

# Lecture 17: Approximate Pattern Matching

Study Chapter 9.6 – 9.8

## Approximate vs. Exact Pattern Matching



- Last time we discussed exact pattern matching algorithms
- Usually, because of mutations, it makes much more biological sense to find approximate pattern matches
- Biologists often use **fast heuristic** approaches (rather than local alignment) to find approximate matches



# Heuristic Similarity Searches



- Genomes are huge: Smith-Waterman quadratic alignment algorithms are too slow
- Alignment of two sequences usually has short identical or highly similar fragments
- Many heuristic methods (i.e., FASTA) are based on the same idea of *filtration*
  - Find short exact matches, and use them as seeds for potential match extension
  - “Filter” out positions with no extendable matches



# Dot Matrices



- Dot matrices show similarities between two sequences
- FASTA makes an implicit dot matrix from short exact matches, and tries to find long diagonals (allowing for some mismatches)

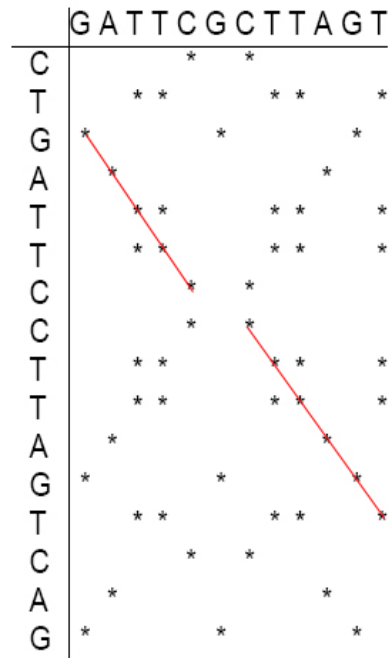
	G	A	T	T	C	G	C	T	T	A	G	T
C						*			*			
T		*	*					*	*			*
G	*					*						*
A		*								*		
T		*	*					*	*			*
T		*	*					*	*			*
C					*		*					
C					*		*					
T		*	*					*	*			*
T		*	*					*	*			*
A		*								*		
G	*					*					*	
T		*	*					*	*			*
C					*		*					
A		*								*		
G	*					*					*	



# Dot Matrices



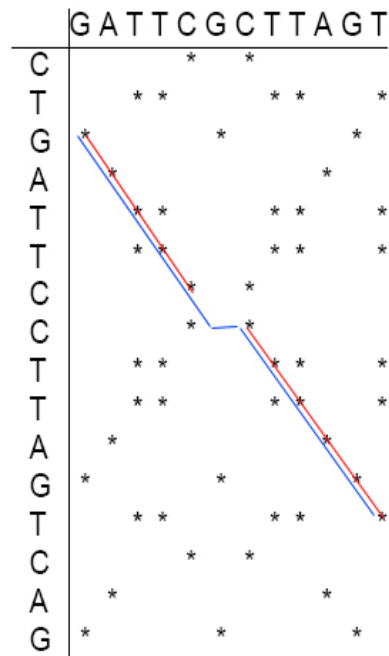
- Identify diagonals above a threshold length
- Diagonals in the dot matrix indicate exact substring matching



# Diagonals in Dot Matrices



- Extend diagonals and try to link them together, allowing for minimal mismatches/indels
- Linking diagonals reveals approximate matches over longer substrings



# Approximate Pattern Matching Problem



- Goal: Find all approximate occurrences of a pattern in a text
- Input:
  - pattern  $\mathbf{p} = p_1 \dots p_n$
  - text  $\mathbf{t} = t_1 \dots t_m$
  - the maximum number of mismatches  $k$
- Output: All positions  $1 \leq i \leq (m - n + 1)$  such that  $t_i \dots t_{i+n-1}$  and  $p_1 \dots p_n$  have at most  $k$  mismatches
  - i.e., Hamming distance between  $t_i \dots t_{i+n-1}$  and  $\mathbf{p} \leq k$



## Approximate Pattern Matching: A Brute-Force Algorithm



### ApproximatePatternMatching(p, t, k)

```
1  n ← length of pattern p
2  m ← length of text t
3  for i ← 1 to m - n + 1
4    dist ← 0
5    for j ← 1 to n
6      if  $t_{i+j-1} \neq p_j$ 
7        dist ← dist + 1
8  if dist ≤ k
9    output i
```



## Approximate Pattern Matching: Running Time



- That algorithm runs in  $O(nm)$ .
- We can generalize the “Approximate Pattern Matching Problem” into a “Query Matching Problem”:
  - We want to match substrings in a query to substrings in a text with at most  $k$  mismatches
  - **Motivation:** we want to see similarities to some gene, but we may not know which parts of the gene to look for



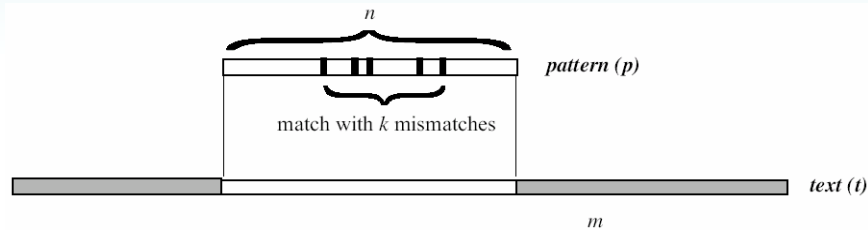
## Query Matching Problem



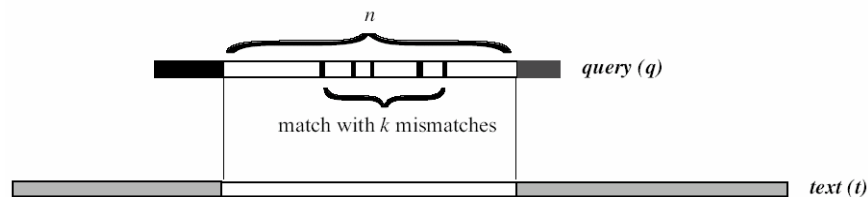
- Goal: Find all substrings of the query that approximately match the text
- Input: Query  $\mathbf{q} = q_1 \dots q_w$   
text  $\mathbf{t} = t_1 \dots t_m$   
 $n$  (length of matching substrings),  
 $k$  (maximum number of mismatches)
- Output: All pairs of positions  $(i, j)$  such that the  
 $n$ -letter substring of  $\mathbf{q}$  starting at  $i$  approximately matches the  
 $n$ -letter substring of  $\mathbf{t}$  starting at  $j$ ,  
with at most  $k$  mismatches



# Approximate Pattern Matching vs Query Matching



(a) Approximate Pattern Matching



(b) Query Matching



## Query Matching: Main Idea



- Approximately matching strings share some perfectly matching substrings.
- Instead of searching for approximately matching strings (difficult) search for perfectly matching substrings first (easy).



# Filtration in Query Matching



- We want all  $n$ -matches between a query and a text with up to  $k$  mismatches
- “Filter” out positions that do not match between text and query
- **Potential match detection:** find all matches of  $\ell$ -tuples in query and text for some small  $\ell$
- **Potential match verification:** Verify each potential match by extending it to the left and right, until  $(k + 1)$  mismatches are found



## Filtration: Match Detection



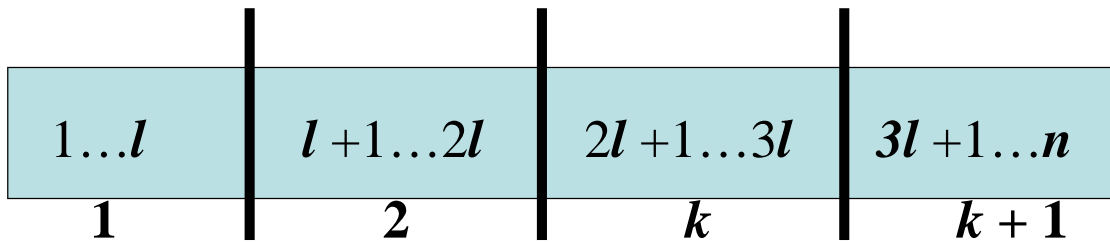
- If  $x_1 \dots x_n$  and  $y_1 \dots y_n$  match with at most  $k$  mismatches, they must share an  $\ell$ -tuple that is perfectly matched, with  $\ell = \lfloor n / (k + 1) \rfloor$
- Break string of length  $n$  into  $k+1$  parts, each of length  $\lfloor n / (k + 1) \rfloor$ 
  - $k$  mismatches can affect at most  $k$  of these  $k+1$  parts
  - At least one of these  $k+1$  parts is perfectly matched



## Filtration: Match Detection (cont'd)



- Suppose  $k = 3$ . We would then have  $l = n / (k + 1) = n / 4$ :



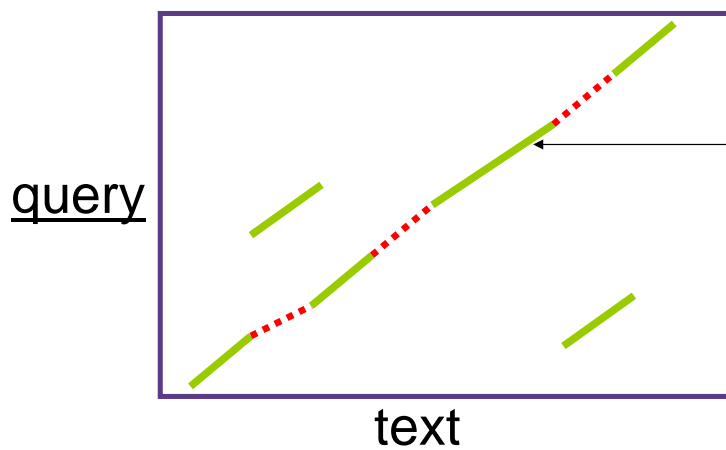
- There are at most  $k$  mismatches in  $n$ , so at the very least there must be one out of the  $k + 1$   $l$ -tuples without a mismatch



## Filtration: Match Verification



- For each  $l$ -match we find, try to extend the match further to see if it is substantial



Extend perfect match of length  $l$  until we find an approximate match of length  $n$  with  $k$  mismatches



# Filtration: Example



	$k = 0$	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$
$\ell$ -tuple length	$n$	$n/2$	$n/3$	$n/4$	$n/5$	$n/6$

Shorter perfect matches required

Performance decreases



# Local alignment is too slow...



- Quadratic local alignment is too slow while looking for similarities between long strings (e.g. the entire GenBank database)

$$s_{i,j} = \max \begin{cases} 0 \\ s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \\ s_{i-1,j-1} + \delta(v_i, w_j) \end{cases}$$



# Local alignment is too slow...



- Quadratic local alignment is too slow while looking for similarities between long strings (e.g. the entire GenBank database)
- Guaranteed to find the optimal local alignment
- Sets the standard for sensitivity

$$s_{i,j} = \max \begin{cases} 0 \\ s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \\ s_{i-1,j-1} + \delta(v_i, w_j) \end{cases}$$



# Local alignment is too slow...



- Quadratic local alignment is too slow while looking for similarities between long strings (e.g. the entire GenBank database)
- **Basic Local Alignment Search Tool**
  - Altschul, S., Gish, W., Miller, W., Myers, E. & Lipman, D.J.  
Journal of Mol. Biol., 1990
- Search sequence databases for local alignments to a query

$$s_{i,j} = \max \begin{cases} 0 \\ s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \\ s_{i-1,j-1} + \delta(v_i, w_j) \end{cases}$$



# BLAST



- Great improvement in speed, with a modest decrease in sensitivity
- Minimizes search space instead of exploring entire search space between two sequences
- Finds short exact matches (“seeds”), only explores locally around these “hits”



## What Similarity Reveals



- BLASTing a new gene
  - Evolutionary relationship
  - Similarity between protein function
- BLASTing a genome
  - Potential genes



# Measuring Similarity



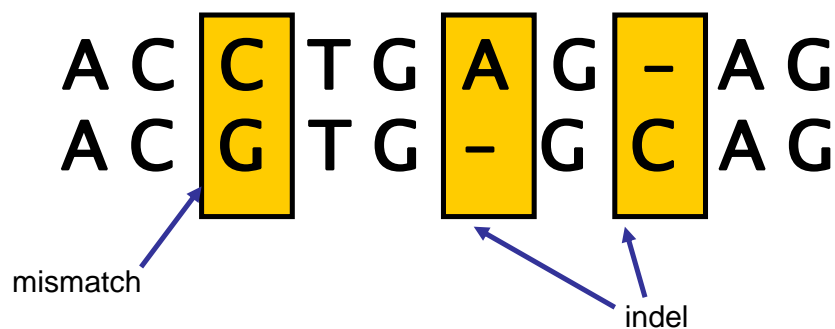
- Measuring the extent of similarity between two sequences
  - Based on percent sequence identity
  - Based on conservation



# Percent Sequence Identity



- The extent to which two nucleotide or amino acid sequences are invariant



# Conservation



- Amino acid changes that preserve the physico-chemical properties of the original residue
  - Polar to polar
    - aspartate → glutamate
  - Nonpolar to nonpolar
    - alanine → valine
  - Similarly behaving residues
    - leucine to isoleucine



# Scoring matrices



- Amino acid substitution matrices
  - PAM
  - BLOSUM
- DNA substitution matrices
  - DNA: less conserved than protein sequences
  - Less effective to compare coding regions at nucleotide level



# PAM



- **Point Accepted Mutation** (Dayhoff et al.)
- 1 PAM = PAM<sub>1</sub> = 1% average change of all amino acid positions
  - After 100 PAMs of evolution, not every residue will have changed
    - May have mutated several times
    - Returned to their original state
    - Not changed at all

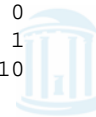


# PAM<sub>x</sub>



- PAM<sub>250</sub> is most widely used:

	ORIGINAL AMINO ACID												
	Ala	Arg	Asn	Asp	Cys	Gln	Glu	Gly	His	Ile	Leu	Lys	...
	A	R	N	D	C	Q	E	G	H	I	L	K	...
Ala A	13	6	9	9	5	8	9	12	6	8	6	7	...
Arg R	3	17	4	3	2	5	3	2	6	3	2	9	
Asn N	4	4	6	7	2	5	6	4	6	3	2	5	
Asp D	5	4	8	11	1	7	10	5	6	3	2	5	
Cys C	2	1	1	1	52	1	1	2	2	2	1	1	
Gln Q	3	5	5	6	1	10	7	3	7	2	3	5	
...													
Trp W	0	2	0	0	0	0	0	0	1	0	1	0	
Tyr Y	1	1	2	1	3	1	1	1	3	2	2	1	
Val V	7	4	4	4	4	4	4	4	5	4	15	10	



# BLOSUM



- **B**locks **S**ubstitution **M**atrix (Henikoff and Henikoff)
- Scores derived from *observations* of the frequencies of substitutions
  - From blocks of local alignments in related proteins
- Matrix name indicates evolutionary distance
  - BLOSUM62 was created using sequences sharing no more than 62% identity



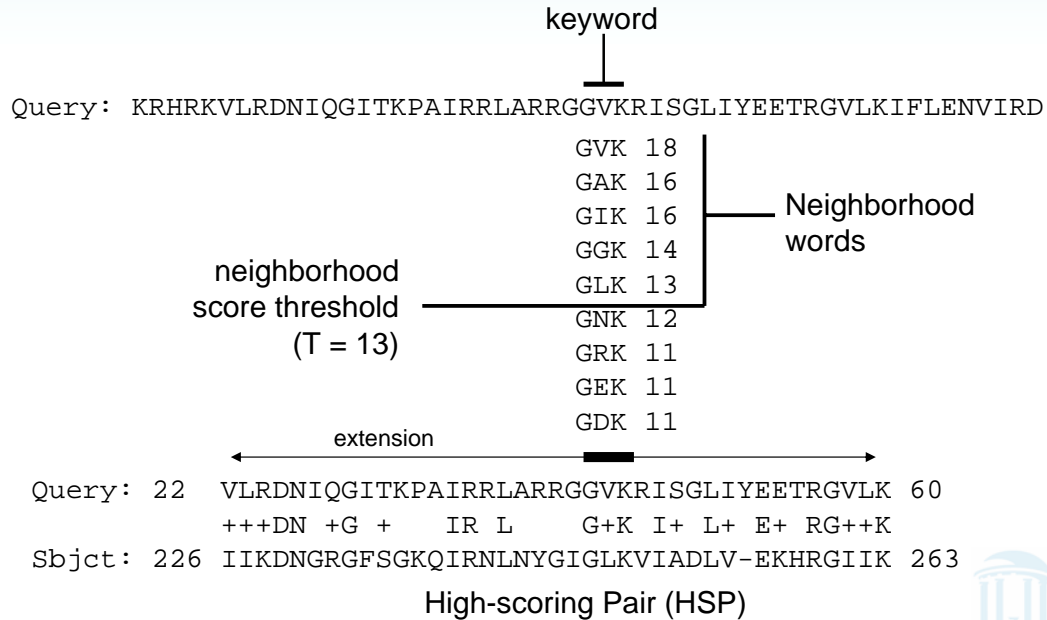
# BLAST algorithm



- **Keyword search** of all words of length  $w$  from the query of length  $n$  in database of length  $m$  with score above threshold
  - $w = 11$  for DNA queries,  $w = 3$  for proteins
- **Local alignment extension** for each found keyword
  - Extend result until longest match above threshold is achieved
- Running time  $O(nm)$



# BLAST algorithm



# Original BLAST



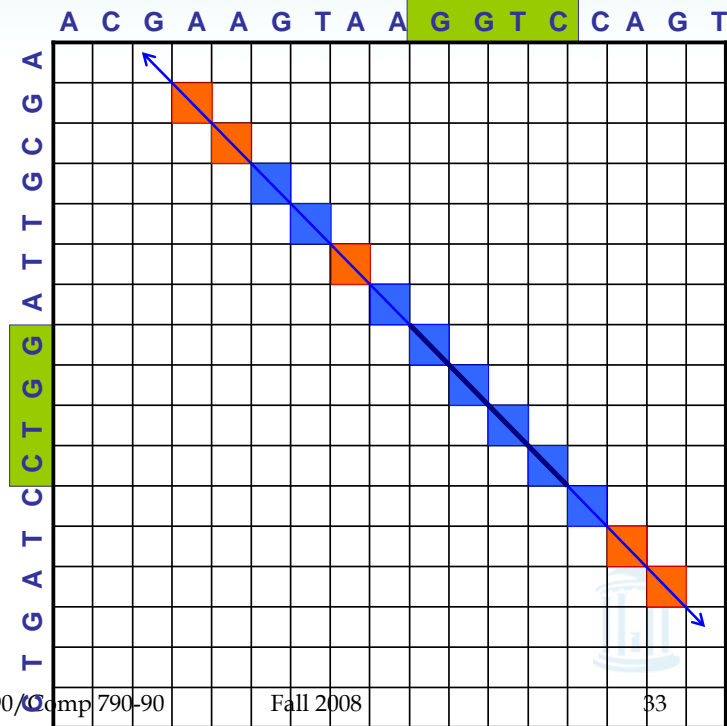
- **Dictionary**
  - All words of length  $w$
- **Alignment**
  - Ungapped extensions until score falls below some statistical threshold
- **Output**
  - All local alignments with score  $>$  threshold



# Original BLAST: Example



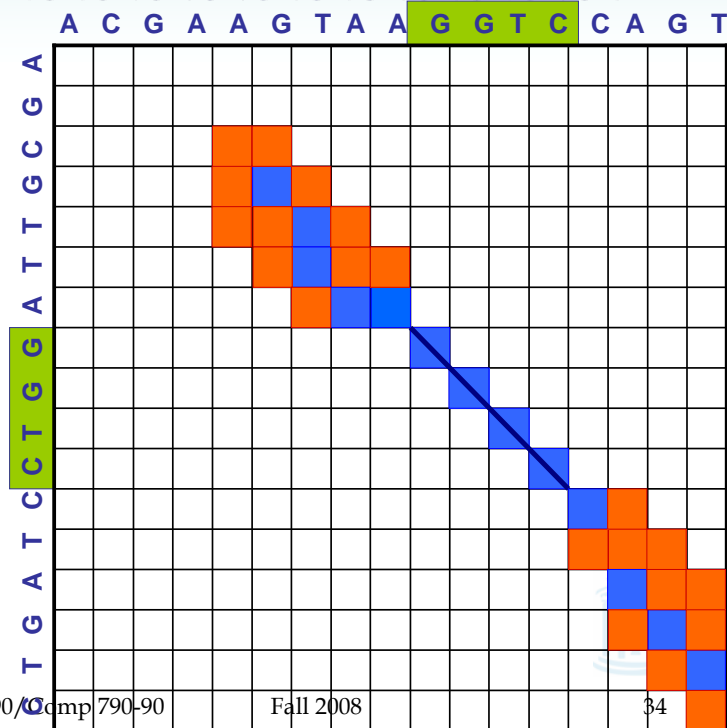
- $w = 4$
- Exact keyword match of **GGTC**
- Extend diagonals with mismatches until score is under 50%
- Output result  
**GTAAGGTCC**  
**GTTAGGTCC**



# Gapped BLAST : Example



- Original BLAST exact keyword search, THEN:
- Extend with gaps around ends of exact match **until score < threshold**
- Output result  
**GTAAGGTCCAGT**  
**GTTAGGTC-AGT**



# Incarnations of BLAST



- blastn: Nucleotide-nucleotide
- blastp: Protein-protein
- blastx: Translated query vs. protein database
- tblastn: Protein query vs. translated database
- tblastx: Translated query vs. translated database (6 frames each)



# Incarnations of BLAST (cont'd)



- PSI-BLAST
  - Find members of a protein family or build a custom position-specific score matrix
- Megablast:
  - Search longer sequences with fewer differences
- WU-BLAST: (Wash U BLAST)
  - Optimized, added features



# Assessing sequence similarity



- Need to know how strong an alignment can be expected from chance alone
- “Chance” relates to comparison of sequences that are generated randomly based upon a certain sequence model
- Sequence models may take into account:
  - G+C content
  - Poly-A tails
  - “Junk” DNA
  - Codon bias
  - Etc.



# BLAST: Segment Score



- BLAST uses scoring matrices ( $\delta$ ) to improve on efficiency of match detection
  - Some proteins may have very different amino acid sequences, but are still similar
- For any two  $\ell$ -mers  $x_1 \dots x_\ell$  and  $y_1 \dots y_\ell$ :
  - Segment pair: pair of  $\ell$ -mers, one from each sequence
  - Segment score:  $\sum_{i=1}^{\ell} \delta(x_i, y_i)$



## BLAST: Locally Maximal Segment Pairs



- A segment pair is maximal if it has the best score over all segment pairs
- A segment pair is locally maximal if its score can't be improved by extending or shortening
- Statistically significant *locally maximal* segment pairs are of biological interest
- BLAST finds all locally maximal segment pairs with scores above some threshold
  - A significantly high threshold will filter out some statistically insignificant matches



## BLAST: Statistics



- Threshold: Altschul-Dembo-Karlin statistics
  - Identifies smallest segment score that is unlikely to happen by chance
- # matches above  $\theta$  has mean  $E(\theta) = Kmne^{-\lambda\theta}$ ;  $K$  is a constant,  $m$  and  $n$  are the lengths of the two compared sequences
  - Parameter  $\lambda$  is a positive root of:  
$$\sum_{x,y \text{ in } A} (p_x p_y e^{\delta(x,y)}) = 1$$
, where  $p_x$  and  $p_y$  are frequencies of amino acids  $x$  and  $y$ ,  $\delta$  is the scoring matrix, and  $A$  is the twenty letter amino acid alphabet



# P-values



- The probability of finding  $b$  HSPs with a score  $\geq \theta$  is given by:
  - $(e^{-E}E^b)/b!$
- For  $b = 0$ , that chance is:
  - $e^{-E}$
- Thus the probability of finding at least one HSP with a score  $\geq \theta$  is:
  - $P = 1 - e^{-E}$



# Sample BLAST output



- Blast of human beta globin protein against zebra fish

Sequences producing significant alignments: (bits) Value

```
gi|18858329|ref|NP_571095.1| bal globin [Danio rerio] >gi|147757... 171 3e-44
gi|18858331|ref|NP_571096.1| ba2 globin; SI:dZ118J2.3 [Danio rer... 170 7e-44
gi|37606100|emb|CAE48992.1| SI:BY187G17.6 (novel beta globin) [D... 170 7e-44
gi|31419195|gb|AAH53176.1| Bal protein [Danio rerio] 168 3e-43
```

## ALIGNMENTS

```
>gi|18858329|ref|NP_571095.1| bal globin [Danio rerio]
Length = 148
```

Score = 171 bits (434), Expect = 3e-44  
Identities = 76/148 (51%), Positives = 106/148 (71%), Gaps = 1/148 (0%)

```
Query: 1 MVHLTPEEKSAVTALWGKVNVDVEVGGELGRLLVVYPWTQRFESFGDLSTPDAVMGNPK 60
MV T E++A+ LWGK+N+DE+G +AL R L+VYPWTQR+F +FG+LS+P A+MGNPK
Sbjct: 1 MVEWTD AERTAILGLWGLNIDEIGPQALSRLIVYPWTQRYFATFGNLSSPAAIMGNPK 60
```

```
Query: 61 VKAHGKVKLVGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVLCVLAHHPG 120
V AHG+ V+G + ++DN+K T+A LS +H +KLHVDP+NFRL + + A FG
Sbjct: 61 VAAHGRTVMGGLERAIKNMDNVKNTYAALSVMHSEKLHVDPDNFRLLADLCITVCAAMKFG 120
```

```
Query: 121 KE-FTPPVQAAYQKVVAGVANALAHKYH 147
+ F VQ A+QK +A V +AL +YH
Sbjct: 121 QAGFNADVQEAWQKFLAVVVSALCRQYH 148
```





# PatternHunter: faster and even more sensitive



- BLAST: matches short consecutive sequences (consecutive seed)
- Length =  $k$
- Example ( $k = 11$ ):
- PatternHunter: matches short non-consecutive sequences (spaced seed)
- Increases sensitivity by locating homologies that would otherwise be missed
- Example (a spaced seed of length 18 w/ 11 "matches"):

11111111111

111010010100110111

Each 1 represents a "match"

Each 0 represents a "don't care", so there can be a match or a mismatch



## Spaced seeds



Example of a hit using a spaced seed:

```

GAGTACTCAACACCAACATTAGTGGCAATGGAAAAT...
||  |||  |||  |||  |||  |||  |||  |||  |||  |||  |||
GAATACTCAACAGCAACACTAATGGCAGCAGAAAAT...
111010010100110111
    
```

How does this result in better sensitivity?



# Why is PH better?



- BLAST: redundant hits

## ■ PatternHunter

```
TTGACCTCACC?
|||||||?
TTGACCTCACC?
11111111111
11111111111
```

This results in > 1 hit and creates clusters of redundant hits

```
CAA?A??A?C??TA?TGG?
|||?|??|?|??|?|||?
CAA?A??A?C??TA?TGG?
111010010100110111
111010010100110111
```

This results in very few redundant hits



# Why is PH better?



## BLAST may also miss a hit

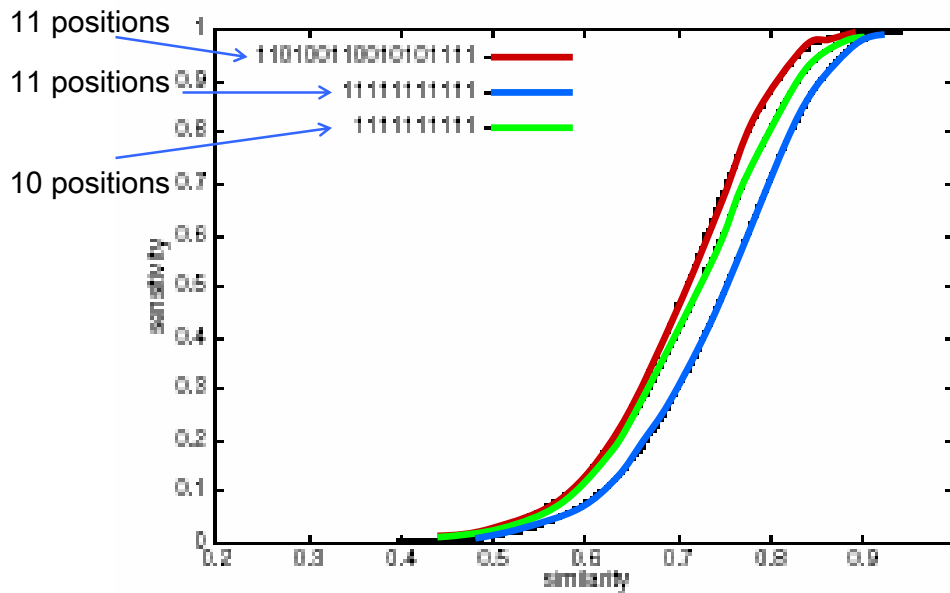
```
GAGTACTCAACACCAACATTAGTGGGCAATGGAAAAT
|| ||| ||| ||| ||| ||| ||| ||| |||
GAATACTCAACAGCAACATCAATGGGCAGCAGAAAAT
↔
9 matches
```

In this example, despite a clear homology, there is no sequence of continuous matches longer than length 9. BLAST uses a length 11 and because of this, BLAST does not recognize this as a hit!

Resolving this would require reducing the seed length to 9, which would have a damaging effect on speed



# Advantage of Gapped Seeds



10/28/2008

Comp 590/Comp 790-90

Fall 2008



49

## Why is PH better?



- Higher hit probability
- Lower expected number of random hits

10/28/2008

Comp 590/Comp 790-90

Fall 2008



50

# Use of Multiple Seeds



## Basic Searching Algorithm

1. Select a group of spaced seed models
2. For each hit of each model, conduct extension to find a homology.



# Another method: BLAT



- BLAT (BLAST-Like Alignment Tool)
- Same idea as BLAST - locate short sequence hits and extend



# BLAT vs. BLAST: Differences



- BLAT builds an index of the database and scans linearly through the query sequence, whereas BLAST builds an index of the query sequence and then scans linearly through the database
- Index is stored in RAM which is memory intensive, but results in faster searches



# BLAT: Fast cDNA Alignments



## Steps:

1. Break cDNA into 500 base chunks.
2. Use an index to find regions in genome similar to each chunk of cDNA.
3. Do a detailed alignment between genomic regions and cDNA chunk.
4. Use dynamic programming to stitch together detailed alignments of chunks into detailed alignment of whole.

A sophisticated divide and conquer approach



## However...



- BLAT was designed to find sequences of 95% and greater similarity of length >40; may miss more divergent or shorter sequence alignments



## PatternHunter and BLAT vs. BLAST



- PatternHunter is 5-100 times faster than Blastn, depending on data size, at the same sensitivity
- BLAT is several times faster than BLAST, but best results are limited to closely related sequences

