. Ici, la seule occurrence exacte de $P = ATT$ dans le texte T se termine en position $j = 6$ du texte.

Vecteurs de bits et recherche approximative [a]

Idée:

- Calculer les vecteurs $\boldsymbol{R_j}$ comme précédemment (maintenant dénotés $\boldsymbol{R_j^0}$).
- Calculer aussi t nouveaux vecteurs, où t est le nombre d'erreurs permis, $\boldsymbol{R_j^1}$, $\boldsymbol{R_j^2}$, ..., $\boldsymbol{R_j^t}$.
- Le vecteur $\boldsymbol{R_j^d}$ contient l'information sur les occurrences de préfixes de P, contenant au plus d erreurs, se terminant en position j du texte.
- On veut pouvoir calculer, avec des opérations vectorielles, le vecteur $\boldsymbol{R_{j+1}^d}$.

---

[a] S. Wu et U. Manber, Fast Text Searching Allowing Errors, Communications of the ACM, 35, pp. 83-91, 1992.

Remarque: Il y a quatre façons différentes d'obtenir une occurrence de $p_1 \dots p_i$, contenant au plus d erreurs, se terminant en t_{j+1}:

1) Identité: il y a une occurrence de $p_1 \dots p_{i-1}$, contenant au plus d erreurs, se terminant en position j du texte et $t_{j+1} = p_i$.

$$\Longrightarrow (\uparrow_1 \boldsymbol{R_j^d} \wedge \boldsymbol{t_{j+1}})$$

2) Substitution: il y a une occurrence de $p_1 \dots p_{i-1}$, contenant au plus $d-1$ erreurs, se terminant en position j du texte et $t_{j+1} \neq p_i$.

$$\Longrightarrow (\uparrow_1 \boldsymbol{R_j^{d-1}})$$

3) Suppression de p_i: il y a une occurrence de $p_1 \dots p_{i-1}$, contenant au plus $d-1$ erreurs, se terminant en position $j+1$ du texte.

$$\Longrightarrow (\uparrow_1 \boldsymbol{R_{j+1}^{d-1}})$$

Remarque (suite):

4) Insertion de t_{j+1}: il y a une occurrence de $p_1 \dots p_i$, contenant au plus $d-1$ erreurs, se terminant en position j du texte.

$$\Longrightarrow \boldsymbol{R_j^{d-1}}$$

Lemme: Si $\boldsymbol{R_0^d} = \overbrace{11\dots1}^{d}\overbrace{00\dots0}^{m-d}$, alors

$$\boldsymbol{R_{j+1}^d} = (\uparrow_1 \boldsymbol{R_j^d} \wedge \boldsymbol{t_{j+1}}) \vee \uparrow_1 (\boldsymbol{R_j^{d-1}} \vee \boldsymbol{R_{j+1}^{d-1}}) \vee \boldsymbol{R_j^{d-1}}$$

Complexité:

- Le calcul de chacun des vecteur $\boldsymbol{R_j^d}$, $1 \leq j \leq n$, requiert seulement 6 opérations sur des vecteurs de bits (2 déplacements vers la droite, 3 disjonctions et 1 conjonction) et peut donc être calculé en temps $\mathcal{O}(n)$.
- Comme d varie entre 0 et t, le calcul des occurrences approximatives de P, ayant au plus t erreurs, se fait en temps $\mathcal{O}(tn)$, si la longueur du mot P est plus petite que la longueur d'un mot machine.
- Sinon, $\mathcal{O}(tn\frac{m}{\omega})$

Algorithme de Myers [a]

Idée: Calculer les différences horizontales et verticales entre les éléments de la table des distances D

Définition: On définit $\Delta v_{i,j}$ comme étant la *différence verticale* entre les éléments de la table des distances situés en les positions (i, j) et $(i-1, j)$:

$$\Delta v_{i,j} = D[i, j] - D[i-1, j].$$

De la même façon, on définit $\Delta h_{i,j}$ comme étant la *différence horizontale* entre les éléments de la table des distances situés en les positions (i, j) et $(i, j-1)$:

$$\Delta h_{i,j} = D[i, j] - D[i, j-1].$$

[a] G. Myers, A Fast Bit-Vector Algorithm for Approximate String Matching Based on Dynamic Programming, Journal of the ACM, 46-3, pp. 395-415, 1999.

Pour connaître les valeurs des éléments de la colonne j de la table des distances D, il suffit de connaître le vecteur de différences verticales

$$\boldsymbol{\Delta v_j} = \Delta v_{1,j} \ldots \Delta v_{m,j},$$

étant donné que $D(0, j) = 0$, $\forall j$.

Nouveaux problèmes:

1. Comment calculer rapidement les vecteurs de différences verticales?
   Idée de Myers: Calculer $\boldsymbol{\Delta v_j}$ en temps constant en utilisant des opérations sur les vecteurs de bits (si $m < \omega$).
2. Comment trouver facilement la valeur de $D[m, j]$?
   Posons $score_j = D(m, j)$. Alors, $score_0 = m$, car $D[m, 0] = m$. Ensuite, on a, par définition que

$$score_j = score_{j-1} + \Delta h_{m,j}.$$

Représentation des différences horizontales et verticales par des vecteurs de bits

Valeurs des vecteurs $\boldsymbol{\Delta v_j}$ et $\boldsymbol{\Delta h_j}$ dans l'ensemble $\{-1, 0, 1\}$.

Deux vecteurs de bits pour représenter $\boldsymbol{\Delta v_j}$: $\boldsymbol{Pv_j}$ pour les 1 et $\boldsymbol{Mv_j}$ pour les -1. Les 0 sont obtenus en calculant le vecteur $\neg(\boldsymbol{Pv_j} \vee \boldsymbol{Mv_j})$.

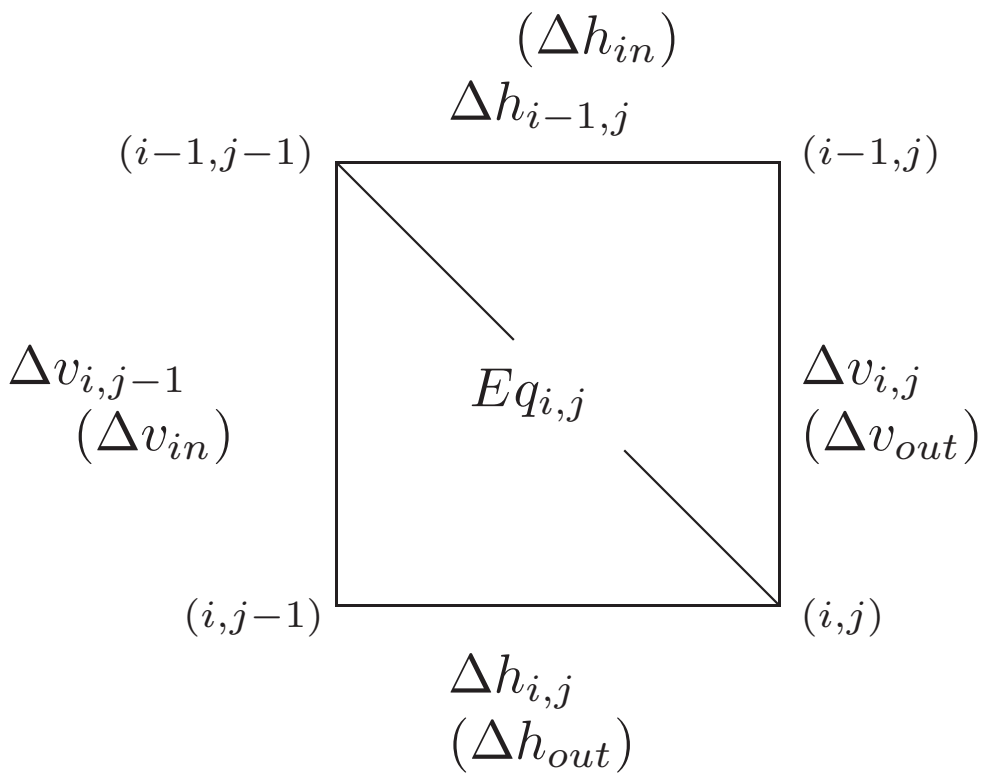
	-	G	G	A	A	C	G
-	0	0	0	0	0	0	0
A	1	1	1	0	0	1	1
A	2	1	2	1	0	1	2
C	3	2	2	2	1	0	1
G	4	2	2	3	2	1	0

0	**1**	**2**	**3**	**4**	**5**	**6**
1	1	1	0	0	1	1
1	0	1	1	0	0	1
1	1	0	1	1	-1	-1
1	0	0	1	1	1	-1

Ici, $\boldsymbol{\Delta v_5}$ est représenté par: $\boldsymbol{Pv_5} = 1001$, $\boldsymbol{Mv_j} = 0010$ et $\neg(\boldsymbol{Pv_j} \vee \boldsymbol{Mv_j}) = 0100$.

Calcul des différences verticales et horizontales

Considérons la cellule suivante, où $Eq_{i,j} = 1$ si $p_i = t_j$ et 0 sinon:



Théorème: On a que

$$\Delta v_{out} = \min\{-Eq_{i,j}, \Delta v_{in}, \Delta h_{in}\} + (1 - \Delta h_{in})$$

$$\Delta h_{out} = \min\{-Eq_{i,j}, \Delta v_{in}, \Delta h_{in}\} + (1 - \Delta v_{in}).$$

Comme Δv_{in} et Δh_{in} peuvent prendre seulement les valeurs -1, 0 ou 1 et que $Eq = 0$ ou 1, il n'y a que 18 entrées différentes possibles pour une cellule. L'étude des différents cas d'entrée a donné l'algorithme suivant:

- Au départ, on a $\boldsymbol{Pv_0}$(i)=1, $\forall i$, $\boldsymbol{Mv_0}$(i)=0, $\forall i$ et $score_0 = m$
- Pour j de 1 à n, on veut calculer $\boldsymbol{\Delta v_j}$ et $score_j$ on procède alors comme suit:

1. Les différences verticales de la colonne $j-1$, $\boldsymbol{\Delta v_{j-1}}$, sont utilisées pour calculer les différences horizontales $\boldsymbol{\Delta h_j}$ comme suit:

$$\begin{aligned} \boldsymbol{Ph_j} &= \boldsymbol{Mv_{j-1}} \vee \neg(\boldsymbol{Xh_j} \vee \boldsymbol{Pv_{j-1}}) \\ \boldsymbol{Mh_j} &= \boldsymbol{Pv_{j-1}} \wedge \boldsymbol{Xh_j} \end{aligned}$$

où $\boldsymbol{Xh_j} = (((\boldsymbol{t_j} \wedge \boldsymbol{Pv_{j-1}}) +_{\boldsymbol{b}} \boldsymbol{Pv_{j-1}}) \oplus \boldsymbol{Pv_{j-1}}) \vee \boldsymbol{t_j}$

2. On modifie le score de la façon suivante:

$$score_j = score_{j-1} + \begin{cases} 1 & \text{si } \boldsymbol{Ph_j}(m) = 1 \\ -1 & \text{si } \boldsymbol{Mh_j}(m) = 1 \\ 0 & \text{sinon} \end{cases}$$

3. On calcule $\boldsymbol{\Delta v_j}$:

$$\begin{aligned} \boldsymbol{Pv_j} &= \uparrow_1 \boldsymbol{Mh_j} \vee \neg(\boldsymbol{Xv_j} \vee \uparrow_1 \boldsymbol{Ph_j}) \\ \boldsymbol{Mv_j} &= \uparrow_1 \boldsymbol{Ph_j} \wedge \boldsymbol{Xv_j} \end{aligned}$$

où $\boldsymbol{Xv_j = t_j \vee Mv_{j-1}}$

Donc, lorsque la longueur du mot P cherché est plus petite que la longueur d'un mot machine, Myers a développé un algorithme vectoriel permettant le calcul des occurrences approximatives de P dans T en temps $\mathcal{O}(n)$. Le pseudocode, en C, de l'algorithme est disponible dans l'article.