Optimisations des algorithmes d'alignement et de recherche approchée

Algorithmes de programmation dynamique classiques:

- Alignement local ou global: $O(mn)$ en temps et en espace (pire des cas et meilleur des cas)
- Recherche à k erreurs: $O(mn)$ en temps et en espace (pire des cas et meilleur des cas)

Une grande variété de méthodes d'optimisation en temps, en espace, pire des cas, cas moyen.

Une bonne référence: *G. Navarro,"A guided tour to approximate string matching", ACM Computing Surveys, Vol.33, No.1, March 2001, pp. 31-88*

I - Complexité en espace de l'alignement de séquences

S de taille m, T de taille n.

Valeur d'un alignement global optimal: $V(S,T)$

Algorithme classique pour l'alignement global: Espace $O(mn)$

Impossibilité de considérer de longues séquences d'ADN.

Calcul de $V(S,T)$ sans alignement: Pour le calcul des valeurs d'une ligne i, on a besoin uniquement de la ligne $i-1$. À chaque étape, remplir la ligne courante à l'aide de la ligne précédente $\longrightarrow$ Espace nécessaire $2m$

On peut encore réduire l'espace à $m+1$

Comment utiliser un espace linéaire pour trouver un alignement, sans trop augmenter le temps de calcul?

Algorithme de Hirschberg

D.S. Hirschberg, *Algorithms for the longest common subsequence problem*, J. ACM, 24, pp. 664- 675, 1977

Complexité en espace en $O(m)$ $(m \leq n)$, et complexité en temps dans le pire des cas doublé % à l'algorithme classique pour l'alignement global, donc, toujours en $O(mn)$.

Idée générale: Résoudre une famille de sous-problèmes par une stratégie "divide and conquer", et combiner les résultats obtenus pour produire l'alignement global.

Notations:

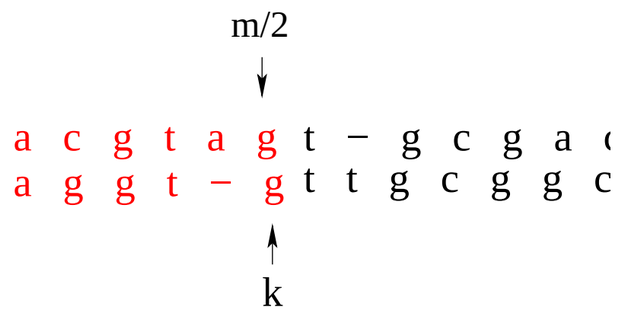
- Pour une séquence M, M^r l'inverse de M, i.e., Si t est la taille de M, $M^r[i] = M[t - i + 1]$.

  Exemple: $M = agctagt$, $M^r = tgatcga$.

- $V^r(i, j)$: valeur de l'alignement optimal entre $S^r[1..i]$ et $T^r[1..j]$, i.e entre $S[m - i + 1..m]$ et $T[n - j + 1..n]$

Lemme 1:

$$V(m,n) = max_{0 \leq k \leq n}[V(m/2,k) + V^r(m/2, n-k)]$$



Preuve: Pour tout k, $V(m,n) =$

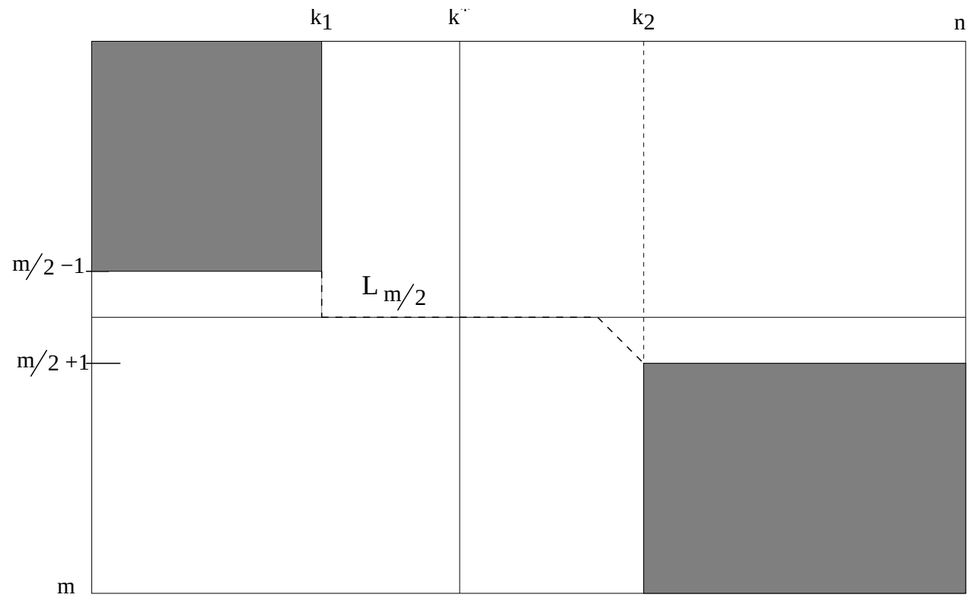
$$\text{Optimal}\begin{pmatrix} S[1..m/2] \\ T[1..k] \end{pmatrix} \quad + \quad \text{Optimal}\begin{pmatrix} S[m/2+1..m] \\ T[k+1..n] \end{pmatrix}$$

$$V(m/2,k) \qquad\qquad V^r(m/2, n-k)$$

Notation: k^* maximise $[V(m/2,k) + V^r(m/2, n-k)]$.

Il existe un chemin optimal L allant de $(0,0)$ à (m,n) qui passe par $(m/2, k^*)$

$L_{m/2}$: sous-chemin de L qui commence à la dernière case de la ligne $m/2-1$ de L et fini à la $1^{ère}$ case de la ligne $m/2+1$ de L.



Lemme 2: k^* peut être trouvé en un temps $O(mn)$ et en espace $O(m)$. Même complexité pour produire le sous-chemin $L_{m/2}$.

Preuve:

- Remplir la table pour S et T, mais s'arrêter après la ligne $m/2$. Conserver les pointeurs à la ligne $m/2 \longrightarrow$ temps $O(mn)$, espace $O(m)$. $V(m/2, k)$ connu pour tout k.
- Même chose pour S^r et T^r.
- Trouver la valeur maximale $V(m/2, k) + V(m/2, n-k) \hookrightarrow k^*$ $\longrightarrow$ temps $O(n)$.
- Produire le chemin $L_{m/2}$ en suivant les pointeurs, d'abord de $(m/2, k^*)$ à la case $(m/2-1, k_1)$, ensuite de $(m/2, k^*)$ à la case $(m/2+1, k_2)$.

Deux sous-problèmes: Alignement de $S[1, m/2-1]$ et $T[1..k_1]$, et alignement de $S[m/2+1..m]$ et $T[k_2..n]$

Procedure $OPTA(l, l', r, r')$;
Début
    $h = (l' - l)/2$;
    Trouver k^*, k_1, k_2 et L_h;
    $OPTA(l, h-1, r, k_1)$; (problème supérieur)
    Produire L_h;
    $OPTA(h+1, l', k_2, r')$; (problème inférieur)
Fin.

Appel initial: $OPTA(1, m, 1, n)$

À la sortie de la procédure, le chemin L produit est un alignement optimal de S et T.

Valeur $V(m, n)$ obtenue dès la premier appel de la procédure, i.e. pour $h = m/2$.

Complexité en temps: On note cpq, au lieu de $O(pq)$, le temps nécessaire pour remplir une table $p \times q$. c: constante.

- 1^{er} niveau de récursion: cmn
- $2^{ème}$ niveau de récursion, 2 sous-problèmes:

$$ck^*m/2 + c(n - k^*)m/2 = cnm/2$$

- $i^{ème}$ niveau de récursion, 2^{i-1} sous-problèmes.
  Chaque sous-problème a $m/2^{i-1}$ lignes et un nombre variables de colonnes. Mais chaque sous-problème a des colonnes distinctes des autres $\Rightarrow$ Au plus $cnm/2^{i-1}$

Théorème: Complexité en espace en $O(m)$ et complexité en temps

$$\sum_{i=1}^{\log n} cmn/2^{i-1} \leq 2cnm$$

Algorithme classique de l'alignement global en temps cmn.

II - Optimisation en temps pour l'alignement global

1. Nombre borné d'erreurs

Deux séquences S et T et un entier k.

Déterminer si un alignement global optimal de S et T a au plus k erreurs.

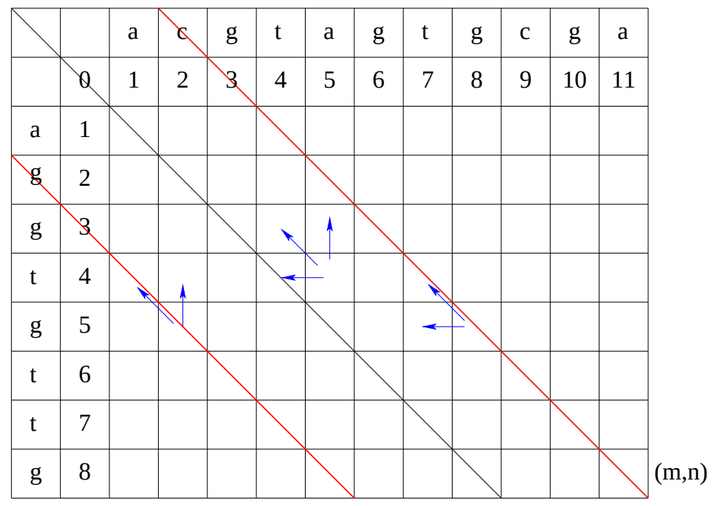
Exemples d'applications:

- Comparaison de gènes homologues, séquences très similaires.
- Vérification d'une nouvelle longue séquence avec des fragments déja séquencés par d'autres laboratoires: Problème si trop d'erreurs.
- Construction d'une banque de données non redondante, pour accelérer la recherche, pour des évaluations statistiques plus significatives. Par exemple, combiner les séquences de nucléotides ou d'acides aminés ayant plus de 97% de similarité.

Observation: Tout chemin définissant un alignement contenant moins de k erreurs ne passe par aucune case $(i, i+l)$ ou $(i, i-l)$, avec $l > k$

$\Rightarrow$ compléter la table dans une bande centrée sur la diagonale principale, déterminée par les deux diagonales $+k$ et $-k$. Bande qui contient $2k+1$ cases à chaque ligne

k=3



Si $m = n$, la taille de la bande peut être réduite de moitié

Si (m, n) contient une valeur $v < k$, alors un alignement optimal avec v erreurs existe. Alignement trouvé en suivant les pointeurs de (m, n) à $(0, 0)$.

Réciproquement, si un alignement optimal avec moins de k erreurs existe, alors (m, n) contient une valeur $< k$.

Si (m, n) contient une valeur plus grande que k, alors cette valeur est inférieur à la valeur d'un alignement optimal.

Complexité: en temps et en espace $O(km)$.

Nombre d'erreurs non spécifié

S et T deux séquences, et k^* valeur d'un alignement optimal (k^* non connu, mais petit).

"Deviner" la valeur de k^* en appliquant l'algorithme pour nombre d'erreurs borné avec différentes valeurs de k

Commencer avec $k = 1$, et doubler k jusqu'à obtenir un alignement global de valeur $\leq k$.

Complexité: Soit k' la plus grande valeur de k utilisée. $k' \leq 2k^*$.

Complexité totale en temps et en espace:

$$O(k'm + k'm/2 + k'm/4 + \cdots + m) = O(k'm) = O(k^*m)$$

II.2. - Méthode des quatre russes pour l'alignement global

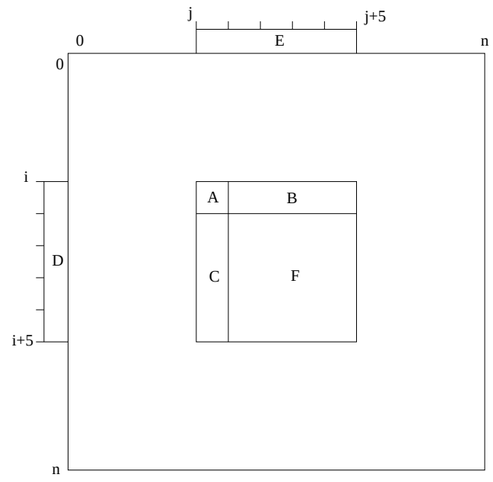
(Masek et Paterson, 1980)

Méthode d'accélération d'algorithmes basés sur la programmation dynamique. Comparaison de deux séquences de taille n en temps $O(n^2/\log n)$

Plusieurs améliorations de la méthode originelle existent (Myers), et s'appliquent à différentes versions du problème.

t-bloc: Sous-tableau de taille $t \times t$

Idée générale: Partitionner le tableau de prog. dyn. en t-blocs, "pas trop" chevauchant. Calcul élémentaire effectué pour chaque t-bloc plutôt que pour chaque case. Valeur de chaque t-bloc calculé en temps $O(t)$



Observation: Valeurs d'un t-bloc débutant à la case (i, j) fonction de la première ligne et de la première colonne du t-bloc, et des deux sous-mots $S_1[i..i+t-1]$ et $S_2[j..j+t-1]$.

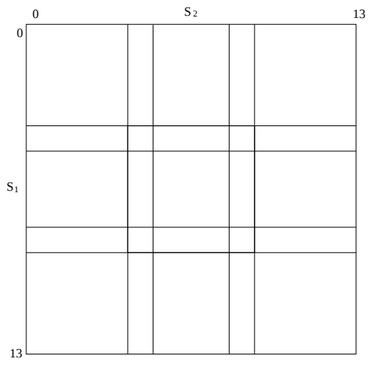
Fonction "bloc": Calcul des valeurs d'un $t - bloc$. Paramètres (A, B, C, D, E).

Fonction "bloc restreint": Calcul de la dernière ligne et de la dernière colonne d'un t-bloc.

Algorithme général

Pour simplifier, S_1 et S_2 sont de taille $n = k(t-1)$.

1. Subdiviser le tableau $(n+1) \times (n+1)$ en t-blocs:



2. Initialiser la première ligne et la première colonne du tableau
3. Examiner le tableau ligne par ligne. Fonction “bloc restreint” détermine les valeurs de la dernière ligne colonne de chaque t-bloc.
4. Valeur d’édition de l’alignement global optimal en (n, n).

Cœur du problème: Étape (3).

Possibilité: Précalculer tous les $t - blocs$ possibles. Énumérer tous les inputs possibles, calculer les outputs correspondant (ligne et colonne de taille t). Conserver les output indexés par les inputs.

Complexité:

- Inputs et outputs de taille $O(t)$. Chaque input peut être retrouvé en un temps $O(t)$. Il y a (n^2/t^2) t-blocs $\Rightarrow$ algorithme en $O(n^2/t)$. Si $t \sim \log n$, algorithme en $O(n^2/\log n)$.
- Phase de prétraitement: Combien de choix d'inputs? Chaque case peut contenir les valeurs de 0 à $n \Rightarrow (n+1)^t$ valeurs possibles pour chaque ligne ou colonne de taille t. D'autre part, $|\Sigma|^t$ mots possibles de taille t

$\Longrightarrow (n+1)^{2t}|\Sigma|^{2t}$ inputs différents

$\Longrightarrow$ Temps de prétraitement en $O((n+1)^{2t}|\Sigma|^{2t}t^2)$, et comme $t \geq 1$, temps au moins en $O(n^2)$ $\Rightarrow$ Pas de progrès !!!

Optimisation

Terme prédominant: $(n+1)^{2t}$, nombre de choix possibles pour chaque ligne et colonne. Mais ces choix pas tous possibles:

Lemme: Deux cases adjacentes d'une ligne, colonne ou diagonale du tableau de la prog. dyn. peuvent avoir des valeurs qui diffèrent d'au plus 1

Chaque ligne (ou colonne) d'un t-bloc peut être représentée par la valeur la plus à gauche de la ligne, suivie d'un vecteur de compensation, de valeurs $\{-1, 0, 1\}$.

Exemple: ligne 5, 4, 4, 5 $\longrightarrow$ 5, -1, 0, +1.

Comment se débarasser du premier terme?

Vecteur offset: Vecteur de taille t, de valeurs $\{-1, 0, 1\}$, où la première valeur doit être 0.

Théorème: t-bloc commençant à la position (i, j). Vecteurs offset de la dernière ligne et colonne du t-bloc, peuvent être déterminés à partir des vecteurs offset correspondant à la première ligne et colonne du t-bloc, et des mots $S_1[i..i + t - 1]$ et $S_2[j..j + t - 1]$.

Algorithme:

1. Subdiviser le tableau en t-blocs.
2. Initialiser la première ligne et colonne. Calculer les vecteurs offset de la première ligne et colonne.
3. Examiner le tableau ligne par ligne, et déterminer les vecteurs offset à la dernière ligne et à la dernière colonne de chaque bloc.
4. Si Q est la valeur totale des offsets calculés à la dernière ligne, alors $D(n, n) = D(n, 0) + Q = n + Q$.

(a)

		C	A	G	T	C	A
	0 / 0	+1	+1	+1 / +1	+1	+1	+1
A	+1			−1			−1
T	+1			0			−1
G	+1 / +1	0	0	0 / −1	+1	0	0 / +1
T	+1			+1			0
C	+1			+1			−1
A	+1 / +1	−1	−1	+1 / +1	−1	−1	−1 / −1

D(6,0) =6

D(6,6) = D(6,0)−1−1+1−1−1−1=6−4=2

(b)

	T	C	A
	0	1	2	3
A	−1	0	1	2
T	−1	−1	0	1
G	−1	0	0	1

Complexité: Il existe maintenant $3^{2(t-1)}|\Sigma|^{2t}$ inputs possibles. Pour chaque input, le output correspondant est calculé en temps $O(t^2)$.

$\Longrightarrow$ si $t = (\log_{3|\Sigma|} n)/2$, prétraitement effectué en un temps

$$O(n(\log n)^2)$$

Théorème: La distance d'édition entre deux séquences de taille n peut être calculée en un temps $O(n^2/\log n)$

Dans la pratique, valeur de t choisie telle que calcul des vecteur offsets et recherche d'un input en temps constant. Par exemple, choisir la taille d'un mot machine pour t. Également, utiliser une table de hashage pour ranger les inputs.