Rapid sequence alignment and Database search

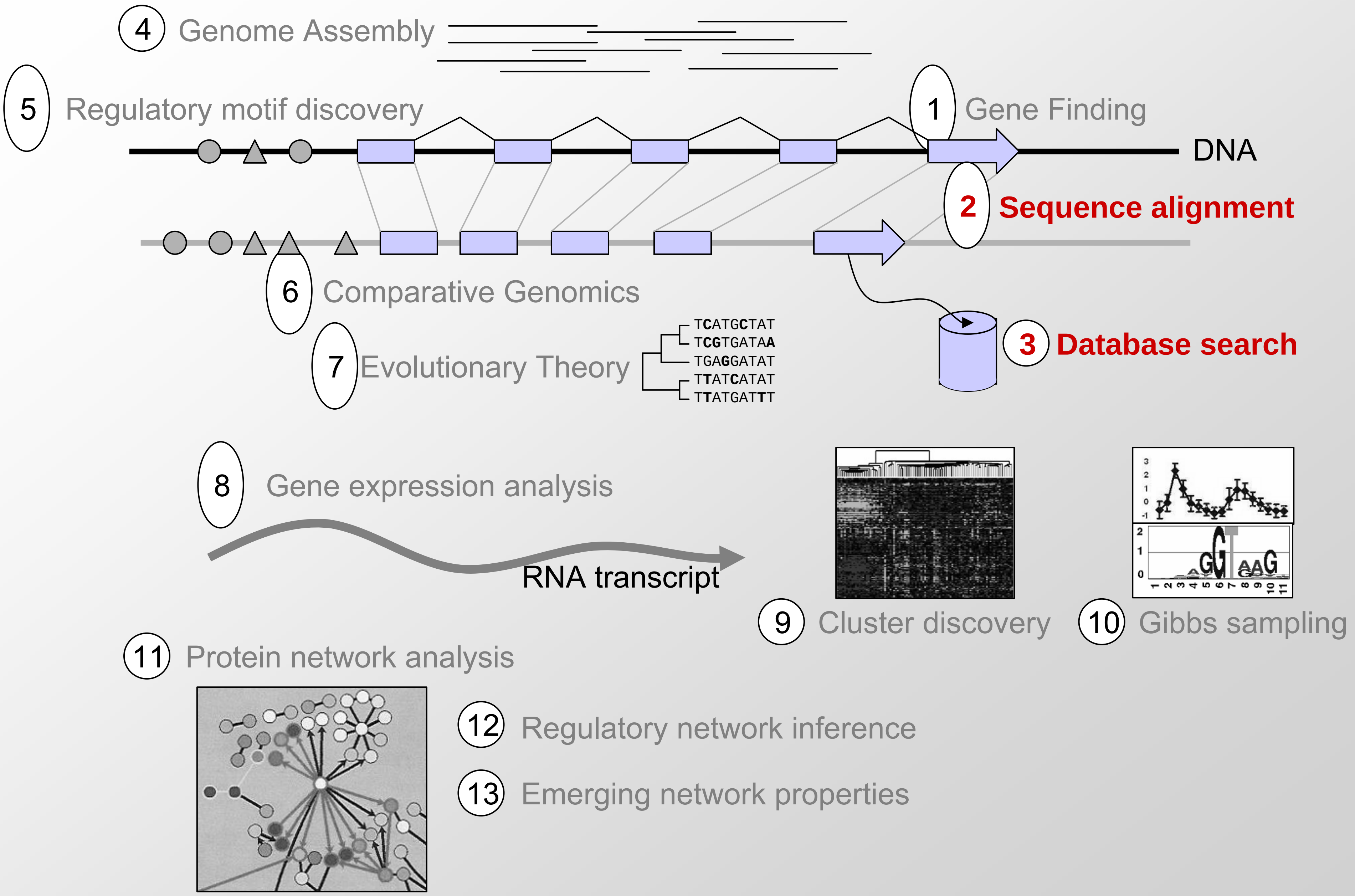
Local alignment, varying gap penalties

Karp-Rabin: Semi-numerical methods

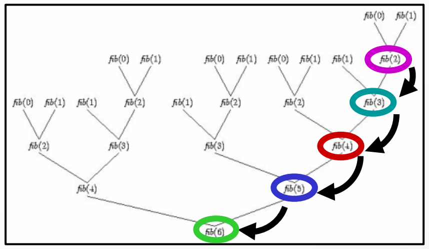
BLAST: dB search, neighborhood search

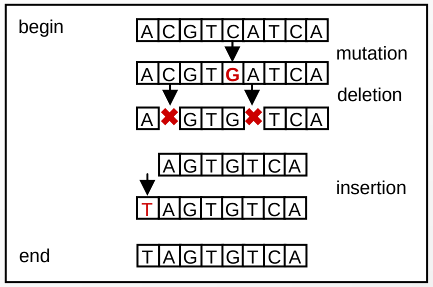
Statistics of alignment scores (recitation)

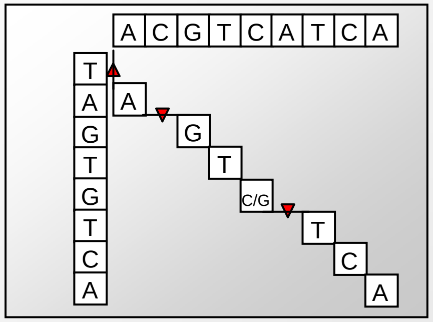
Challenges in Computational Biology

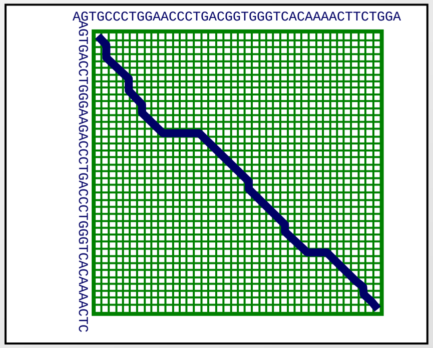


Tues: Sequence alignment + dynamic programming








AGTGCCCTGGAACCCTGACGGTGGGTCACAAAACTTCTGGA
AGTGACCTGGGAAGACCCTGACCCTGGGTCACAAAACTC

- Dynamic programming
  - Problems that can be decomposed into subparts
  - Identical sub-problems: reuse computation
  - Bottom-up approach: systematically fill table
- The sequence alignment problem
  - Genomes change: mutation, insertions, deletions
  - Alignment: infer evolutionary events
  - Scoring metric reflects evolutionary properties
- Dynamic programming and sequence alignment
  - Alignment scores are additive: decomposable
  - Represent sub-problem scores in M(i,j) matrix
  - Duality between alignment and path through matrix
- Needleman-Wunsch algorithm
  - Local update rule: F(i,j) = max{up, left, diagonal}
  - Save choice pointers for traceback
  - Bottom-right corner gives optimal alignment score
  - Trace-back of pointers gives optimal path/alignment

Today’s Goal: Diving deeper into alignments

1. **Global alignment vs. Local alignment**
   - Needleman-Wunsch and Smith-Waterman
   - Varying gap penalties and algorithmic speedups

2. **Linear-time exact string matching**
   - Karp-Rabin algorithm and semi-numerical methods
   - Hash functions and randomized algorithms

3. **The BLAST algorithm and inexact matching**
   - Hashing with neighborhood search
   - Two-hit blast and hashing with combs

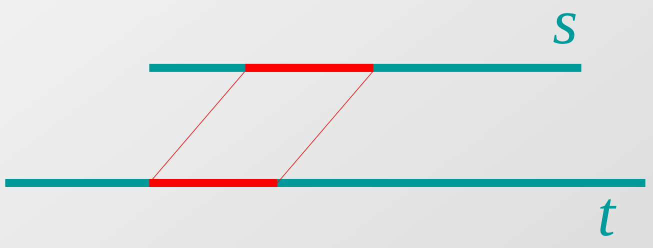
4. **Probabilistic foundations of sequence alignment**
   - Mismatch penalties, BLOSUM and PAM matrices
   - Statistical significance of an alignment score

Today's Goal: Diving deeper into alignments

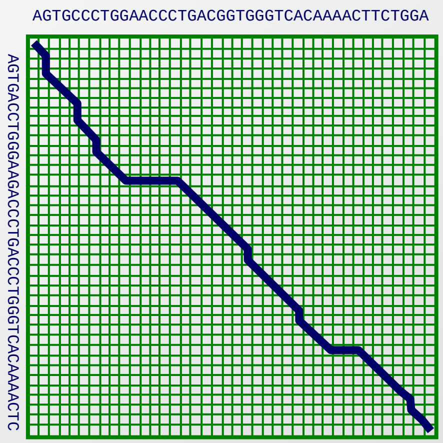
1. **Global alignment vs. Local alignment**
   - Needleman-Wunsch and Smith-Waterman
   - Varying gap penalties and algorithmic speedups
2. **Linear-time exact string matching**
   - Karp-Rabin algorithm and semi-numerical methods
   - Hash functions and randomized algorithms
3. **The BLAST algorithm and inexact matching**
   - Hashing with neighborhood search
   - Two-hit blast and hashing with combs
4. **Probabilistic foundations of sequence alignment**
   - Mismatch penalties, BLOSUM and PAM matrices
   - Statistical significance of an alignment score

Intro to Local Alignments

- Statement of the problem
  - A *local alignment* of strings *s* and *t* is an alignment of a substring of *s* with a substring of *t*

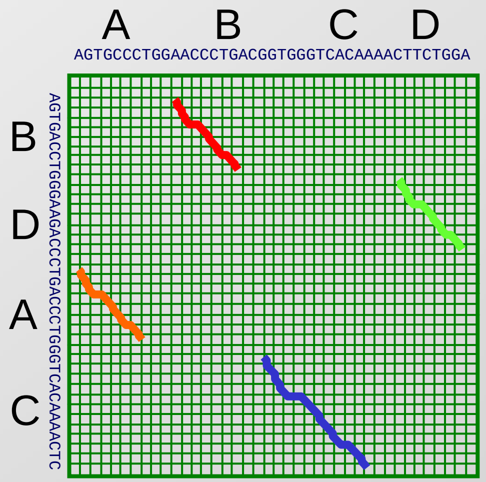


- Why local alignments?
  - Small domains of a gene may be only conserved portions
  - Looking for a small gene in a large chromosome (search)
  - Large segments often undergo rearrangements



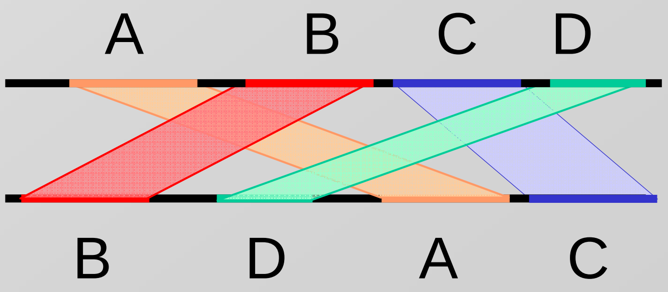
AGTGCCCTGGAACCCTGACGGTGGGTCACAAAACTTCTGGA
AGTGACCTGGGAAGACCCTGACCCTGGGTCACAAAACTC

Global alignment

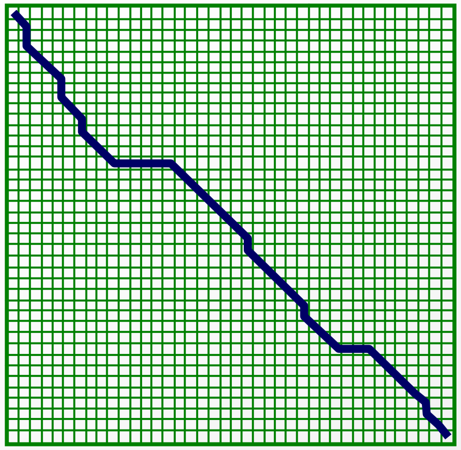


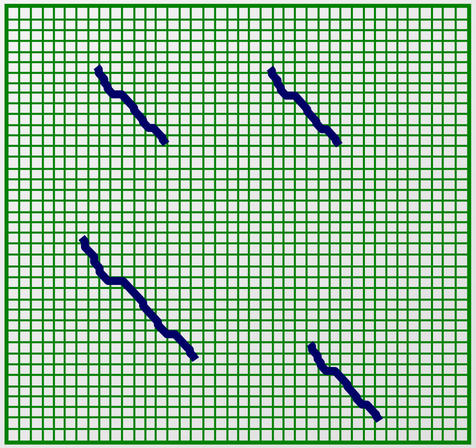
A
B
C
D
AGTGCCCTGGAACCCTGACGGTGGGTCACAAAACTTCTGGA
B
D
A
C
AGTGACCTGGGAAGACCCTGACCCTGGGTCACAAAACTC

Local alignment



Global Alignment vs. Local alignment





Needleman-Wunsch algorithm

Initialization: $F(0, 0) = 0$

Iteration:

$$F(i, j) = \max \begin{cases} F(i-1, j) - d \\ F(i, j-1) - d \\ F(i-1, j-1) + s(x_i, y_j) \end{cases}$$

Termination: Bottom right

Smith-Waterman algorithm

Initialization: $F(0, j) = F(i, 0) = 0$

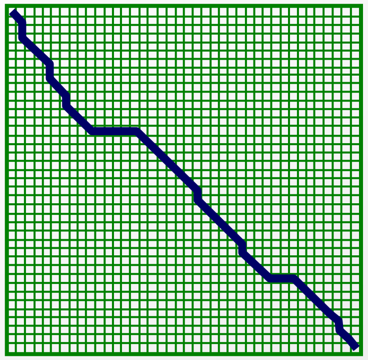
Iteration:

$$F(i, j) = \max \begin{cases} 0 \\ F(i-1, j) - d \\ F(i, j-1) - d \\ F(i-1, j-1) + s(xi, yj) \end{cases}$$

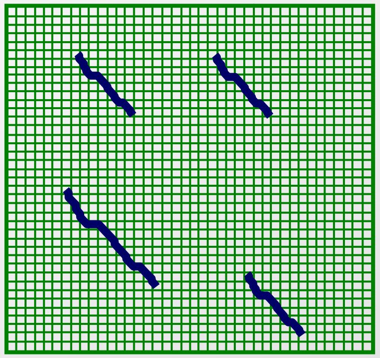
Termination: **Anywhere**

More variations on the theme: semi-global alignment

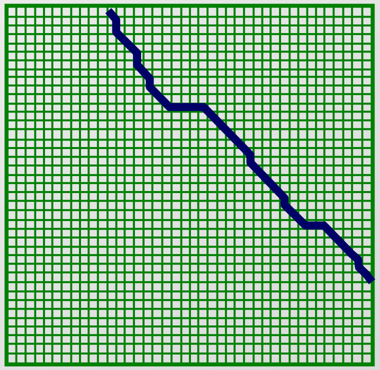
- Sequence alignment variations



Global



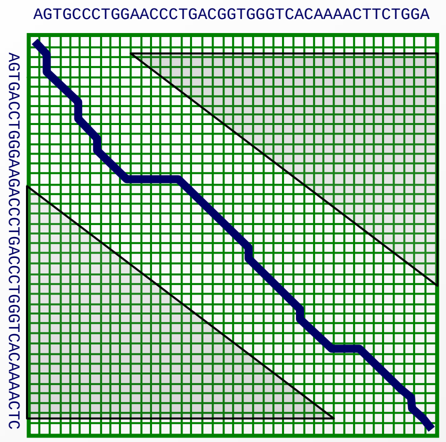
Local



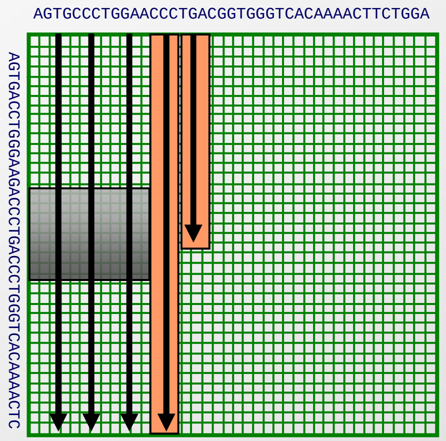
Semi-global

	Global	Local	Semi-global
Initialization	Top left	Top row/left col.	Top row
Iteration:max	$F(i-1, j) - d$ $F(i, j-1) - d$ $F(i-1, j-1) + s(x_i, y_j)$	0 $F(i-1, j) - d$ $F(i, j-1) - d$ $F(i-1, j-1) + s(x_i, y_j)$	$F(i-1, j) - d$ $F(i, j-1) - d$ $F(i-1, j-1) + s(x_i, y_j)$
Termination	Bottom right	Anywhere	Right column

Some algorithmic variations to save time/space


AGTGCCCTGGAACCCTGACGGTGGGTCACAAAACTTCTGGA
AGTGACCTGGGAAGACCCTGACCCTGGGTCACAAAACTC

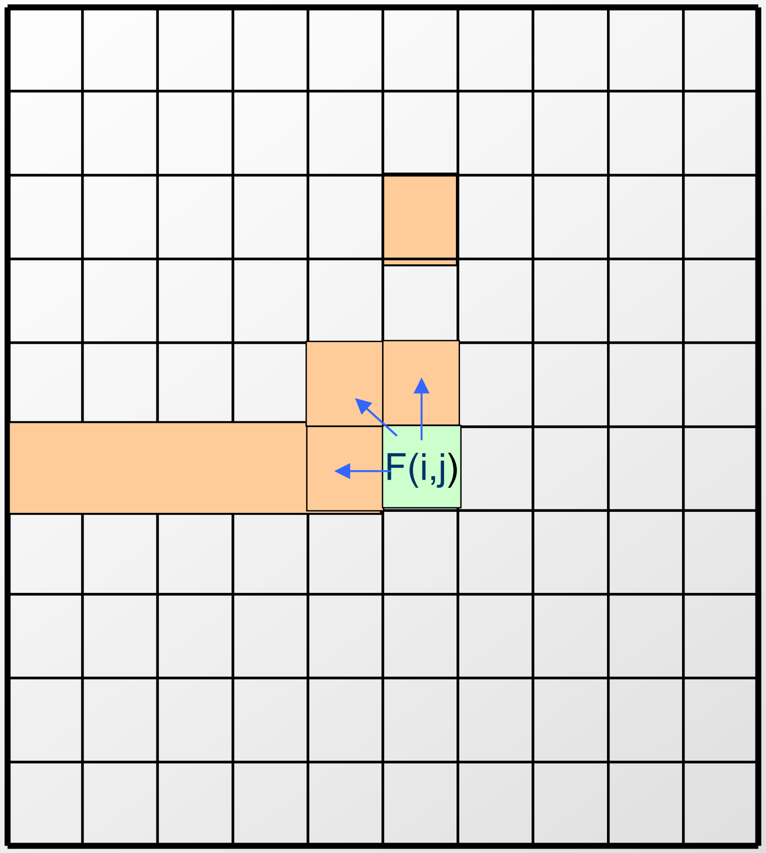
- Save time: Bounded-space computation
  - Space: O(k*m)
  - Time: O(k*m), where k = radius explored
  - Heuristic
    - Not guaranteed optimal answer
    - Works very well in practice
  - Practical interest


AGTGCCCTGGAACCCTGACGGTGGGTCACAAAACTTCTGGA
AGTGACCTGGGAAGACCCTGACCCTGGGTCACAAAACTC

- Save space: Linear-space computation
  - Save only one col / row / diag at a time
  - Computes optimal score easily
  - Recursive call modification allows traceback
  - Theoretical interest
    - Effective running time slower
    - Optimal answer guaranteed

Sequence alignment with generalized gap penalties

- **Implementing a generalized gap penalty function F(gap_length)**



Initialization: same

Iteration:

$$F(i, j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_j) \\ \max_{k=0\ldots i-1} F(k,j) - \gamma(i-k) \\ \max_{k=0\ldots j-1} F(i,k) - \gamma(j-k) \end{cases}$$

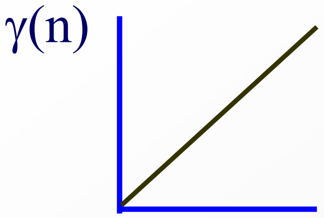
Termination: same

Running Time: $O(N^2M)$ (cubic)

Space: $O(NM)$

Do we have to be so general?

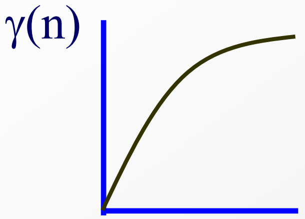
Algorithmic trade-offs of varying gap penalty functions



Linear gap penalty: w(k) = k*p

– State: Current index tells if in a gap or not

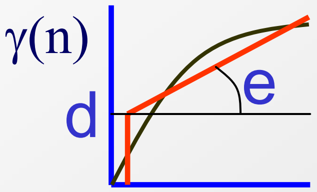
– Achievable using quadratic algorithm (even w/ linear space)



Quadratic: w(k) = p+q*k+rk^2.

– State: needs to encode the length of the gap, which can be O(n)

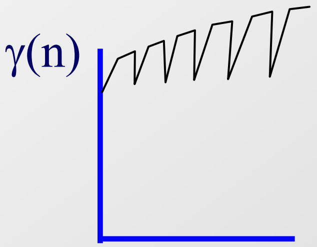
– To encode it we need O(log n) bits of information. Not feasible



Affine gap penalty: w(k) = p + q*k, where q<p

– State: add binary value for each sequence: starting a gap or not

– Implementation: add second matrix for already-in-gap (recitation)



Length (mod 3) gap penalty for protein-coding regions

– Gaps of length divisible by 3 are penalized less: conserve frame

– This is feasible, but requires more possible states

– Possible states are: starting, mod 3=1, mod 3=2, mod 3=0

Today's Goal: Diving deeper into alignments

1. Global alignment vs. Local alignment
   - Needleman-Wunsch and Smith-Waterman
   - Varying gap penalties and algorithmic speedups
2. **Linear-time exact string matching**
   - Karp-Rabin algorithm and semi-numerical methods
   - Hash functions and randomized algorithms
3. The BLAST algorithm and inexact matching
   - Hashing with neighborhood search
   - Two-hit blast and hashing with combs
4. Probabilistic foundations of sequence alignment
   - Mismatch penalties, BLOSUM and PAM matrices
   - Statistical significance of an alignment score

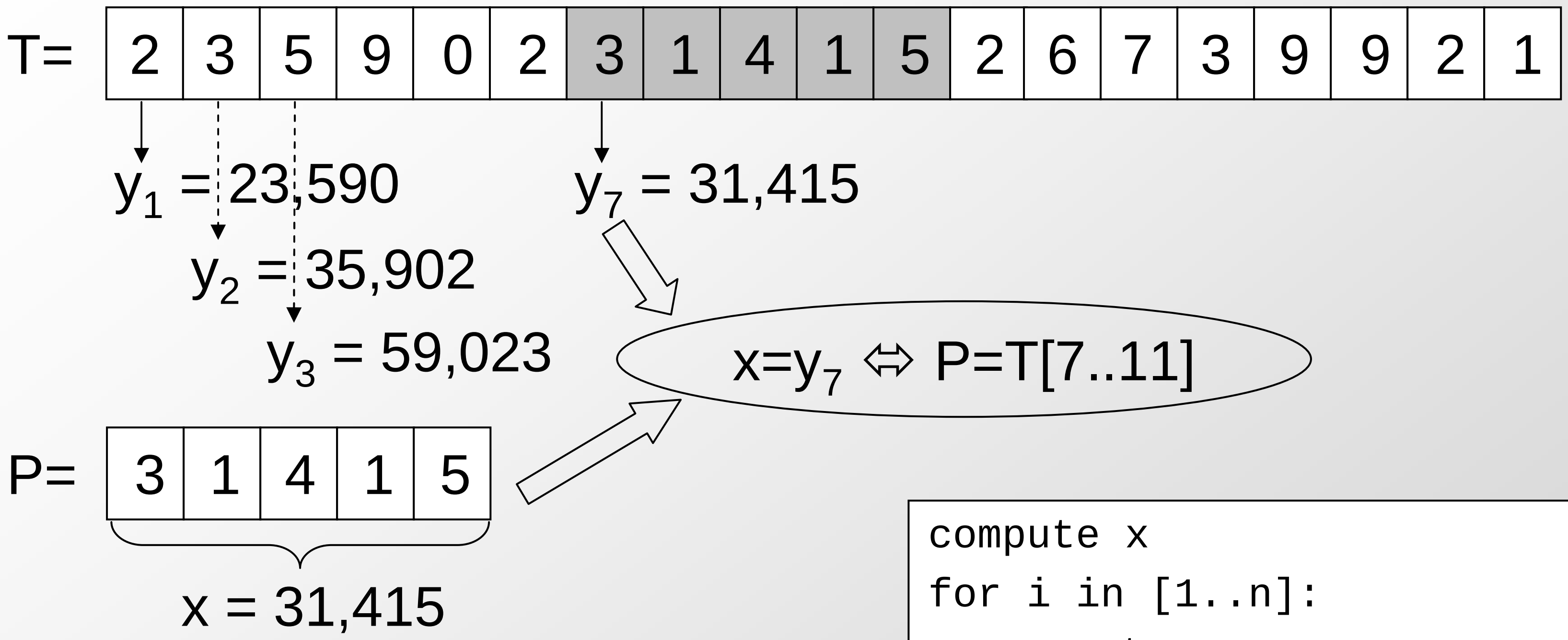
Linear-time string matching

- When looking for exact matches of a pattern

- Karp-Rabin algorithm: interpret it numerically
  - Start with ‘broken’ version of the algorithm
  - Progressively fix it to make it work

- Several other solutions exist, not covered today:
  - Z-algorithm / fundamental pre-processing, Gusfield
  - Boyer-Moore and Knuth-Morris-Pratt algorithms are earliest instantiations, similar in spirit
  - Suffix trees: beautiful algorithms, many different variations and applications, limited use in CompBio
  - Suffix arrays: practical variation, Gene Myers

Karp-Rabin algorithm

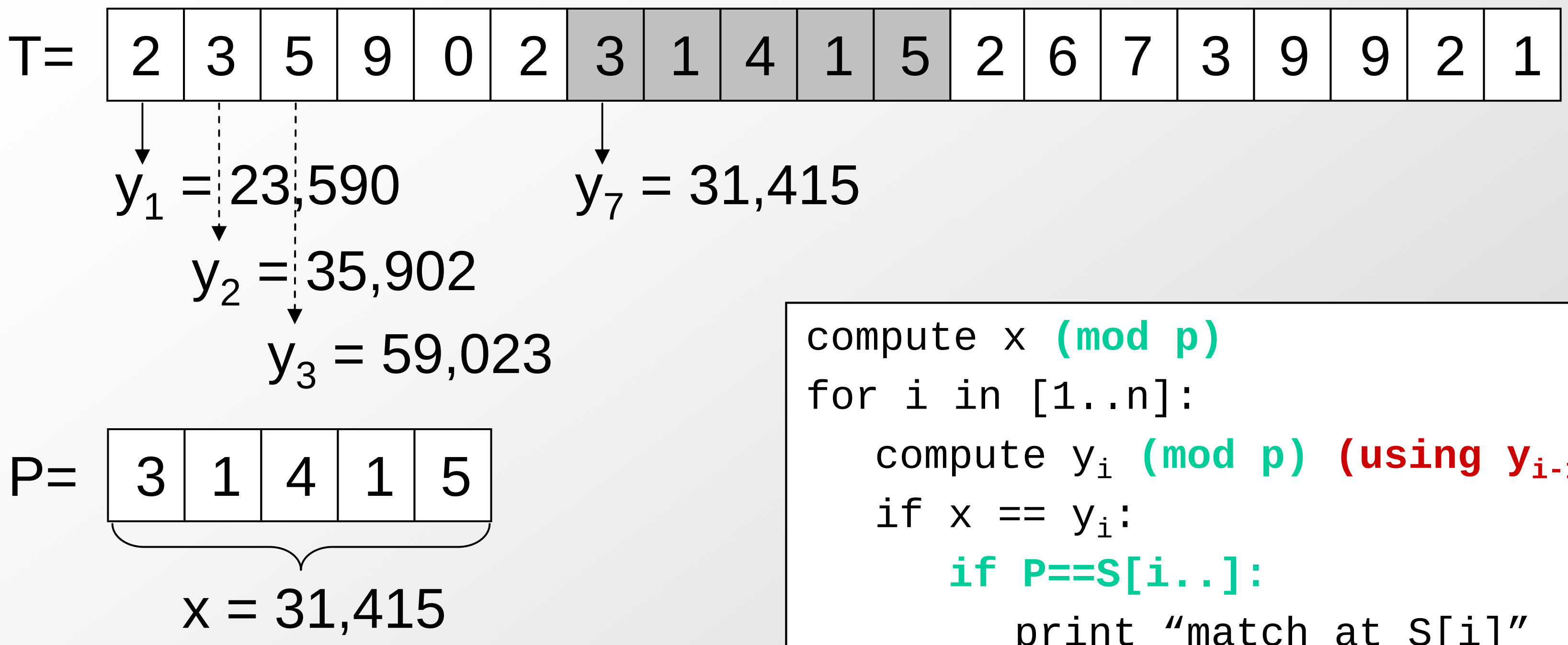


```
compute x
for i in [1..n]:
   compute y_i
   if x == y_i:
        print “match at S[i]”
```

(this does not actually work)

- Key idea:
  - Interpret strings as numbers: fast comparison

Karp-Rabin algorithm

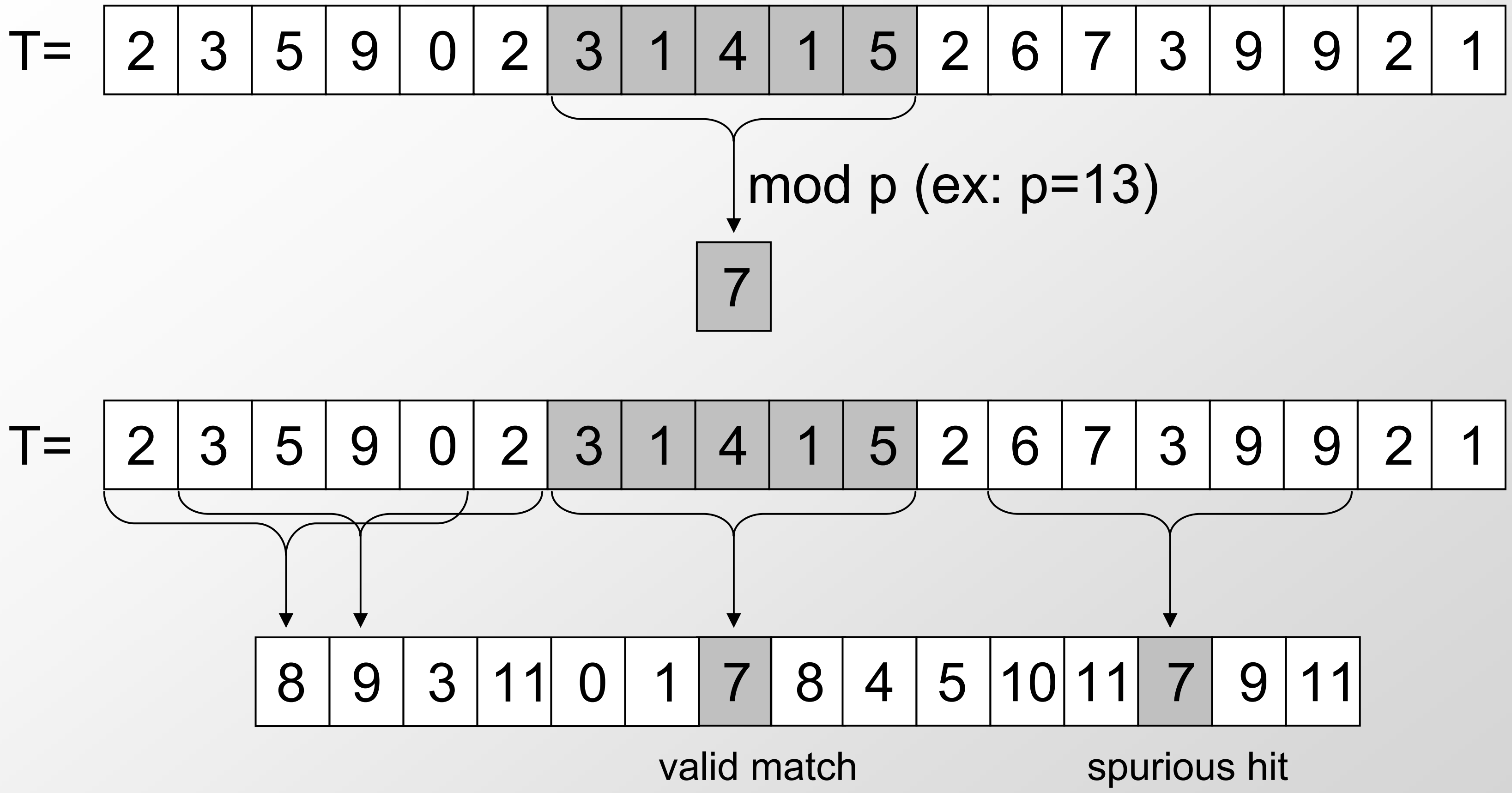


```
compute x (mod p)
for i in [1..n]:
   compute y_i (mod p) (using y_i-1)
   if x == y_i:
      if P==S[i..]:
         print “match at S[i]”
      else:
         (spurious hit)
```

(this actually works)

- Key idea:
  - Interpret strings as numbers: fast comparison
- To make it work:
  - **Compute next number based on previous one → O(1)**
  - **Hashing (mod p) → keep the numbers small → O(1)**

Hashing is good, but leads to collisions



- Consequences of (mod p) ‘hashing’
  - Good: Enable fast computation (use small numbers)
  - Bad: Leads to spurious hits (collisions)

➔ Complete algorithm must deal with the bad

Karp Rabin key idea: Semi-numerical approach

- Idea 1: ***semi-numerical*** approach:
  - Consider all m-mers:
    T[1…m], T[2…m+1], …, T[m-n+1…n]
  - Map each T[s+1…s+m] into a ***number*** t_s
  - Map the pattern P[1…m] into a ***number*** p
  - Report the m-mers that map to the same value as p

Semi-numerical approach: implementation

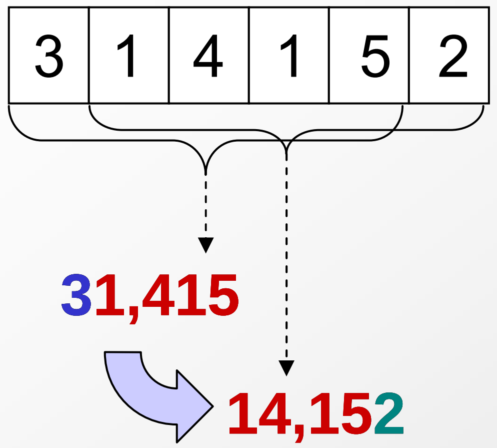
- First attempt:
  - Assume $\Sigma=\{0,1\}$

    (for {A,G,T,C} convert: A→00, G →01, A→10, G →11)
  - Think about each $T[s+1\ldots s+m]$ as a number in binary representation, i.e.,

    $$t_s=T[s+1]2^{m-1}+T[s+2]2^{m-2}+\ldots+T[s+m]2^0$$
  - Output all s such that t_s is equal to the number p represented by P

- Problem: how to map all m-mers in O(n) time ?
  - Find a fast way of computing t_{s+1} given t_s

Computing t_{s+1} based on t_s in constant time



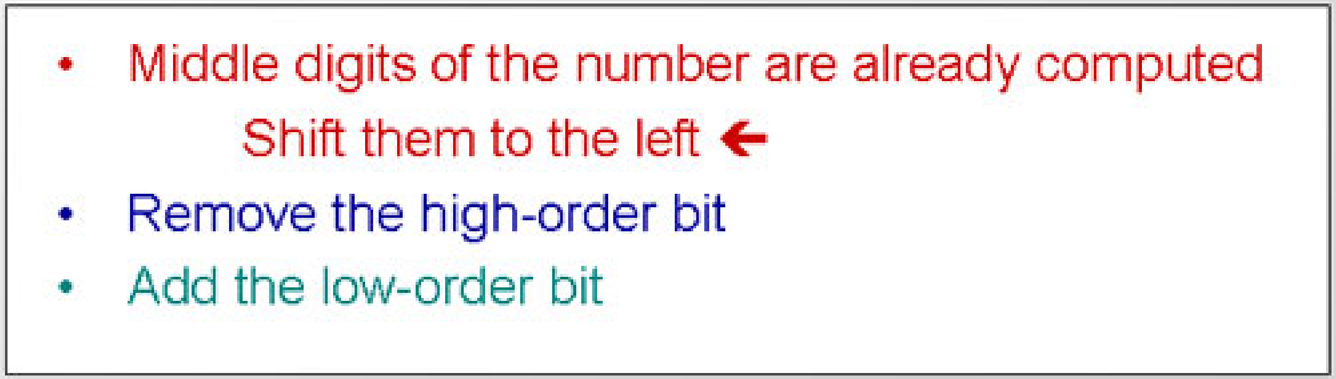
left shift

new low-order digit

old high-order bit

$$14{,}152 = (31{,}415 - 3 * 10{,}000) * 10 + 2$$

$$14{,}152 =? \text{ function } (31{,}415)$$



- Middle digits of the number are already computed

  Shift them to the left ←
- Remove the high-order bit
- Add the low-order bit

Idea 2: Computing all numbers in linear time

- How to transform

$$t_s=\underline{T[s+1]2^{m-1}}+T[s+2]2^{m-2}+\ldots+T[s+m]2^0$$

Into

$$t_{s+1}=T[s+2]2^{m-1}+T[s+3]2^{m-2}+\ldots+\underline{T[s+m+1]2^0}\ ?$$

- Can compute t_{s+1} from t_s using 3 arithmetic operations:
  - Subtract $T[s+1]2^{m-1}$
  - Multiply by 2 (i.e., shift the bits by one position)
  - Add $T[s+m+1]2^0$

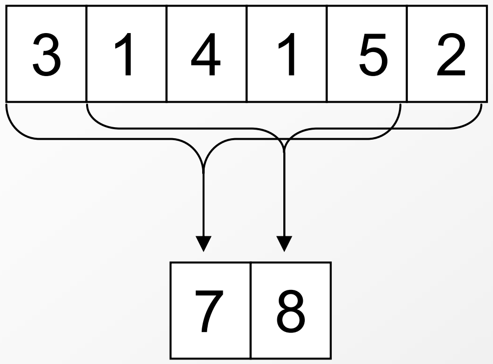
- Therefore: $t_{s+1}= (t_s- T[s+1]2^{m-1})*2 + T[s+m+1]2^0$

- Therefore, we can compute all $t_0,t_1,\ldots,t_{n-m}$ using $O(n)$ arithmetic operations, and a number for P in $O(m)$

Problem: Long strings = big numbers

- To get $O(n)$ time, we would need to perform each arithmetic operation in $O(1)$ time
- However, the arguments are m-bit long !
- If m large, it is unreasonable to assume that operations on such big numbers can be done in $O(1)$ time
- We need to reduce the number range to something more manageable

Dealing with long numbers in constant time



old high-order bit

shift

new low-order digit

14,152 = (31,415 - **3** * 10,000) *** 10** + **2** (mod 13)

= (7-3*3)*10+2 (mod 13)

= 8 (mod 13)

Idea 3: Hashing

- We will instead compute

  $$t'_s=T[s+1]2^{m-1}+T[s+2]2^{m-2}+\ldots+T[s+m]2^0 \textbf{ mod q}$$

  where q is an "appropriate" prime number

- One can still compute t'_{s+1} from t'_s :

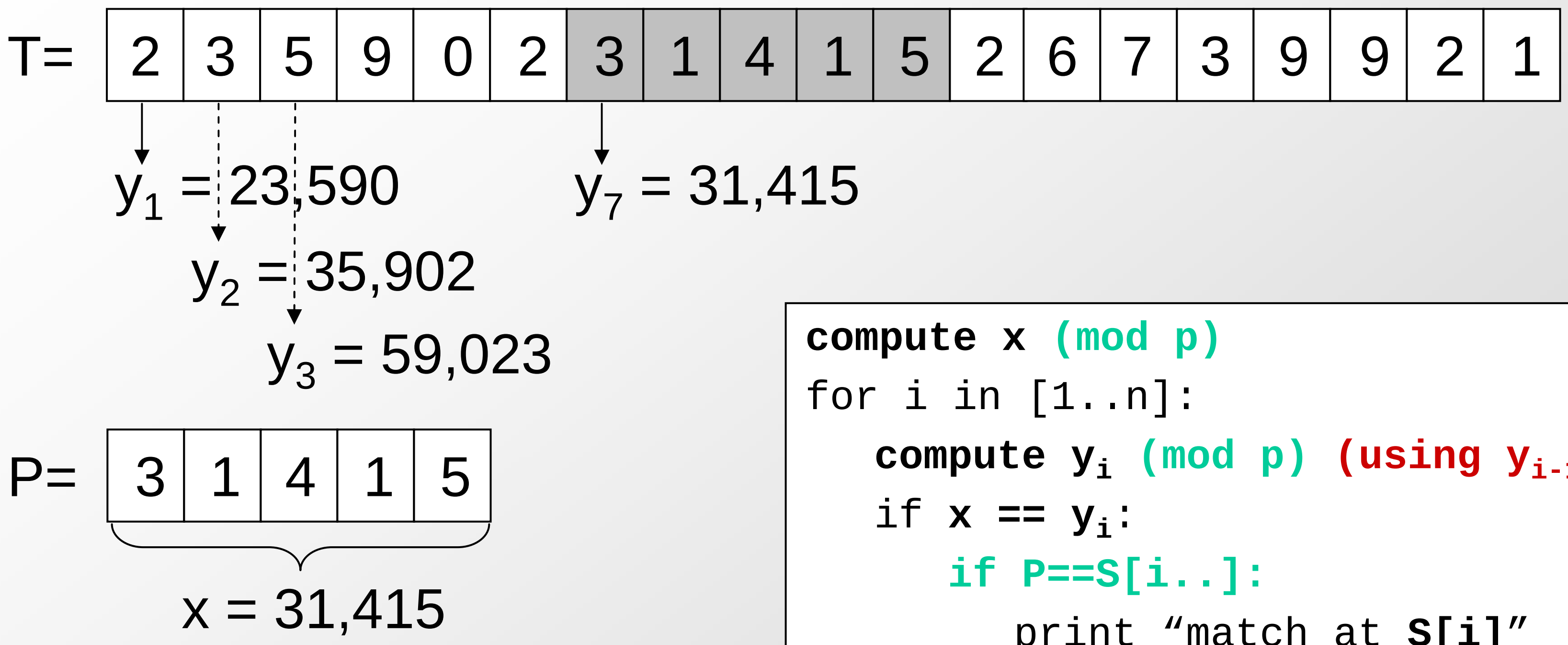
  $$t'_{s+1}= (t'_s- T[s+1]2^{m-1})*2+T[s+m+1]2^0 \text{ mod } q$$

- If q is not large, we can compute all t'_s (and p') in $O(n)$ time

Problem: hashing leads to false positives

- Unfortunately, we can have false positives, i.e., $T[s+1 \ldots s+m] \neq P$ but $t_s \bmod q = p \bmod q$
- Our approach:
  - Use a random q
  - Show that the probability of a false positive is small → randomized algorithm

Karp-Rabin algorithm: Putting it all together



```
compute x (mod p)
for i in [1..n]:
    compute $y_i$ (mod p) (using $y_{i-1}$)
    if x == $y_i$:
        if P==S[i..]:
            print “match at S[i]”
        else:
            (spurious hit)
```

(this actually works)

- Key idea: Semi-numerical computation
  - **Idea 1: Interpret strings as numbers => fast comparison**
    (other semi-numerical methods: Fast Fourier Transform, Shift-And)
- To make it work:
  - **Idea 2: Compute next number based on previous one → O(1)**
  - **Idea 3: Hashing (mod p) → keep the numbers small → O(1)**

Today’s Goal: Diving deeper into alignments

1. Global alignment vs. Local alignment
   - Needleman-Wunsch and Smith-Waterman
   - Varying gap penalties and algorithmic speedups
2. Linear-time exact string matching
   - Karp-Rabin algorithm and semi-numerical methods
   - Hash functions and randomized algorithms
3. **The BLAST algorithm and inexact matching**
   - Hashing with neighborhood search
   - Two-hit blast and hashing with combs
4. Probabilistic foundations of sequence alignment
   - Mismatch penalties, BLOSUM and PAM matrices
   - Statistical significance of an alignment score

Increased sequence availability → new problems

- Global Alignment and Dyn. Prog. Applications
  - Assume sequences have some common ancestry
  - Finding the “right” alignment between two sequences
    - Find minimum number of transformation operations
  - Understanding evolutionary events: mutations, indels
- Sequence databases
  - Query: new sequence. Subject: many old sequences
  - Goal: which sequences are related to the one at hand
  - most sequences will be completely unrelated to query
  - Individual alignment needs not be perfect.
    - Once initial matches are reported, can fine-tune them later
  - Query must be very fast for a new sequence

Speeding up your searches

- Exploit nature of the problem
  - If you’re going to reject any match with idperc <= 90, then why bother even looking at sequences which don’t have a stretch of 10 nucleotides in a row.
  - Pre-screen sequences for common long stretches
- Put the speed where you need it
  - Pre-processing the database is off-line.
  - Once the query arrives, must act fast
- Solution: content-based indexing and BLAST
  - Example: index 10-mers.
  - Only one 10-mer in 4^{10} will match, one in a million.
  - (even with 500 k-mers, only 1 in 2000 will match).
  - Additional speedups…

BLAST

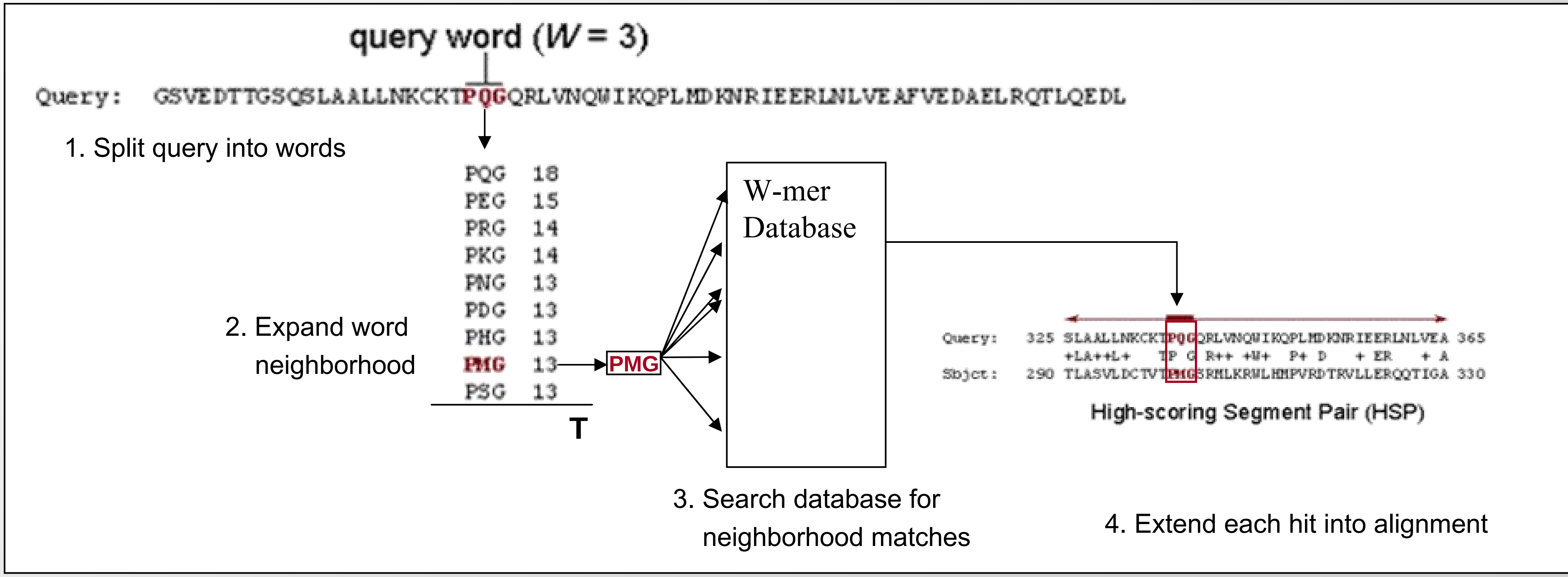
Basic local alignment search tool - all 46 versions »
SF Altschul, W Gish, W Miller, EW Myers, DJ Lipman - J. Mol. Biol, 1990

Gish', Webb Miller2 Eugene W. Myers3 and David J. Lipmanl **...**
Cited by **21457** - Related Articles - View as HTML - Web Search

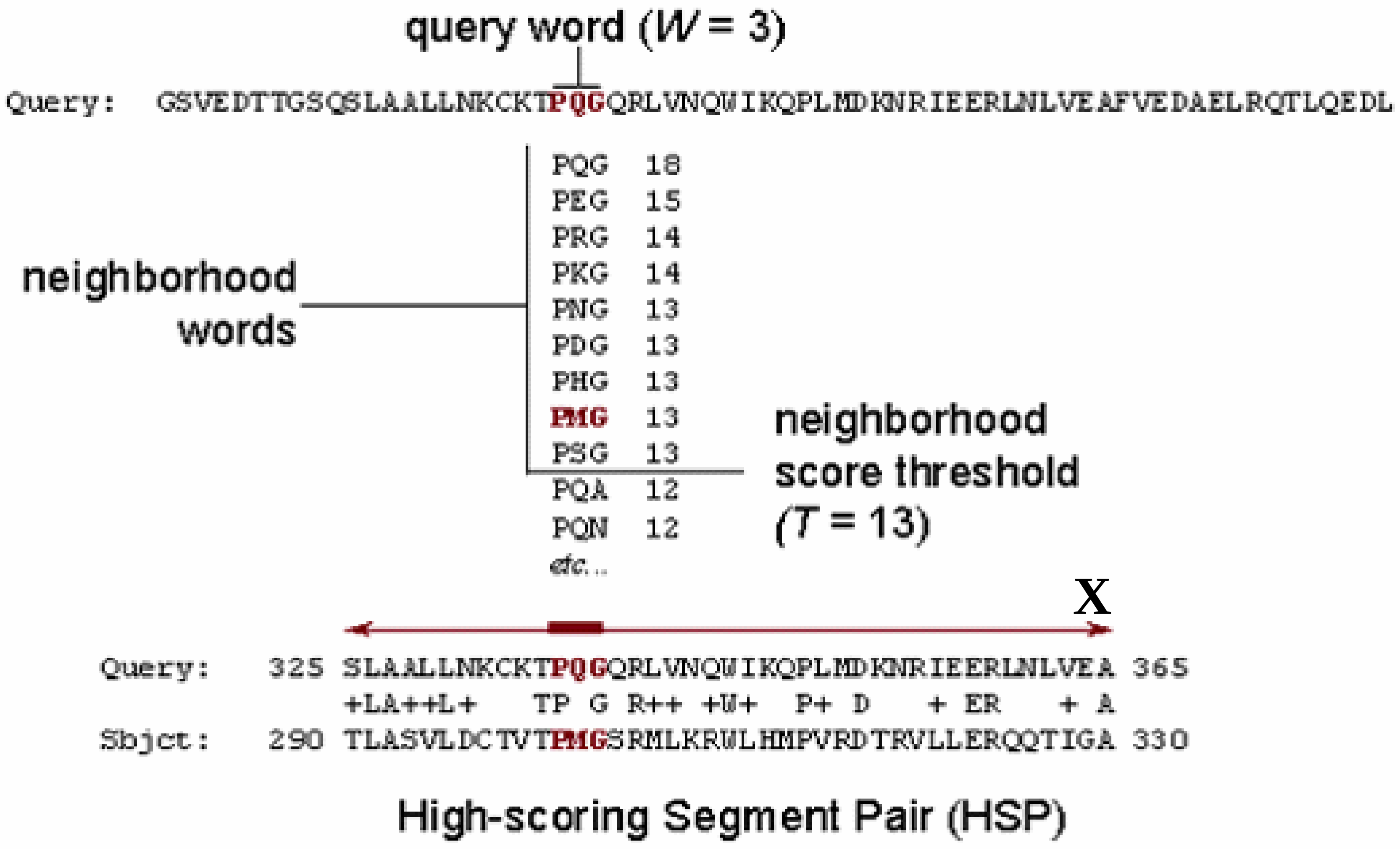
(Gapped blast: 24000 citations!)

Blast Algorithm Overview

- Receive query
  1. Split query into overlapping words of length W
  2. Find neighborhood words for each word until threshold T
  3. Look into the table where these neighbor words occur: seeds S
  4. Extend seeds S until score drops off under X
- Report significance and alignment of each match

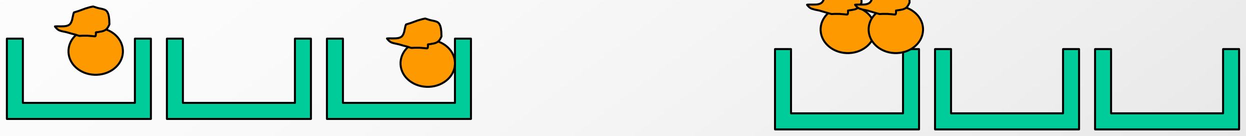

query word (W = 3)
Query: GSVEDTTGSQSLAALLNKCKTPQGQRLVNQWIKQPLMDKNRIEERLNLVEAFVEDAELRQTLQEDL
1. Split query into words
PQG 18
PEG 15
PRG 14
PKG 14
PNG 13
PDG 13
PHG 13
PMG 13
PSG 13
T
2. Expand word neighborhood
PMG
W-mer Database
3. Search database for neighborhood matches
Query: 325 SLAALLNKCKTPQGQRLVNQWIKQPLMDKNRIEERLNLVEA 365
+LA++L+ TP G R++ +W+ P+ D + ER + A
Sbjct: 290 TLASVLDCTVTPMGSRMLKRWLHMPVRDTRVLLERQQTIGA 330
High-scoring Segment Pair (HSP)
4. Extend each hit into alignment

The BLAST Search Algorithm

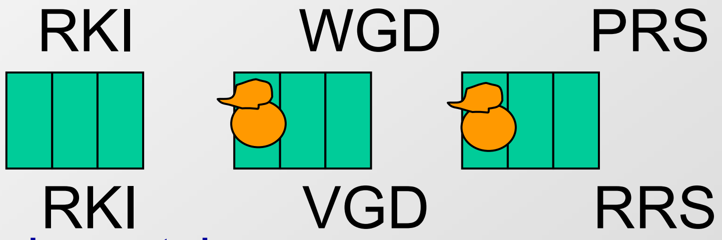

query word (*W* = 3)
neighborhood words
neighborhood score threshold (*T* = 13)
X
High-scoring Segment Pair (HSP)

```
Query:  GSVEDTTGSQSLAALLNKCKTPQGQRLVNQWIKQPLMDKNRIEERLNLVEAFVEDAELRQTLQEDL

PQG  18
PEG  15
PRG  14
PKG  14
PNG  13
PDG  13
PHG  13
PMG  13
PSG  13
PQA  12
PQN  12
etc...

Query:  325 SLAALLNKCKTPQGQRLVNQWIKQPLMDKNRIEERLNLVEA 365
            +LA++L+    TP G R++ +W+   P+ D    + ER    + A
Sbjct:  290 TLASVLDCTVTPMGSRMLKRWLHMPVRDTRVLLERQQTIGA 330
```

Why BLAST works(1): Pigeonhole and W-mers

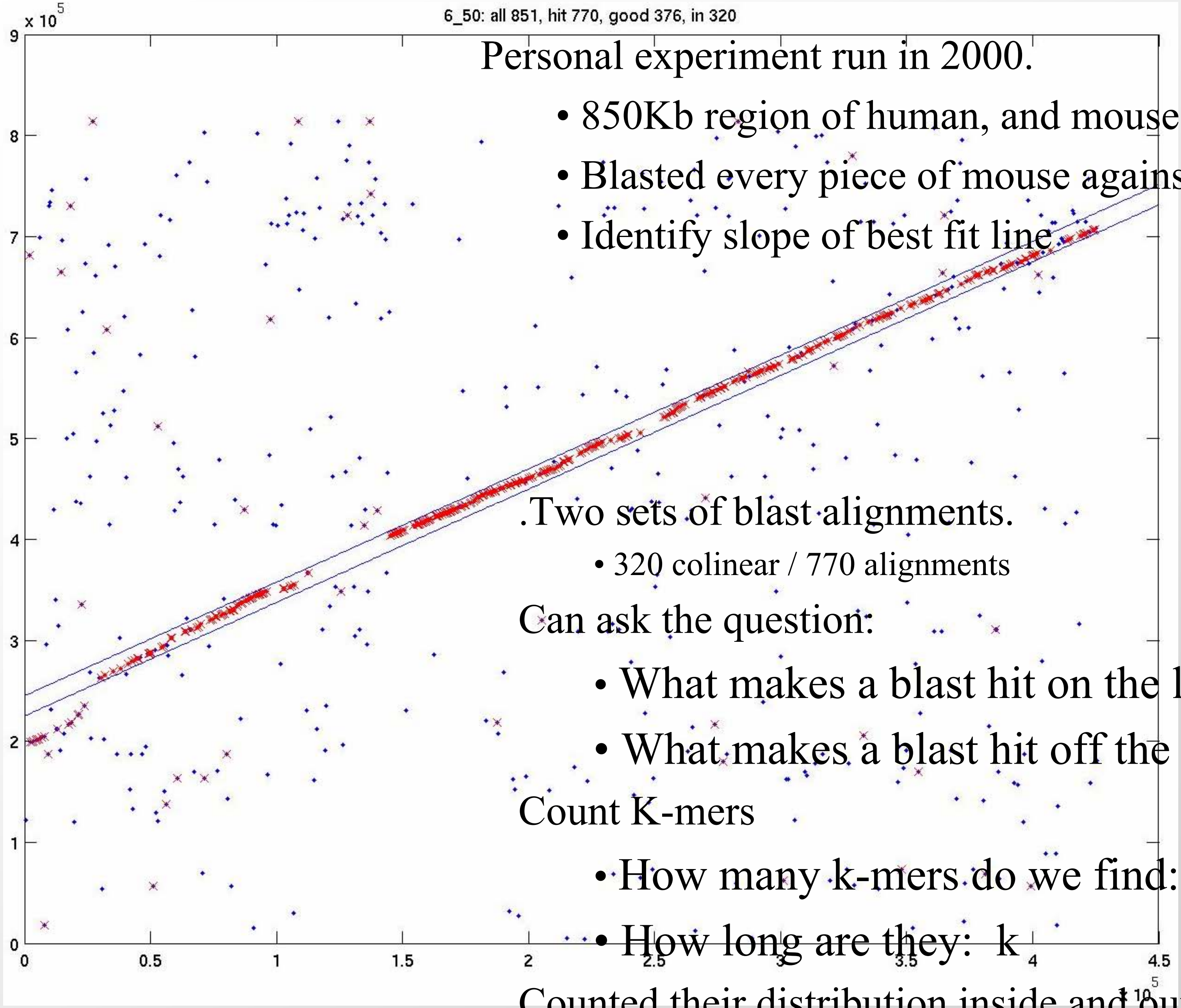


- Pigeonhole principle
  - If you have 2 pigeons and 3 holes, there must be at least one hole with no pigeon



- Pigeonholing mis-matches
  - Two sequences, each 9 amino-acids, with 7 identities
  - There is a stretch of 3 amino-acids perfectly conserved
- In general:
  - Sequence length: n
  - Identities: t
  - Can use W-mers for W= [n/(n-t+1)]

Why BLAST works(2): K-mer matches in practice



Personal experiment run in 2000.

- 850Kb region of human, and mouse 450Kb ortholog.
- Blasted every piece of mouse against human (6,50)
- Identify slope of best fit line

.Two sets of blast alignments.

- 320 colinear / 770 alignments

Can ask the question:

- What makes a blast hit on the line look good.
- What makes a blast hit off the diagonal look ba

Count K-mers

- How many k-mers do we find: n
- How long are they: k

Counted their distribution inside and outside the sequence.

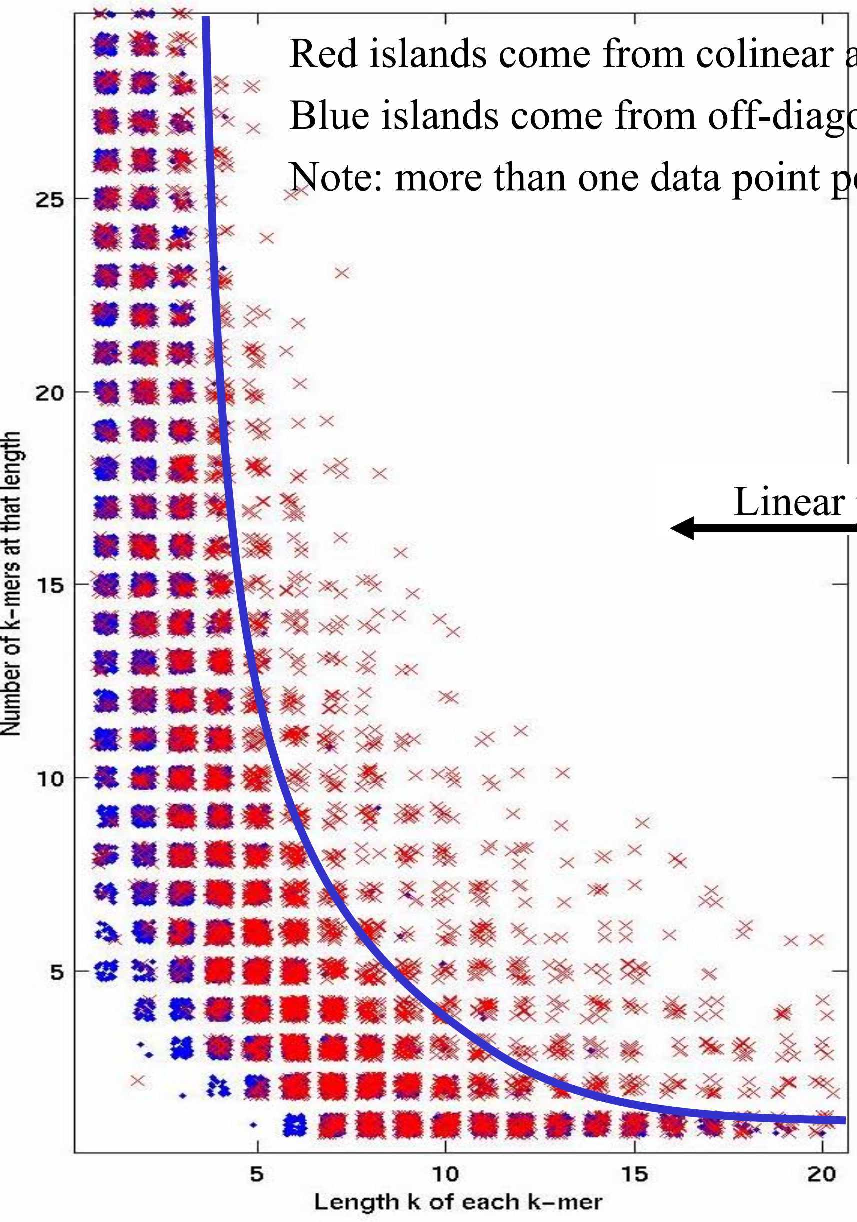
True alignments: Looking for K-mers

number of k-mers that happen for each length of k-mer.

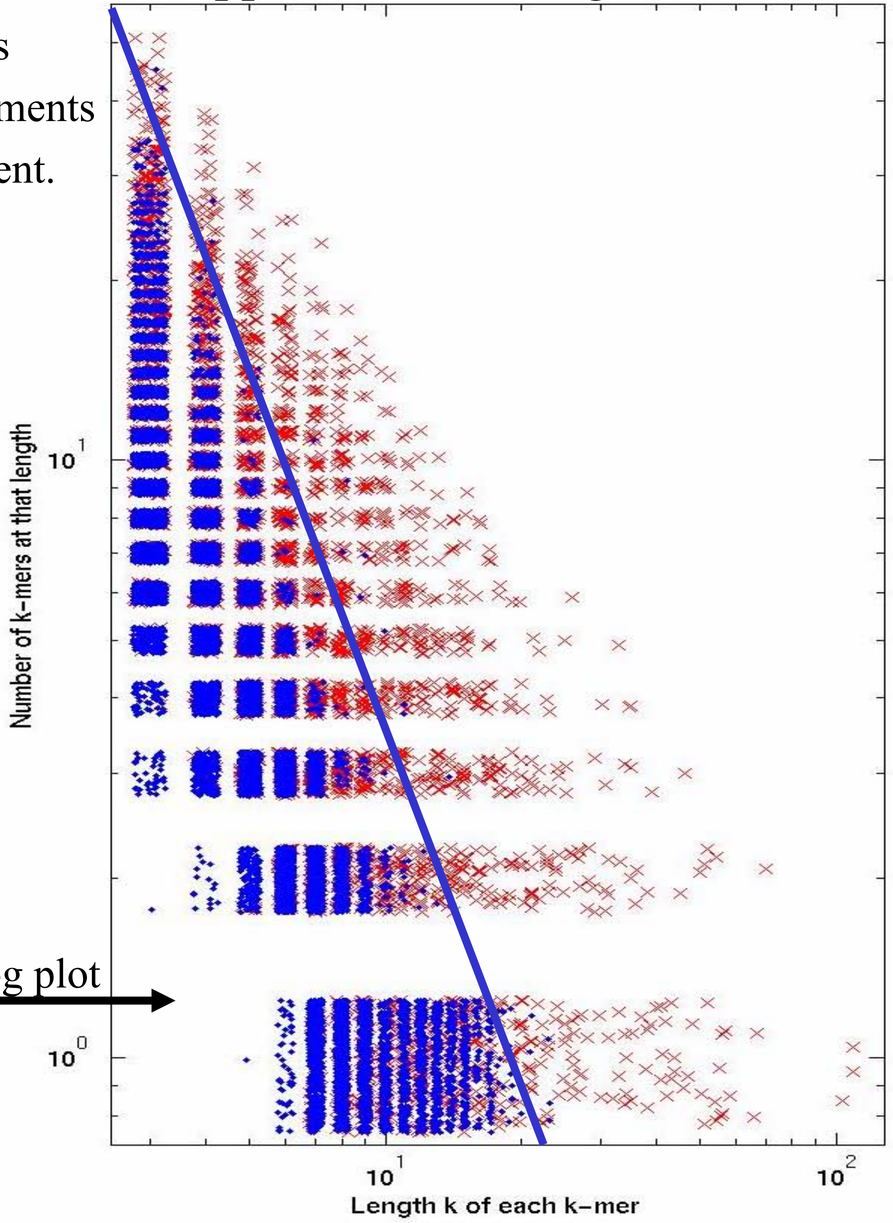
Red islands come from colinear alignments

Blue islands come from off-diagonal alignments

Note: more than one data point per alignment.



Linear plot



Log Log plot

Extensions to the basic algorithm

- Ideas beyond W-mer indexing ?
  - Faster
  - Better sensitivity (less false negatives)

1. Filtering: Low complexity regions cause spurious hits
   - Filter out low complexity in your query
   - Filter most over-represented items in your database
2. Two-hit BLAST
   - Two smaller W-mers are more likely than one longer one
   - Therefore it's a more sensitive searching method to look for two hits instead of one, with the same speed.
   - Improves sensitivity for any speed, speed for any sensitivity
3. Beyond W-mers, hashing with Combs

Extension(3): Combs and Random Projections

Key idea:

- No reason to use only consecutive symbols
- Instead, we could use combs, e.g.,

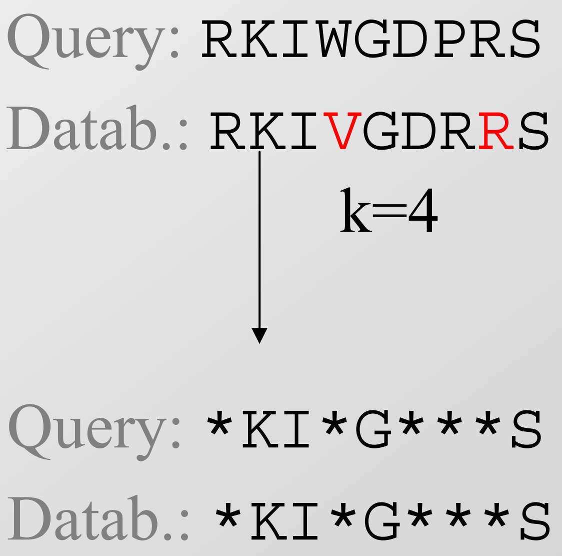
  `RGIKW → R*IK* , RG**W, …`
- Indexing same as for W-mers:
  - For each comb, store the list of positions in the database where it occurs
  - Perform lookups to answer the query
- How to choose the combs? At random
  - Randomized projection:

    Califano-Rigoutsos'93, Buhler'01, Indyk-Motwani'98
  - Choose the positions of * at random
  - Analyze false positives and false negatives

Extension(3): Combs and Random Projections

Performance Analysis:

- Assume we select k positions, which do not contain *, at random with replacement
- What is the probability of a false negative ?
  - At most: 1-idperc^k
  - In our case: $1-(7/9)^4 = 0.63...$
- What is we repeat the process l times, independently ?
  - Miss prob. = 0.63^l
  - For l=5, it is less than 10%



Today's Goal: Diving deeper into alignments

1. **Global alignment vs. Local alignment**
   - Needleman-Wunsch and Smith-Waterman
   - Varying gap penalties and algorithmic speedups
2. **Linear-time exact string matching**
   - Karp-Rabin algorithm and semi-numerical methods
   - Hash functions and randomized algorithms
3. **The BLAST algorithm and inexact matching**
   - Hashing with neighborhood search
   - Two-hit blast and hashing with combs
4. **Probabilistic foundations of sequence alignment**
   - Mismatch penalties, BLOSUM and PAM matrices
   - Statistical significance of an alignment score

Varying scores/penalties for matches/mismatches

Nucleotide sequences

	A	G	T	C
A	+1	-½	-1	-1
G	-½	+1	-1	-1
T	-1	-1	+1	-½
C	-1	-1	-½	+1

purine pyrimid.

Transitions:

A⇔G, C⇔T common (lower penalty)

Transversions:

All other operations

Protein space: amino-acid similarities

	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	
C	9																				C
S	-1	4																			S
T	-1	1	5																		T
P	-3	-1	-1	7																	P
A	0	1	0	-1	4																A
G	-3	0	-2	-2	0	6															G
N	-3	1	0	-2	-2	0	6														N
D	-3	0	-1	-1	-2	-1	1	6													D
E	-4	0	-1	-1	-1	-2	0	2	5												E
Q	-3	0	-1	-1	-1	-2	0	0	2	5											Q
H	-3	-1	-2	-2	-2	-2	1	-1	0	0	8										H
R	-3	-1	-1	-2	-1	-2	0	-2	0	1	0	5									R
K	-3	0	-1	-1	-1	-2	0	-1	1	1	-1	2	5								K
M	-1	-1	-1	-2	-1	-3	-2	-3	-2	0	-2	-1	-1	5							M
I	-1	-2	-1	-3	-1	-4	-3	-3	-3	-3	-3	-3	-3	1	4						I
L	-1	-2	-1	-3	-1	-4	-3	-4	-3	-2	-3	-2	-2	2	2	4					L
V	-1	-2	0	-2	0	-3	-3	-3	-2	-2	-3	-3	-2	1	3	1	4				V
F	-2	-2	-2	-4	-2	-3	-3	-3	-3	-3	-1	-3	-3	0	0	0	-1	6			F
Y	-2	-2	-2	-3	-2	-3	-2	-3	-2	-1	2	-2	-2	-1	-1	-1	-1	3	7		Y
W	-2	-3	-2	-4	-3	-2	-4	-4	-3	-2	-2	-3	-3	-1	-3	-2	-3	1	2	11	W

BLOSUM matrix of AA similarity scores

- Where do these scores come from?
- Are two aligned sequences actually related?

Probabilistic Model of Alignments

- we'll focus on protein alignments without gaps
- given an alignment, we can consider two possibilities

  R: the sequences are related by evolution

  U: the sequences are unrelated

- How can we distinguish these possibilities?
- How is this view related to amino-acid substitution matrices?

Model for Unrelated Sequences

- we'll assume that each position in the alignment is sampled randomly from some distribution of amino acids
- let q_a be the probability of amino acid a
- the probability of an n-character alignment of x and y is given by

$$\Pr(x, y \mid U) = \prod_{i=1}^{n} q_{x_i} \prod_{i=1}^{n} q_{y_i}$$

Model for Related Sequences

- we'll assume that each pair of aligned amino acids evolved from a common ancestor
- let p_{ab} be the probability that evolution gave rise to amino acid *a* in one sequence and *b* in another sequence
- the probability of an alignment of *x* and *y* is given by

$$\Pr(x, y \mid R) = \prod_{i=1}^{n} p_{x_i y_i}$$

Probabilistic Model of Alignments

- How can we decide which possibility (*U* or *R*) is more likely?
- one principled way is to consider the relative likelihood of the two possibilities (the odds ratio)

$$\frac{\Pr(x, y \mid R)}{\Pr(x, y \mid U)} = \frac{\prod_i p_{x_i y_i}}{\prod_i q_{x_i} \prod_i q_{y_i}} = \frac{\prod_i p_{x_i y_i}}{\prod_i q_{x_i} q_{y_i}}$$

- taking the log, we get

$$\log \frac{\Pr(x, y \mid R)}{\Pr(x, y \mid U)} = \sum_i \log\left(\frac{p_{x_i y_i}}{q_{x_i} q_{y_i}} \right)$$

Probabilistic Model of Alignments

- the score for an alignment is thus given by:

$$S = \sum_i s(x_i, y_i) = \log \frac{\Pr(x, y \mid R)}{\Pr(x, y \mid U)}$$

- the substitution matrix score for the pair *a*, *b* should thus be given by:

$$s(a,b) = \log\left(\frac{p_{ab}}{q_a q_b}\right)$$

Substitution Matrices

- two popular sets of matrices for protein sequences
  - PAM matrices [Dayhoff *et al.*, 1978]
  - BLOSUM matrices
    [Henikoff & Henikoff, 1992]
- both try to capture the the relative substitutability of amino acid pairs in the context of evolution

BLOSUM62

A	4																				
R	-1	5																			
N	-2	0	6																		
D	-2	-2	1	6																	
C	0	-3	-3	-3	9																
Q	-1	1	0	0	-3	5															
E	-1	0	0	2	-4	2	5														
G	0	-2	0	-1	-3	-2	-2	6													
H	-2	0	1	-1	-3	0	0	-2	8												
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4											
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4										
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5									
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5								
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6							
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7						
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4					
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5				
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11			
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7		
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4	
X	0	-1	-1	-1	-2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-2	0	0	-2	-1	-1	-1
	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	X

Positive for chemically similar substitution

Common amino acids have low weights

Rare amino acids have high weight

Substitution Matrices

- the substitution matrix score for the pair *a*, *b* is given by:

$$s(a,b) = \log\left(\frac{p_{ab}}{q_a q_b}\right)$$

- but how do we get values for p_{ab} (probability that *a* and *b* arose from a common ancestor)?
- it depends on how long ago sequences diverged
  - diverged recently: $p_{ab} \approx 0$ for $a \neq b$
  - diverged long ago: $p_{ab} \approx q_a q_b$

Substitution Matrices

- <u>key idea</u>: trusted alignments of related sequences provide information about biologically permissible mutations

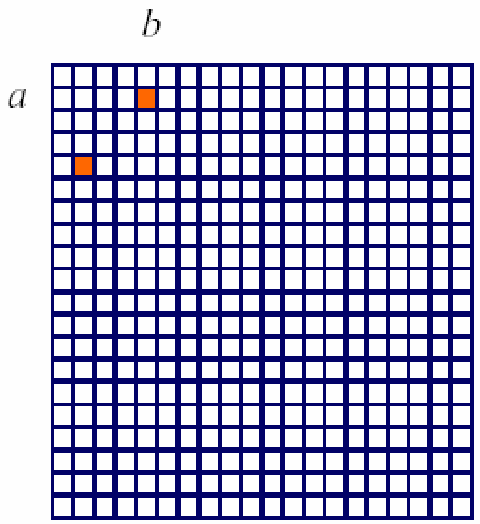
BLOSUM Matrices

- [Henikoff & Henikoff, *PNAS* 1992]
- probabilities estimated from “blocks” of sequence fragments that represent structurally conserved regions in proteins
- transition frequencies observed directly by identifying blocks that are at least
  - 45% identical (BLOSUM-45)
  - 50% identical (BLOSUM-50)
  - 62% identical (BLOSUM-62)
  - etc.

BLOSUM Matrices

- given: a set of sequences in a block
- fill in matrix *A* with number of observed substitutions (we won't worry about details of some normalization that happens here)

a paired with *b* →



$$p_{ab} = \frac{A_{ab}}{\sum_{c,d} A_{cd}} \qquad q_a = \frac{\sum_b A_{ab}}{\sum_{c,d} A_{cd}}$$

(you are not responsible for the remainder of this section)

Statistics of Alignment Scores

Q: How do we assess whether an alignment provides good evidence for homology?

A: determine how likely it is that such an alignment score would result from chance.

3 ways to calculate chance; look at alignment scores for

- real but non-homologous sequences
- real sequences shuffled to preserve compositional properties
- sequences generated randomly based upon a DNA/protein sequence model

Statistics of Alignment Scores

- earlier we considered how do decide if a <u>single</u> alignment was more likely due to relatedness or chance
- but what if we're considered many alignments?
  - e.g. what if we're doing a BLAST search against a large protein database?
- we'd like to know how many high-scoring alignments we're likely to get by chance

Distribution of Scores

- Karlin & Altschul, *PNAS*, 1990
- consider a random model in which
  - we're looking for ungapped local alignments
  - the lengths of the sequences in each pair are m and n
- the expected number of alignments, E, with score at least S is given by:

$$E(S) = Kmne^{-\lambda S}$$

Distribution of Scores

$$E(S) = Kmne^{-\lambda S}$$

- S is a given score threshold
- m and n are the lengths of the sequences under consideration
- K and λ are constants that can be calculated from
  - the substitution matrix
  - the frequencies of the individual amino acids

K = measure of the relative indpdce of points in context of MSP score

λ = the unique positive-valued solution to $S_{i,j}\ P_x(i)\ P_y(j)\ e^{\lambda S_{ij}}=1$

Statistics of Alignment Scores

- to generalize this to searching a database, have n represent the summed length of the sequences in the DB
- the NCBI BLAST server does just this
- with this analysis, can also calculate p-values (the probability of a random alignment scoring at least S)
- theory for gapped alignments not as well developed
- computational experiments suggest this analysis holds for gapped alignments (but K and λ must be estimated from data)

Summary: Diving deeper into sequence alignment

1. **Global alignment vs. Local alignment**
   - Needleman-Wunsch and Smith-Waterman
   - Varying gap penalties and algorithmic speedups
2. **Linear-time exact string matching**
   - Karp-Rabin algorithm and semi-numerical methods
   - Hash functions and randomized algorithms
3. **The BLAST algorithm and inexact matching**
   - Hashing with neighborhood search
   - Two-hit blast and hashing with combs
4. **Probabilistic foundations of sequence alignment**
   - Mismatch penalties, BLOSUM and PAM matrices
   - Statistical significance of an alignment score

Tomorrow’s recitation: Deeper into Alignments

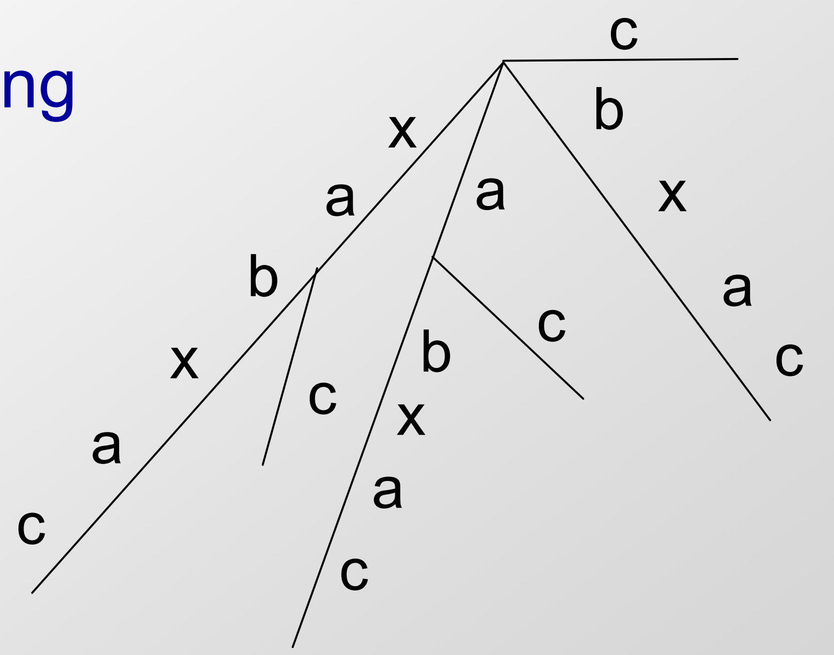
- Affine gap penalties
  - Augmenting the state-space
  - Linear, affine, piecewise linear, general gap penalty
- Statistical significance of alignment
  - Where does $\mathbf{s(x_i, y_j)}$ come from?
  - Are two aligned sequences actually related

3c. Massive pre-processing

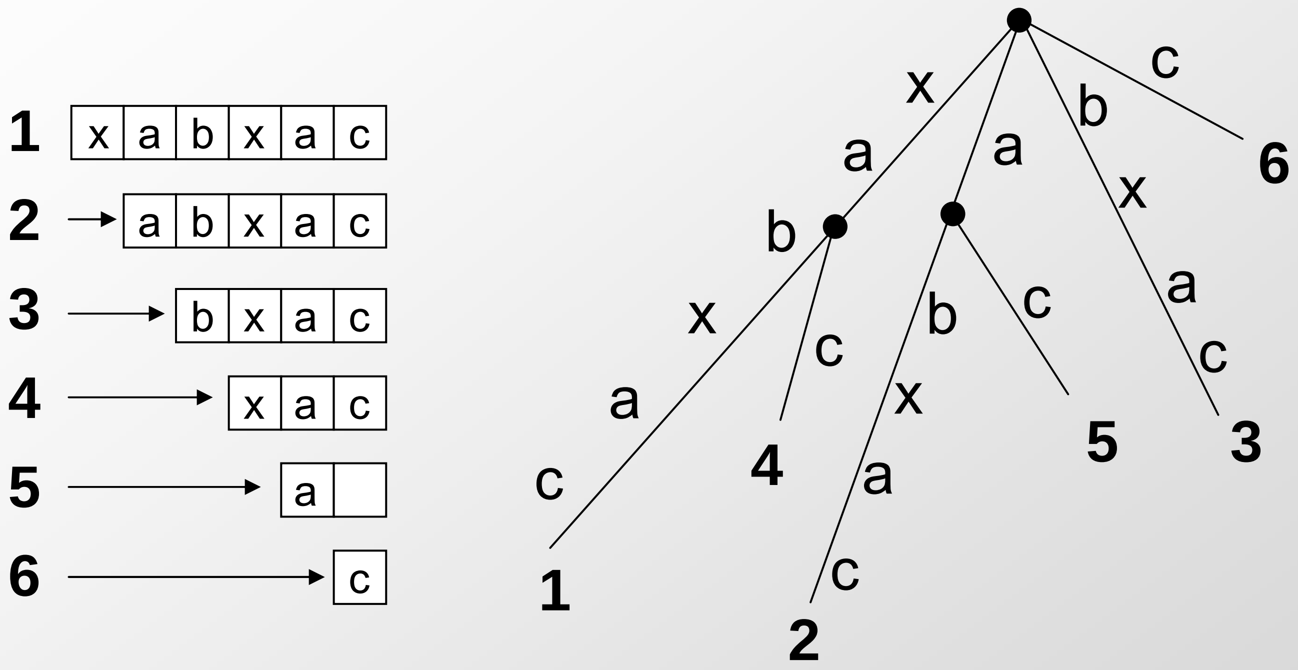
Suffix Trees

Suffix trees

- Great tool for text processing
  - E.g., searching for exact occurrence of a pattern
- Suffix tree for: xabxac

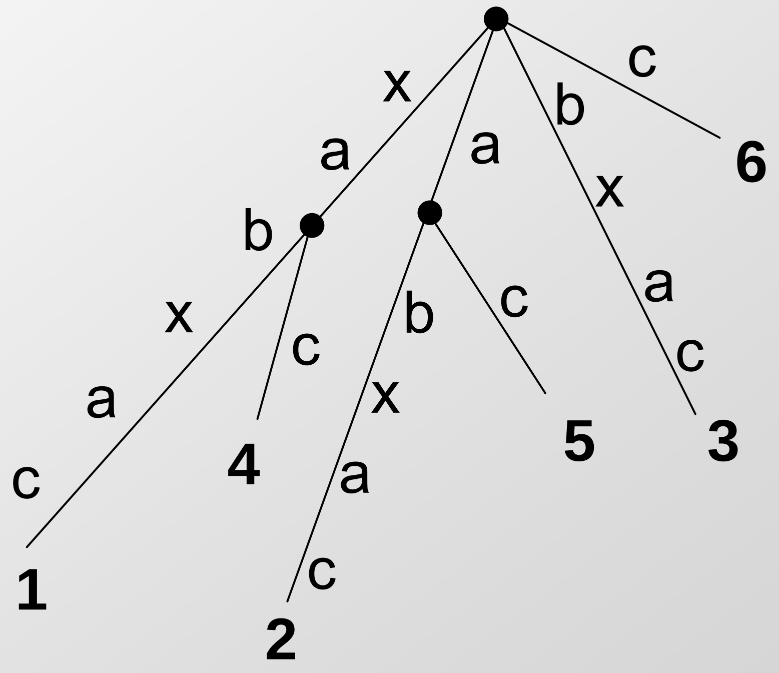


Suffix tree definition



- Definition: Suffix tree ST for text T[1..n]
  - Rooted, directed tree T, n leaves, numbered 1..n
  - Text labels on the edges
  - Path to leaf i spells out the suffix S[i..] , by concatenating edge labels
  - Common prefixes share common paths, diverge to form internal nodes

Properties of suffix trees



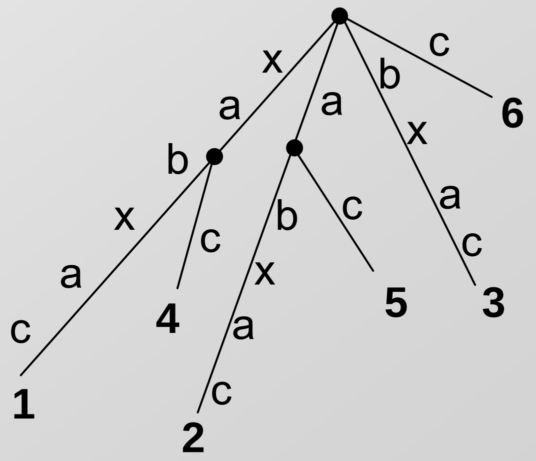
- How much space do we need to represent a suffix tree of T[1..n] ?
- Only O(n)
  - At most O(n) edges
  - Each edge label can be represented as T[i…j]

Exact string matching with suffix trees

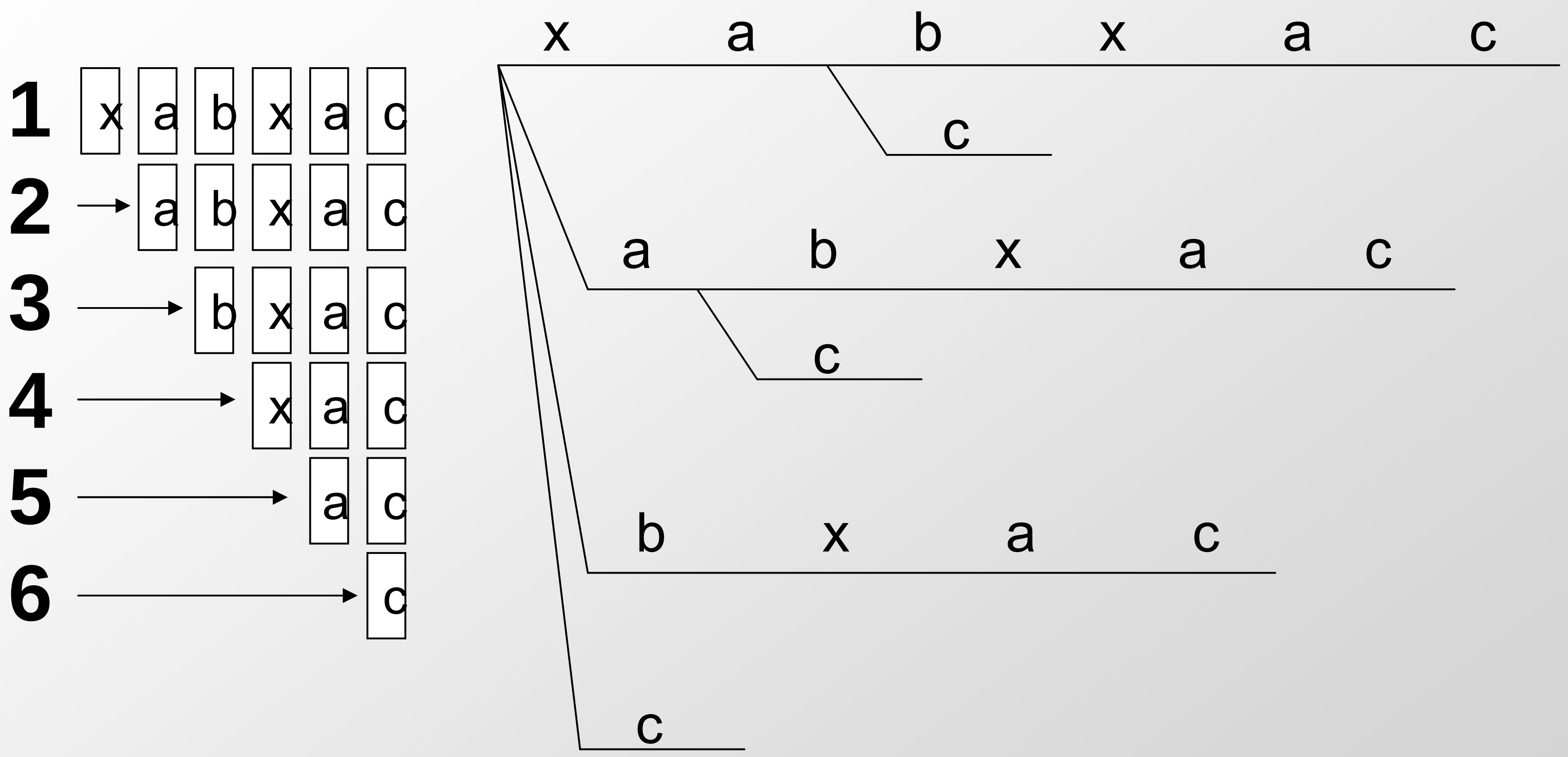
- Given the suffix tree for text T
- Search for pattern P[1…m]
  - For every character in P, traverse the appropriate path of the tree, reading one character each time
  - If P is not found in a path, P does not occur in T
  - If P is found in its entirety, then all occurrences of P in T are exactly the children of that node
    - Every child corresponds to exactly one occurrence
    - Simply list each of the leaf indices
- Time: O(m)

T: xabxac

P: abx



Suffix Tree Construction



- Running time: $O(n^2)$
- Can be improved to $O(n)$