

# Εισαγωγή στη Βιοπληροφορική

Μερικές διαφάνειες είναι από από Neil C. Jones, Pavel A. Pevzner,  
Εισαγωγή στους Αλγορίθμους Βιοπληροφορικής εκδόσεις Κλειδάριθμος  
(ελληνική μετάφραση) και MIT Press  
(αγγλόφωνη)(<http://bix.ucsd.edu/bioalgorithms/slides.php> και από  
Algorithms on Strings, Trees and Sequences, D. Gusfield, Cambridge  
University Press, 10th edition 2007

# ΕΠΕΚΤΑΣΕΙΣ

- Longest Common Subsequence (match weight 1, otherwise 0)
- End-space free variant that encourages one string to align in the interior of the other, or the suffix of one to align with the prefix of the other (initial conditions 0, everything is countable in the last row and the last column) – shotgun sequence assembly
- Approximate occurrence of  $P$  in  $T$  (the optimal alignment of  $P$  to a substring of  $T$  has distance  $\delta$  from the optimal alignment) (initial conditions 0).  $V(0,j)=0$ .
  - locate a cell  $(m,j)$  with value greater than  $\delta$ .
  - traverse backpointers from  $(m,j)$  to  $(0,k)$ .
  - occurrence in  $T[k,j]$

## 1. Map longest common subsequence to longest increasing subsequence

Έστω οι δύο ακολουθίες  $S1$  και  $S2$  των οποίων θέλουμε να βρούμε τη μέγιστη κοινή υποακολουθία. Έστω ότι η  $S2$  είναι μικρότερη σε μέγεθος από την  $S1$ . Για κάθε χαρακτήρα  $x$  του αλφαβήτου που εμφανίζεται τουλάχιστον μία φορά στην  $S1$ , δημιουργήστε μια λίστα από τις θέσεις όπου ο χαρακτήρας  $x$  εμφανίζεται στη συμβολοσειρά  $S2$ . γράψτε αυτή τη λίστα ανάποδα.

Το πρόβλημα longest common subsequence (εύρεση μέγιστης κοινής υποακολουθίας των  $S1$  και  $S2$ ) ανάγεται σε πρόβλημα εύρεσης της μεγαλύτερης αύξουσας υποακολουθίας στην λίστα.

## 2. Compute the Longest increasing subsequence algorithm

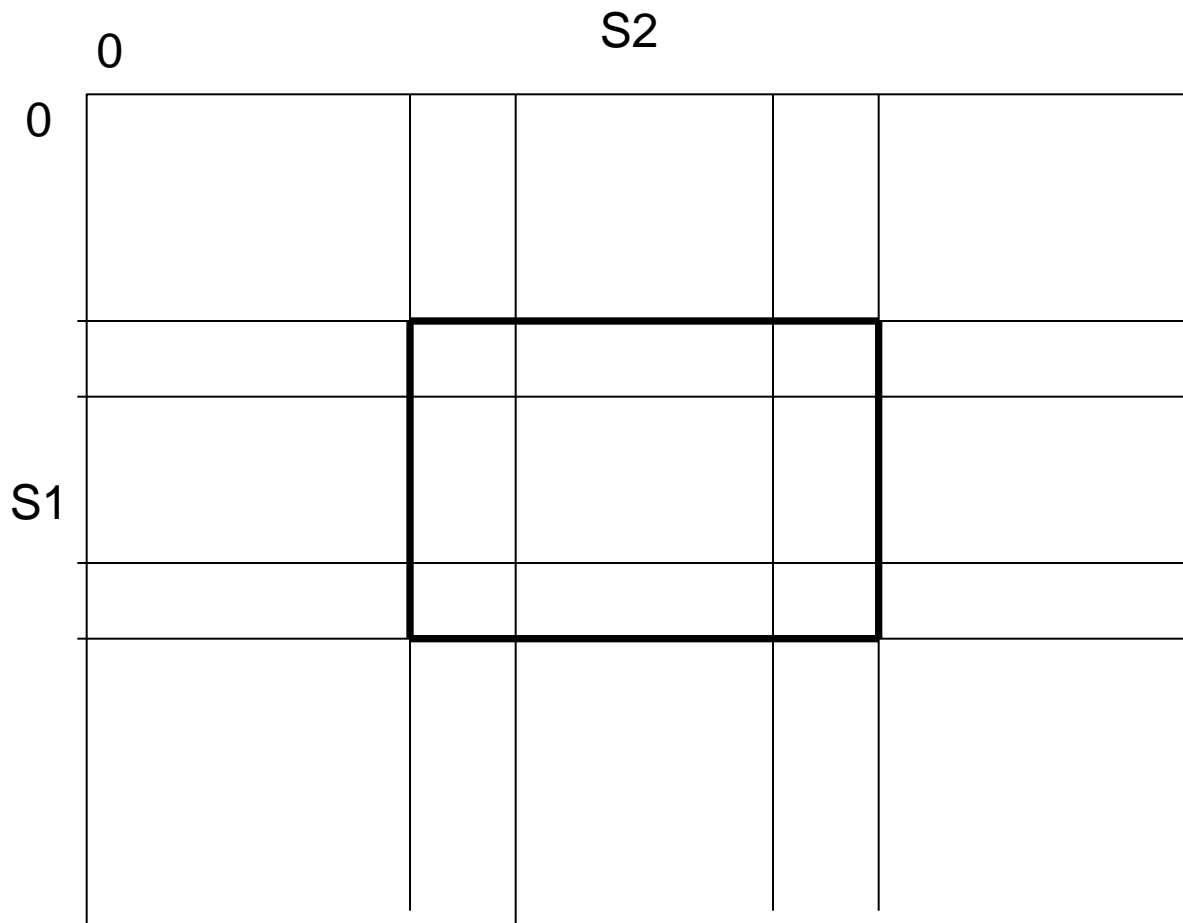
Ξεκινώντας από τα αριστερά, εξετάστε κάθε διαδοχικό αριθμό και τοποθετήστε τον στο τέλος της πρώτης (αριστερότερης) φθίνουσας υποακολουθίας που μπορεί να επεκτείνει. Εάν δεν υπάρχουν φθίνουσες υποακολουθίες που μπορούν να επεκταθούν τότε ξεκινήστε μια νέα (φθίνουσα) υποακολουθία δεξιά από όλες τις υπάρχουσες φθίνουσες υποακολουθίες.

Ενδιαφέρον paper: Maxime Crochemore, Ely Porat: Fast computation of a longest increasing subsequence and application. Inf. Comput. 208(9): 1054-1059 (2010)



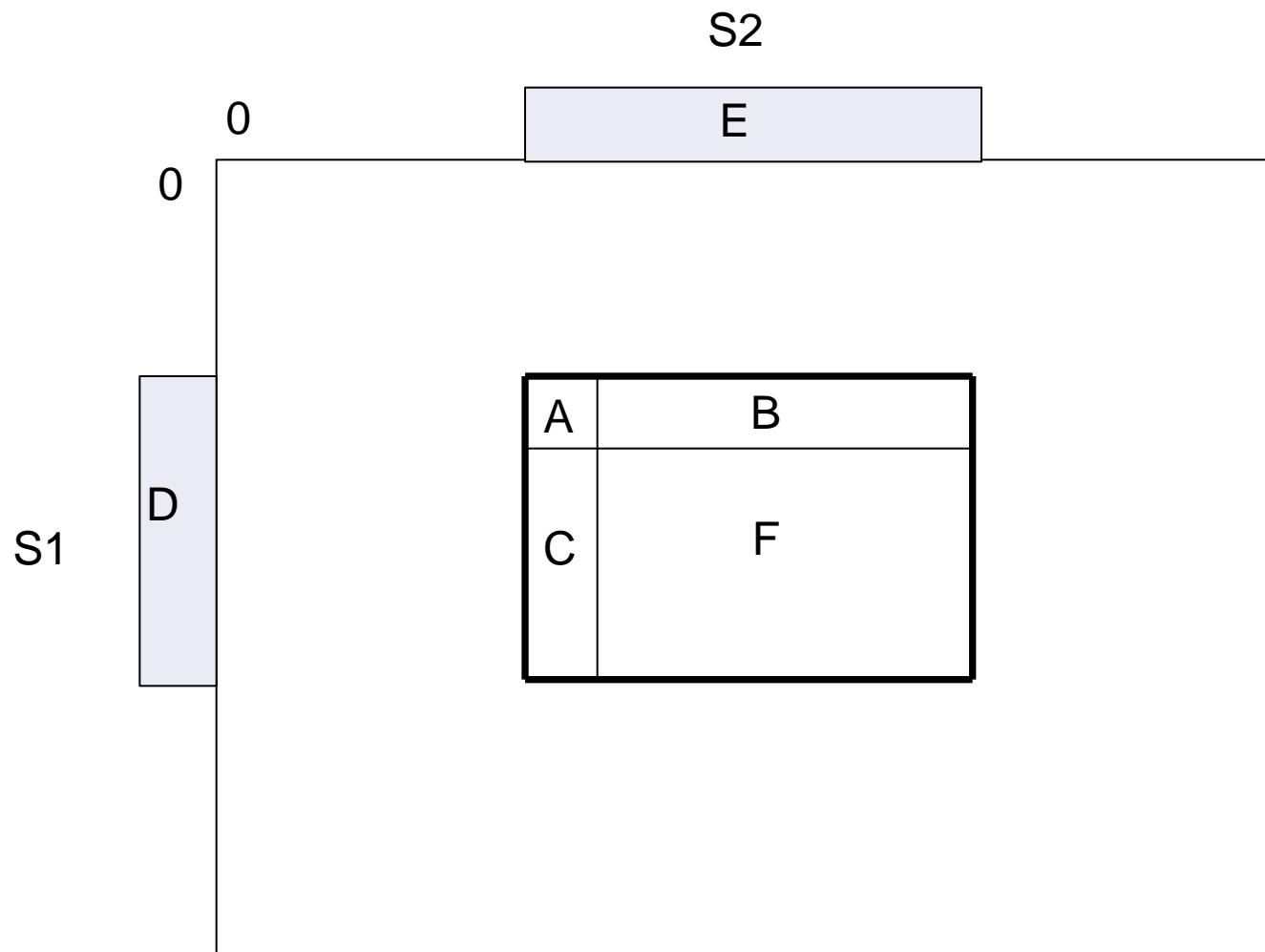
## 4-Russians speedup (Alizarin, Dinic, Kronrod, and Faradzev)

- [Arlazarov, V.](#); *Dinic, E.*; *Kronrod, M.*; *Faradžev, I.* (1970), "On economical construction of the transitive closure of a directed graph", *Dokl. Akad. Nauk SSSR*, **194** (11). Original title: "Об экономном построении транзитивного замыкания ориентированного графа", published in [Доклады Академии Наук СССР](#) **134** (3), 1970.



# The Four-Russians speedup-t-blocks

- Definition: A t-block is a t by t square in the dynamic programming table.
- The rough idea of the Four-Russians method is to partition the dynamic programming table into t-blocks and compute the essential values in the table one t-block at a time, rather than one cell at a time.
- Lemma: The distance values in a t-block starting in position (i,j) are a function of the values in its first row and column and the substrings  $S_1[i \dots i+t-1]$  and  $S_2[j \dots j+t-1]$ .
- Definition: Given above lemma and the following figure, we define the block function as the function from the five inputs (A,B,C,D,E) to the output F
- The values in the last row and column of a r-block are also a function of the inputs (A, B, C, D, E). We call the function from those inputs to the values in the last row and column of a r-block, the restricted block function.



# Accounting detail

- block edit distance algorithm

- the sizes of the input and the output of the restricted block function are both  $O(t)$ . There are  $\Theta(n^2/t^2)$  blocks, hence the total time used by the block edit distance algorithm is  $O(n^2/t)$ . Setting  $t$  to  $\Theta(\log n)$ , the time is  $O(n^2/\log n)$ . However in the unit-cost RAM model of computation, each output value can be retrieved in constant time since  $t=O(\log n)$ .  $\Rightarrow$  the method is reduced to  $O(n^2 / (\log n)^2)$ .

- precomputation time

- The key issue involves the number of input choices to the restricted block function.
- Every cell has an integer from zero to  $n \Rightarrow (n+1)^t$  possible values for any  $t$ -length row or column.
- If the alphabet has size  $\sigma$ , then there are  $\sigma^t$  possible substrings of length  $t \Rightarrow$  #distinct input combinations to the restricted block function: is  $(n + 1)^{2t} \sigma^{2t}$
- For each input, it takes  $\Theta(t^2)$  time to evaluate the last row and column of the resulting  $r$ -block (by running the standard dynamic program). Thus the overall time used in this way to precompute the function outputs to all possible input choices is  $\Theta((n + 1)^{2t} \sigma^{2t} t^2)$ .



# The trick: offset encoding

- Lemma: In any row, column, or diagonal of the dynamic programming table for edit distance, two adjacent cells can have a value that differs by at most one.
- Definition: The offset vector is a  $t$ -length vector of values from  $\{-1,0,1\}$ , where the first entry must be zero.
- Theorem: Consider a  $t$ -block with upper left corner in position  $(i,j)$ . The two offset vectors for the last row and last column of the block can be determined from the two offset vectors for the first row and column of the block and from substrings  $S1[1..i]$  and  $S2[1..j]$ . That is, no  $D$  value is needed in the input in order to determine the offset vectors in the last row and column of the block.
- Definition: The function that determines the two offset vectors for the last row and last column from the two offset vectors for the first row and column of a block together with substrings  $S1[1..i]$  and  $S2[1..j]$  is called the offset function
- We have  $3\sigma^{2t} * t^2$  precomputation time, that for  $t=(\log_{3\sigma} n)/2$  is  $O(n(\log n)^2)$

## 4-Russians αλγόριθμος

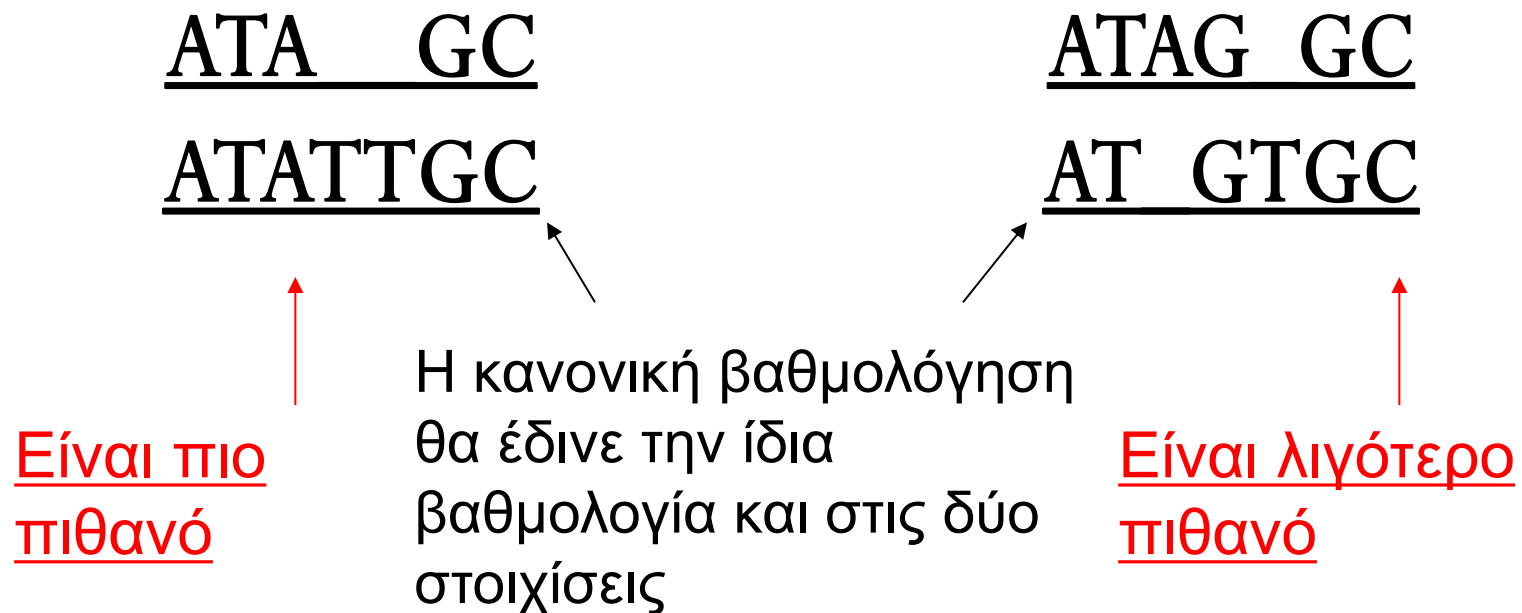
1. Καλύψτε τον πίνακα δυναμικού προγραμματισμού  $n$  by  $n$  με  $t$ -blocks, όπου η τελευταία στήλη κάθε  $t$ -block μοιράζεται με την πρώτη στήλη του  $t$ -block στα δεξιά του (εάν υπάρχει) και η τελευταία σειρά κάθε  $t$ -block μοιράζεται με την πρώτη σειρά του  $t$ -block κάτω από αυτό (εάν υπάρχει).
2. Αρχικοποιήστε τις τιμές στην πρώτη γραμμή και στήλη του πλήρους πίνακα σύμφωνα με τις βασικές συνθήκες της αναδρομής. Υπολογίστε τις τιμές μετατόπισης στην πρώτη γραμμή και στήλη.
3. Κατά σειρά, χρησιμοποιήστε τη συνάρτηση μπλοκ μετατόπισης για να προσδιορίσετε διαδοχικά τα διανύσματα μετατόπισης της τελευταίας γραμμής και στήλης κάθε μπλοκ. Λόγω της επικαλυπτόμενης φύσης των μπλοκ, το διάνυσμα μετατόπισης στην τελευταία στήλη (ή γραμμή) ενός μπλοκ παρέχει το επόμενο διάνυσμα μετατόπισης στην πρώτη στήλη (ή γραμμή) του μπλοκ στα δεξιά του (ή κάτω από αυτό).
4. Έστω  $Q$  το σύνολο των τιμών μετατόπισης που υπολογίζονται για τα κελιά της σειράς  $n$ .  $D(n, n) = D(n, 0) + Q = n + Q$ .

# Στοίχιση Ακολουθιών με κενά

- **Έννοια κενού:** συνεχόμενα spaces, θέλουμε να ελέγχουμε την κατανομή των κενών.
- **Εισαγωγή Κενών:** Για να συμπεριλάβουμε το κόστος που η εισαγωγή κενών εισάγει στη στοίχιση 2 ακολουθιών, μπορούμε σε μια απλή προσέγγιση να θεωρήσουμε ότι κάθε κενό συνεισφέρει ένα σταθερό βάρος  $W_g$ , ανεξάρτητα από το μήκος του.
- τιμή στοίχισης που περιέχει “k” κενά: 
$$\sum_{i=1}^l s(S'_1(i), S'_2(i)) - kW_g$$
- μία καλύτερη προσέγγιση είναι η χρησιμοποίηση μίας συνάρτησης του μήκους του κενού. Τότε μπορούμε να γεμίσουμε ένα πίνακα:  $V(i,j) = \max[E(i,j), F(i,j), G(i,j)]$

# Συγγενικές ποινές κενού

- Στη φύση, μια σειρά  $k$  προσθαφαιρέσεων εμφανίζεται συχνά ως ένα μόνο συμβάν αντί για μια σειρά  $k$  συμβάντων που αφορούν μεμονωμένα νουκλεοτίδια:



# Λαμβάνοντας υπόψη τα κενά

- Κενά – συνεχόμενη ακολουθία (κενών) διαστημάτων σε μία από τις γραμμές
- Η βαθμολογία για ένα κενό μήκους  $x$  είναι:  
 $-(\rho + \sigma x)$   
όπου  $\rho > 0$  είναι η ποινή για την εισαγωγή του κενού:  
**ποινή ανοίγματος κενού**  
το  $\rho$  θα είναι μεγάλο σε σχέση με το  $\sigma$ :  
**ποινή επέκτασης κοινού**  
επειδή δεν θέλουμε να έχουμε πολύ μεγάλη ποινή για την επέκταση του κενού.

# Συγγενικές ποινές κοινού

## ■ Ποινές κενού:

- $-ρ-σ$  όταν υπάρχει 1 προσθαφαίρεση
- $-ρ-2σ$  όταν υπάρχουν 2 προσθαφαιρέσεις
- $-ρ-3σ$  όταν υπάρχουν 3 προσθαφαιρέσεις, κλπ.
- $-ρ-χ·σ$  (-άνοιγμα κενού -  $χ$  επεκτάσεις κενού)

## ■ Επιβάλλονται σχετικά μειωμένες ποινές (σε σύγκριση με την απλοϊκή βαθμολόγηση) σε ακολουθίες οριζόντιων και κάθετων ακμών

## Στοιχίση Ακολουθιών με κενά

- **Έννοια κενού:** συνεχόμενα spaces, θέλουμε να ελέγχουμε την κατανομή των κενών.
- **Εισαγωγή Κενών:** Για να συμπεριλάβουμε το κόστος που η εισαγωγή κενών εισάγει στη στοιχίση 2 ακολουθιών, μπορούμε σε μια απλή προσέγγιση να θεωρήσουμε ότι κάθε κενό συνεισφέρει ένα σταθερό βάρος  $W_g$ , ανεξάρτητα από το μήκος του.
- τιμή στοιχίσης που περιέχει “k” κενά: 
$$\sum_{i=1}^l s(S'_1(i), S'_2(i)) - kW_g$$
- μία καλύτερη προσέγγιση είναι η χρησιμοποίηση μίας συνάρτησης του μήκους του κενού  $w(x)$ . Τότε μπορούμε να γεμίσουμε ένα πίνακα:  $V(i,j) = \max[E(i,j), F(i,j), G(i,j)]$

# Στοιχισή με arbitrary gap weights

$$V(i,j) = \max[E(i,j), F(i,j), G(i,j)]$$

*i*



*i*

$$G(i,j) = V(i-1, j-1) + \text{cost}(i \rightarrow j)$$

$$E(i,j) = \max_K V(i,k) - w(j-k) \quad (0 \leq k \leq j-1)$$

*i*



**gaps**



*i*

$$F(i,j) = \max_l V(l,j) - w(i-l) \quad (0 \leq l \leq i-1)$$

*i*



**gaps**

*i*

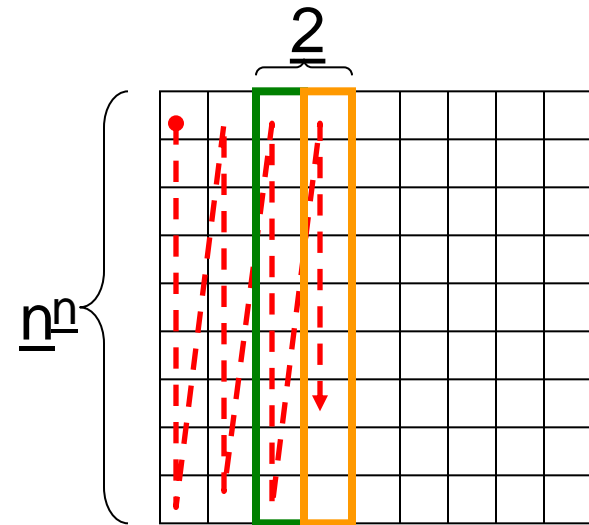


- $V(i,j)=\max\{E(i,j), F(i,j), G(i,j)\}$
- $V(i,0)=-w(i)$
- $V(0,j)=-w(j)$
- $E(i,0)=-w(i)$
- $F(0,j)=-w(j)$
- $G(0,0)=0$
- Assuming that  $|S_1|=n$  and  $|S_2|=m$  the recurrences can be evaluated in  $O(nm^2+n^2m)$

# Υπολογισμός βαθμολογίας της στοίχισης με χρήση γραμμικής μνήμης

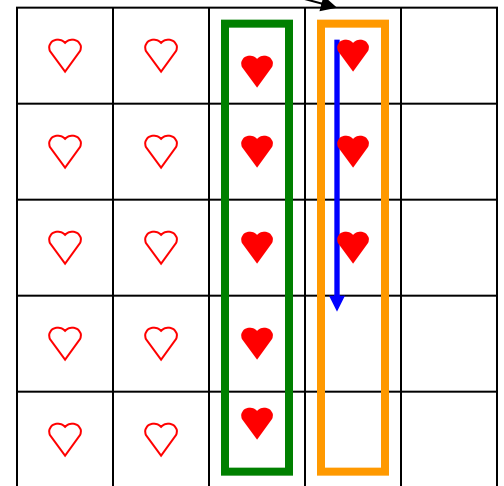
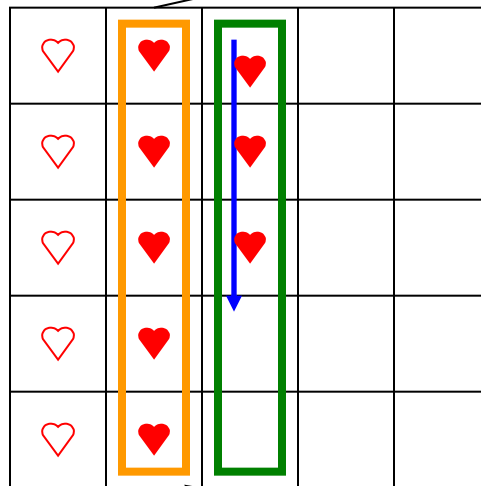
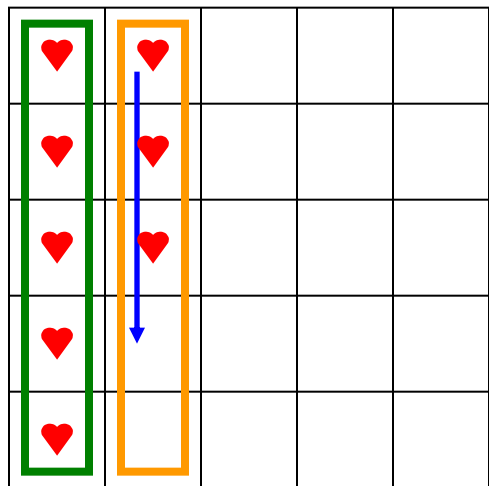
## Βαθμολογία στοίχισης

- Η χωρική πολυπλοκότητα για τον υπολογισμό μόνο της βαθμολογίας είναι  $O(n)$
- Χρειαζόμαστε μόνο την προηγούμενη στήλη για να υπολογίσουμε την τρέχουσα στήλη, και μπορούμε να «απαλλαγούμε» από την προηγούμενη στήλη μόλις ολοκληρώσουμε τη χρήση της



# Υπολογισμός βαθμολογίας της στοίχισης: ανακύκλωση στηλών

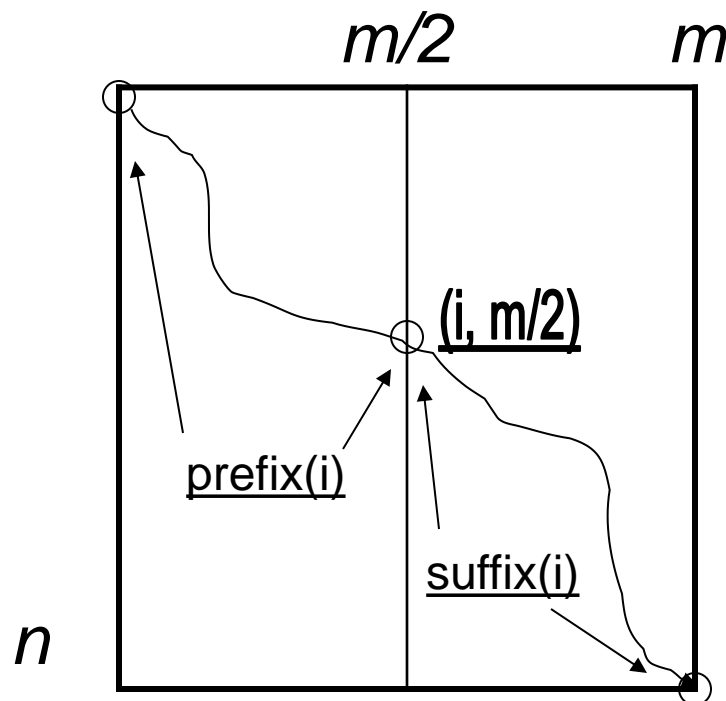
Μόνο δύο στήλες βαθμολογιών αποθηκεύονται σε  
οποιαδήποτε χρονική στιγμή



η μνήμη για τη στήλη 1  
χρησιμοποιείται για τον  
υπολογισμό της στήλης 3

η μνήμη για τη στήλη 2  
χρησιμοποιείται για τον  
υπολογισμό της στήλης 4

# Περνώντας από τη μεσαία γραμμή



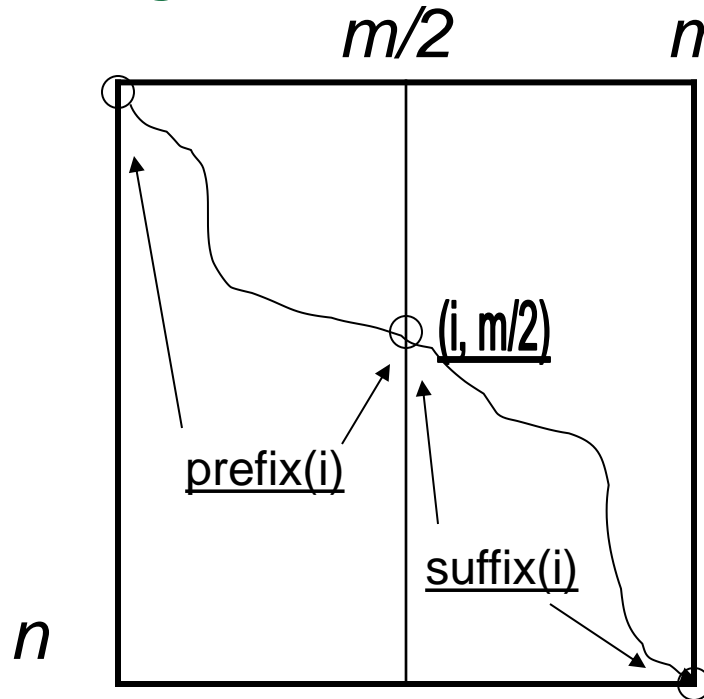
Θέλουμε να υπολογίσουμε τη μεγαλύτερη διαδρομή από την κορυφή  $(0,0)$  προς την κορυφή  $(n, m)$  που διέρχεται από την κορυφή  $(i, m/2)$ , όπου το  $i$  παίρνει τιμές από 0 έως  $n$  και αναπαριστά την  $i$ -οστή γραμμή

Ορίζουμε το

$length(i)$

ως το μήκος της μεγαλύτερης διαδρομής μεταξύ των  $(0,0)$  και  $(n, m)$  που διέρχεται από την κορυφή  $(i, m/2)$

# Περνώντας από τη μεσαία γραμμή



Ορίζουμε το  $(mid, m/2)$  ως την κορυφή όπου η μεγαλύτερη διαδρομή τέμνει τη μεσαία στήλη

$$length(mid) = \text{βέλτιστο μήκος} = \max_{0 \leq i \leq n} length(i)$$

# Sequence Database Searching

## Βήματα καθορισμού πρωτεϊνικής ακολουθίας

1. Σύγκριση της νέας ακολουθίας με PROSITE και BLOCKS για εύρεση well-characterized sequence motifs.
2. Ψάξιμο στις DNA και protein sequence databases (Genbank, Swiss-Prot, etc.) για εντοπισμό ακολουθιών τοπικά παρόμοιων (με χρήση ενός κριτηρίου τοπικής ομοιότητας) – χρήση FASTA και BLAST
3. Εάν τα παραπάνω ψαξίματα δίνουν ενδιαφέρον αποτέλεσμα, τότε καταφεύγουμε στη χρήση της τεχνικής του δυναμικού προγραμματισμού.
4. Όταν χρειαστεί να εμπλακούν amino acid substitution matrices, συνήθως χρησιμοποιείται μία παραλλαγή του Dayhoff PAM matrix και του BLOSUM matrix.

# Μήτρες κουκκίδων

- Οι μήτρες κουκκίδων δείχνουν τις ομοιότητες μεταξύ δύο αλληλουχιών
- Το εργαλείο FASTA φτιάχνει μια «έμμεση» μήτρα κουκκίδων από μικρά ακριβή ταιριάσματα, και προσπαθεί να βρει μεγάλες διαγωνίους (επιτρέποντας κάποιες ασυμφωνίες)

	G	A	T	C	G	C	T	A	G	T
C				*		*				
T		*	*				*	*		*
G	*				*					*
A		*						*		
T		*	*			*	*			*
T		*	*			*	*			*
C				*		*				
C				*		*				
T		*	*				*	*		*
T		*	*				*	*		*
A		*							*	
G	*				*				*	
T		*	*			*	*			*
C				*		*				
A		*						*		
G	*				*				*	

# Ο Αλγόριθμος FASTA

- FASTA: Fast – All, Lipman et al. 1985
- Κεντρική ιδέα: εύρεση μικρών λέξεων (words ή k-tuples) που εμφανίζονται και στις δύο ακολουθίες. Στην περίπτωση πρωτεϊνικών ακολουθιών το μήκος των λέξεων είναι 1-2 residues ενώ για ακολουθίες DNA το μήκος μίας λέξης μπορεί να φτάνει στις 6 βάσεις



# Τα βήματα του αλγορίθμου FASTA

- 1ο βήμα: αναζητούμε λέξεις μήκους  $k$  στον πίνακα :  
'hot spots' (pairs  $(i,j)$  )
- 2ο βήμα: εντοπίζουμε τις δέκα καλύτερες διαγώνιες τροχιές – diagonal runs από 'hot-spots' στον πίνακα  
(a hot spot define the  $(i-j)$ -diagonal, the score is the sum of scores of hot-spots plus weighted decreasing as the distance increases)
- 3ο βήμα: συνδυάζουμε «καλές υπο-στοιχίσεις»
- 4ο βήμα: παράγουμε το βέλτιστο μονοπάτι

# Ο Αλγόριθμος BLAST

- BLAST: Basic Local Alignment Search Tool, Altschul et. Al. 1990
- Βασική ιδέα: εντοπισμός κοινών υπο-ακολουθιών ίδιου μήκους (segment pairs) που εμφανίζονται και στη δοσμένη ακολουθία μικρού μήκους (input query sequence) και στο σύνολο των ακολουθιών μίας βάσης δεδομένων. Στη συνέχεια επέκταση για εύρεση maximal segment pairs

Αλγόριθμος	Είδος query	Είδος sequence
BLASTP	Πρωτεΐνη	Πρωτεΐνη
BLASTN	Νουκλεοτίδιο	Νουκλεοτίδιο
BLASTX	Νουκλεοτίδιο	Πρωτεΐνη
TBLASTN	Πρωτεΐνη	Νουκλεοτίδιο
TBLASTX	Νουκλεοτίδιο	Νουκλεοτίδιο

1<sup>ο</sup> βήμα: τμηματοποίηση της δοσμένης ακολουθίας σε διαδοχικές υπο-λέξεις μεγέθους  $w=3$

Query sequence:

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Words

[illegible][illegible][illegible][illegible]

2<sup>ο</sup> βήμα: Εντοπισμός των υπο-λέξεων με μέγιστη τιμή στοίχισης για το όλες τις ακολουθίες

*High-scoring matching words:*

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

[illegible][illegible]

d	s	k	o	w	j	j	d	f	k	s	l	m	n	k	d	k	j	d	f	k	k	j	d	f	f
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

[illegible]

m	s	l	z	m	s	o	w	u	r	n	f	k	s	a	d	e	f	a	q	m	a	z	m	s	l
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

[illegible]

3<sup>ο</sup> βήμα: επέκταση των high-scoring words

a	b	c	d	w	f	h	h	f	j	s	l	m	n	k	d	k	j	d	e	h	k	k	j	f	f
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

[illegible]

# BLAST

- Μεγάλη βελτίωση της ταχύτητας με μέτρια μείωση της ευαισθησίας
- Ελαχιστοποιεί αντί να ψάχνει ολόκληρο το χώρο αναζήτησης μεταξύ δύο αλληλουχιών
- Βρίσκει μικρά ακριβή ταιριάσματα («φύτρα»), και ψάχνει μόνο τοπικά γύρω από αυτά

# Τι αποκαλύπτει η ομοιότητα

- Εκτέλεση του BLAST σε ένα νέο γονίδιο
  - Εξελικτική σχέση
  - Ομοιότητα μεταξύ των λειτουργιών διαφορετικών πρωτεϊνών
- Εκτέλεση του BLAST σε ένα γονιδίωμα
  - Πιθανά γονίδια

# Ο αλγόριθμος BLAST

- Αναζήτηση με λέξεις-κλειδιά για όλες τις λέξεις μήκους  $w$  από το ερώτημα μήκους  $n$  στη βάση δεδομένων μήκους  $m$  με βαθμολογίες μεγαλύτερες από κάποιο κατώφλι
  - $w = 11$  για ερωτήματα DNA,  $w = 3$  για πρωτεΐνες
- Επέκταση τοπικής στοίχισης για κάθε λέξη-κλειδί που εντοπίζεται
  - Επέκταση του αποτελέσματος μέχρι να επιτευχθεί το μεγαλύτερο ταίριασμα που υπερβαίνει το κατώφλι
- Χρόνος εκτέλεσης  $O(nm)$

# Ο αλγόριθμος BLAST (συνέχεια)

λέξη-κλειδί

Query: KRHRKVLRLDNIQGITKPAIRRLARRGGVKRISGLIYEETRGVLKIFLENVIRD

GVK 18

GAK 16

GIK 16

GGK 14

GLK 13

GNK 12

GRK 11

GEK 11

GDK 11

λέξεις γειτονιάς

κατώφλι  
βαθμολογίας γειτονιάς  
( $T = 13$ )

επέκταση

Query: 22 VLRDNIQGITKPAIRRLARRGGVKRISGLIYEETRGVLK 60

+++DN +G + IR L G+K I+ L+ E+ RG++K

Sbjct: 226 IIKDNGRGFSKGQIRNLNYGIGLKVIADLV-EKHRGIK 263

Ζεύγος υψηλής βαθμολογίας (High-scoring pair, HSP)

# Ο αρχικός αλγόριθμος BLAST

- Λεξικό

- Όλες οι λέξεις μήκους  $w$

- Στοιχίση

- Επεκτάσεις χωρίς κενά μέχρι η βαθμολογία να γίνει μικρότερη από κάποιο στατιστικό κατώφλι

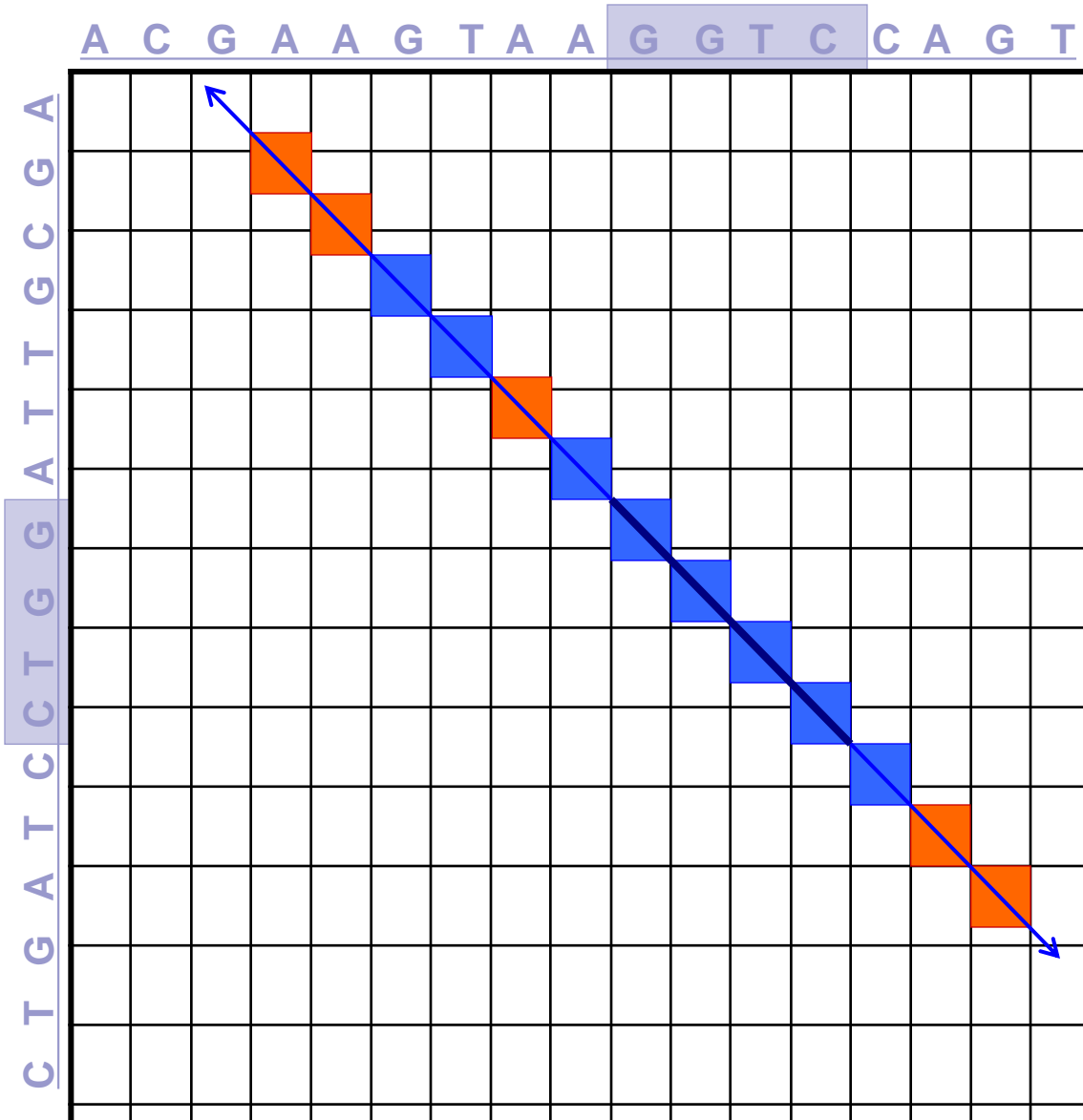
- Έξοδος

- Όλες οι τοπικές στοιχίσεις με βαθμολογία  $>$  κατώφλι



# Ο αρχικός αλγόριθμος BLAST: παράδειγμα

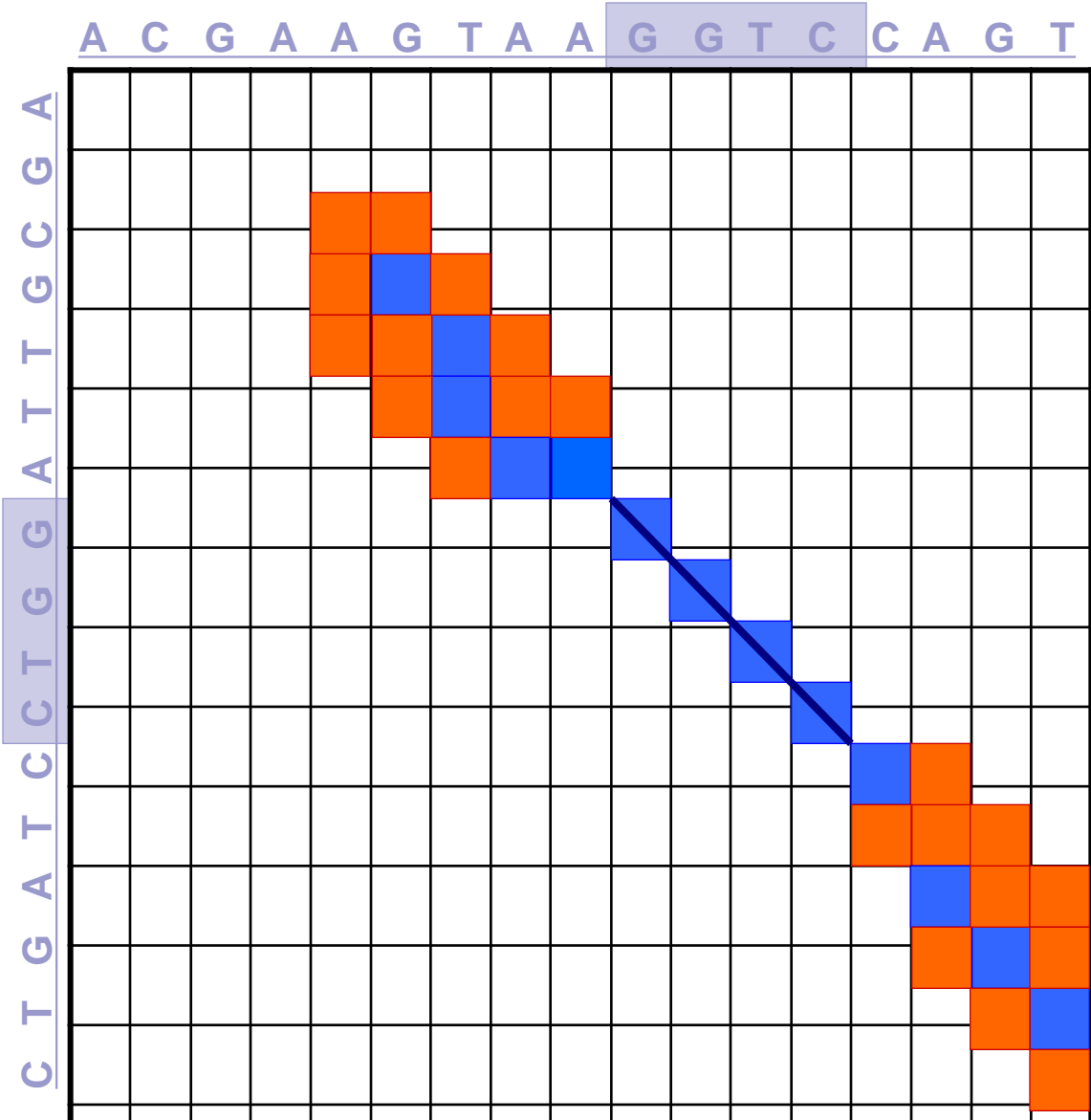
- $w = 4$
- Ακριβές ταίριασμα  
λέξης-κλειδιού του  
GGTC
- Επέκταση  
διαγωνίων με  
ασυμφωνίες μέχρι  
η βαθμολογία να  
πέσει κάτω από  
50%
- Έξοδος  
GTAAGGTCC  
GTTAGGTCC



# BLAST με κενά: παράδειγμα

- Ακριβής αναζήτηση με λέξεις-κλειδιά του αρχικού BLAST και META:
- Επέκταση με κενά γύρω από τα άκρα του ακριβούς ταιριάσματος **μέχρι να ισχύει βαθμολογία < κατώφλι**
- Έξοδος  
GTAAGGTCCAGT  
GTTAGGTC-AGT

Από τις διαλέξεις του Serafim Batzoglou (Stanford)



# Υλοποιήσεις του BLAST

- blastn: Νουκλεοτιδική-νουκλεοτιδική
- blastp: Πρωτεϊνική-πρωτεϊνική
- blastx: Μεταφρασμένη αλληλουχία-ερώτημα έναντι πρωτεϊνικής βάσης δεδομένων
- tblastn: Ερώτημα πρωτεΐνης έναντι μεταφρασμένης νουκλεοτιδικής βάσης δεδομένων
- tblastx: Μεταφρασμένη αλληλουχία-ερώτημα έναντι μεταφρασμένης νουκλεοτιδικής βάσης δεδομένων (6 πλαίσια το καθένα)

\_Neil C. Jones, Pavel A. Pevzner Εισαγωγή στους αλγορίθμους Βιοπληροφορικής, εκδόσεις Κλειδάριθμος

# Υλοποιήσεις του BLAST (συνέχεια)

- PSI-BLAST

- Βρίσκει μέλη μιας οικογένειας πρωτεϊνών ή κατασκευάζει μια προσαρμοσμένη μήτρα βαθμολόγησης εξειδικευμένη για θέσεις

- Megablast:

- Αναζητά μεγαλύτερες αλληλουχίες με λιγότερες διαφορές

- WU-BLAST: (Wash U BLAST)

- Βελτιστοποιημένα, πρόσθετα χαρακτηριστικά

# Αξιολόγηση της ομοιότητας αλληλουχιών

- Πρέπει να ξέρουμε πόσο «ισχυρή» αναμένεται να είναι μια στοίχιση που προκύπτει από καθαρή τύχη
- Η «τύχη» σχετίζεται με τη σύγκριση αλληλουχιών που παράγονται τυχαία με βάση ένα ορισμένο μοντέλο αλληλουχίας
- Τα μοντέλα αλληλουχίας μπορεί να λαμβάνουν υπόψη:
  - ☐ το περιεχόμενο G+C
  - ☐ τις ουρές πολύ-A
  - ☐ το «άχρηστο» DNA
  - ☐ τη μεροληψία κωδικονίων
  - ☐ κλπ.

# BLAST: βαθμολογία τμήματος

- Ο αλγόριθμος BLAST χρησιμοποιεί μήτρες βαθμολόγησης ( $\delta$ ) για να βελτιώσει την αποδοτικότητα της ανίχνευσης ταιριασμάτων
  - Μερικές πρωτεΐνες μπορεί να έχουν πολύ διαφορετικές αμινοξικές αλληλουχίες, αλλά εξακολουθούν να είναι παρόμοιες
- Για δύο οποιαδήποτε  $\ell$ -μερή  $x_1 \dots x_\ell$  και  $y_1 \dots y_\ell$ :
  - Ζεύγος τμημάτων: ζεύγος  $\ell$ -μερών, ένα από κάθε αλληλουχία
  - Βαθμολογία τμημάτων:  $\sum_{i=1}^{\ell} \delta(x_i, y_i)$

# BLAST: Τοπικά μέγιστα ζεύγη τμημάτων

- Ένα ζεύγος τμημάτων είναι μέγιστο αν έχει την καλύτερη βαθμολογία ανάμεσα σε όλα τα ζεύγη τμημάτων
- Ένα ζεύγος τμημάτων είναι τοπικά μέγιστο αν η βαθμολογία του δεν μπορεί να βελτιωθεί με επέκταση ή με περικοπή
- Τα στατιστικά σημαντικά *τοπικά μέγιστα* ζεύγη τμημάτων έχουν βιολογικό ενδιαφέρον
- Ο BLAST βρίσκει όλα τα τοπικά μέγιστα ζεύγη τμημάτων με βαθμολογίες μεγαλύτερες από κάποιο κατώφλι
  - Ένα πολύ υψηλό κατώφλι θα φιλτράρει ορισμένα στατιστικά ασήμαντα ταιριάσματα

# BLAST: στατιστικές

- Κατώφλι: στατιστικές Altschul-Dembo-Karlin
  - Προσδιορίζει τη μικρότερη βαθμολογία τμημάτων που είναι απίθανο να συμβεί κατά τύχη
- Ο αριθμός ταιριασμάτων πάνω από το  $\theta$  έχει μέση τιμή  $E(\theta) = Kmne^{-\lambda\theta}$ , όπου το  $K$  είναι μια σταθερά, τα  $m$  και  $n$  είναι τα μήκη των δύο συγκρινόμενων αλληλουχιών
  - Η παράμετρος  $\lambda$  είναι θετική ρίζα της εξίσωσης  $\sum_{x,y \text{ in } A} (p_x p_y e^{\delta(x,y)}) = 1$ , όπου τα  $p_x$  και  $p_y$  είναι οι συχνότητες των αμινοξέων  $x$  και  $y$ , και το  $A$  είναι το αλφάβητο αμινοξέων με 20 γράμματα



# P-τιμές

- Η πιθανότητα να βρεθούν  $b$  HSP (ζεύγη υψηλής βαθμολογίας) με βαθμολογία  $\geq S$  δίνεται από τη σχέση

$$\square (e^{-E} E^b) / b!$$

- Για  $b = 0$ , η πιθανότητα είναι ίση με

$$\square e^{-E}$$

- Άρα, η πιθανότητα να βρεθεί τουλάχιστον ένα HSP με βαθμολογία  $\geq S$  είναι ίση με

$$\square P = 1 - e^{-E}$$

# Χρονοδιάγραμμα

- 1970: Αλγόριθμος καθολικής στοίχισης των Needleman-Wunsch
- 1981: Αλγόριθμος τοπικής στοίχισης των Smith-Waterman
- 1985: FASTA
- 1990: BLAST (βασικό εργαλείο εύρεσης τοπικών στοιχίσεων)
- Δεκαετία του 2000: το BLAST έχει γίνει πολύ αργό για συγκρίσεις ολόκληρων γονιδιωμάτων – αναπτύσσονται νέοι ταχύτεροι αλγόριθμοι!
  - PatternHunter
  - BLAT

# PatternHunter: ταχύτερος και ακόμα πιο ευαίσθητος

- BLAST: ταιριάζει μικρές συνεχόμενες αλληλουχίες (συνεχόμενο φύτρο)
- Μήκος =  $k$
- Παράδειγμα ( $k = 11$ ):

11111111111

Κάθε 1 αναπαριστά ένα «ταίριασμα»

- PatternHunter: ταιριάζει μικρές μη συνεχόμενες αλληλουχίες (φύτρο με διαστήματα)
- Αυξάνει την ευαισθησία εντοπίζοντας ομολογίες που θα περνούσαν απαρατήρητες διαφορετικά
- Παράδειγμα (ένα φύτρο με διαστήματα, με μήκος 18 w/ 11 «ταίριασματα»):

111010010100110111

Κάθε 0 αναπαριστά μια «αδιάφορη» περίπτωση, οπότε μπορεί να υπάρχει ταίριασμα ή ασυμφωνία

# Μια άλλη μέθοδος: BLAT

- BLAT (BLAST-Like Alignment Tool, εργαλείο στοίχισης τύπου BLAST)
- Ίδια ιδέα με το BLAST – εντοπισμός μικρών ταιριασμάτων αλληλουχιών και επέκταση

# BLAT και BLAST: διαφορές

- Το BLAT δημιουργεί ένα ευρετήριο της βάσης δεδομένων και σαρώνει γραμμικά την αλληλουχία ερωτήματος, ενώ το BLAST δημιουργεί ένα ευρετήριο της αλληλουχίας ερωτήματος και μετά σαρώνει γραμμικά τη βάση δεδομένων
- Το ευρετήριο αποθηκεύεται στη μνήμη RAM, γεγονός που είναι απαιτητικό σε μνήμη αλλά οδηγεί σε γρηγορότερες αναζητήσεις

# BLAT: γρήγορες στοιχίσεις cDNA

## Βήματα:

1. Χωρίζει το cDNA σε τμήματα 500 βάσεων.
2. Χρησιμοποιεί ένα ευρετήριο για να βρει περιοχές στο γονιδίωμα που είναι παρόμοιες με κάθε τμήμα του cDNA.
3. Εκτελεί μια λεπτομερή στοίχιση μεταξύ των γονιδιωματικών περιοχών του τμήματος cDNA.
4. Χρησιμοποιεί δυναμικό προγραμματισμό για να «συρράψει» λεπτομερείς στοιχίσεις τεμαχίων σε μία συνολική λεπτομερή στοίχιση.

# BLAT: δημιουργία ευρετηρίου

- Δημιουργείται ένα ευρετήριο που περιέχει τις θέσεις κάθε  $k$ -μερούς στο γονιδίωμα
- Κάθε  $k$ -μερές στην αλληλουχία ερωτήματος συγκρίνεται με κάθε  $k$ -μερές στο ευρετήριο
- Παράγεται μια λίστα «επιτυχιών» - θέσεις στο cDNA και στο γονιδίωμα που ταιριάζουν για  $k$  βάσεις

# Δημιουργία ευρετηρίου: ένα παράδειγμα

Ακολουθεί ένα παράδειγμα με  $k = 3$ :

**Genome:** cacaattatcacgaccgc

**3-mers (non-overlapping):** cac aat tat cac gac cgc

**Index:**    aat 3            gac 12  
          cac 0,9        tat 6  
          cgc 15

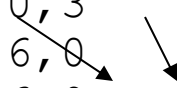
Πολλές περιπτώσεις  
αντιστοιχίζονται σε ένα ευρετήριο

**cDNA (query sequence):** aattctcac

**3-mers (overlapping):** aat att ttc tct ctc tca cac  
                          0    1    2    3    4    5    6

Θέση του 3-μερούς στο ερώτημα, γονιδίωμα

**Hits:** aat 0,3  
          cac 6,0  
          cac 6,9



**clump:** cac**AAT**tat**CAC**gaccgc





# Όμως...

- Το BLAT σχεδιάστηκε να βρίσκει αλληλουχίες με ομοιότητα 95% ή μεγαλύτερη για μήκος  $>40$ . Ενδέχεται να μην εντοπίσει περισσότερες αποκλίνουσες ή μικρότερες στοιχίσεις αλληλουχιών.

# PatternHunter και BLAT εναντίον BLAST

- Το PatternHunter είναι 5-100 φορές ταχύτερο από το Blastn, ανάλογα με το μέγεθος των δεδομένων, για την ίδια ευαισθησία
- Το BLAT είναι αρκετές φορές ταχύτερο από το BLAST, αλλά τα καλύτερα αποτελέσματα περιορίζονται σε εξελικτικά κοντινές αλληλουχίες