



Searching (Sub-)Strings

Ulf Leser

This Lecture

- Exact substring search
 - Naïve
 - Boyer-Moore
- Searching with profiles
 - Sequence profiles
 - Ungapped approximate search
 - Statistical evaluation of search results

Searching / Comparing Strings

- Exact matching
 - Given strings p and t : Find all occurrences of p in t
 - Given a set P and t : Find all occurrences of any $p \in P$ in t
- Approximate matching
 - Given p and t : Find all approximate occurrences of p in t
 - Given p and t : Find p' , t' such that p' is similar to t' and p' is a substring of p and t' is a substring of t
 - Given p and a set of strings T
 - Find all $t \in T$ that are similar to p
 - Find all $t \in T$ containing a t' similar to a p' contained in p
- Many more variants ...

Applications

- Given strings p and t : Find all occurrences of p in t
 - Restriction enzyme cut positions; fixed patterns in gene structure; **seeds for approximate searching**
- Given a set P and t : Find all occurrences of any $p \in P$ in t
 - Same with multiple patterns / enzymes
- Given p and t : Find all approximate occurrences of p in t
 - Less conserved patterns; read mapping; **TF binding sites**
- Given p and t : Find p' , t' such that p' similar to t' and p' is a substring of s and t' is a substring of t
 - **Local alignment; homologous genes**; cross-species searches

Strings

- A **string (or sequence) p** is an ordered list of characters from an alphabet Σ
 - $|p|$ is the length of p
 - $p[i]$ is the character at position i in p (starting from 1)
 - $p[i..j]$ is the substring from position i to position j in p
 - $p[i..j]$ is an empty string if $i > j$
 - $p[1..i]$ is a **prefix** of p ending at position i
 - $p[i..]$ is a **suffix** of p starting at position i
- **Alphabet**
 - Usually: $\Sigma = \{A, C, G, T\}$
 - Often, we need blanks: $\Sigma' = \{A, C, G, T, _ \}$
- **Lower/upper case:** P may denote a set of strings, or a sequence of characters (a string)

Exact Matching

- Given P , T with $|P| \ll |T|$
- Find all occurrences of P in T
- Example of application: **Restriction enzymes**
 - Cut at precisely defined sequence motifs of length 4-10
 - Are used to generate fragments (for later sequencing)
 - Example: Eco RV - **GATATC**

```
tcagcttactaattaaattctttctagtaagtgctaagatcaagaaaataaattaaataatggaacatggcacatcttctaaactcttcacagattgctaagat
tattaattaaagaataaatggtataatcttttatggtaacggaatttctctaaaatattaattcaagcaccatggaatgcaaataagaaggactctgttaattgggtactat
tcaactcaatgcaagtggaaactaagttgggtattaataactctttttacatatatatgtagttatcttttaggaagcgaaggacaatttcatctgctaataaagggattacga
aaaactcttttaataacaaagttaaataatcattttgggaattgaaatgtcaaagataattacttcacgataagtagttgaagatagtttaaattttctttttgtattac
ttcaatgaaggtaacgcaacaagattagagtatatatggccaataagggttgctgttaggaaaattattctaaggagatacgcgagaggggcttctcaaatttattcagaga
tggatgttttagatgggtgggttaagaaaagcagttataaatccagcaaaactagaccttaggtttattaaagcagaggcaataagtttaattggaattgtaaaagatatc
aattcttcttcatttggtaggaaaactagtttaacttcttaccocatgcagggccataggggtcgaatacgatctgtcactaagcaaaggaaaatgtgagtgtagacttt
aaaccatttttattaatgacttttagagaatcatgcatttgatgttactttcttaacaatgtgaacatatttatgcgattaagatgagttatgaaaaaggcgaatatatta
ttcagttacatagagattatagctgggtctattcttagttataggacttttgacaagatagcttagaaaataagattatagagcttaataaaaagagaacttcttggaaatta
gctgcctttgggtgcagctgtaatggctattgggtatggctccagcttactgggttaggttttaatagaaaaattccccatgattgctaattatatactatctattgagaaca
acgtgcgaagatgagtggaattgggtcattattaactgctgggtgctatagtagttatccttagaaaagatatataaatctgataaaagcaaaatcctggggaaaatattg
ctaactgggtgctggtaggggtttggggattggatttttctctacaagaaatttgggtgttactgatatccttataaataatagagaaaaaattaataaagatgatat
```

How to do it?

- The straight-forward way (**naïve algorithm**)
 - We use two counters: t , p
 - One (outer, t) runs through T
 - One (inner, p) runs through P
 - Compare characters at position $T[t+p-1]$ and $P[p]$

```
for t = 1 to |T|
    match := true;
    p := 1;
    while ((match) and (p <= |P|))
        if (T[t + p - 1] <> P[p]) then
            match := false;
        else
            p := p + 1;
    end while;
    if (match) then
        -> OUTPUT t
end for;
```

Examples

Typical case

T ctgagatcgcgta
P gagatc
gagatc
gagatc
gagatc
gagatc
gatatc
gatatc
gatatc

Worst case

T aaaaaaaaaaaaaa
P aaaaat
aaaaat
aaaaat
aaaaat
...

- How many comparisons do we need in the **worst case**?
 - t always runs through T
 - p runs through the entire P for every position in t (worst case)
 - Thus: $O(|P|*|T|)$
 - A lot: $|T|=250\text{M}$ (chromosome), $|P|=250$ (exon) $\Rightarrow \sim 62\text{E}9$ ops

Other Algorithms

- Exact substring search has been researched for decades
 - Boyer-Moore, Z-Box, Knuth-Morris-Pratt, Karp-Rabin, Shift-AND, ...
 - All have WC complexity $O(|P| + |T|)$
 - **Real performance** depends a lot on size of alphabet and composition of strings (most have strengths in certain settings)
- One simple and popular algorithm: **Boyer-Moore**
 - We present a simplified form
 - BM is among the **fastest algorithms in practice**
- Much better performance if **T can be preprocessed**
 - Best algorithms reach $O(|P|)$

This Lecture

- Exact substring search
 - Naïve
 - Boyer-Moore
- Searching with profiles
 - Sequence profiles
 - Ungapped approximate search
 - Statistical evaluation of search results

Boyer-Moore Algorithm

- R.S. Boyer /J.S. Moore. „A Fast String Searching Algorithm“, Communications of the ACM, 1977
- Main idea
 - Again, we use two counters (inner loop, outer loop)
 - Inner loop runs from **right-to-left**
 - If we have a complete match – shift by one (outer loop)
 - If we find a mismatch, we know
 - The character in T we have not just seen
 - This is captured by the **bad character rule**
 - The suffix in P we have just seen
 - This is captured by the **good suffix rule**
- Use this knowledge to make **longer shifts in T**

Bad Character Rule

- Setting 1

- We are at position t in T and compare right-to-left
- Let i be the position of the first mismatch in P
 - We saw $n-i$ matches before
- Let x be the character at the corresponding pos ($t-n+i$) in T
- Candidates for matching x in P
 - Case 1: x does not appear in P at all – we can move t such that $t-n+i$ is not covered by P anymore

T xabx**f**abzzabwzzbzzb
P abwx**y**abzz
 ←

T xabx**f**abzzab**w**zzbzzb
P abwx**y**a**b**zz
 ←

What next?

Bad Character Rule 2

- Setting 2

- We are at position t in T and compare right-to-left
- Let i be the position of the first mismatch in P
- Let x be the character at the corresponding pos $(t-n+i)$ in T
- Candidates for matching x in P
 - Case 1: x does not appear in P at all
 - Case 2: Let j be the right-most appearance of x in P with $j < i$ – we can move t such that j and i align

T xabxkabzzabwzzbzzb
P abzwyabzz

 ↑ ↑
 j i

T xabxkabzzabwzzbzzb
P abzwyabzz

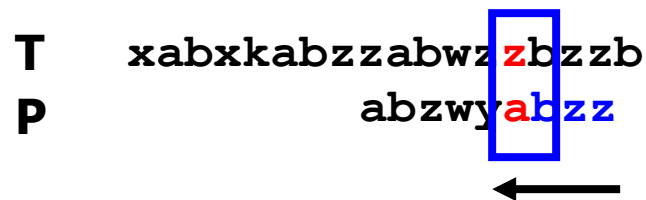
 ←

What next?

Bad Character Rule 3

- Setting 3
 - We are at position t in T and compare right-to-left
 - Let i be the position of the first mismatch in P
 - Let x be the character at the corresponding pos $(t-n+i)$ in T
 - Candidates for matching x in P
 - Case 1: x does not appear in P at all
 - Case 2: Let j be the right-most appearance of x in P with $j < i$
 - Case 3: *As case 2, but $j > i$* – we need some more knowledge

T xabxkabzzabwz**zb**zzb
P abzwy**ab**zz



Preprocessing 1

- In case 3, there is at least one “x” right from position i
 - For small alphabets (DNA), this will almost always be the case
 - Thus, case 3 is the usual one
- These “x” are irrelevant – we need the **right-most x left of i**
- This can (and should!) be **pre-computed**
 - Build a two-dimensional array $A[|\Sigma|, |P|]$
 - Run through P from left-to-right (pointer i)
 - If character c appears at position i , set all $A[c, j] := i$ for all $j > i$
 - Possible in $O(|A|)$; negligible because P is small
 - **Constant lookup time** during search

(Extended) Bad Character Rule

- Simple, effective for larger alphabets
- For random DNA, **average shift-length is ~ 2**
 - Expected distances to the next match using EBCR
 - Per position in T, the expected length of the match also is ~ 2
 - Thus, we expect $\sim 2 * |T| / 2 = |T|$ comparisons
- Worst-Case complexity?

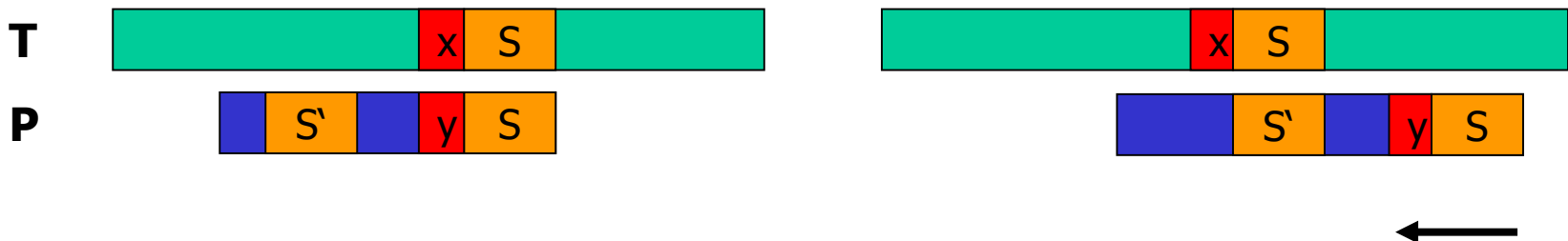
(Extended) Bad Character Rule

- Simple, effective for larger alphabets
- For random DNA, average shift-length is ~ 2
 - Expected distances to the next match using EBCR
 - Per position in T, the expected length of the match also is ~ 2
 - Thus, we expect $\sim 2 * |T| / 2 = |T|$ comparisons
- Worst-Case complexity?



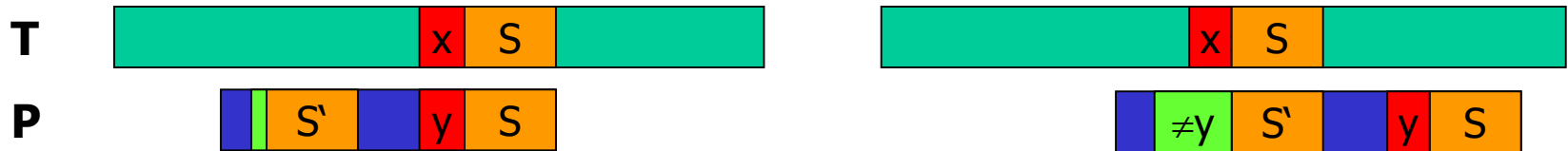
Good-Suffix Rule

- Recall: If we reach a mismatch, we know ...
 - The character in T we have not just seen
 - The **suffix in P** we have just seen
- **Good suffix rule**
 - We have just seen some matches in P; let this suffix be S
 - Where else does S appear in P?
 - If we know the **right-most appearance S'** of S in P, we can immediately align S' with the current match in T
 - If S does not appear at least twice in P, we shift t by $|P| - |S| + 1$



Good-Suffix Rule – One Improvement

- Actually, we can do a little better
- Not all S' are of interest to us



- We only need S' whose next character to the left is not y
- Why don't we directly require that this character is x ?

Complete Algorithm

```
t := 0;
while (t<=|T|-|P|) do                                \\ outer loop
  p := |P|;
  match := true;
  while (match and p>=1) do                          \\ inner loop
    if (T[t+p]=P[p]) then p := p-1                  \\ matching chars
    else match := false;                             \\ mismatch
  end while;
  if match then
    print t;                                         \\ complete match
    t := t+1;
  else
    compute shift s1 using BCR(t,p);
    compute shift s2 using GSR(t,p);
    t := t + max(s1, s2);                          \\ shift maximal
  end if;
end while;
```

GSR Preprocessing

- We need to find all occurrences of all suffixes of P in P with restrictions on the character left of the suffix
- Could be computed using naïve algorithm for each suffix
- Or, **more complicated, in linear time** (not this lecture)
- Runtime negligible since we assume P being short

Concluding Remarks

- Worst-case complexity of Boyer-Moore is $O(|P|*|T|)$
 - WC complexity can be reduced to linear (not this lecture)
- Empirical runtime **is sub-linear**
 - The larger the alphabet (with roughly equal character frequencies), the faster
- Faster variants
 - Often, using the GSR does not pay off
 - BM-Horspool: Instead of looking at the mismatch character x , always look at the **symbol in T aligned to the last position of P**
 - Generates longer shifts on average (i is maximal)
- In practice, naïve algorithm is also quite competitive for random strings and **non-trivial alphabets** (not for DNA)
 - Empirical results much better than worst-case estimations

Example

b b c g g b c b a a g g b b a a c a b a a b g b a a c g c a b a a b c a b
c a b a a b g b a a

b b c g g b c b a a g g b b a a c a b a a b g b a a c g c a b a a b c a b

EBCR wins

c a b a a b g b a a

b b c g g b c b a a g g b b a a c a b a a b g b a a c g c a b a a b c a b

GSR wins

c a b a a b g b a a

b b c g g b c b a a g g b b a a c a b a a b g b a a c g c a b a a b c a b

GSR wins

c a b a a b g b a a

b b c g g b c b a a g g b b a a c a b a a b g b a a c g c a b a a b c a b

Match Good suffix
Mismatch Ext. Bad character

c a b a a b g b a a

???

This Lecture

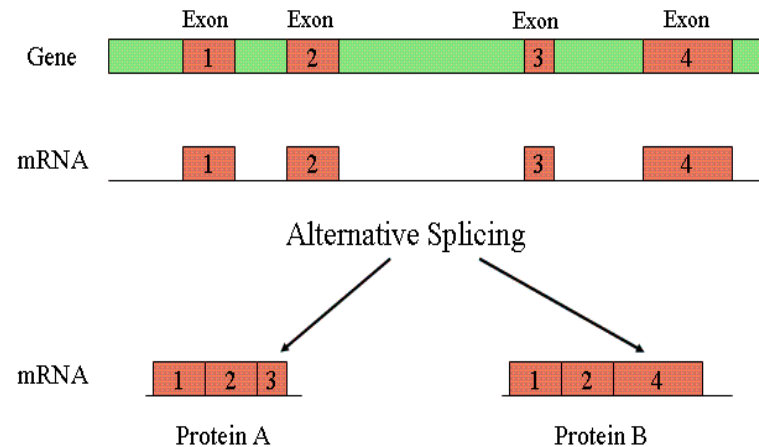
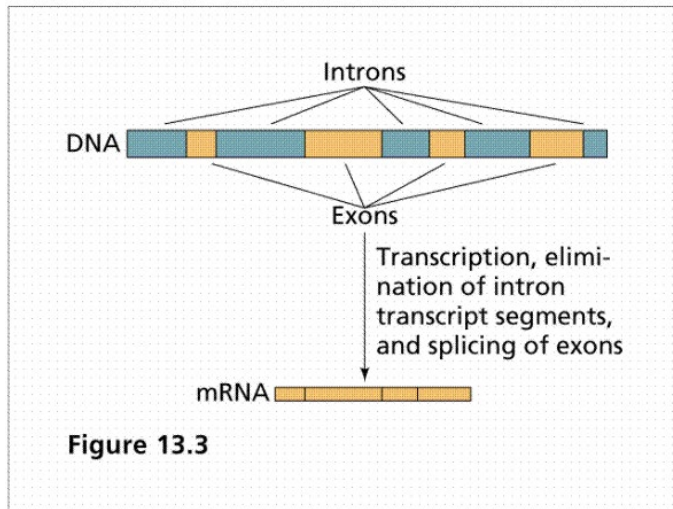
- Exact substring search
- Searching with profiles
 - Splicing
 - Position Specific Weight Matrices
 - Likelihood scores

Approximate Search (First Step)

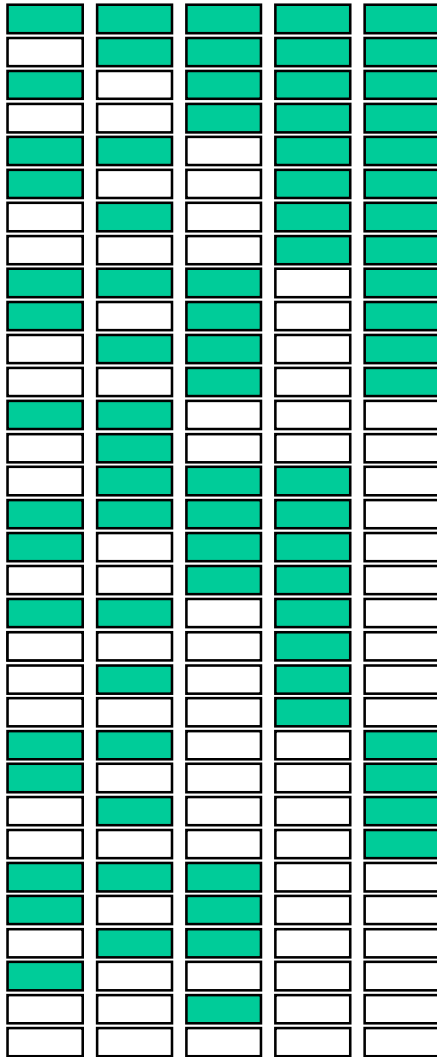
- Requiring an exact match is too strict in most bioinformatics applications
 - Sequencing errors, mutations, individual differences, ...
- More often, one is interested in **matches similar to P**
- Many definitions of “similar” are possible
- Now: **Position Specific Weight Matrices (PSWM)**
 - Also called **profiles**
 - Powerful tool with many bioinformatics applications
 - We develop the idea using an example taken from Spang et al. “Genome Statistics”, Lecture 2004/2005, FU Berlin

Splicing

- Not all DNA of a gene is translated into amino acid
- **Splicing**: Removal of introns
- **Alternative splicing**: Removal of some exons



Diversity



- From a gene with n exons, alternative splicing can create $2^n - 1$ proteins
- Example: Troponin T (muscle protein)
 - 18 exons
 - 64 different known isoforms
 - 10 exons present in all isoforms
- Source: Eurasnet, „Alternative Splicing“

Recognizing Splice Sites

- A special enzyme (spliceosome) very precisely recognizes **exon-intron boundaries** in mRNA as a sequence **motif**
- Spliceosome recognizes certain sequence **motifs**
- How are these motifs characterized? Can we find them?
 - Very often, introns start with GT and end with AG
 - But that is **not specific enough** - why?
 - In random sequences, we expect a GT (AG) at every 16th position
 - Thus, the average distance between a GT and an AG is 16, and we find such pairs very often
 - But: Introns typically are larger than 100 bases

Context of a Splice Site

CTCCGAAGTAGGATT

TCAGAAGGTGAGGGC

TTGGAAGGTTTCGCAG

TACTCAGGTACTCAC

CGCCCAGGTGACCGG

AGAAAGAGTAAGCTC

CAATGCTGTATGTGT

GGTCTCGGTAAGTGC

CCTGCTGGTAAGGCC

TGTTGCGGTAGGTCC

CTCCGAAGTAGGATT

TCAGAAGGTGAGGGC

TTGGAAGGTTTCGCAG

TACTCAGGTACTCAC

CGCCCAGGTGACCGG

AGAAAGAGTAAGCTC

CAATGCTGTATGTGT

GGTCTCGGTAAGTGC

CCTGCTGGTAAGGCC

TGTTGCGGTAGGTCC

- Observing real splice sites, we find **no crisp context**
- But: columns are not composed at random
- How can we capture and quantify this knowledge?

Vizualization: Sequence Logos

- Very popular
- Based on **information content** of each base at each position
 - Which, in turn, is based on the entropy of the columns

```
CTCCGAAGTAGGATT
TCAGAAGGTGAGGGC
TTGGAAGGTTTCGCAG
TACTCAGGTACTCAC
CGCCCAGGTGACCGG
AGAAAGAGTAAGCTC
CAATGCTGTATGTGT
GGTCTCGGTAAGTGC
CCTGCTGTAAGGCC
TGTTGCGTAGGTCC
```



Position-Specific Weight Matrices

```
# DONOR FREQUENCY MATRIX from http://genomic.sanger.ac.uk/spldb/SpliceDB.html
  1      2      3      4      5      6      7      8      9
A 34.08  60.36   9.14   0.00   0.00  52.57  71.26   7.08  15.98
C 36.24  12.90   3.27   0.00   0.00   2.82   7.56   5.50  16.46
G 18.31  12.48  80.34 100.00   0.00  41.94  11.76  81.35  20.90
T 11.38  14.25   7.24   0.00 100.00   2.55   9.29   5.88  46.16
```

- Count in every column the frequencies of all bases
- Store the **relative frequencies** in an array of size $|P| * |\Sigma|$
 - With $|P|$ being the size of the context around the splice sites
- At "GT", all values except one are 0% and one is 100%
 - Actually, GT is not perfectly conserved in real sequences
- In **random sequences**, all values should be 25%

Scoring with a PSWM

- Eventually, we want to find **potential splice sites** in a genome G
 - For instance for gene prediction
- We need a way to decide, given a sequence S and a PSWM A (both of the same length): **Does S match A ?**
 - We devise a function **assigning a score** to S given A
 - With this function, we score all subsequences of length $|A|$ in G
 - Subsequences above a **given threshold** are considered candidates
- We give this question a **probabilistic interpretation**
 - Assume, for each column, a dice with four faces; each face is thrown with probability equal to the relative frequencies as given in the PSWM A for this column
 - What is the probability that this **dice generates S** ?

Examples

- In **random sequences**, all values in A are 25%, and all possible S would get the same probability: $\frac{1}{4}^{|S|}$

- But

	1	2	3	4	5	6	7	8	9
A	34.08	60.36	9.14	0.00	0.00	52.57	71.26	7.08	15.98
C	36.24	12.90	3.27	0.00	0.00	2.82	7.56	5.50	16.46
G	18.31	12.48	80.34	100.00	0.00	41.94	11.76	81.35	20.90
T	11.38	14.25	7.24	0.00	100.00	2.55	9.29	5.88	46.16

- $P(\text{AAGGTAAGT}) \sim 0.3 * 0.6 * 0.8 * 1 * 1 * 0.5 * 0.7 * 0.8 * 0.5 \sim 0.023$
- $P(\text{CCCGTCCCC}) \sim 0.4 * 0.1 * 0.03 * 1 * 1 * 0.02 * 0.08 * 0.05 * 0.2 \sim 3\text{E-}8$
- $P(\text{AGTCTGAAG}) \sim 0.3 * 0.1 * 0.1 * 0 * 1 * 0.4 * 0.7 * 0.07 * 0.2 = 0$

- 1st sequence matches A **much better** than the second
- 3rd sequence hints towards **overfitting**
 - Nothing is impossible in biology

This Lecture

- Exact substring search
- Searching with profiles
 - Splicing
 - Position Specific Weight Matrices
 - Likelihood scores

I am not Convinced (yet)

- Is S **actually** a match for A?
- We need to quantify the “goodness” of a score
 - By comparing it to other / best / worst scores
- Observations
 - The first match on the previous slide is about **as good as it can get**: Best possible sequence has a score of 0.025 (compared to 0.023)
 - If match S is not a splice site, it is an “ordinary” sequence. How likely is it that S is generated under this **zero model (Z)**?
 - “Zero model” often means: Equal probability for all bases
 - Could include species bias, coding region bias, CpG island bias, ...
 - $p(S|“zero”) = 1/4^9 \sim 3.8E-6$
 - Thus, is it **much more likely** (app. 6000 times more likely) that S was generated under the A model than that it was generated under the Z model

Likelihood (Odds) Ratios

- Given two models A, Z. The **likelihood ratio score** of a sequence S is the ratio of $p(S|A) / p(S|Z)$

- $\text{score}(\text{AAGGTACGT}) \sim 6000$
- $\text{score}(\text{CCCGTCCCC}) \sim 1/140$
- $\text{score}(\text{CTGGTCCGA}) \sim 3$
- $\text{score}(\text{TCCGTCCCC}) < 1$

	1	2	3	4	5	6	7	8	9
A	34.08	60.36	9.14	0.00	0.00	52.57	71.26	7.08	15.98
C	36.24	12.90	3.27	0.00	0.00	2.82	7.56	5.50	16.46
G	18.31	12.48	80.34	100.00	0.00	41.94	11.76	81.35	20.90
T	11.38	14.25	7.24	0.00	100.00	2.55	9.29	5.88	46.16

1. $P(\text{AAGGTACGT}) \approx 0.34 * 0.6 * 0.8 * 1 * 1 * 0.53 * 0.71 * 0.81 * 0.46 = 0.023$
2. $P(\text{CCCGTCCCC}) \approx 0.36 * 0.13 * 0.03 * 1 * 1 * 0.03 * 0.08 * 0.05 * 0.16 = 2.7e-08$
3. $P(\text{CTGGTCCGA}) \approx 0.36 * 0.14 * 0.8 * 1 * 1 * 0.03 * 0.08 * 0.81 * 0.16 = 1.25e-05$
4. $P(\text{TACCTCCGT}) = 0$

- Also called **odds score**

- This is just one (popular) method for computing a “goodness” score

Matching with a PSWM

- Given genome G , models A and Z , and a **threshold t** : Find all S in G with $\text{likelihood}(S) > t$
- Method: For all S with $|S| = |A|$, compute $\text{likelihood}(S)$
 - This requires $\sim |G|^{|A|}$ divisions and multiplications
 - Divisions can be saved easily (how?)

Numerical trick

- Values get quite small (close to 0) for longer A
- This yields problems with **numeric stability** in programs
- Better: Compute **log-likelihood score** $s' = \log_2(\text{score}(\dots))$
 - Also faster: Replaces multiplication with addition
 - Pre-compute divisions

$$\begin{aligned} s'(S) &= \log\left(\frac{p(S | A)}{p(S | Z)}\right) = \log\left(\frac{p(S_1 | A_1) * \dots * p(S_n | A_n)}{p(S_1 | Z_1) * \dots * p(S_n | Z_n)}\right) \\ &= \log\left(\frac{p(S_1 | A_1)}{p(S_1 | Z_1)}\right) + \dots + \log\left(\frac{p(S_n | A_n)}{p(S_n | Z_n)}\right) \end{aligned}$$

Beware

- Assume a highly conserved motif A of length 8
 - The chance that an arbitrary S, $|S|=8$, matches A is only 0.000015
 - But: $|G|=3.000.000.000$
 - Only by chance, we will have $\sim 45,000$ perfect matches
 - This applies even if we set the threshold at maximum
 - Help: For $|A|=16$, we expect less than 1 match by chance
- Generally: Number of false hits depend on the threshold t
 - Higher t: Stricter search, less false hits, but may incur misses
 - Lower t: Less strict, less misses, but more false hits
- Note: A match is a hypothesis calling for further analysis
 - By additional knowledge (e.g.: is S part of a gene?)
 - By experimentation (e.g.: can we find an isoform spliced at S?)

Pattern Matching

- We discussed exact matching and matching with a PSWM
- But **motifs** may also look quite differently
 - Motifs (domains) in protein sequences
 - Some important positions and much “glue” of unspecified length
 - Pattern here may be: `[AV].*FGKG[SIV]2.*[LI]`...
 - Which positions in S should we compare to which columns in P?
 - How can we **derive a specific pattern P** from S_1 - S_6 ?

```

S1: M---AIDE----NKQKALAAALGQ--KQFGKGSIMRLGEDR-SMDVETISTGSLSLDI
S2: MSDN-----KKQQALELALKQI-KQFGKGSIMKLGDG-ADHSIEAIPSGSIALDI
S3: M----AINTDTSGKQKALTMVLNQUIERSEFGKGVIMRLGDA-TRMRVETISTGALTLDL
S4: M-----DRQKALEAAVSQ--RAFGKGSIM-LGGKD---ETEVVSTRILGLDV
S5: M-----DE---NKKRALAAALGQI-KQFGKGVSMRMGDHE-RQAIPAISTGSLGLDI
S6: MD-----K-EKSEFGKGSIMRMGEE-VVEQVEVIPTGSIA---

```

Further Reading

- On string matching algorithms
 - Gusfield
- On sequence logos and TFBS-identification
 - Christianini & Hahn, chapter 10
 - Merkl & Waack, chapter 10