## Motifs

Certain patterns on DNA and protein sequences are strongly conserved by evolution, meaning they likely have important biological functions.

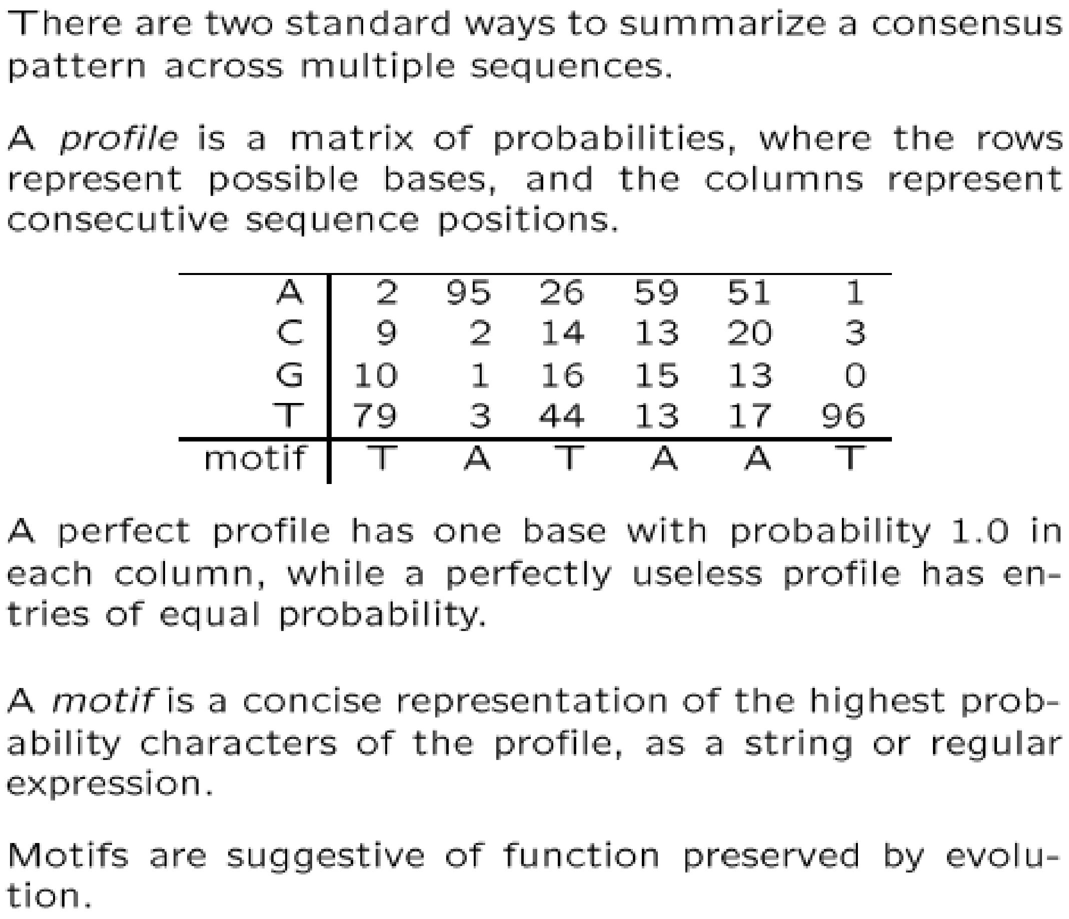
In DNA sequences, *promoter binding sites* are typically marked by a pattern of 5-10 nucleotides shortly up-stream of each *co-regulated* gene.

In protein sequences, highly conserved regions among similar proteins likely define the most important folds or binding sites.

Highly conserved does not mean completely conserved, and such short sequence matches may not stand out in any pairwise alignment.

Thus it is important to be able to identify possible *motifs* from a set of related sequences, and be able to test whether a given string contains a given motif.

# Profiles and motifs



There are two standard ways to summarize a consensus pattern across multiple sequences.

A *profile* is a matrix of probabilities, where the rows represent possible bases, and the columns represent consecutive sequence positions.

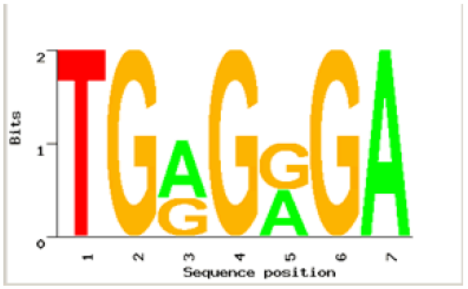
| | | | | | | |
|---|---|---|---|---|---|---|
| A | 2 | 95 | 26 | 59 | 51 | 1 |
| C | 9 | 2 | 14 | 13 | 20 | 3 |
| G | 10 | 1 | 16 | 15 | 13 | 0 |
| T | 79 | 3 | 44 | 13 | 17 | 96 |
| motif | T | A | T | A | A | T |

A perfect profile has one base with probability 1.0 in each column, while a perfectly useless profile has entries of equal probability.

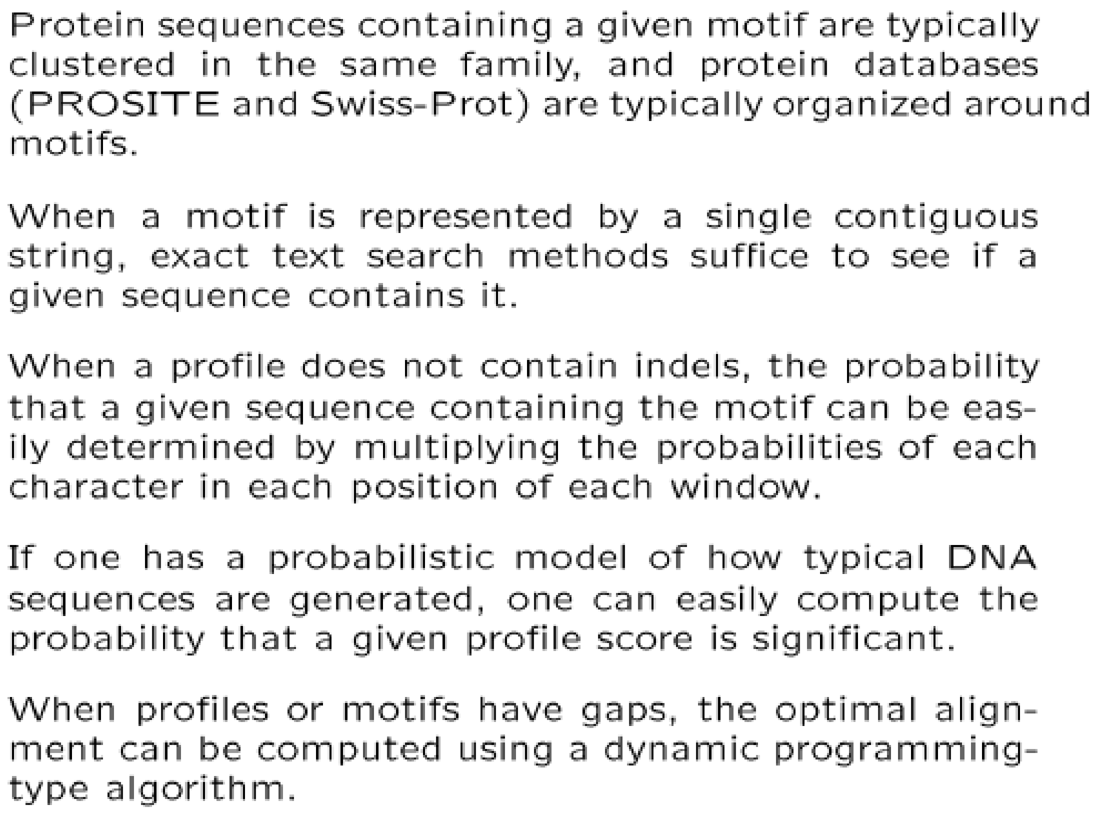
A *motif* is a concise representation of the highest probability characters of the profile, as a string or regular expression.

Motifs are suggestive of function preserved by evolution.

TGGGGGA
TGAGAGA
TGGGGGA
TGAGAGA
TGAGGGA



# Searching against profiles and motifs



Protein sequences containing a given motif are typically clustered in the same family, and protein databases (PROSITE and Swiss-Prot) are typically organized around motifs.

When a motif is represented by a single contiguous string, exact text search methods suffice to see if a given sequence contains it.

When a profile does not contain indels, the probability that a given sequence containing the motif can be easily determined by multiplying the probabilities of each character in each position of each window.

If one has a probabilistic model of how typical DNA sequences are generated, one can easily compute the probability that a given profile score is significant.

When profiles or motifs have gaps, the optimal alignment can be computed using a dynamic programming-type algorithm.

# Why finding motifs is hard

## Random Sample

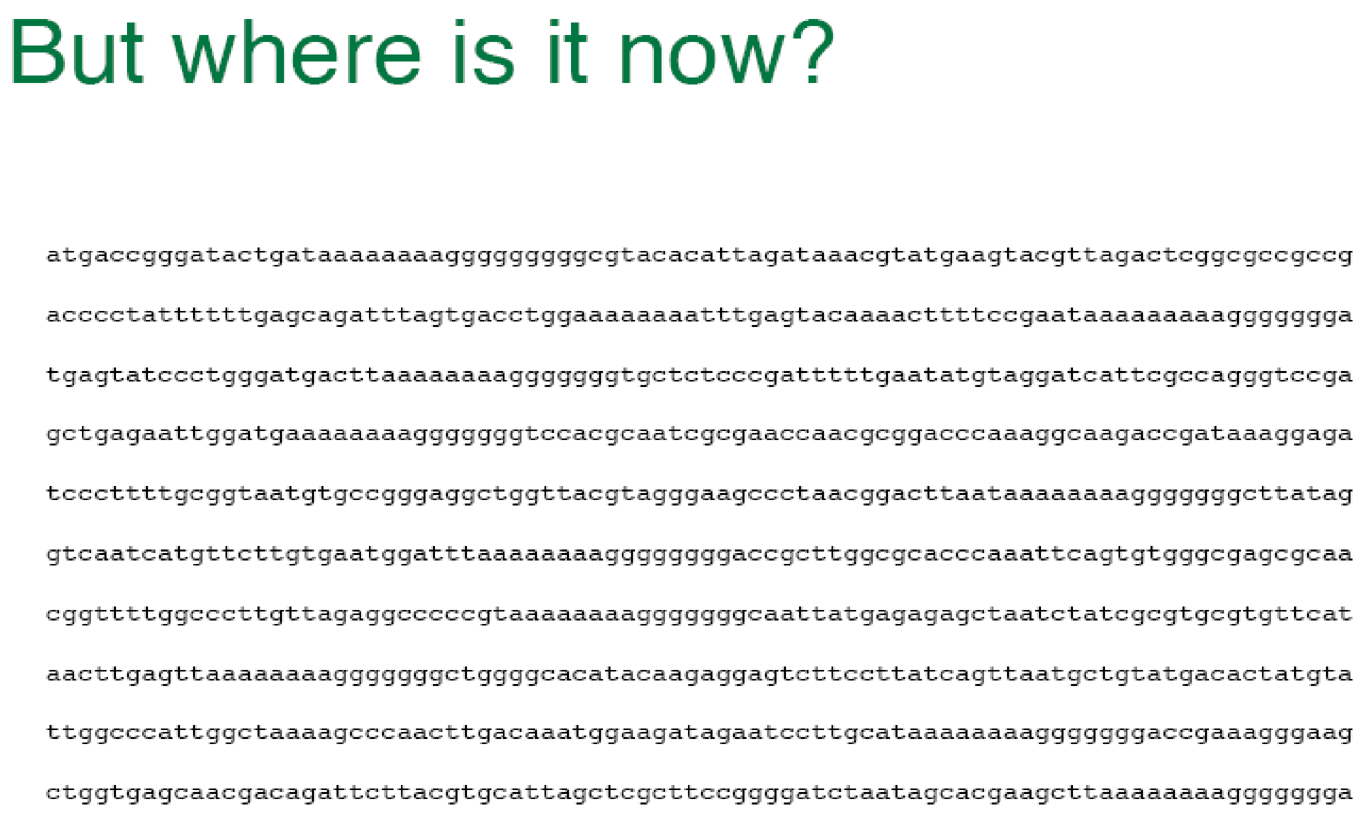
```
atgaccgggatactgataccgtatttggcctaggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctattttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaatactgggcataaggtaca

tgagtatccctgggatgacttttgggaacactatagtgctctcccgatttttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgaccttgtaagtgttttccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tcccttttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatggcccacttagtccacttatag

gtcaatcatgttcttgtgaatggatttttaactgagggcatagaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgtactgatggaaactttcaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttggtttcgaaaatgctctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatttcaacgtatgccgaaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttctgggtactgatagca
```

# Why finding motifs is hard

## Implanting the Motif: AAAAAAAGGGGGGG

```
atgaccgggatactgatAAAAAAAAGGGGGGGggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg
acccctattttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaataAAAAAAAAGGGGGGGa
tgagtatccctgggatgacttAAAAAAAAGGGGGGGtgctctcccgatttttgaatatgtaggatcattcgccagggtccga
gctgagaattggatgAAAAAAAAGGGGGGGtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga
tcccttttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatAAAAAAAAGGGGGGGcttatag
gtcaatcatgttcttgtgaatggatttAAAAAAAAGGGGGGGgaccgcttggcgcacccaaattcagtgtgggcgagcgcaa
cggttttggcccttgttagaggcccccgtAAAAAAAAGGGGGGGcaattatgagagagctaatctatcgcgtgcgtgttcat
aacttgagttAAAAAAAAGGGGGGGctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta
ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatAAAAAAAAGGGGGGGaccgaaagggaag
ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttAAAAAAAAGGGGGGGa
```

## Why finding motifs is hard



But where is it now?

```
atgaccgggatactgataaaaaaaagggggggggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg
acccctattttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaataaaaaaaaaggggggga
tgagtatccctgggatgacttaaaaaaaaggggggtgctctcccgatttttgaatatgtaggatcattcgccagggtccga
gctgagaattggatgaaaaaaaaagggggggtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga
tcccttttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaataaaaaaaaagggggggcttatag
gtcaatcatgttcttgtgaatggatttaaaaaaaaggggggggaccgcttggcgcacccaaattcagtgtgggcgagcgcaa
cggttttggcccttgttagaggcccccgtaaaaaaaagggggggcaattatgagagagctaatctatcgcgtgcgtgttcat
aacttgagttaaaaaaaagggggggctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta
ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcataaaaaaaaagggggggaccgaaagggaag
ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttaaaaaaaaggggggga
```

# Why finding motifs is hard

## Implanting the Motif with Four Mutations

```
atgaccgggatactgatAgAAgAAAGGttGGGggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg
acccctattttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaatacAAtAAAAcGGcGGGa
tgagtatccctgggatgacttAAAAtAAtGGaGtGGtgctctcccgatttttgaatatgtaggatcattcgccagggtccga
gctgagaattggatgcAAAAAAAGGGattGtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga
tcccttttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatAtAAtAAAGGaaGGGcttatag
gtcaatcatgttcttgtgaatggatttAAcAAtAAGGGctGGgaccgcttggcgcacccaaattcagtgtgggcgagcgcaa
cggttttggcccttgttagaggcccccgtAtAAAcAAGGaGGGccaattatgagagagctaatctatcgcgtgcgtgttcat
aacttgagttAAAAAAtAGGGaGccctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta
ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatActAAAAAGGaGcGGaccgaaagggaag
ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttActAAAAAGGaGcGGa
```

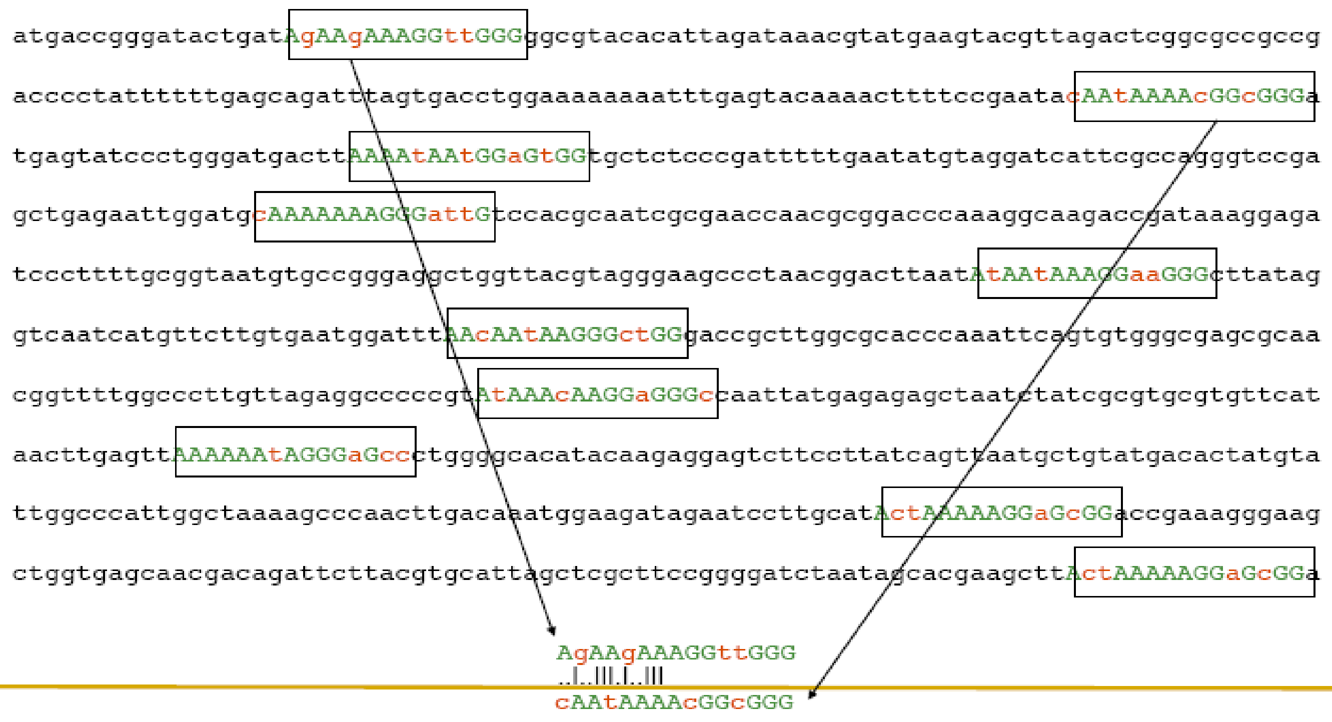
## Why finding motifs is hard

Oh, geez.  Where is it now?!

```
atgaccgggatactgatagaagaaaggttgggggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg

acccctatttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaatacaataaaacggcggga

tgagtatccctgggatgacttaaaataatggagtggtgctctcccgatttttgaatatgtaggatcattcgccagggtccga

gctgagaattggatgcaaaaaaagggattgtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga

tcccttttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatataataaaggaagggcttatag

gtcaatcatgttcttgtgaatggatttaacaataagggctgggaccgcttggcgcacccaaattcagtgtgggcgagcgcaa

cggttttggcccttgttagaggcccccgtataaacaaggagggccaattatgagagagctaatctatcgcgtgcgtgttcat

aacttgagttaaaaaatagggagccctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta

ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatactaaaaaggagcggaccgaaagggaag

ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttactaaaaaggagcgga
```

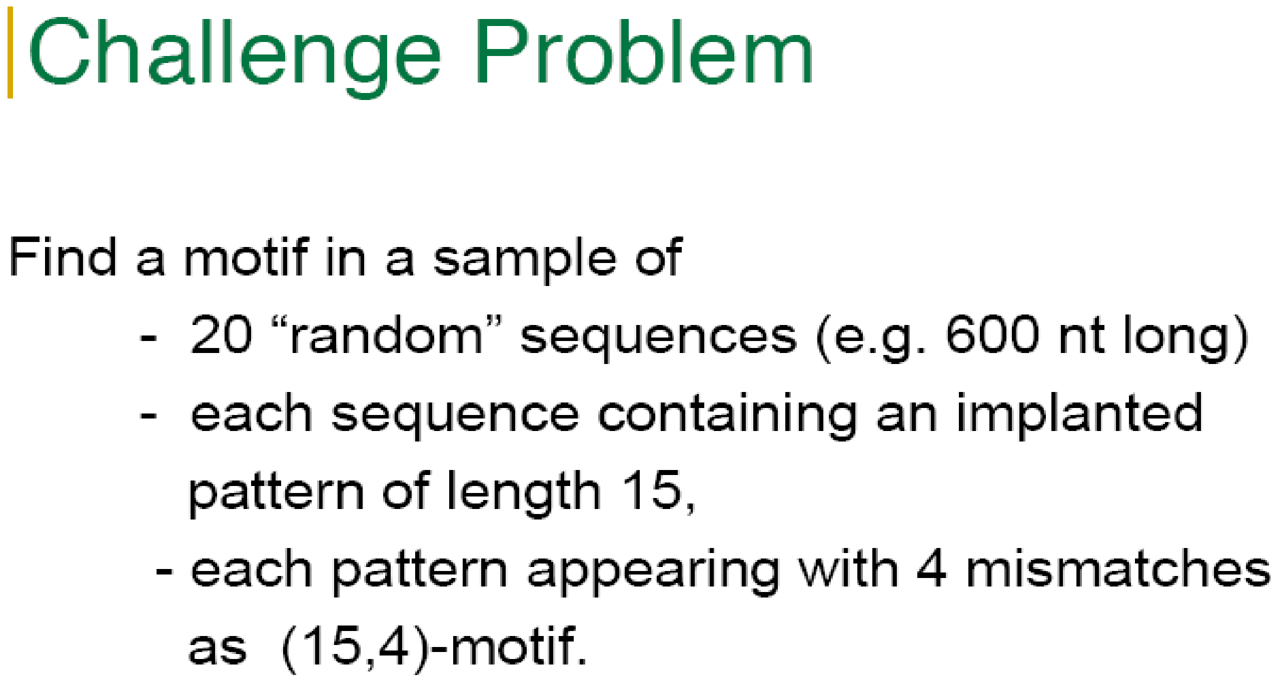
# Why finding motifs is hard

## Why is Finding a (15,4) Motif Hard?



```
atgaccgggatactgatAgAAgAAAGGttGGGggcgtacacattagataaacgtatgaagtacgttagactcggcgccgccg
acccctattttttgagcagatttagtgacctggaaaaaaaatttgagtacaaaacttttccgaatacAAtAAAAcGGcGGGa
tgagtatccctgggatgacttAAAAtAAtGGaGtGGtgctctcccgatttttgaatatgtaggatcattcgccagggtccga
gctgagaattggatgcAAAAAAAGGGattGtccacgcaatcgcgaaccaacgcggacccaaaggcaagaccgataaaggaga
tcccttttgcggtaatgtgccgggaggctggttacgtagggaagccctaacggacttaatAtAAtAAAGGaaGGGcttatag
gtcaatcatgttcttgtgaatggatttAAcAAtAAGGGctGGgaccgcttggcgcacccaaattcagtgtgggcgagcgcaa
cggttttggcccttgttagaggcccccgtAtAAAcAAGGaGGGccaattatgagagagctaatctatcgcgtgcgtgttcat
aacttgagttAAAAAAtAGGGaGccctggggcacatacaagaggagtcttccttatcagttaatgctgtatgacactatgta
ttggcccattggctaaaagcccaacttgacaaatggaagatagaatccttgcatActAAAAAGGaGcGGaccgaaagggaag
ctggtgagcaacgacagattcttacgtgcattagctcgcttccggggatctaatagcacgaagcttActAAAAAGGaGcGGa

AgAAgAAAGGttGGG
..|..|||.|..|||
cAAtAAAAcGGcGGG
```

## Why finding motifs is hard



### Challenge Problem

Find a motif in a sample of

- 20 “random” sequences (e.g. 600 nt long)
- each sequence containing an implanted pattern of length 15,
- each pattern appearing with 4 mismatches as (15,4)-motif.

## Combinatorial gene regulation and Regulatory Proteins

A microarray experiment showed that when gene X is knocked out, 20 other genes are not expressed

- **How can one gene have such drastic effects?**

Gene X encodes regulatory protein, a.k.a. a ***transcription factor*** (TF)

The 20 unexpressed genes rely on gene X's TF to induce transcription

A single TF may regulate multiple genes

## Regulatory Regions

Every gene contains a regulatory region (RR) typically stretching 100-1000 bp upstream of the transcriptional start site

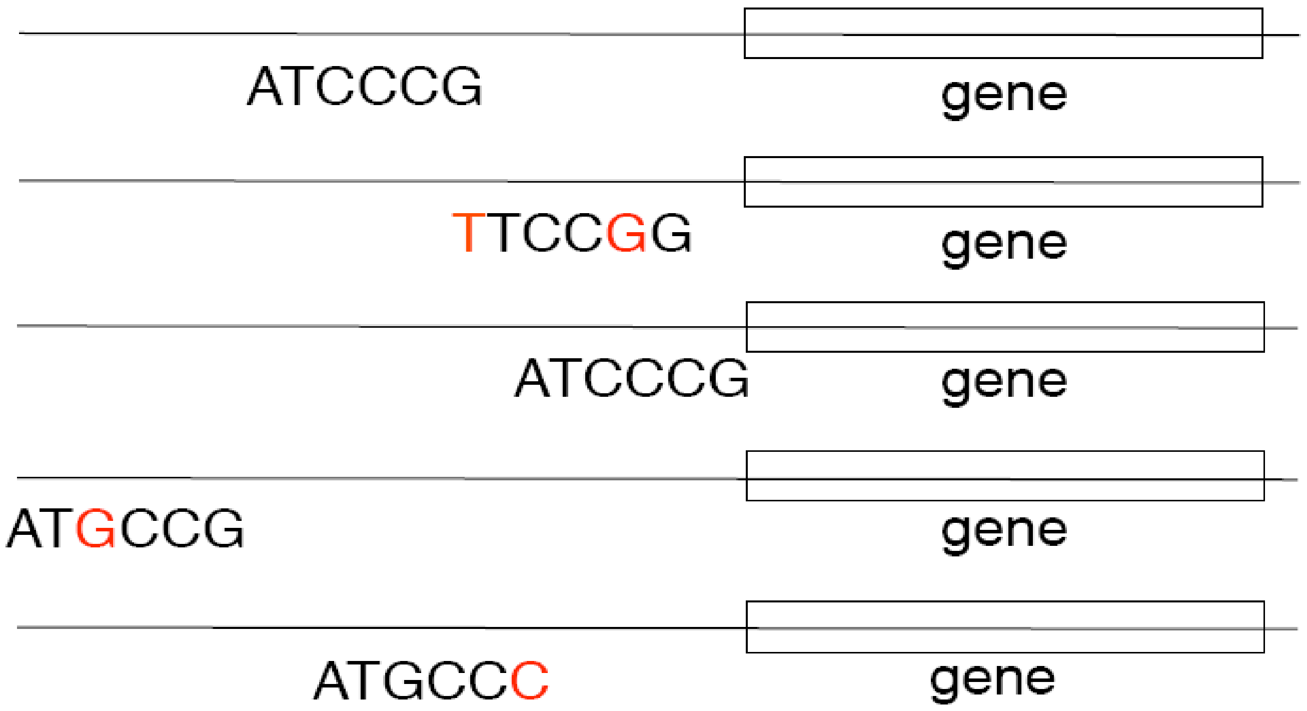
Located within the RR are the ***Transcription Factor Binding Sites*** (TFBS), also known as ***motifs***, specific for a given transcription factor

TFs influence gene expression by binding to a specific location in the respective gene's regulatory region - TFBS

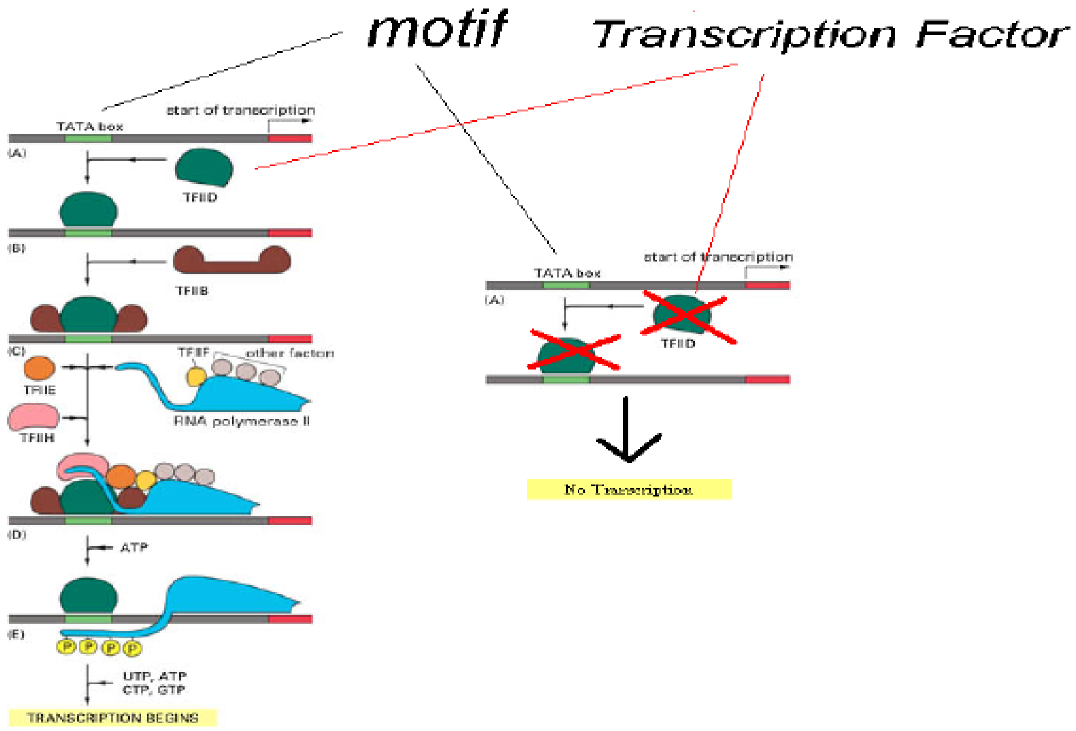
A TFBS can be located anywhere within the Regulatory Region.

TFBS may vary slightly across different regulatory regions since non-essential bases could mutate

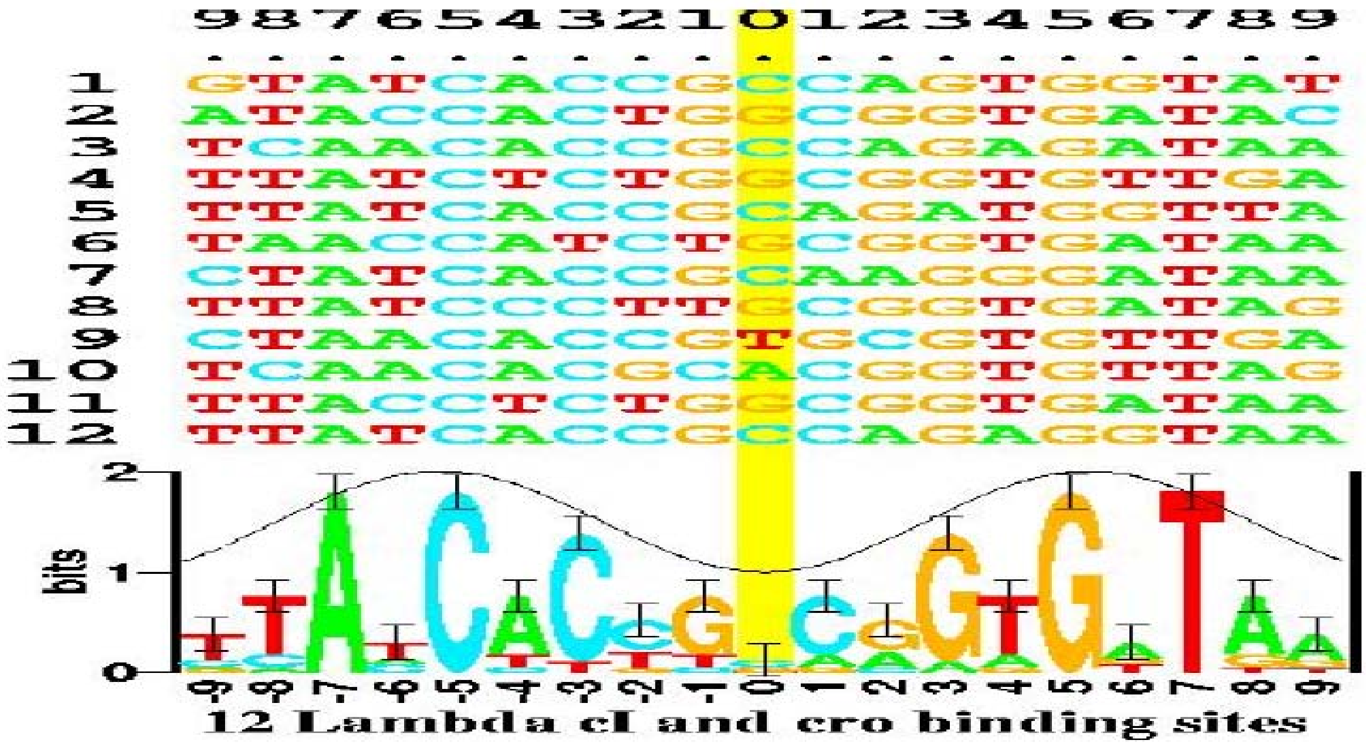
# Motifs and transcriptional start sites



# Transcription factors and motifs



# Motif Profiles



```
    98765432101234567890
    . . . . . . . . . .
 1  GTATCACCGCCAGTGGTAT
 2  ATACCACTGGCGGTGATAC
 3  TCAACACCGCCAGAGATAA
 4  TTATCTCTGGCGGTGTTGA
 5  TTATCACCGCAGATGGTTA
 6  TAACCATCTGCGGTGATAA
 7  CTATCACCGCAAGGGATAA
 8  TTATCCCTTGCGGTGATAG
 9  CTAACACCGTGCGTGTTGA
10  TCAACACGCACGGTGTTAG
11  TTACCTCTGGCGGTGATAA
12  TTATCACCGCCAGAGGTAA
```

# Identifying motifs and complications

Genes are turned on or off by regulatory proteins

These proteins bind to upstream regulatory regions of genes to either attract or block an RNA polymerase

Regulatory protein (TF) binds to a short DNA sequence called a motif (TFBS)

So finding the same motif in multiple genes' regulatory regions suggests a regulatory relationship amongst those genes

We do not know the motif sequence

We do not know where it is located relative to the genes start

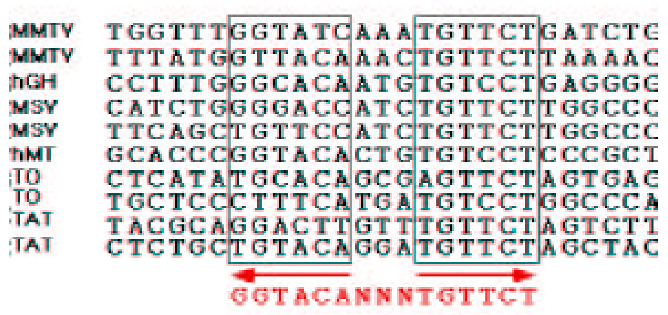
Motifs can differ slightly from one gene to the next

How to discern it from “random” motifs?

# Finding motifs / alignments

Motifs stand out as highly conserved regions in a *multiple sequence alignment*

GRE Consensus Sequence

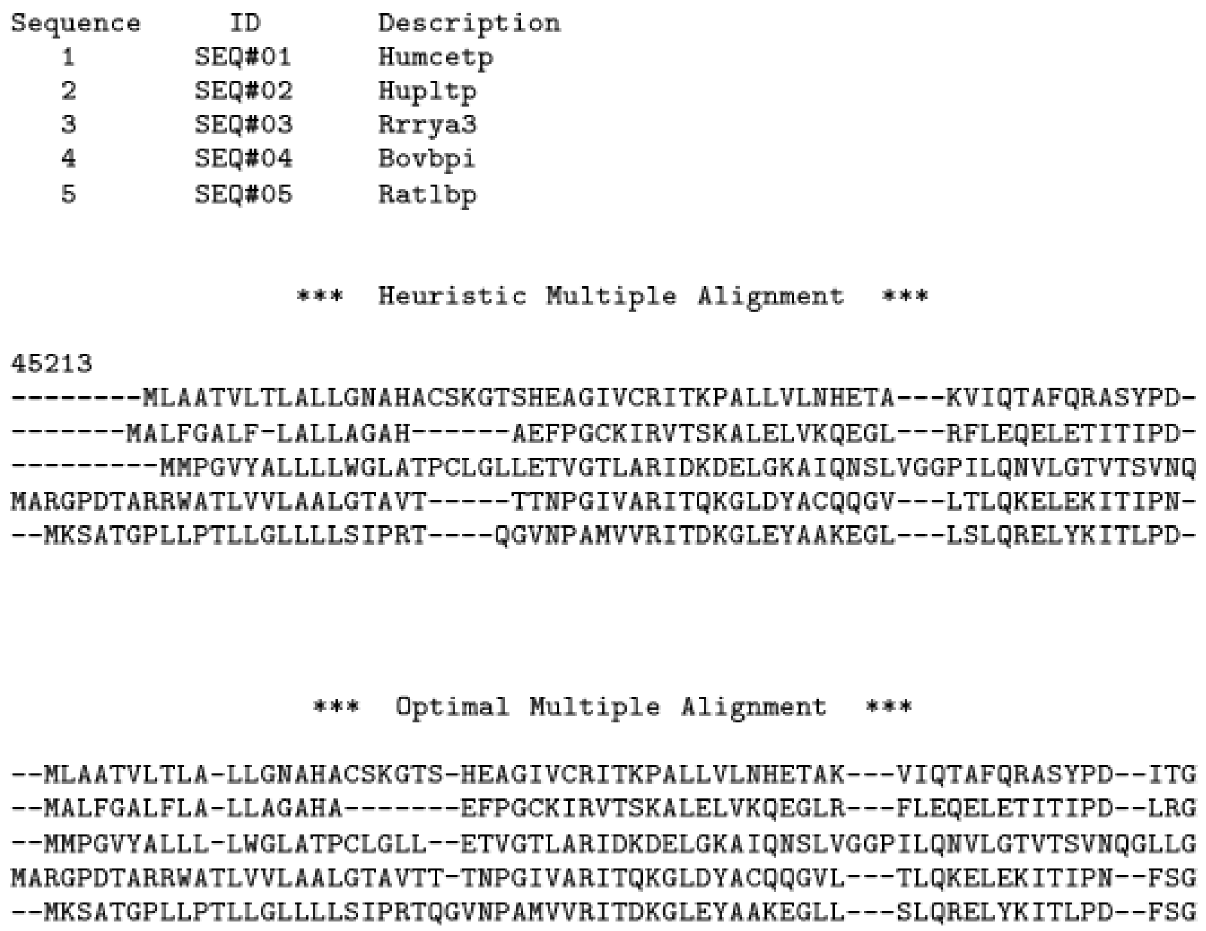


```
MMTV   TGGTTTGGTATCAAATGTTCTGATCTG
MMTV   TTTATGGTTACAAACTGTTCTTAAAAC
hGH    CCTTTGGGCACAATGTGTCCTGAGGGG
MSV    CATCTGGGGACCATCTGTTCTTGGCCC
MSV    TTCAGCTGTTCCATCTGTTCTTGGCCC
hMT    GCACCCGGTACACTGTGTCCTCCCGCT
TO     CTCATATGCACAGCGAGTTCTAGTGAG
TO     TGCTCCCTTTCATGATGTCCTGGCCCA
TAT    TACGCAGGACTTGTTTGTTCTAGTCTT
TAT    CTCTGCTGTACAGGATGTTCTAGCTAC

             GGTACANNNTGTTCT
```

MMTV = mouse mammary tumor virus
hGH = human growth hormone
MSV = murine sarcoma virus
hMT = human metallothionein
TO = tyrosine oxidase
TAT = tyrosine aminotransferase

Fragment assembly programs must have a multiple sequence alignment code to extract a consensus call to define each base in the final sequence, but base calling accuracy is high enough that this sequence alignment problem is much easier than for conserved regions under evolution.

# Heuristic vs optimal alignments

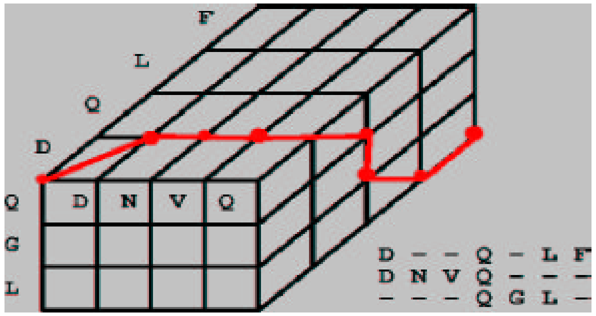


```
Sequence      ID       Description
   1        SEQ#01     Humcetp
   2        SEQ#02     Hupltp
   3        SEQ#03     Rrrya3
   4        SEQ#04     Bovbpi
   5        SEQ#05     Ratlbp


                 ***  Heuristic Multiple Alignment  ***

45213
--------MLAATVLTLALLGNAHACSKGTSHEAGIVCRITKPALLVLNHETA---KVIQTAFQRASYPD-
-------MALFGALF-LALLAGAH------AEFPGCKIRVTSKALELVKQEGL---RFLEQELETITIPD-
---------MMPGVYALLLLWGLATPCLGLLETVGTLARIDKDELGKAIQNSLVGGPILQNVLGTVTSVNQ
MARGPDTARRWATLVVLAALGTAVT-----TTNPGIVARITQKGLDYACQQGV---LTLQKELEKITIPN-
--MKSATGPLLPTLLGLLLLSIPRT----QGVNPAMVVRITDKGLEYAAKEGL---LSLQRELYKITLPD-


                  ***  Optimal Multiple Alignment  ***

--MLAATVLTLA-LLGNAHACSKGTS-HEAGIVCRITKPALLVLNHETAK---VIQTAFQRASYPD--ITG
--MALFGALFLA-LLAGAHA-------EFPGCKIRVTSKALELVKQEGLR---FLEQELETITIPD--LRG
--MMPGVYALLL-LWGLATPCLGLL--ETVGTLARIDKDELGKAIQNSLVGGPILQNVLGTVTSVNQGLLG
MARGPDTARRWATLVVLAALGTAVTT-TNPGIVARITQKGLDYACQQGVL---TLQKELEKITIPN--FSG
--MKSATGPLLPTLLGLLLLSIPRTQGVNPAMVVRITDKGLEYAAKEGLL---SLQRELYKITLPD--FSG
```

# Dynamic programming

Pairwise sequence comparison algorithms based on dynamic programming can be natural generalized to 3 or more sequences, but at a cost.

The possible states of our dynamic programming algorithm are all subsets of positions in each sequence, so the dynamic programming matrix of $k$ sequences each of length $n$ has size $O(n^k)$.

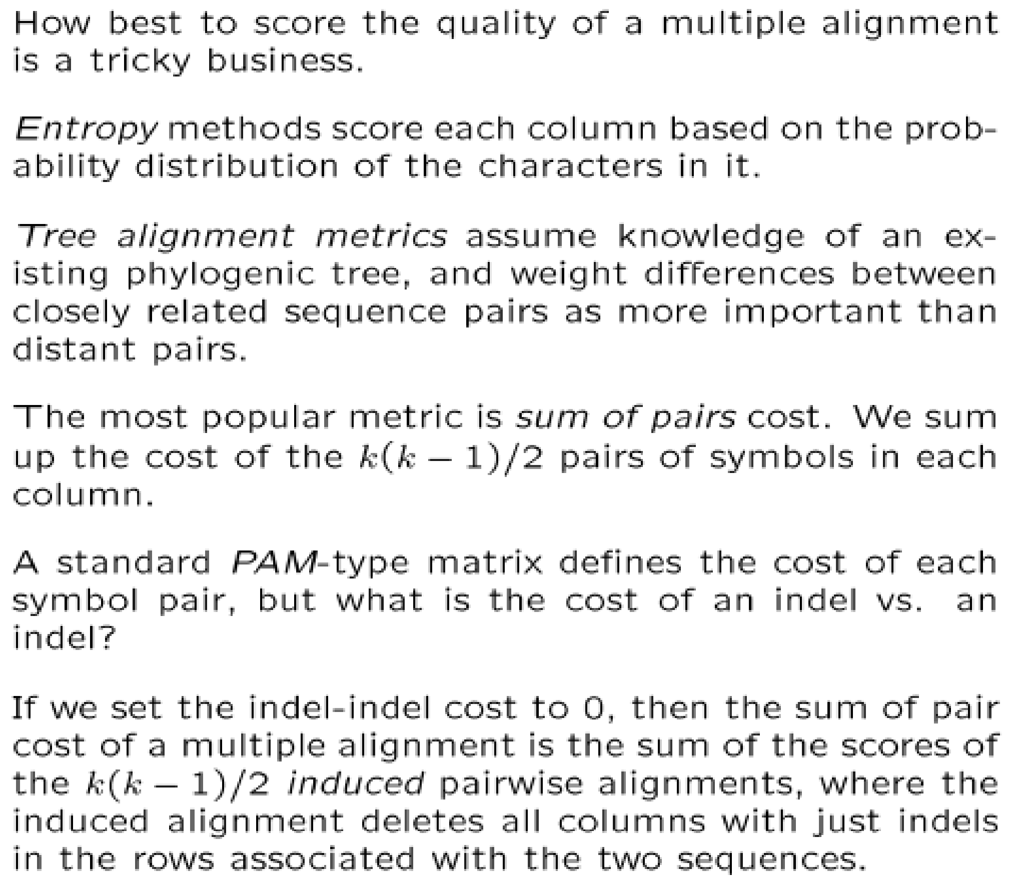


Since the last column of each state can contain any non-empty subset of sequences, the time to fill this matrix is $O(2^k n^k)$.

This quickly gets hopeless as $n$ and $k$ get large, e.g. $n = 100$ and $k = 10$.

Further, even more information must be stored if we are to charge lower per-base deletion penalties for gaps. For each position state, we must keep track of the cost for all $2^k - 1$ possibilities of ending each sequences alignment with a blank/gap.

## Scoring criteria for multiple alignment



How best to score the quality of a multiple alignment is a tricky business.

*Entropy* methods score each column based on the probability distribution of the characters in it.

*Tree alignment metrics* assume knowledge of an existing phylogenic tree, and weight differences between closely related sequence pairs as more important than distant pairs.

The most popular metric is *sum of pairs* cost. We sum up the cost of the $k(k-1)/2$ pairs of symbols in each column.

A standard *PAM*-type matrix defines the cost of each symbol pair, but what is the cost of an indel vs. an indel?

If we set the indel-indel cost to 0, then the sum of pair cost of a multiple alignment is the sum of the scores of the $k(k-1)/2$ *induced* pairwise alignments, where the induced alignment deletes all columns with just indels in the rows associated with the two sequences.

## Scoring criteria for multiple alignment

Sum of pairs is a nice measure because (1) it is quite natural, and (2) a good upper bound on the best possible alignment follows by finding all pairwise alignments and summing the scores.

Finding the optimal sum of pairs alignment is NP-complete.

However, by exploiting the lower bounds given by each pairwise DP matrix, one can heuristically reduce the number of states in the multiple dynamic programming matrix and hope to find the optimal alignment of (say) 6-7 sequences of (say) 200 characters in a reasonable amount of time.

## Heuristic alignments

Since the most interesting applications of multiple alignment lie beyond the range of exact solution algorithms, we must use heuristic methods.

A natural method exploits our ability to do pairwise alignments, and inserts sequences one by one into the alignment.

Note that aligning two *consensus alignments* is not conceptually different than aligning two sequences, since we can treat indels as just a space character.

Thus heuristics differ according to the order in which they merge the sequences, which is defined by a *tree*.

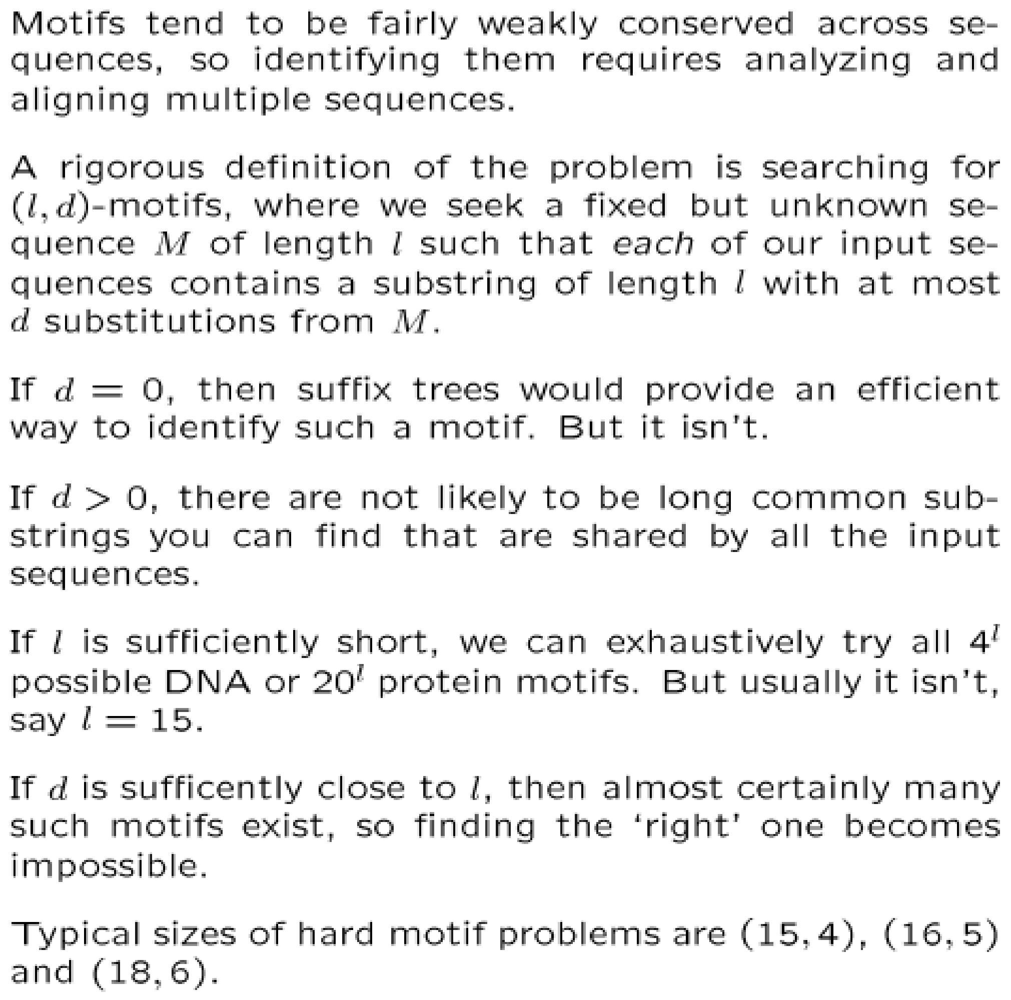
Reasonable trees are (1) based on known phylogenic trees, or similarity clustering dendograms, (2) of fixed topologies, such as paths or binary trees.

Fixed topologies require associating each node with an input sequence.

In *star* alignments, one sequence is selected as the center, and all other sequences merged on a pairwise basis into it.

Picking as the center the sequence "most similar to the others" gives the best results.

## Complexity of motif finding



Motifs tend to be fairly weakly conserved across sequences, so identifying them requires analyzing and aligning multiple sequences.

A rigorous definition of the problem is searching for $(l, d)$-motifs, where we seek a fixed but unknown sequence $M$ of length $l$ such that *each* of our input sequences contains a substring of length $l$ with at most $d$ substitutions from $M$.

If $d = 0$, then suffix trees would provide an efficient way to identify such a motif. But it isn't.

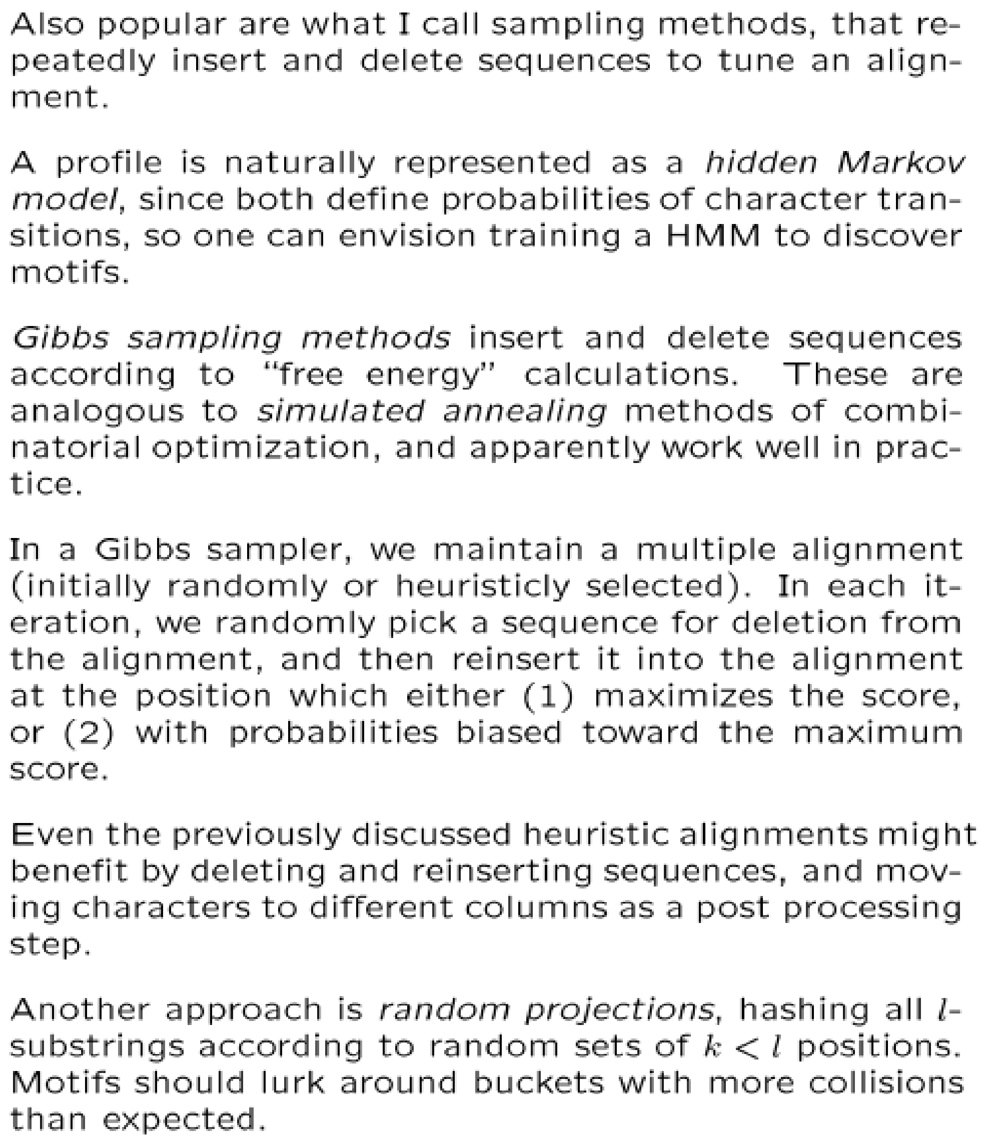
If $d > 0$, there are not likely to be long common substrings you can find that are shared by all the input sequences.

If $l$ is sufficiently short, we can exhaustively try all $4^l$ possible DNA or $20^l$ protein motifs. But usually it isn't, say $l = 15$.

If $d$ is sufficently close to $l$, then almost certainly many such motifs exist, so finding the 'right' one becomes impossible.

Typical sizes of hard motif problems are $(15, 4)$, $(16, 5)$ and $(18, 6)$.

## Sampling methods



Also popular are what I call sampling methods, that repeatedly insert and delete sequences to tune an alignment.

A profile is naturally represented as a *hidden Markov model*, since both define probabilities of character transitions, so one can envision training a HMM to discover motifs.

*Gibbs sampling methods* insert and delete sequences according to "free energy" calculations. These are analogous to *simulated annealing* methods of combinatorial optimization, and apparently work well in practice.

In a Gibbs sampler, we maintain a multiple alignment (initially randomly or heuristicly selected). In each iteration, we randomly pick a sequence for deletion from the alignment, and then reinsert it into the alignment at the position which either (1) maximizes the score, or (2) with probabilities biased toward the maximum score.

Even the previously discussed heuristic alignments might benefit by deleting and reinserting sequences, and moving characters to different columns as a post processing step.

Another approach is *random projections*, hashing all $l$-substrings according to random sets of $k < l$ positions. Motifs should lurk around buckets with more collisions than expected.