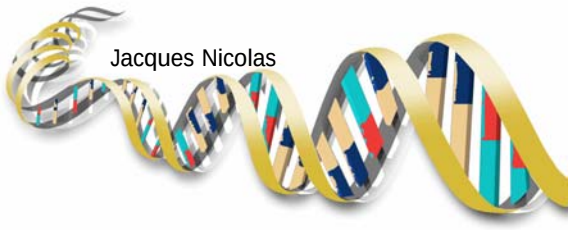
Algorithmes de recherche de motifs dans les séquences

Liffré Novembre 2005

Jacques Nicolas



IRISA

Participants

LEROY	Hugues	ing. rech.	Symbiose
ASSI	Anthony	ing. expert	Symbiose
BIMBOT	Frédéric	chercheur Cnrs	Metiss
GUESDON	Maxence	ing. rech. Rocq	Miriad, Cristal
CENAC	Peggy	doctorante	Rocq Preval
CORDIER	Marie-Odile	professeure	Dream
BAYOUDH	Sabri	doctorant	Cordial
QUINIOU	René	chercheur Inria	Dream
ROSSI	Vivien	postdoc	Aspi
GAIGNARD	Alban	ing. associé	Visages
ILLANES MANRIQUEZ	Alfredo	doctorant	Sosso 2
SERICOLA	Bruno	chercheur	Armor
TUFFIN	Bruno	chercheur	Armor
CAMBEFORT	Jeanne	ingénieure	Symbiose

Recherche de motifs (String matching)

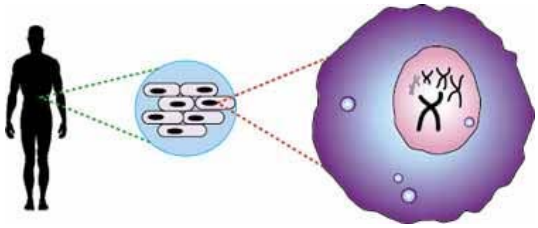
- **Problème fondamental dans de nombreux domaines**
  - Initialement recherche dans des fichiers ou des documents;
  - Regain d'intérêt depuis le web et la génomique.
- **Problème de complexité faible** (algorithmes linéaires en temps/espace) **mais algorithmique sophistiquée** (difficile à implémenter).
- **Recherche exacte de mots ok, erreurs** (fréquent en biologie) **et motifs plus difficiles à prendre en compte.**

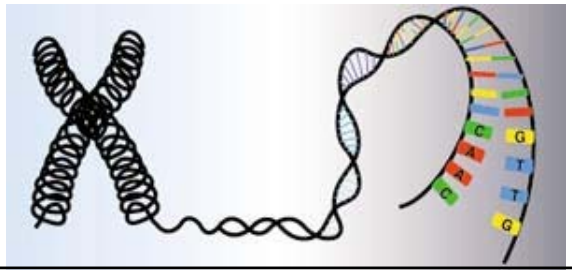
Le cours d'aujourd'hui

- Introduit rapidement la biologie moléculaire;
- Présente les concepts fondamentaux de la recherche exacte de chaînes;
- Propose un choix des meilleurs algorithmes;
- Termine par quelques problèmes de recherche.

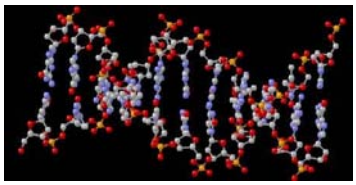
Chromosomes: le support de l'hérédité

- 10 ^{14} cellules chez chaque individu
- Eukaryotes : chromosomes dans un noyau (sinon prokaryote bactéries +archées)
- Homme : génome= 23 paires de chromosomes

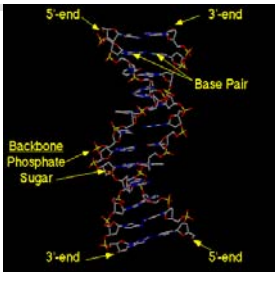


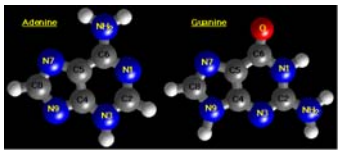


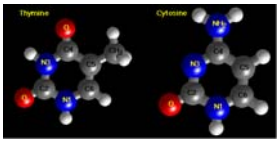
Structure de l'ADN : une double hélice



Deux brins complémentaires orientés







Alphabet à 4 lettres: Purines A et T Pyrimidines C et G

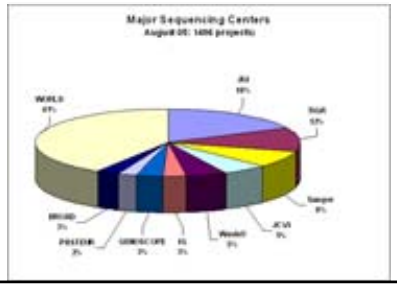
Le séquençage

5'P 3'OH
ACGTATGCCCATACGCGCG
TGCATACGGGTATGCGCGC
3'OH 5'P

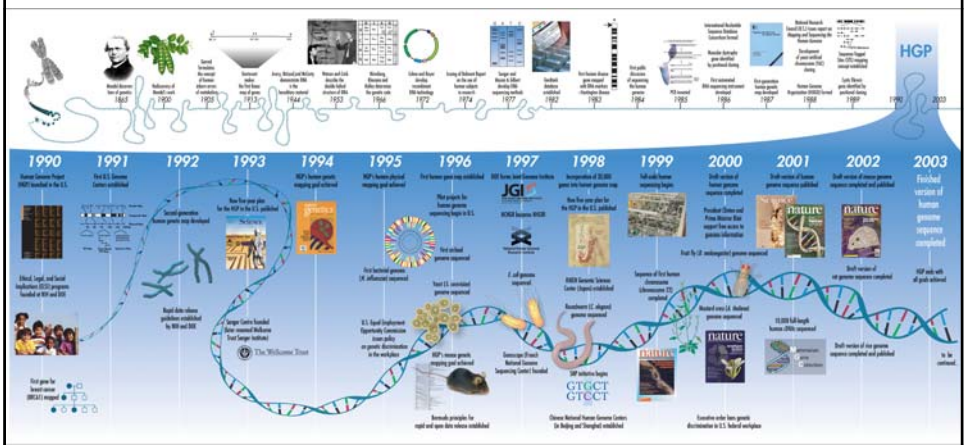
- Séquencer un génome, c'est obtenir la série de nucléotides des chromosomes;
- De nombreux génomes sont maintenant disponibles.
- Combien de génomes complètement séquencés ?

Données Gold oct. 2005 :
315 génomes complets

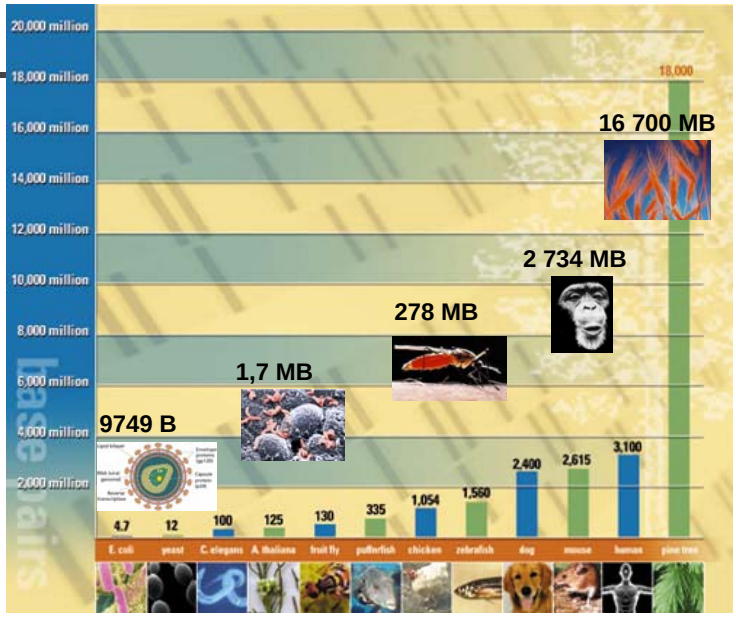
804 prokaryotes,
547 eukaryotes en cours



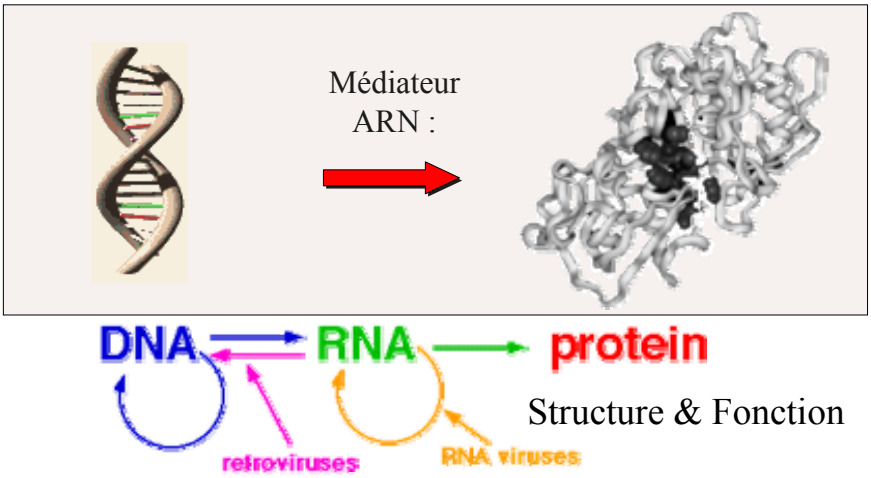
HGP : le projet génome humain



Taille de quelques génomes

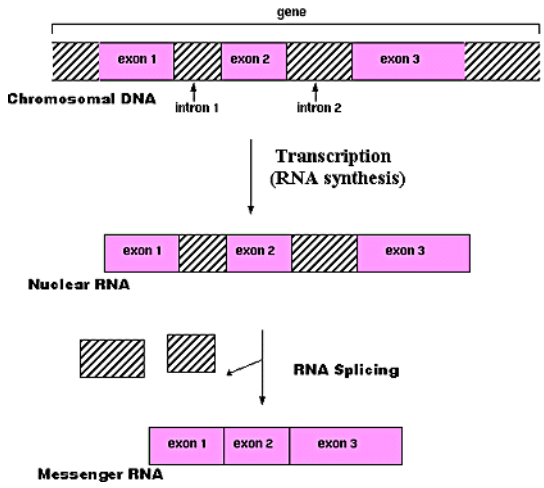


Les Gènes codent pour des protéines



Structure & Fonction

Transcription : synthèse de l'ARN

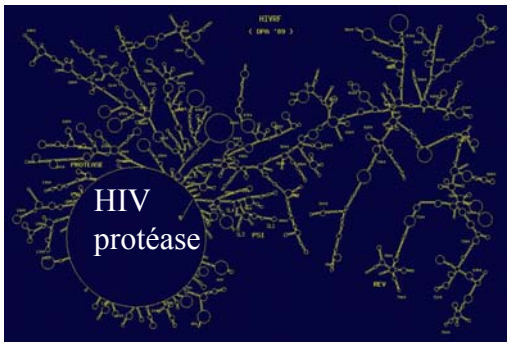


Alphabet équivalent

A, U, G, C

Structure de l'ARN : simple brin et palindromes...

5'P AUCGGCUUAGACGAUGAAGCCGUCCCGGAAA 3'OH

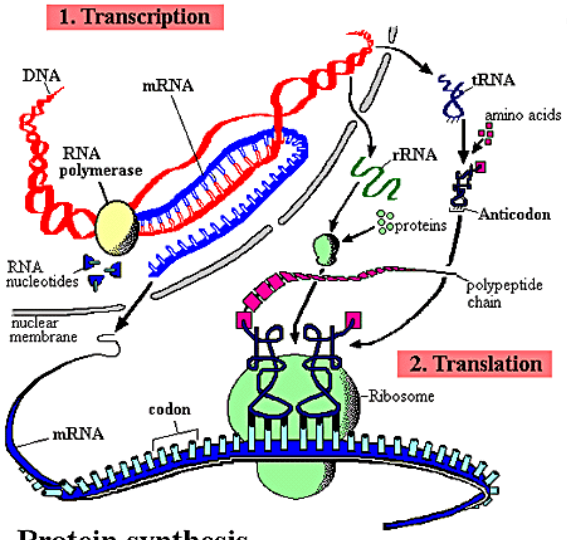


A
G A
U G
A C
G C
C G
A U
G C

5'P AUCGGCUUA CCGGAAA 3'OH

Traduction : synthèse des protéines

Alphabet des acides aminés : 20 lettres



Protein synthesis

Le code génétique

	U	C	A	G	
U	UUU Phe	UCU Ser	UAU Tyr	UGU Cys	U
	UUC Phe	UCC Ser	UAC Tyr	UGC Cys	C
	UUA Leu	UCA Ser	UAA	UGA	A
	UUG Leu	UCG Ser	UAG	UGG Trp	G
C	CUU Leu	CCU Pro	CAU His	CGU Arg	U
	CUC Leu	CCC Pro	CAC His	CGC Arg	C
	CUA Leu	CCA Pro	CAA Gln	CGA Arg	A
	CUG Leu	CCG Pro	CAG Gln	CGG Arg	G
A	AUU Ile	ACU Thr	AAU Asn	AGU Ser	U
	AUC Ile	ACC Thr	AAC Asn	AGC Ser	C
	AUA Ile	ACA Thr	AAA Lys	AGA Arg	A
	AUG Met	ACG Thr	AAG Lys	AGG Arg	G
G	GUU Val	GCU Ala	GAU Asp	GGU Gly	U
	GUC Val	GCC Ala	GAC Asp	GGC Gly	C
	GUA Val	GCA Ala	GAA Glu	GGA Gly	A
	GUG Val	GCG Ala	GAG Glu	GGG Gly	G

L'art des mots...

Quelques notations sur les mots

- A word is a finite suite of letters, elements of a finite non empty alphabet.
- |S| or len(S) denotes the length of word S.
  The empty word, of size 0, is denoted ε or λ .
  Positions in a word start from 0 : S[0] is the 1st letter of S.
- A word x is a subword (substring) of a word y if there exists 2 words u and v such that **y=uxv**. A proper subword is an non empty subword.
  If u= ε, then x is a prefix, if v= ε , then x is a suffix of y.
  More generally, x is a subsequence of y if there exists |x+1| words w_i such that y= w_0 x[0] w_1 x[1] w_2 … $w_{|x-1|}$ x[|x-1|] $w_{|x|}$
- S[a..b] is an occurrence of a subword of S between positions a and b.

Recherche du mot ATA

```
S=      GATATAAATACATATATG
          ATAT          ATATAT
           1            11  13

S[0]=G,  S[1]=A,   S[1..4]=ATAT     len(S)=18
```

Deux notions fondamentales: periode et bord

- Period of word x : Positive integer p such that

$$\forall i \in [0, |x| - p - 1],\ x[i+p] = x[i]$$

The period of x is its smallest period.

- Border of word x : non empty word which is a proper prefix and suffix of x.
  The border of x is the largest border of x.
- If p is the period of x and b the border of x, then p=|x|-|b|

Exercice

- Quels sont les facteurs de taille 3 du mot aaababbbaa ?
- Trouvez toutes les périodes du mot babbababbab.
- Trouver la taille des bords des préfixes du mot abbabaabbabb.

Solution

- aaa, aab, aba, abb, baa, bab, bba, bbb
  1 occurrence, all words on {a,b} : word of de Bruijn.
- babbababbab 5
  babbababbab 8
  babbababbab 10
  babbababbab 11
- abbabaabbabb
  ---121123453
  period of abbab : 5-2=3, period of the word : 12-3=9

Complexité : hypothèses d'opérations en temps constant

- Access to a position in a sequence of size n;
- Comparison of 2 letters or 2 numbers $\leq$ n;
- Boolean operations on $o(\log_2(n))$- bit vectors where n is the length of the sequence.

Exercice : complexité

- What is the complexity (in time and space) of a naive string matching approach for the search of a word of length m in a sequence of length n ?

- How many time will require the naïve search of a 1.6kb segment in the human genome (3. 10^9 b) with a computer doing 300 millions letter comparisons per second ?

Recherche exacte de chaîne : algorithme naïf

Naive Exact string matching (Pattern,Sequence):

n

For each Position in Sequence

Take a Substring of length
the length of the pattern at Position in Sequence

m

if for each letter of the Pattern
Substring and Pattern are equal
then print Position

Complexité : o(m) space, o(n.m) time 16000s ~4H

Recherche exacte de chaîne : deux voies

- Pretreatment of the pattern
  - one marks multiple starts of the pattern in it;
  - algorithm in O(n+m) (typically sublinear !);
  - Knuth-Morris-Pratt 1977 (Aho-Corasick 1975), Boyer-Moore 1977 (Horspool 1980, Apostolico-Giancarlo 1986), BDM 1994 (BOM 2001).
- Pretreatment of the sequence
  - one builds the dictionary of words in the sequence;
  - algorithm in O(m) + initial step in O(n);
  - suffix tree : McCreight 1976, Ukonnen 1995.

Prétraitement du motif

Three types of approaches exist

1. Reading of the text, one character at a time, updating variables and doing inclusion test on pattern prefixes at the current position (Knuth Morris Pratt)
2. Reading of the text with a sliding window and inclusion test on pattern suffixes in this window. (Boyer Moore)
3. Reading of the text with a sliding window and inclusion test on pattern subwords or reverse prefixes in this window. (BDM)

Prétraitement du motif
Quelques notations

But :
Calculer tous les départs possibles (via Z_i) du mot

Z_i
longueur du plus long mot commun aux positions 0 et i≠0

[li , ri]
Position du bord le plus à droite calculé par Z_j , j= 1 à i

Prétraitement du motif, un exemple

ATAT

AT présent en 0 et 2

$Z_1= 0,\ Z_2= 2,\ Z_3= 0$

[l1,r1]= [0,0] [l2,r2]= [l3,r3] = [2,3]

The naive algorithm computing Z_i works on a smaller string, but has the same complexity than the initial problem : m^2

How to compute it linearly with respect to m?

Exercice

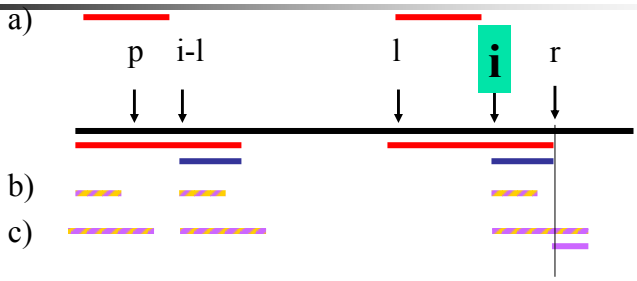
- Calculer les valeurs de Z_i pour le motif attataattataa.

Solution

➢

```
attataattataa
 002017002011
```

Comment calculer Z_i incrémentalement



a) Z-word at l (i.e. r) does not reach i : compare from i

b) Z-word at l (i.e. r) passes i and $Z_{i-l} < r-i$: do nothing !

c) Z-word at l (i.e. r) passes i and $Z_{i-l} \geq r-i$: compare from r

Prétraitement du motif, algorithme

```
def update_Z (Seq,Z,i,l,r):
    % Compute Z[i], updates  initial & final positions l and r
If  (i>r) : % a) Complete comparison necessary
      Z[i]  = prefix length (Seq,Seq,i, length(Seq)-i+1, 0)
      if non empty common word  :  [l, r] = [i, i+Z[i]-1]

  else if  (Z[i-l] < r +1 –i ) : % b) no comparison
        Z[i]  = Z[i-l] % No update of l and r

  else : % c) comparison from the previous r
        q = prefix length (Seq,Seq,r+1, length(Seq)-r, r+1-i)
        Z[i]  = q+r+1-i ;      [l, r] = [i, r+q]
```

Exercice

- Quelles sont les valeurs de Zi, l_i et ri pour la séquence **ATATCATATA ?**

Exemple de prétraitement

ATATCATATA

i:1 Z[1]=0 , l=0, r=0
i:2 Z[2]=2, l=2, r=3

Case a)

Current rightmost Zbox

AT[AT]CATATA Case b)

i:3 l=2, r=3, i-l=1, r+1-i=1, Z[1]<1, Z[3]=Z[1] =0

AT(AT)C[AT{AT]A} Case c)

i:7 l=5, r=8, Z[2]=2, q=1 Z[7]=1+8+1-7=3 , l=7, r=9

Table des résultats

i	Z_i	new l	new r	i-l	r+1-i	case
1 (T)	0	0	0			a)
2 (A)	2	2	3	2	-1	a)
3 (T)	0	2	3	1	1	b)
4 (C)	0	2	3	2	0	a)
5 (A)	4	5	8	3	-2	a)
6 (T)	0	5	8	1	3	b)
7 (A)	3	7	9	2	2	c)
8 (T)	0	7	9	1	2	b)
9 (A)	1	7	9	2	1	c)

Sur le nombre de comparaisons dans update_Z

If **(i>r)** : % a) Complete comparison necessary
Z[i] = prefix length (Seq,Seq,i, length(Seq)-i+1, 0)
if non empty common word : [l, r] = [i, i+Z[i]-1]
r increases by the number of good comparisons

else if **(Z[i-l] < r +1 –i)** : % b) no comparison
Z[i] = Z[i-l] % No update of l and r
r stable and no comparison

else : % c) comparison from the previous r
q = prefix length (Seq,Seq,r+1, length(Seq)-r, r+1-i)
Z[i] = q+r+1-i ; [l, r] = [i, r+q]
r increases by the number of good comparisons

Complexité linéaire du prétraitement

Pretreatment is linear with respect to the length m of the sequence, independent of the length of the alphabet.

- m loops on update_Z ;
- Other loops are calls to prefix length, making a number of comparisons;
  - r increases each time by the number of successful comparisons and r is bounded by m.
  - at each step, there is at most 1 failure comparison, that is at most m altogether.

Un algorithme linéaire
(Main & Lorentz 1984)

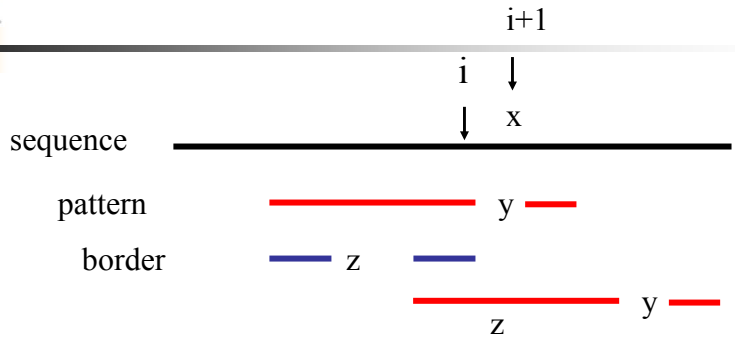
- Append pattern, $,and sequence;
- Compute Z on the whole sequence;
- Print positions where Z=m.

```
ATAT$GATATAAATACATATATG
 02000403011301040400200
```

Note sur l' algorithme équivalent KMP

- KMP, in a similar way, looks at every position in the sequence for the longest prefix of the pattern that is also a suffix of the sequence.
- In practice, these algorithms are not the most efficient. They are overperformed by the « subword » approach for long patterns and the « numerical » approach for short patterns.

Principe de KMP



$x \neq y, \quad y \neq z$:

In case of matching failure on y, it is sufficient to restart after the border of the correct prefix (MP), or better, a smaller border starting with a different letter (KMP)

Exercice : chromosomes circulaires

La plupart des chromosomes bactériens sont en fait circulaires

Comment rechercher un motif dans ces conditions conditions ?

Solution

Travailler sur la séquence CC !
(longueur C+ motif suffisant)

(pour d'autres problèmes, ce n'est plus forcément si simple…)

Les meilleurs algorithmes actuels

Approche Numérique

Start from a binary representation of words.

« Shift-and » or « shift-or » method depending of used boolean operators.

Shift-or is slightly better than shift-and, working on the complement, but slightly less easy to explain.

Notations pour la méthode Shift-And

- R. Baeza-Yates, G. Gonnet 1992.
- Efficient search of short patterns.

- Alphabet of size t, coded from 0 to t-1. Pattern of size m.
  Looking for a match of a prefix of the pattern
  ending at current position k in the sequence.

- Data structures : Bit Matrix I(m,t) (Vector of t integers)
  on the pattern Bit Vector M(m) (Integer)

- I[i,j]=1 iff pattern [i]==j,
- M[i]=1 iff pattern [0..i]==Seq[k-i..k]

Beware,
position 0 right !

Structures de données du Shift-And

ATAT Pattern

ATATCATATA Sequence

I

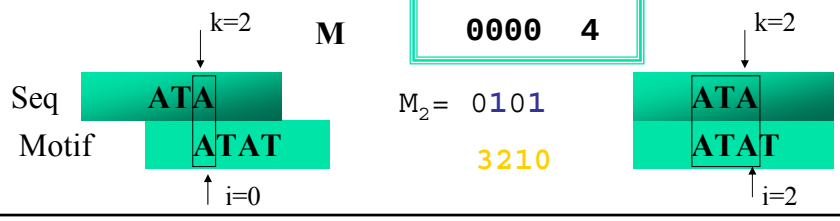
	TATA
A	0101
C	0000
G	0000
T	1010
	0000

0	
1	
2	
3	
4	

M

M_2= 0101

3210

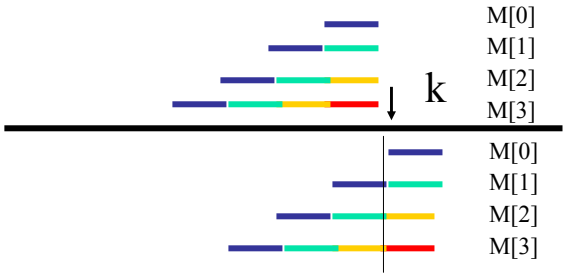


«Shift-And» : l' algorithme

- Let's consider an increasing position k in sequence Seq
- $M=0^m$; $I=0^{m}$; for i from 0 to m-1, I[Pattern[i]]= I[Pattern[i]] or $0^{m-i-1}10^i$
- For k from 0 to n-1

  M = (*1 Or* M<<1) *And* I [.,Seq [k]]

  If M[m-1] , then pattern has an occurrence at k



Exercice

Essayer le Shift-And algorithm et donner les valeurs de M pour chaque valeur de k.

ATAT	**ATATCATATA**
Motif	Séquence

Méthode « Shift-And » : Exemple

ATATCATATA

	TATA
A	0101 0
C	0000 1
G	0000 2
T	1010 3
M	0000 4

K=

0 A M= 0001∧0101= 0001
1 T M= 0011∧1010= 0010
2 A M= 0101∧0101= 0101
3 T M= 1011∧1010= 1010
4 C M= 0101∧0000= 0000
5 A M= 0001∧0101= 0001
6 T M= 0011∧1010= 0010
7 A M= 0101∧0101= 0101
8 T M= 1011∧1010= 1010
9 A M= 0101∧0101= 0101

L'approche par facteurs

- G. Navarro & M. Raffinot 2000 :
  BNDM (Backward Nondeterministic Dawg Matching)
  Improvement of algorithm BDM of Crochemore & al. 94 with a numerical approach.

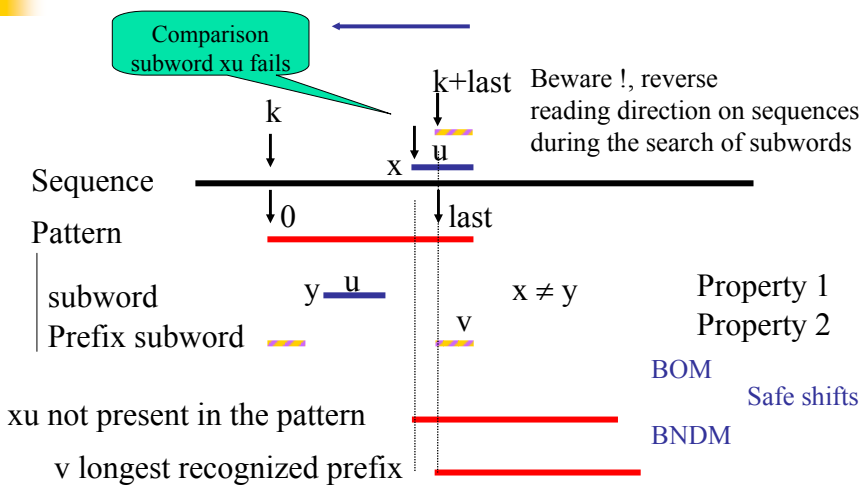
- The idea is to build and use an automaton recognizing all subwords of the pattern.
- Worst case complexity o(n.m) but on average (one assumes that the probability of occurrence of each character is equal and that letters of the sequence are independent) **sub-linear : o(n. log $_{|\Sigma|}$m/m)**

Principle of the subword approach

- Property 1
  Given a pattern p of size m and a sequence Seq, if there exists $0 \leq i < m$, Seq[k+i..k+m-1] is not a subword of p, then there exists no match of p in Seq between position k and k+i.
  (contrapositive of if p matches w, then all its subwords match w).

- Property 2
  Given a pattern p of size m and a sequence Seq, if there exists $last < m$ such that for all i such that $0 \leq i < last$, Seq[k+i..k+m-1] is not a prefix of p, then there exists not match of p in Seq between position k and k+last-1.

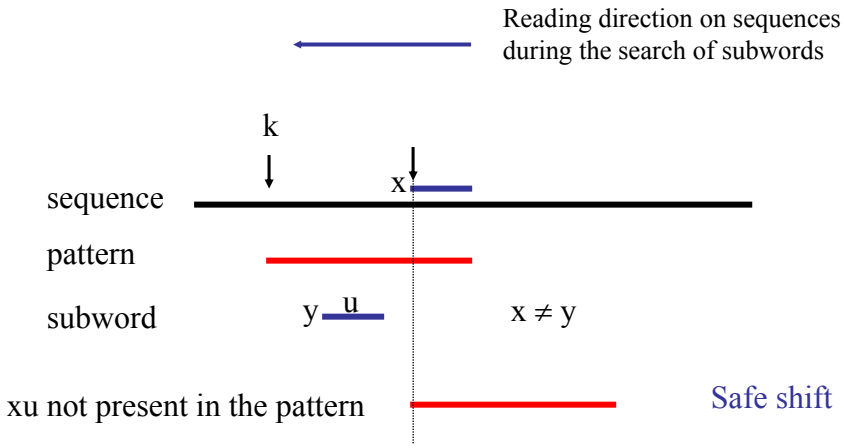
Principe de l'approche facteur



BOM : la structure de données d'oracle des facteurs

- Recent algorithm :
  reference A. Allauzen, M. Crochemore, M. Raffinot 2001
- Applicable for longer patterns (> 1 word of memory).
- Backward Oracle Matching idea : replace suffix automaton with factor oracle, simply indicating words that are not subwords, i.e. recognizing a superset of the set of subwords.
- The algorithm is o(n.m) worst case,
  efficient in practice, «conjectured optimal» on average.

Décalage correct dans BOM



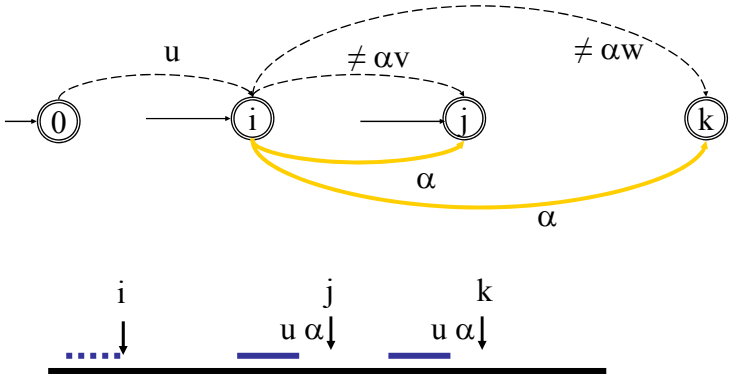
Oracle des facteurs: 1ère caractérisation

- C'est un automate déterministe uniquement défini par sa construction
  (on ne connaît toujours pas le langage qu'il accepte !)

  1. Build the maximal canonical automaton recognizing the pattern;
  2. For increasing state number i in the automaton,
     For each letter α in the alphabet such that no α–transition exist from i,
     if u is the shortest word regognized at i and uα is a subword ending at j, after position i in the sequence, add an α–transition from i to j.

- The factor oracle recognizes a (little) superset of the set of subwords. It has m+1 states and no more than 2m-1 transitions.
  It can be built in linear time.

Schéma



Oracle des facteurs: caractérisation en ligne

1. Build the maximal canonical automaton recognizing the pattern (internal transitions);

2. All Suffixes : All states are final states.
3. All prefixes : For each state from the initial state

- There is an backward link to the «closest» supply state with an in-transition with the same letter (default, initial state). Closest is with respect to the backward path from the previous state.
- Add for all states on the backward path an (external) transition with the same letter to the new state.

Algorithme construisant l'oracle

create-factor-oracle(word):

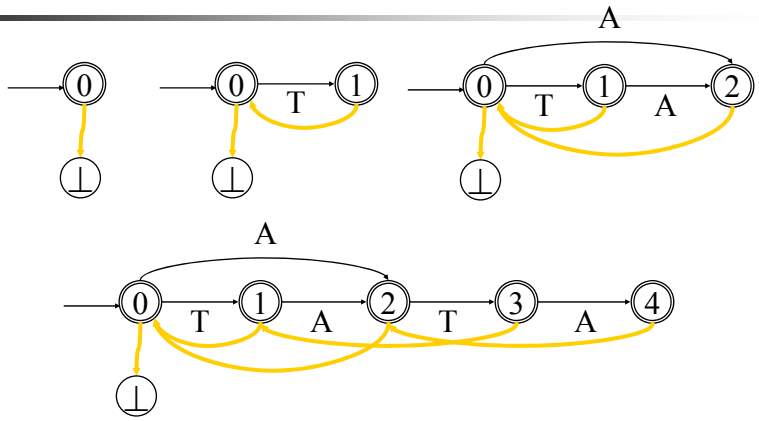
1. Create the initial state q_0; link(0):=⊥;

2. For i from 1 to m do
   1. Create state q_i; $\delta(q_{i-1}, word[i]):=q_i$;
   2. j:= link(i-1);
   3. While ($j \neq \perp$ and $\delta(q_j, word[i])= \perp$) do
      1. $\delta(q_j, word[i]):= q_i$; j:= link(j);
   4. od
   5. If j= ⊥ then link(i):= 0
      else link(i):= k / $q_k=\delta(q_j,word[i])$ fi
3. od

No word [i] transition from j

Go Backward on the link path

Oracle des facteurs pour reverse(ATAT)



The automaton recognizes just factors in this case
It recognizes just one word of the size of the pattern : the pattern itself.

Algorithme BOM

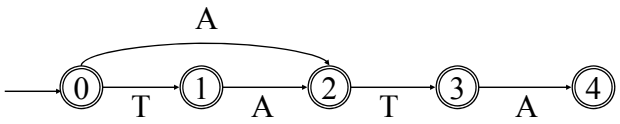
1. k:=0; create-factor-oracle(reverse pattern);
2. While (k ≤ n-m) do % window of size m at k on the sequence
    1. state:= q_0 ; i:= m-1 % initial state, end of window
    2. While (i≥0 & state ≠ ⊥) do % progress on the window
        2. state := δ(state,Seq[k+i]); i:=i-1 % and the automaton
    3. od
    4. If state ≠ ⊥ then write('pattern occurrence at', k)
    5. k:=k+ i +2 %slide the window on the next possible factor
3. od

Exercice

Run BOM algorithm and give the value of used oracle states and i for each value of k.

ATAT
Motif

ATATCATATA
Sequence



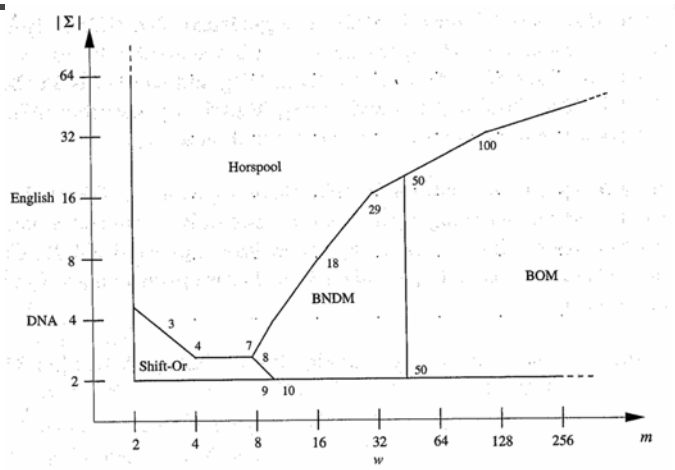
BOM : exemple d'exécution

- k= 0 window = ATAT, state=0, i= 3
  - state=1, i= 2 state=2, i= 1
  - state=3, i= 0
  - state=4, i=-1 occurrence at 0
- k= 1 window = TATC , state=0, i= 3
  - state= ⊥, i= 2
- k= 5 window = ATAT , state=0, i= 3
  - state=1, i= 2 state=2, i= 1
  - state=3, i= 0
  - state=4, i= -1 occurrence at 5
- k= 6 window = TATA, state=0, i= 3
  - state=2, i= 2 state=3, i= 1
  - state=4, i= 0 state=⊥, i=-1
- k=7

Comparaison empirique des algorithmes

Extrait de Navarro et Raffinot “Flexible pattern matching in strings” 2002

Size alphabet



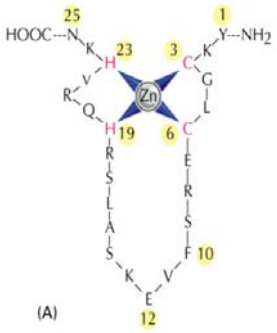
Size Pattern

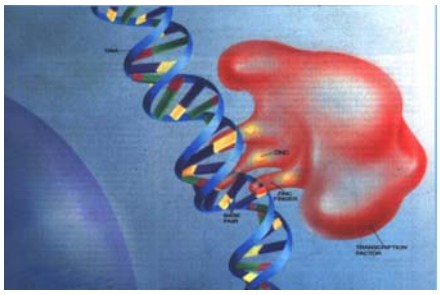
UltraSparc 32 bits Machine, Random Sequences 10Mbytes

Quelques problèmes de recherche de motifs en biologie moléculaire

- Recherche de motifs spécifiques, avec « don’t care » et « erreurs »;
- Visualisation du contenu de génomes;
- Recherche de « répétitions »;
- Recherche de modèles syntaxiques.
- Assemblage de séquences

Exemple de Motif de Protéine/ADN





Motif de la protéine « En Doigt de Zinc » :
utilisation de don't cares pour modéliser les distances.

HVRQH-X(10,20)-CLGC

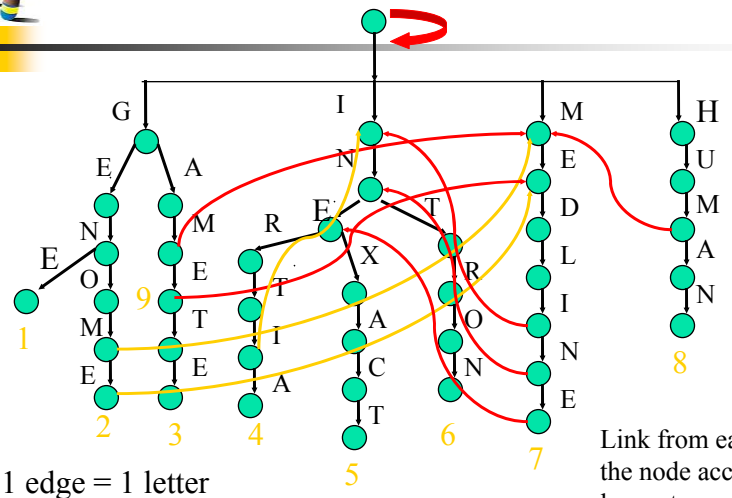
Idée de résolution

- Consider the set of k patterns delimited by Xs;
- Look in parallel for the k patterns;
- Adapt Aho-Corasick's multiple matching algorithm, to manage shifts.

Recherche de chaînes multiples

- If one has to search for a set of k patterns, a trivial solution is to repeat k times the previous algorithm : complexity o(k(n+m)).
- For a large number of patterns (a dictionnary to check or code the sequence), it is possible to design better algorithms, with a complexity not depending on k, with the help of a pretreatment of the set of patterns.
- The most useful structure for this is a pattern tree with suffix links

Arbre des motifs avec liens suffixes

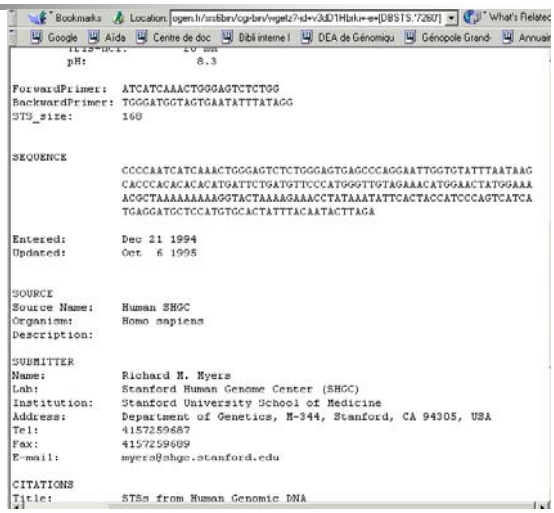


1 edge = 1 letter
1 node = word read on path from root node
1 number= 1 word

Link from each node to the node accepting the longest proper suffix of the word on this node.

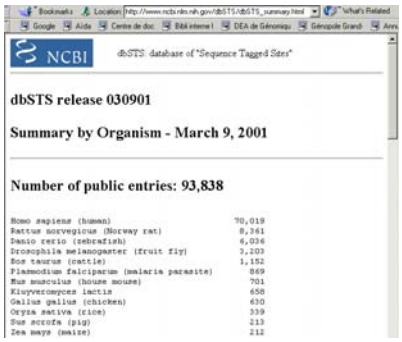
Application : recherche de tags

STSs
(Sequenced Tagged Site)
are small DNA sequences
150-300 bases long
whose extremities
(20-30 nucleotides) are
unique in a whole genome



```
        pH:                 8.3

ForwardPrimer:  ATCATCAAACTGGGAGTCTCTGG
BackwardPrimer: TGGGATGGTAGTGAATATTTATAGG
STS_size:       168

SEQUENCE
                CCCCAATCATCAAACTGGGAGTCTCTGGGAGTGAGCCCAGGAATTGGTGTATTTAATAAG
                CACCCACACACACATGATTCTGATGTTCCCATGGGTTGTAGAAACATGGAACTATGGAAA
                ACGCTAAAAAAAAAAGGTACTAAAAGAAACCTATAAATATTCACTACCATCCCAGTCATCA
                TGAGGATGCTCCATGTGCACTATTTACAATACTTAGA

Entered:        Dec 21 1994
Updated:        Oct  6 1995

SOURCE
Source Name:    Human SHGC
Organism:       Homo sapiens
Description:

SUBMITTER
Name:           Richard M. Myers
Lab:            Stanford Human Genome Center (SHGC)
Institution:    Stanford University School of Medicine
Address:        Department of Genetics, M-344, Stanford, CA 94305, USA
Tel:            4157259687
Fax:            4157259689
E-mail:         myers@shgc.stanford.edu

CITATIONS
Title:          STSs from Human Genomic DNA
```

Recherche de tags : taille des données



NCBI dbSTS: database of "Sequence Tagged Sites"

dbSTS release 030901

Summary by Organism - March 9, 2001

Number of public entries: 93,838

Homo sapiens (human)	70,019
Rattus norvegicus (Norway rat)	8,361
Danio rerio (zebrafish)	6,036
Drosophila melanogaster (fruit fly)	3,203
Bos taurus (cattle)	1,152
Plasmodium falciparum (malaria parasite)	869
Mus musculus (house mouse)	701
Kluyveromyces lactis	658
Gallus gallus (chicken)	630
Oryza sativa (rice)	339
Sus scrofa (pig)	213
Zea mays (maize)	212

Jan 2005 **gbdiv sts[prop] AND**

Homo sapiens	117 925
Mus musculus	120 837
Canis	61 328
Gallus	1 354
Maize	59 271
Arabidopsis	28 584

- Given a new sequence, locate it in the whole genome, wrt these STSs.
- Direct application of set matching, where an algorithm with complexity not depending on the number of STS is welcome !

Aho-Corasick's algorithm

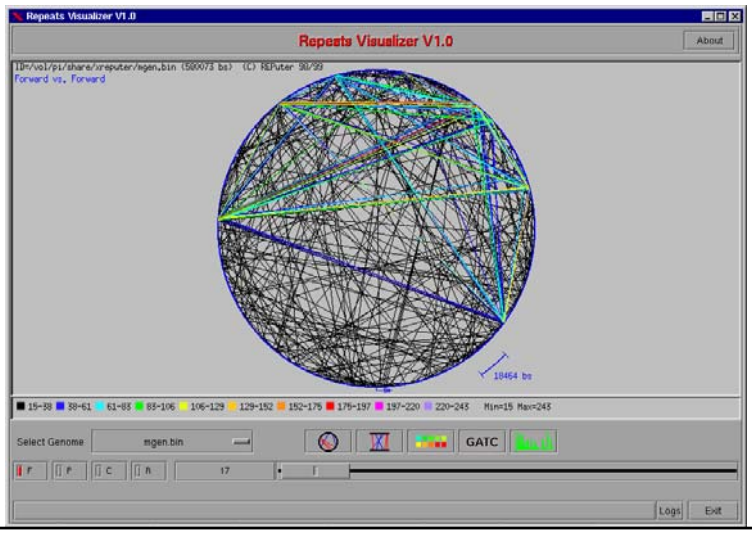
Start from the root of the tree and the beginning of the sequence;

While it is possible, go down in the tree with the current letter in the sequence;

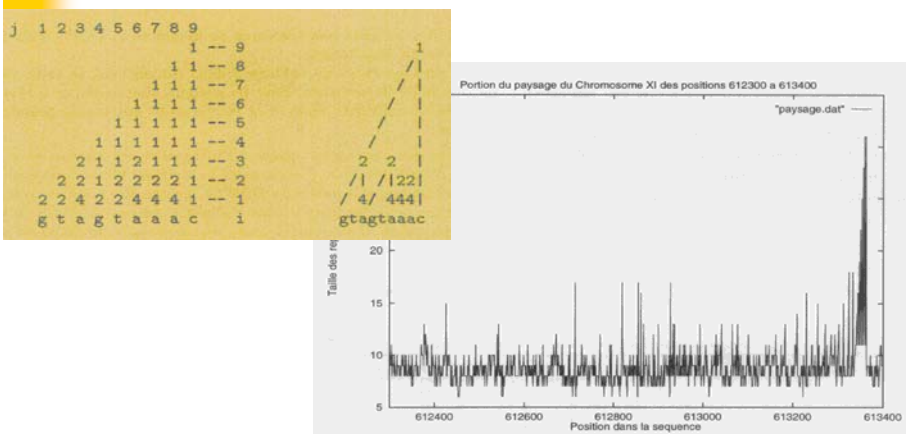
If a labeled node is reached
output presence of a pattern;

Follow the suffix link
to reach the new starting node

Visualisation de génomes (Reputer)



Visualisation de génomes (paysages)



« Landscapes »: Number of occurrences of subwords along the sequence
(B. Clift & al NAR 86, thèse R. Verin Univ. Marne la vallee 98)

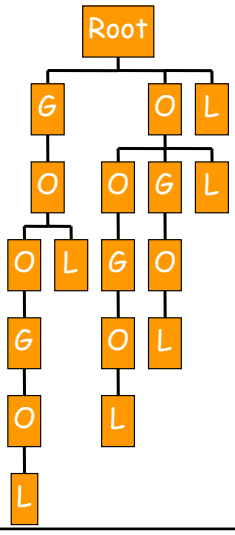
Elementary suffix tree

Aim : a dictionary of all subwords of a sequence

- It is the pattern tree of suffixes of a sequence (Trie).
- Leaves of the tree refer to positions in the sequence.
- Remark :
  Subwords may label either the nodes or the branches of the tree

An elementary Suffix Tree

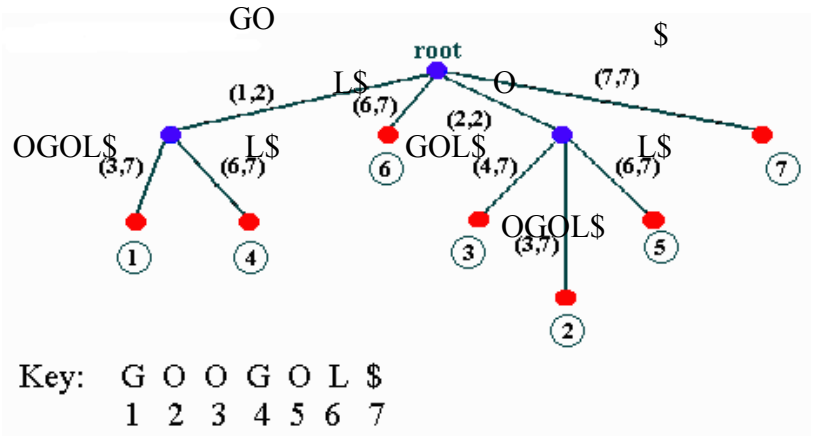
Example : GOOGOL



Compact suffix tree (ST)

- It is the elementary suffix tree of a word W where all branching nodes of degree 1 have been compacted.
- It is the deterministic automaton recognizing the set of suffixes of M, where :
  - There are n final states;
  - Each state that is neither initial or final has at least 2 successors;
  - Transitions are labeled with a subword of W
- In order to get unique suffixes, one considers the string W$, instead of W, where $ is a new letter.

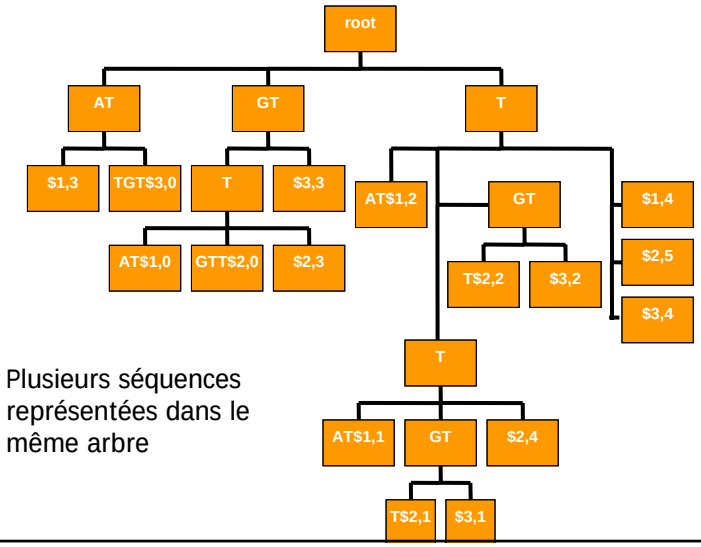
Exemple de ST sur GOOGOL



Subwords may be also represented at the level of nodes

Arbre des suffixes généralisé (GST)

GTTAT, GTTGTT et ATTGT

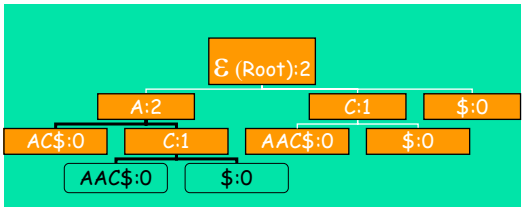


Plusieurs séquences représentées dans le même arbre

Exercice :
Le GST au secours de la contamination

- When one purifies, clones or sequences samples of DNA or proteins in a laboratory, there may be a possible contamination of these samples with the experimenter, the host organism or used products.
- In most cases, sequences of contaminants are known (keratines, yeast, enzymes…). How to filter the produced sequences ?
- It is sufficient to build the GST of potential contaminants and the produced sequence and to filter from it all positions corresponding to a common subword (two leaves with the same label, one from the sequence and the other from a contaminant sequence).

L'arbre des suffixes est une structure "à tout faire"



• A calculus on attributes may be associated to each node : see example of longest repeat with ACAAC$

• A great variety of usages : string matching, dictionary, common subsequence, or on the contrary, superstring , compression ….

Exercice : attributs de ST & GST

How to compute the following entities, using attributes within ST or GST ?

- Longest repeat;
- Length of the longest common word of 2 sequences (Knuth's conjecture in 70 : the problem is not solvable in linear time !)

Solution : attributes de ST et GST

- Longest repeat :
  Synthesized attribute S on internal nodes
  S(Parent)= if (leaf, 0, max $_{Child}$ (S(Child))+ length(Parent))

- Length of the longest common word of 2 sequences :
  Synthesized attributes O, T and S on nodes of the GST
  O(Parent)= if(leaf, pointer 1st sequence, or $_{Child}$ (O(Child)))
  T (Parent)= if(leaf, pointer 2nd sequence, or $_{Child}$ (T (Child)))
  S (Parent)= if(not(O(Parent) and T(Parent)), 0,
  max $_{Child}$ (S(Child))+ length(Parent))

L'arbre des suffixes est une structure de données efficace

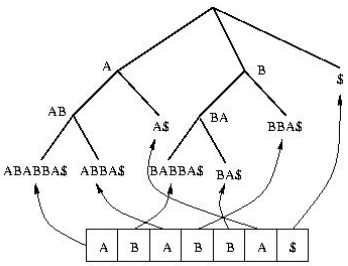
- The size of the sequence is n, size of pattern is m
  - Construction of the tree in O(n) ;
  - Memory space complexity O(n) ;
    (Max 2n+1 nodes, 2n edges, obtained for a « comb » on A^n)
  - String matching in O(m).

- From a practical point of view, considering the value of constants α and β in the complexity formula $\alpha n+\beta$ is important, particularly for memory space.
  The structure is efficient only if it is stored in main memory !
  Several variations exist, more compact (array, automaton…)

Looking for a pattern in a suffix tree

Go down in the tree with the pattern

- If it is not possible : no match
  AA
- If one reaches a leaf : match at position given in the leaf
  ABB
- Else : multiple match, with occurrences pointed by leaves under the reached node
  A

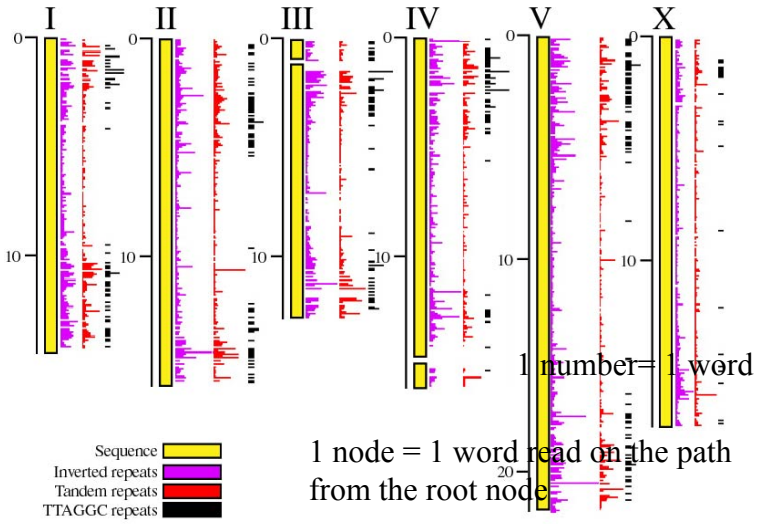


Ukkonen's construction algorithm (1995)

Systematic, lazy exploration of subwords Seq[j..i] :

- 1st loop i : 0..n ([0..i])
  building a suffix tree for increasing prefixes $pref_i$
- 2nd loop j : 0..i ([j..i])
  inserting decreasing suffixes $suf_{j..i}$ (letter Seq[i])

Importance des repeats génome de C. elegans



Implication des répétitions dans les maladies

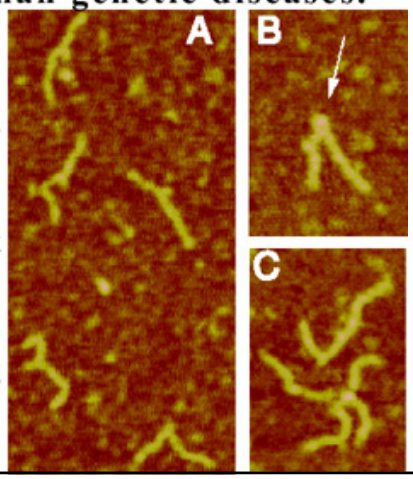
(CTG)n•(CAG)n repeat expansion leads to human genetic diseases.

$(CTG)_{23}$ and $(CAG)_{23}$ loopouts

A. single CTG loopout

B. double loopout, 30 bp spacing

C. dimer molecule



Ubiquité des répétitions dans le génome : exemple du prion

```
>prion2     cac cgc cag
ccc agc tac ccc cgc cag
ccc ggc tac ccc caa aat
ccc ggc tat ccc cat aat
ccg ggc tac ccc cac aac
ccg ggg tac ccc cac aac
ccg ggg tac ccc cac aac
ccg ggg tac ccg cag aac
cct gga tgg

>prion1     cat cgc cag
ccc agc tac ccc cgc cag
ccc ggc tac ccc caa aat
ccc ggc tat ccc cat aat
ccg ggc tac ccc cac aac
ccg ggg tac ccc cac aac
ccg ggg tac ccc cac aac
ccg ggg tac ccc cag aac
cct gga tgg

>owl        cat cgc cag
ccc agc tac ccc cgc cag
ccc ggc tac ccc caa aac
ccc agc tat cct cat aac
ccg ggg tac ccc cac aac
ccg ggg tac ccc cac aac
ccg ggg tac ccc cac aac
ccg ggg tac ccc cac aac
ccg ggg tgg

>crane      cac cgc cag
ccc agc tac ccc cgc cag
ccc ggc tac ccc caa aat
ccc ggc tac ccc cat aat
ccg ggg tac ccc cac aac
cct ggc tat ccc cac aac
cct ggc tat ccc cac aac
ccg ggg tgg

>condor     cac cgc cag
ccc agc tac ccc cgc cag
cct agc tac ccc caa aat
ccc ggc tat ccc cat aat
cca ggc tac ccc cac aac
ccg ggg tat ccc cac aac
ccg ggg tac ccc cac aac
cca ggg tgg

>duck       cac cgc cag
ccc agc tac ccc cgc cag
cct ggc tac ccc cag aat
cct ggt tat ccc cac aac
ccg ggg tat ccc cac aac
ccg ggg tac ccc cat aac
cca ggg tac ccc cac aac
ccc ggc tgg

>chickenXL  cat cgc cag
ccc agc tac ccc cgc cag
ccg ggc tac cct cat aac
cca ggg tac ccc cat aac
cca ggg tac ccc cat aac
cca ggg tac ccc cac aac
cct ggc tat ccc cat aac
ccc ggc tac ccc cag aac
cct ggc tac ccc cat aac
cca ggt tac
cca ggc tgg

>chickenL   cat cgc cag
ccc agc tac ccc cgc cag
ccg ggc tac cct cat aac

cca ggg tac ccc cat aac
cca ggg tac ccc cac aac
cct ggc tat ccc cat aac
ccc ggc tac ccc cag aac
cct ggc tac ccc cat aac
cca ggt tac
cca ggc tgg

>ostrich

                    cat aat
ccc ggc tac ccc cac aac
cca ggc tat ccc cat aat
ccc ggc tac ccc cac aac
cca ggc tac ccc cac aac

cca ggc tgg
```

Reputer : un outil utilisant les arbres de suffixes

- Developed in 1999 at Bielefeld University by S. Kurtz and C. Schleiermacher
- Find all repeats in complete genomes (contrary to programs such as Repeatfinder of GCG).
- Constant α =12.5 (repeats + palindroms on S. cerevisiae, 11.5 Mb, with 160 Mb of memory and <1 mn on Sparc 400 MHz).
- Allows to take into account errors, based on exact repeats and includes a significance calculus.

Recherche de maximal repeats

A maximal repeat of a sequence is a subword occurring at 2 different positions in the sequence such that it cannot be enlarged to the left or to the right without loosing the double occurrence. The issue is to study their construction with a suffix tree.

- Give 2 maximal repeats of xabcyuuzabcvabcywxaw
- Show that if w is a maximal repeat , it is associated to a internal node of the suffix tree.
- Deduce from this fact that there is at most n maximal repeats in a sequence, if n is the size of the sequence.

Recherche de maximal repeats : solution

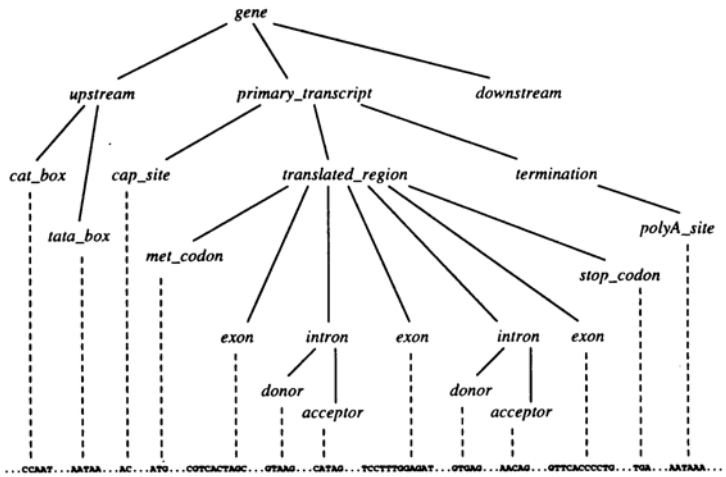
- abc and abcy
- Nodes of a suffix trees are repeats.
  But not all repeats are present in the tree.
  Consider the tree of ACAC$. A is not in the tree because all A are followed with a C in the sequence.
  However given a maximal repeat w, it exists by definition 2 *different letters* x et y (one of them being possibly $) such that words wx and wy appear in the sequence, with a common prefix parent, w that will thus appear in the tree.
- There exists n internal nodes thus at most n maximal repeats

Un calcul de maximal repeats dans l'arbre des suffixes

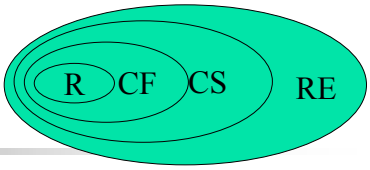
T(Parent):= if (leaf(Parent,i),
Sequence[i-1],
(S:=$\cup_{Child}$(T(Child));
if |S| >1 then true else S))

Solution = words leading to nodes Parent such that T(Parent).

Exemple d'arbre d'analyse de la séquence d'un gène (illustrations de D. Searls)



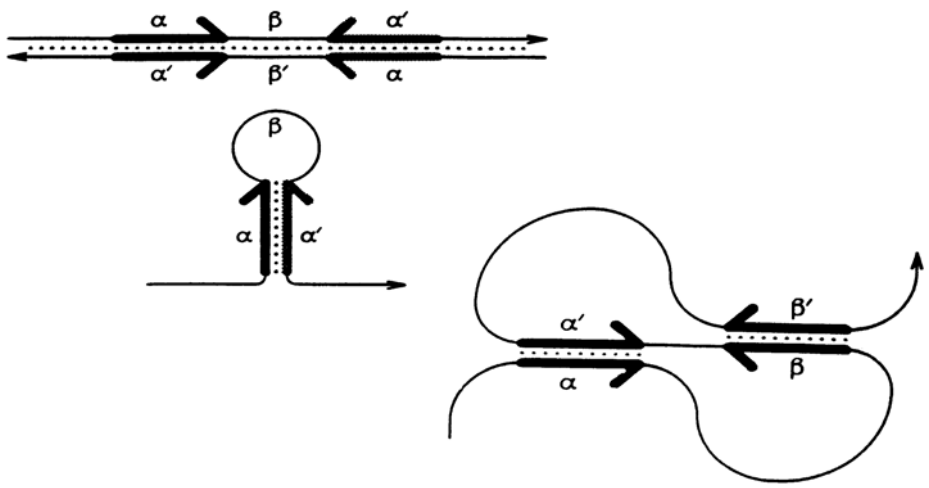
Hiérarchie de Chomsky



Languages	*Automaton*	*Grammar*	*Recognition*	*Dependency*	*Biology*
Recursively Enumerable	Turing Machine	Unrestricted Baa → A	Undecidable	Arbitrary	Unknown
Context-Sensitive	Linear-Bounded	Context-Sensitive At → aA	NP-Complete	Crossing	Pseudoknots, etc.
Context-Free	Pushdown (stack)	Context-Free S → gSc	Polynomial	Nested	Orthodox 2^o Structure
Regular	Finite-State Machine	Regular A → cA	Linear	Strictly Local	Central Dogma

From D. Searls

Inverted repeats (Tige-boucle) et pseudo-noeuds



Exemples simples de grammaires Genlang

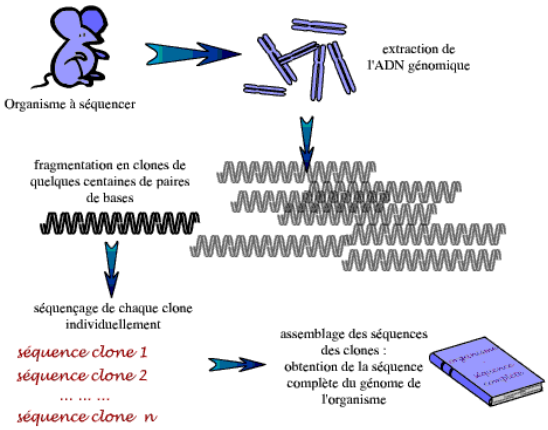
- purine ---> “g” | “a”.
  pyrimidine ---> “c” | “t”.
  purine_rich ---> purine:[cost=0], mostly_purine,
  purine:[cost=0], .
  mostly_purine---> purine, mostly_purine | [].

  (…, purine_rich: [size>25, cost <9], …): P ==> 1.

- tandem_repeat ---> X: [size>9], X.
- palindrom ---> X: [size>9], 2…1000, ~X.

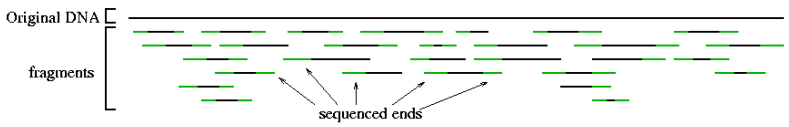
- gene ---> “atg”, @-3, modules, stop_codon.
  modules ---> exon, intron, modules | exon.
  stop_codon ---> “tga” | “ta”, purine.

Schéma de base de l'assemblage



Recherche de recouvrements

Shotgun



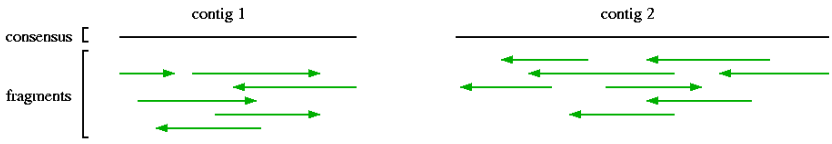
Fragments obtenus par ultrasons ou flux d'air haute pression

Taille fragments =

2-3 Kb + 8-10 Kb

Sequencing reads =

paires de séquences ~ 600-700 b



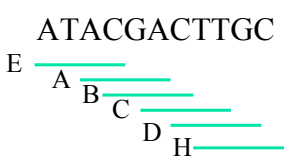
Assemblage : un problème NP-complet/difficile

2 formulations simplifiées possibles

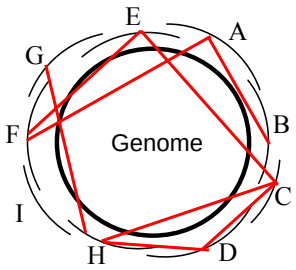
- Recherche de la plus courte superchaîne d'un ensemble de mots;
- Recherche de chemin hamiltonien dans un graphe (nœud = séquence clone, arc = chevauchement clone).

Illustration du problème abstrait d'assemblage

{A=ACGA, B=CGAC, C=ACTT, D=CTTG, E=ATAC, H=TTGC}



Superchaîne la plus courte



Chemin hamiltonien

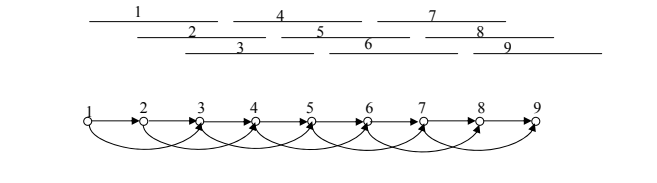
Programmes d'assemblage: état de l'art rapide

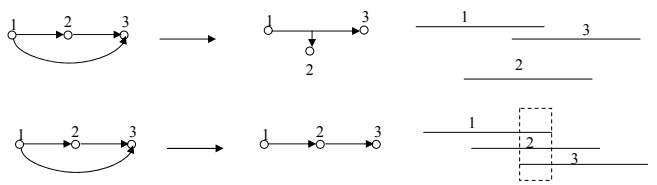
3 méthodes principales

- Algorithme glouton de recherche de superchaîne
- Algorithme Overlap-Layout-Consensus
- Algorithme de recherche de chemin eulérien

[1] M. Pop, S. L. Salzberg, M. Shumway. (2002) Genome Sequence Assembly: Algorithms and Issues. *IEEE Computer* **35(7),** pp. 47-54.

Algorithme overlap-layout-consensus



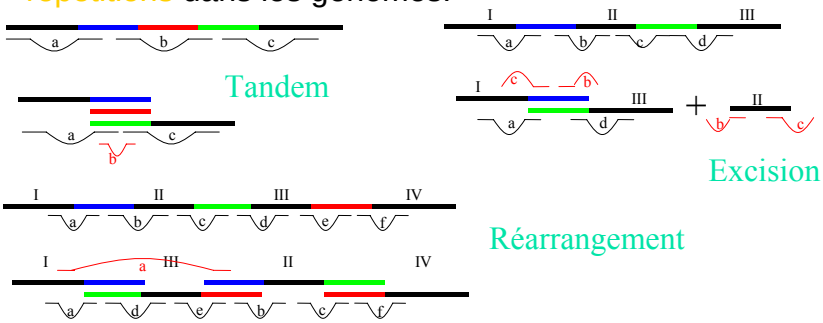


ACCTGA
ACCTGA
AGCTGA
ACCAGA

Consensus
(correction d'erreurs)

De la difficulté de l'assemblage...

- Le problème principal est celui de l'existence de répétitions dans les génomes.



- Ce problème est compliqué par la présence d'erreurs de séquençage et de polymorphisme sur les séquences.

Quelques résultats sur le génome du chien

(Genome Biology 6/10 2005 P. Quignon, M. Giraud, M. Rimbault, P. Lavigne, S. Tacher, E. Retout, E. Morin, A.-S. Valin, K. Linblad-Toh, J. Nicolas, F. Galibert)

- Jeu d'apprentissage de 60 gènes;
- Apprentissage par Pratt de 5 patterns caractéristiques, dont la localisation est distribuée le long du gène;
- Filtrage de 63745 séquences par automates pondérés déduits des motifs (RDISK);
- Nettoyage conduisant à 61321 séquences;
- Assemblage par CAP3 avec 97% d'identité des recouvrements donnant 6727 contigs d'en moyenne 7 fragments;
- Finition par pattern le plus discriminant, Blast et filtrage manuel conduisant à 1050 gènes;
- Travail ensuite sur le génome assemblé quand disponible conduisant à 1000 gènes.


POURQUOI CETTE POMME TOMBE.T.ELLE ?
PARCE QU'IL Y A LA TERRE DESSOUS !
ET POURQUOI CETTE POMME NE TOMBE.T-ELLE PLUS ?
PARCE QU'IL Y A LA TERRE DESSOUS !