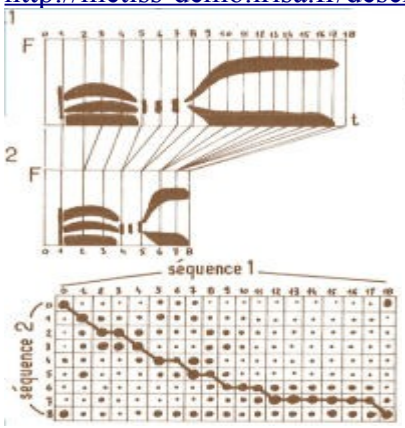
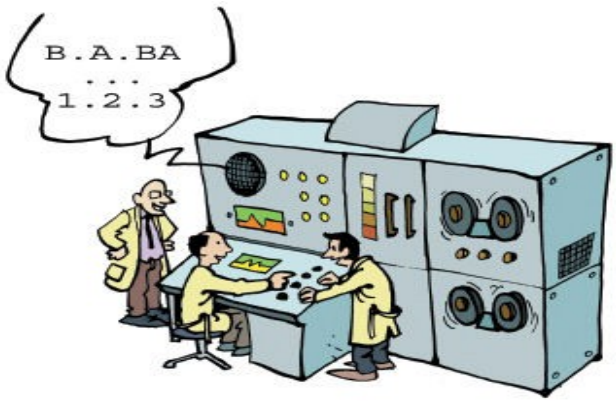
TP2 Langages Perl, Java, C / Programmation Dynamique

Une version heuristique d'alignement telle que la programmation dynamique a donné de très bon résultat en reconnaissance de la parole puis en bio-informatique pour l'alignement de séquence discrètes de tailles différentes.
http://metiss-demo.irisa.fr/descriptions/





Illustrons avec le problème de la plus longue sous-séquence de caractères commune à deux chaînes de caractères S1 et S2 (LCS : Longest Common Subsequence). Notez bien que le problème est différent de la Longest Common Substring (ici ce serait simplement GCG : les caractères de la solution doivent être contigus dans S1 et S2). Par ailleurs ces problèmes sont fortement liés à des analyses de complexité délicates et notamment de problèmes NP-complet (ce qui dépasse l'esprit de ce TP).
Soient deux chaînes de caractères de taille différente :
S1 = **GC**CCTA**G**CG
S2 = **GCG**CA**A**T**G**

La LCS de ces deux chaînes S1 et S2 est LCS(S1,S2)= **GCCAG**
Elle est de taille 5. C'est une façon de **mesurer la similarité entre deux séquences discrètes de taille différente.**

Modélisation récursive :
Soient :
C1 le caractère le plus à droite de S1 soit G
C2 le caractère le plus à droite de S2 soit G

S1 = S1'.C1 (. est le symbole de concaténation)
S2 = S2'.C2

Il y a alors trois problèmes récursifs à traiter :
L1=LCS(S1',S2)
L2=LCS(S1,S2')
L3=LCS(S1',S2')

La solution est la plus longue des trois chaînes suivantes :
L1
L2
L3.C1 si C1=C2 et L3 sinon

Cet algorithme est très coûteux en terme de temps de calcul (complexité exponentielle $O(k^{nm})$). D'où

une méthode plus coûteuse en stockage mémoire mais plus rapide en temps de calcul (complexité quadratique en n et m les tailles de deux séquences O(nm)) : la programmation dynamique.
Le cas trivial est le cas où L1= " " ou L2 = " " (chaînes vides) auquel cas LCS(L1,L2) = " " , ce qui nous servira pour l'initialisation des bords de la matrice de coût que nous allons construire.
Matrice de coûts initial avec coûts initiaux à 0 et remplissage à mi-parcours de l'algorithme.

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

Chaque case correspond à un score (= longueur de la LCS au point courant de résolution) pour le sous-problème considéré. Le tableau se remplit de gauche à droite et de haut en bas.
Pour remplir une case, on considère les positions de case suivantes comme figurés avec les flèches.
V1 : Case à l'Ouest
V2 : Case au Nord
V3 : Case au Nord-Ouest

Dans la cellule à remplir, on affecte le maximum de :
V1
V2
V3+1 si C1=C2 et V3 sinon où C1 et C2 sont respectivement les valeurs de caractères correspondant en colonne et en ligne de la case considérée.

Parallèlement on garde pour chaque case une trace du chemin en affectant une flèche de type V1, V2 ou V3 en fonction de l'origine de la valeur maximale retenue. Ces flèches sont essentielles et vont permettre de réaliser la dernière étape de back-tracking pour reconstruire la LCS à la fin du calcul des scores.

Cas ambigus : En cas d'égalité de valeurs de case, faire un choix arbitraire mais toujours le même par exemple ambiguïté entre case V1 et V3 choisir V3 systématiquement. A la fin de l'algorithme le score final LCS(S1,S2) sera le même quel que soit ce choix systématique. En général privilégiez la case V3.

Pour le tracing back (principe de la programmation dynamique, c'est par essence un processus off-line), on part de la case en bas à droite et on remonte le chemin d'alignement selon l'algorithme suivant :
Si Cell.V3

Si Cell.V3.value = Cell.value -1, ajouter les caractères C1 (et C2 car C1=C2) correspondant à la cellule Cell courante

Sinon Si Cell.V3.value = Cell.value, on passe le caractère courant,

Si Cell.V1 ou Cell.V2

on passe le caractère C1 en colonne ou C2 en ligne respectivement.

L'alignement de séquence de type Needleman-Wunsch (NW) ou Smith-Waterman (SW) utilise le même principe de la programmation dynamique avec des fonctions de coût différentes (*mismatch,* **match** et *gap (-)*) pour remplir les scores (valeurs des cases) et obtenir un alignement plus renseigné

LCS :

GCCAG

Alignement Global (NW) :

S1 = **G** **C** *C* **C** T **A** *G* *C* **G**

S2 = **G** **C** G **C** - **A** *A* *T* **G**

Alignement local (SW) :

S1 = GCCCTA**GCG**

S2 = **GCG**CAATG

Perl, Java, C

Récupérez tous les fichiers ici : http://www.math-info.univ-paris5.fr/~lomn/Cours/BI/Data/LCS/

Tester le programme *Perl* suivant de calcul de la plus grande sous-chaîne commune entre deux chaînes de caractères (Longest Common Substring)

```
$perl LCS.prl "coucou" "corageco"

$perl LCS.prl "AGGGTTTTGGGAA" "AGCCCCGTTAT"
```

Compiler le code *LCS.c (regardez le fichier Usage.txt à l'occasion).*

```
$gcc -c LCS.c
$gcc -o LCS LCS.o
```

et exécuter le :

```
$./LCS "coucou" "corageco"
```

Compiler le code *LCS.java*

```
$javac LCS.java
```

et exécuter le

```
$java LCS ou $java -classpath . LCS
```

Et Python dans tout ça

La plupart des gros serveurs de logiciels bio-informatiques est écrit en C ou C++. BLAST était à l'origine écrit en C et maintenant il existe une version C++. Mais la plupart des petites applications écrites par les chercheurs (biologiste, bio-informaticien, biologiste computationnel) sont écrits en Perl. Sans doute car la plus grande base de ressources bio-informatiques, Bioperl, est écrite en Perl. Mais Python (et BioPython) gagne du terrain de façon exponentielle.

Perl ou Python ? Ou autre

https://www.biostars.org/p/2737/ - 2011

« CPAN do your life more easy, and I think that is one of the missing pieces in Python. »– 2013

« There are also other languages to consider such as Ruby, with the advantage that is even newer than Python and it uses Gems, so it is easy to install new modules, or Java, perhaps more difficult to learn, but with much better performance than Perl, Python or Ruby. Perhaps we can get one idea of the use of these languages in the scientific community looking into number of citations of Bioperl (Stajich JE et al 2002), Biopython (Cock PJA et al 2009), Bioruby (Goto N. et al 2010) and Biojava (Prlic A et al. 2012) in 2012. Bioperl 128, Biopython 75, Bioruby 20 and Biojava 2. So I agree with Dr. Podicheti about Python is catching up Perl, but for now is one of the most used languages in bioinformatics. »

« Perl has a problem. It kind of encourages writing a short program which next month is faster to write again than understand what it's doing. One needs discipline to write readable code. Python, on the other hand, encourages readability. For me and my friends, ease of comprehending code written in Python was the reason to abandon Perl. Both own code and somebody's else. »

« Personally I am drifting away from Perl. I started out with Perl, I still own Perl for its support for me. But I find it is very difficult to transit from small programs to big programs in Perl. This can be done via OOP, but jumping from procedural Perl to OOP Perl was a pain for me. It may not be a problem to someone with good background in OOP, I started out with procedural, and Perl is not a good OOP starter. I can learn OOP with other languages, i.e. Python or Java and then come back to Perl? Not quite so, it is better learn Python and Java and keep using them if possible. »

« In addition, it is Big Data age, and to my experience, Perl is not doing much to it. Last time I tried with parallel and distributed Perl modules, they are very difficult to start up. I felt one has to be a hardcore Perl programmer and quite a geek in computer science to make use of them. At least that is what the authors of those modules are :-) On the other hand, it is much easier for me to understand and write parallel programs in Python or Java. The cause might be traced back to Perl's OOP and readability. »

« The fasta and genbank parsers of python are great for instance, but I think the phylo module in bioperl is better than the one in biopython (at least for the moment). »

« One place I find Perl useful is the command line, because of its stream processing and in-place editing capabilities. For me it is easier to use than awk and sed. To do things like taking the first, fourth and fifth column of a GFF file and writing them out in a chromosome:start-end format instead: $ cat myfile.gff | perl -p -i -e "substitution/regexp/here/"
Maybe Python and other languages have these possibilities too, but the terse code of Perl is not only more tolerable on the command line for me, it is outright desirable there. »

« I am a masters student and I have worked with both Perl and Python. Python was easy for me to learn than Perl. but when I learnt Perl i felt it was so useful in Bioinformatics to analyse sequences and others. But when i learnt python and started working with it I felt it is more easy to dynamically segment the data and work simultaneously with the data. also its easy because I feel Python handles BIGDATA better than Perl. Also with the various packages and its interface with R to do statistics Python is what i prefer and feel is better. But I would prefer Bioperl over BIopython. »