

Algorithms in Sequence Analysis:

Solon P. Pissis

Lecture 14

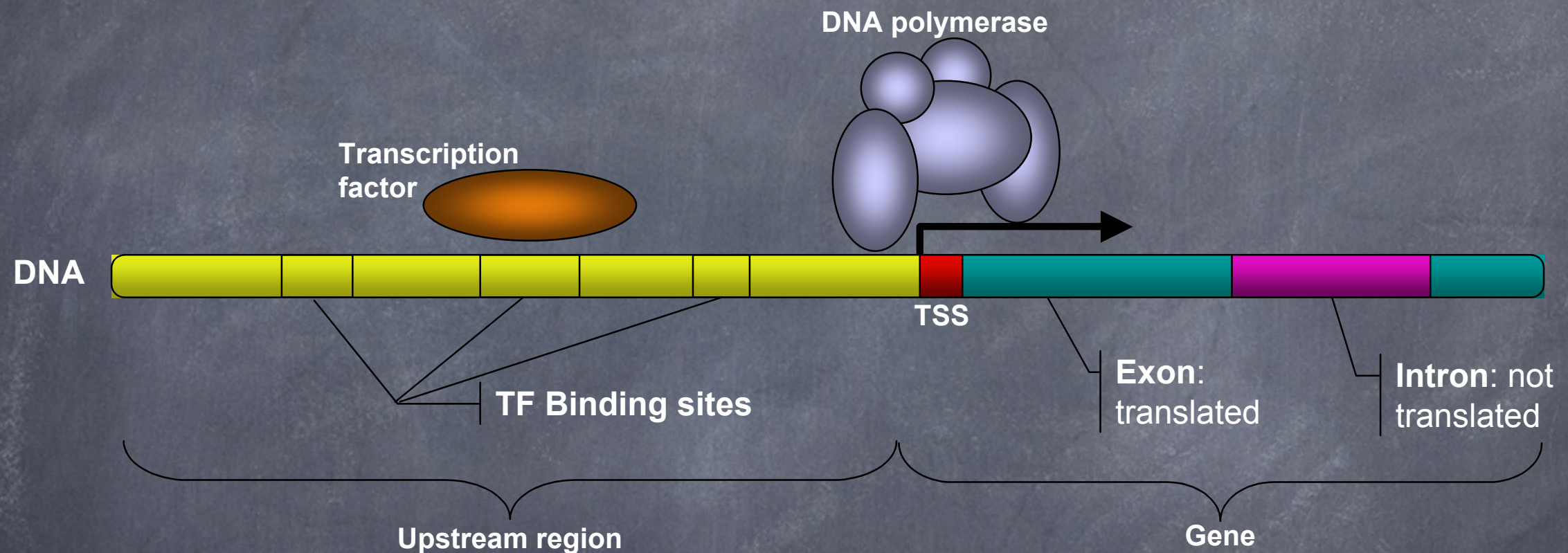
Motif finding

Slides courtesy of

Carl Kingsford

Carnegie Mellon University

DNA → mRNA → Protein



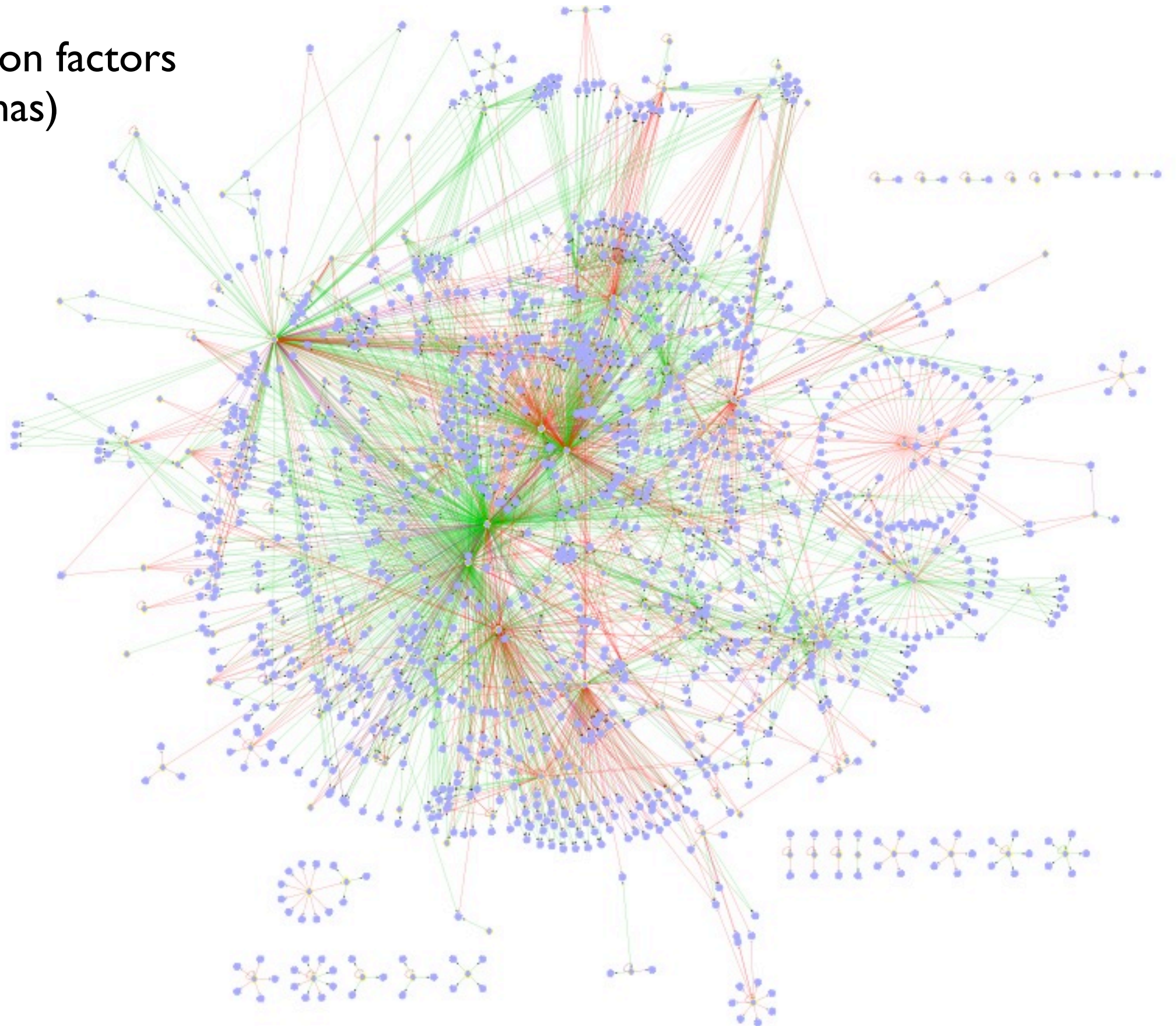
- Finding transcription factor binding sites can tell us about the cell's regulatory network.

Transcription Network

169 transcription factors
(excluding sigmas)

3322 edges

1753 activation,
1369 repression,
185 both,
3 unknown



RNA Polymerase

b/c it makes RNA

into a polymer

is an enzyme

Discovered in 1960; Nobel prize for its discovery in 1959... oops

1959 Nobel awarded to Severo Ochoa and Arthur Kornberg for discovering what was mistakenly believed to be RNA pol.

1960 Sam Weiss and Jared Hurwitz discover the *real* RNA pol.

2006 Nobel awarded to **Roger** Kornberg (son of Arthur) for detailed structure of RNA pol.

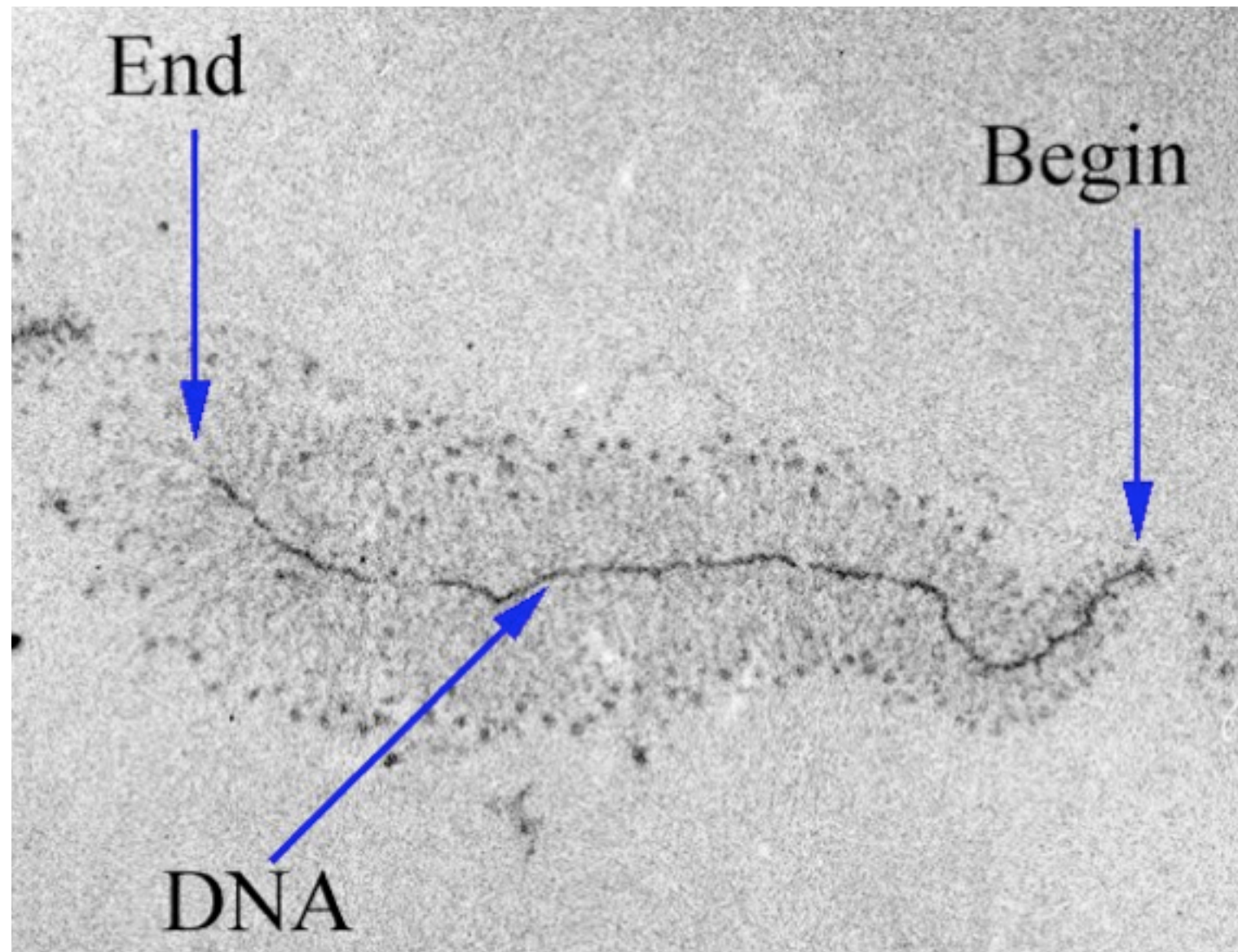
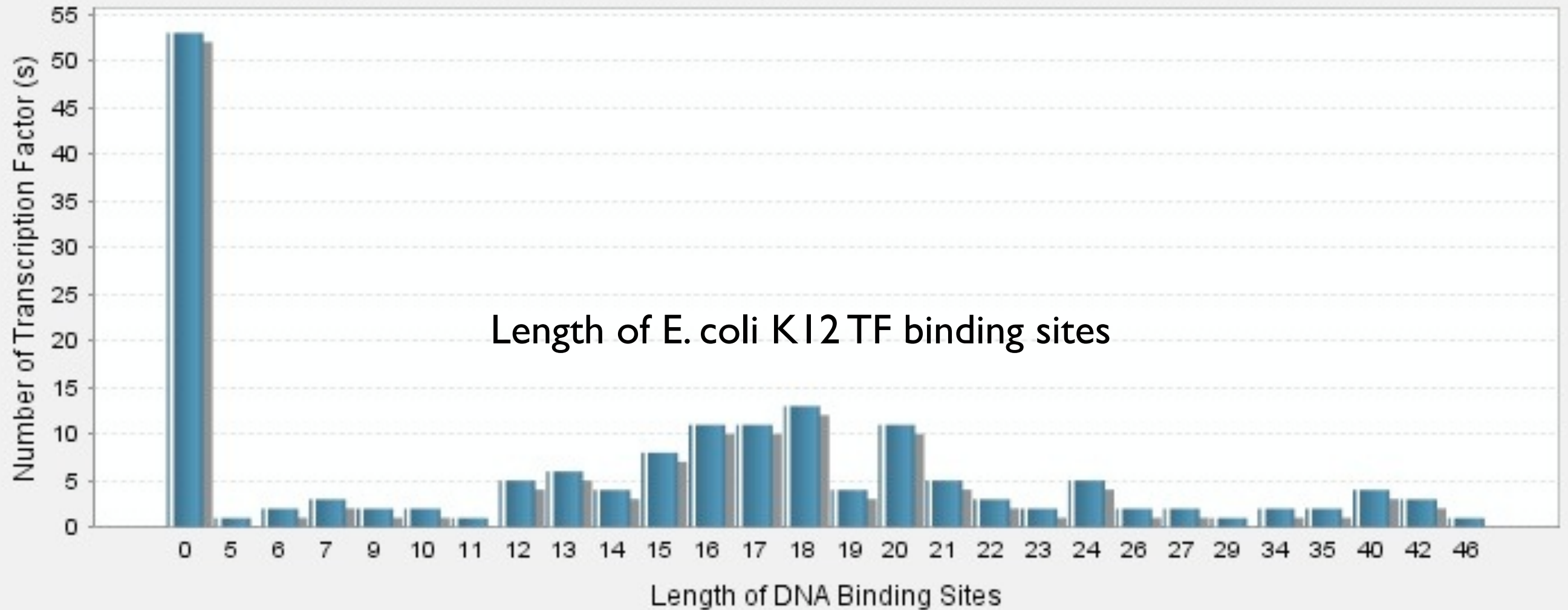


Image of transcription occurring.

Each “hair” is a piece of RNA that RNA pol is growing off of the DNA.

Transcription Factor Binding Sites

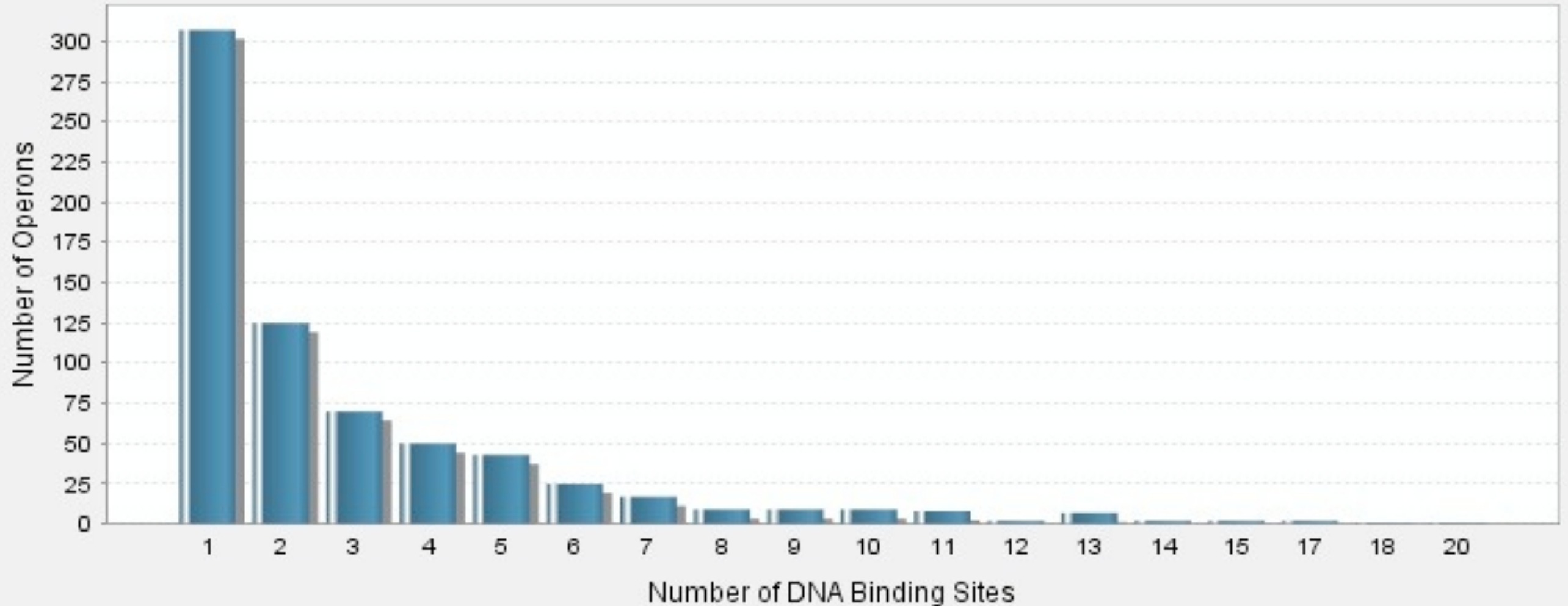
Length of DNA Binding Sites per Transcription Factor



RegulonDB (Feb 27, 2010)

Transcription Factor Binding Sites

Number of DNA Binding Sites of Transcription Factors per Operon



RegulonDB (Feb 27, 2010)

Motif Finding



Transcription factor

1. ttgccacaaaataatccgccttcgcaaattgacc**TACCTCAATAGCGGTA**gaaaaacgcaccactgcctgacag
2. gtaagtacctgaaagttacggtctgcgaacgctattccac**TGCTCCTTTATAGGTA**caacagtatagtctgatgga
3. ccacacggcaaataaggag**TAACTCTTTCCGGTA**tgggtatacttcagccaatagccgagaatactgccattccag
4. ccatacccggaaagagttactccttattttgccgtgtggtagtcgctt**TACATCGGTAAGGGTA**gggatttttacagca
5. aaactattaagatttttatgcagatgggtattaagga**GTATTCCCATGGGTA**acataattaatggctctta
6. ttacagtctgttatgtggtggctgttaa**TTATCCTAAAGGGGTA**tcttaggaatttactt

Given p sequences, find the **most mutually similar** length- k subsequences, one from each sequence:

$$\operatorname{argmin}_{s_1, \dots, s_p} \sum_{i < j} \operatorname{dist}(s_i, s_j)$$

$\operatorname{dist}(s_i, s_j) =$ **Hamming distance** between s_i and s_j .

Hundreds of papers, many formulations (Tompa05)

Motif-finding by Gibbs Sampling

Problem. Given p strings and a length k , find the most “mutually similar” length- k substring from each string.

Not exactly the same though

“Gibbs sampling” is the basis behind a general class of algorithms that is a type of local search.

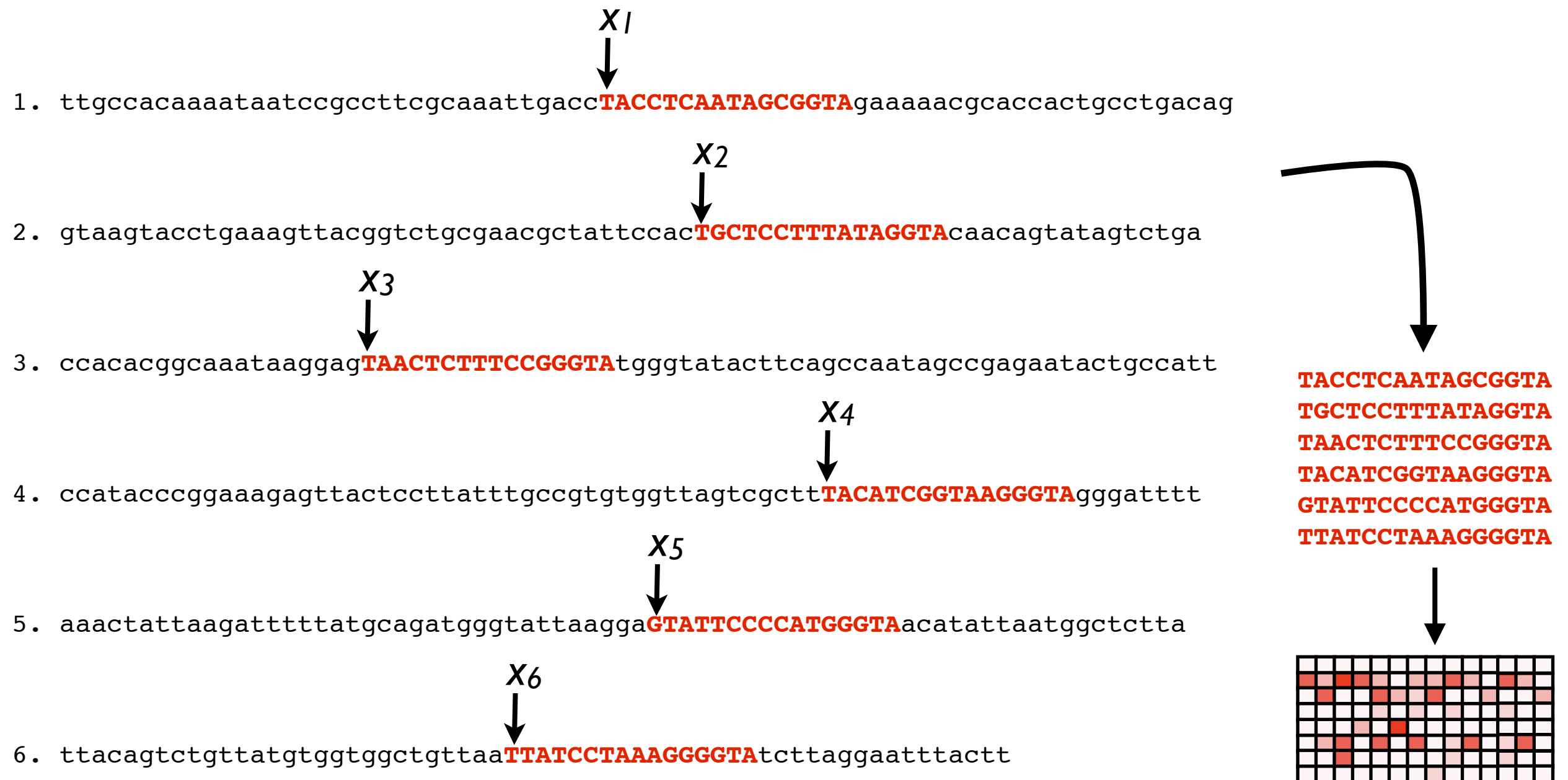
It doesn't guarantee good performance, but often works well in practice.

Assumes:

1. we know the length k of the motif we are looking for.
2. each input sequence contains exactly 1 real instance of the motif.

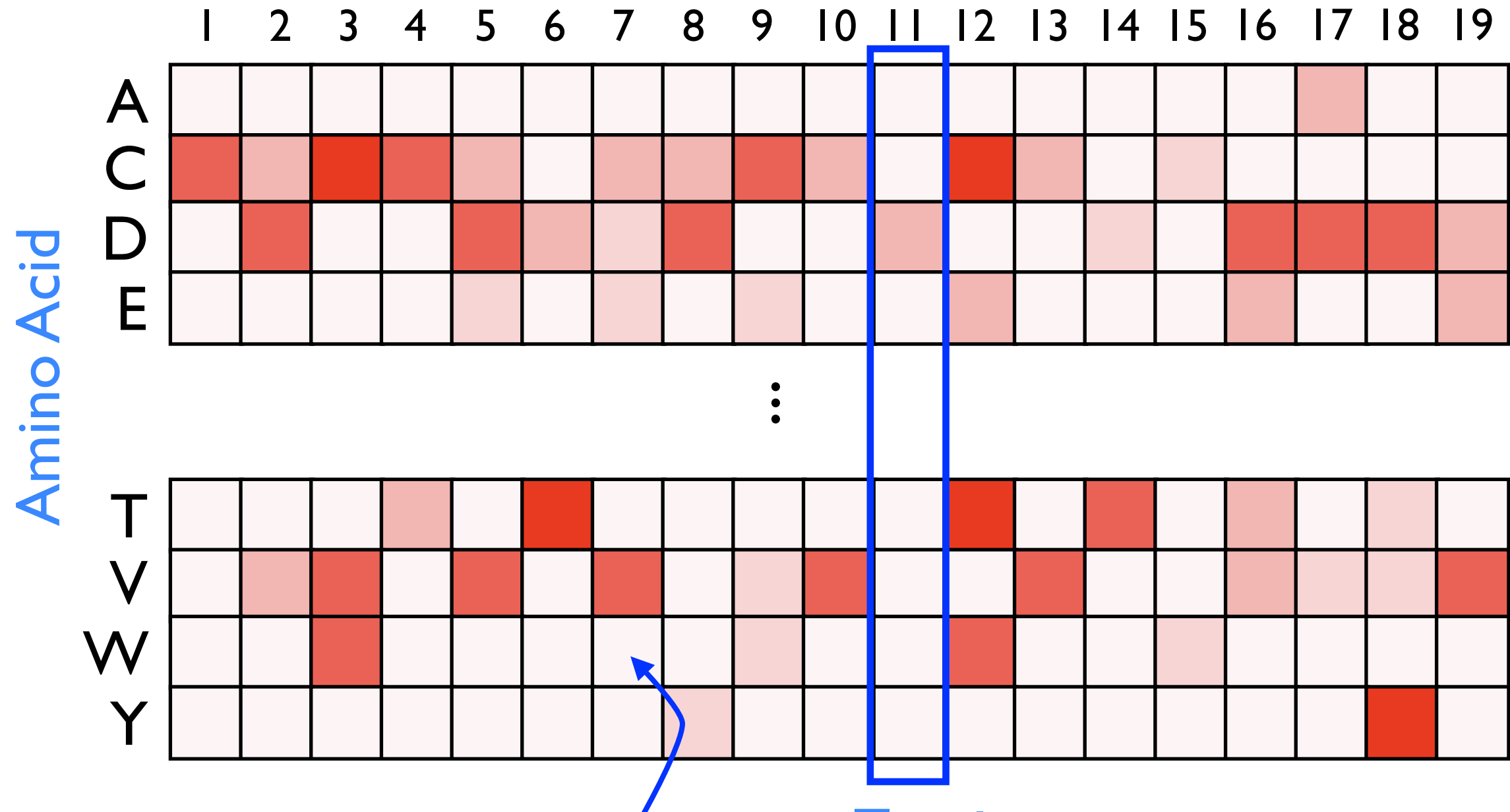
Gibbs Sampling: Profiles

If we knew the starting point of the motif in each sequence, we could construct a Sequence Profile (PSSM) for the motif:



Sequence Profiles (PSSM)

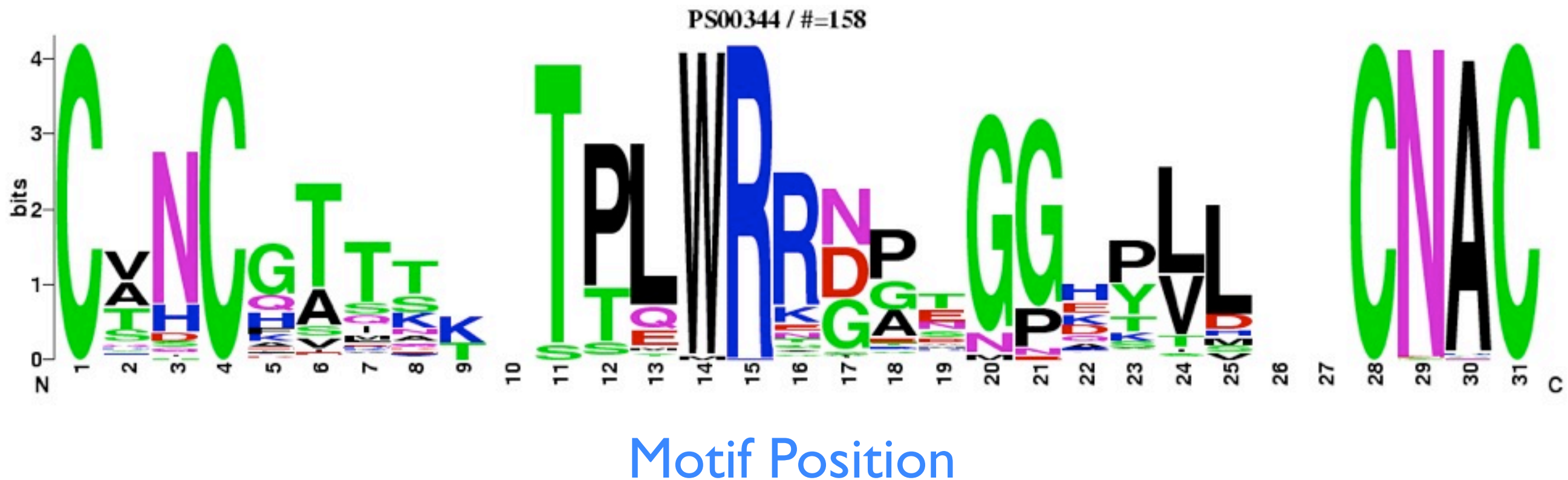
Motif Position



Sequence Logos

Height of letter \approx fraction of time that letter is observed at that position.

(Height of all the letters in a column \approx to how **conserved** the column is)



Gibbs Sampling, Version 1: Pseudocode

Set (x_1, x_2, \dots, x_p) to random positions in each input string.

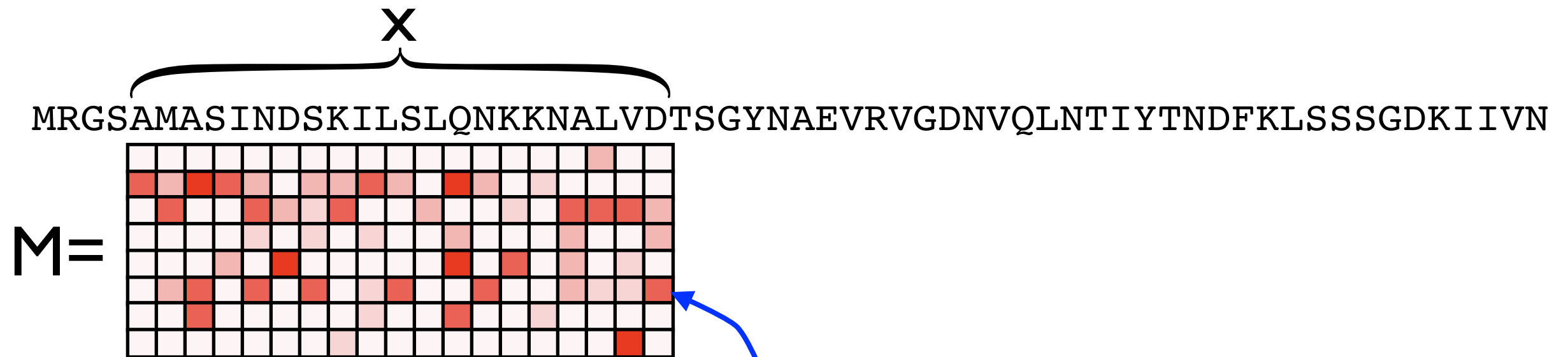
repeat until the answer (x_1, x_2, \dots, x_p) doesn't change

for $i = 1 \dots p$:

 Build a profile Q using sequences at (x_1, x_2, \dots, x_p) except x_i

 Set x_i to where the profile Q matches **best** in string i .

Scoring a Sequence




$$\text{Score}(x) = \Pr(x \mid M) = \prod_{i=1}^L e_i(x_i)$$

Score of a string according to profile M =
Product of the probabilities you would
observe the given letters.

Background Frequencies

Interested in how different this motif position is from we expect by chance.

Correct for “expect by chance” by dividing by the probability of observing x in a random string:

$$\text{ScoreCorrected}(x) = \frac{\Pr(x \mid M)}{\Pr(x \mid \text{background})} = \prod_{i=1}^L \frac{e_i(x_i)}{b(x_i)}$$


$b(x_i) :=$ probability of observing character x_i at random.
Usually computed as (# x_i in entire string) / (length of string)

Often, to avoid multiplying lots of terms, we take the log and then sum:

$$\text{ScoreCorrectedLog}(x) = \log \prod_{i=1}^L \frac{e_i(x_i)}{b(x_i)} = \sum_{i=1}^L \log \left(\frac{e_i(x_i)}{b(x_i)} \right)$$

Gibbs Example

```
gibbs(["thequickdog", "browndog", "dogwood"], k=3)
```

```
1: [8, 1, 2] ['dog', 'row', 'gwo'] ← random starting positions
```

```
2: [8, 5, 0] ['dog', 'dog', 'dog']
```

```
F: [8, 5, 0] ['dog', 'dog', 'dog']
```

Small bias toward “o” in the middle is correct.

```
gibbs(["thequickdog", "browndog", "dogwood"], k=3)
```

```
1: [4, 3, 1] ['uic', 'wnd', 'ogw']
```

```
2: [6, 2, 4] ['ckd', 'own', 'ood']
```

```
3: [8, 5, 0] ['dog', 'dog', 'dog']
```

```
F: [8, 5, 0] ['dog', 'dog', 'dog']
```

```
gibbs(["thequickdog", "browndog", "dogwood"], k=3)
```

```
1: [2, 0, 1] ['equ', 'bro', 'ogw']
```

```
2: [7, 4, 2] ['kdo', 'ndo', 'gwo']
```

```
F: [7, 4, 2] ['kdo', 'ndo', 'gwo'] ←
```

Might not find the optimal.


```

def gibbs(Seqs, k):
    """Seqs is a list of strings. Find the best motif."""

    # start with random indices
    I = [random.randint(0, len(x) - k) for x in Seqs]

    LastI = None
    while I != LastI:        # repeat until nothing changes
        LastI = list(I)

        # iterate through every string
        for i in xrange(len(Seqs)):
            # compute the profile for the sequences except i
            P = profile_for([
                x[j : j + k] for q, (x, j) in enumerate(zip(Seqs, I))
                    if q != i
            ])

            # find the place the profile matches best
            best = None
            for j in xrange(len(Seqs[i]) - k + 1):
                score = profile_score(P, Seqs[i][j : j + k])
                if score > best or best is None:
                    best = score
                    bestpos = j
            # update the ith position with the best
            I[i] = bestpos

    return I, [x[j : j + k] for x, j in zip(Seqs, I)]

```

Another Example

```
gibbs(["aaa123", "678aaa45", "9a7aaab", "32aa19a8aaa"], 3)
1: [0, 5, 0, 2] ['aaa', 'a45', '9a7', 'aa1']
2: [1, 3, 3, 8] ['aa1', 'aaa', 'aaa', 'aaa']
3: [0, 3, 3, 8] ['aaa', 'aaa', 'aaa', 'aaa']
F: [0, 3, 3, 8] ['aaa', 'aaa', 'aaa', 'aaa']
```



Bias toward “a” in the profile
quickly leads to finding the
implanted “aaa”

Can be multiple
optimal answers



```
gibbs(["aaabbb", "bbbaaabb", 'babaaab', 'ababacaaabac', 'abbbababaaabbbaba'], 3)
1: [1, 4, 0, 4, 11] ['aab', 'aab', 'bab', 'aca', 'bbb']
2: [1, 4, 4, 7, 9] ['aab', 'aab', 'aab', 'aab', 'aab']
F: [1, 4, 4, 7, 9] ['aab', 'aab', 'aab', 'aab', 'aab']
gibbs(["aaabbb", "bbbaaabb", 'babaaab', 'ababacaaabac', 'abbbababaaabbbaba'], 3)
1: [0, 3, 3, 3, 8] ['aaa', 'aaa', 'aaa', 'bac', 'aaa']
2: [0, 3, 3, 6, 8] ['aaa', 'aaa', 'aaa', 'aaa', 'aaa']
F: [0, 3, 3, 6, 8] ['aaa', 'aaa', 'aaa', 'aaa', 'aaa']
```


Randomness: Gibbs Sampling

- Run the Gibbs sampling multiple times to make it more likely you find the global optimal.
- Can increase the use of randomness to further avoid getting stuck in local optima by choosing new x_i randomly.

Set (x_1, x_2, \dots, x_p) to random positions in each input string.

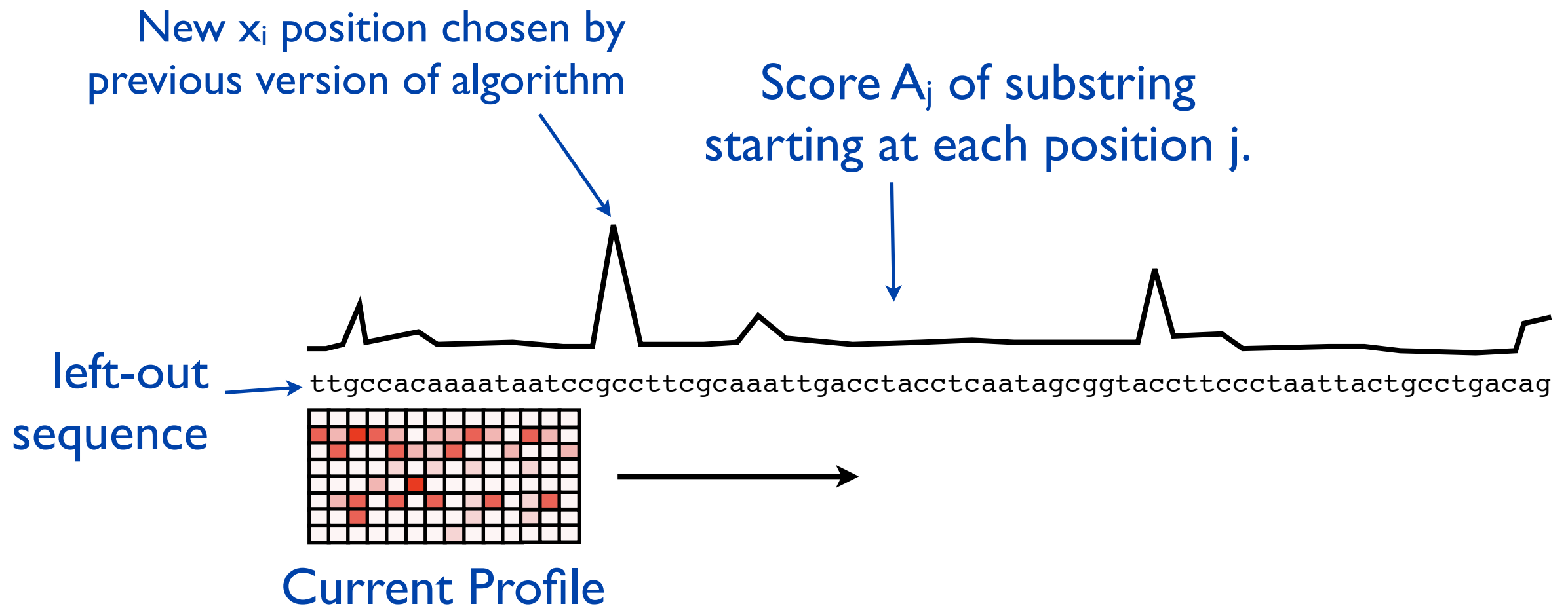
repeat until the best (x_1, x_2, \dots, x_p) doesn't change too often

for $i = 1 \dots p$:

Build a profile Q using sequences at (x_1, x_2, \dots, x_p) except x_i

Choose x_i according to the profile probability distribution of Q in string i .

Profile Probability Distribution



Instead of choosing the position with the best match,
choose a position randomly such that:

$$\text{Probability of choosing position } j = \frac{A_j}{\sum_i A_i}$$

(Lawrence, et al., *Science*, 1994)

Recap

- “Motif finding” is the problem of finding a set of common substrings within a set of strings.
- Useful for finding transcription factor binding sites.
- **Gibbs sampling:** repeatedly leave one sequence out and optimize the motif location in the left-out sequence.
- Doesn't guarantee finding a good solution, but often works.

Over successive iterations, motifs that better fit the consensus (high-probability regions) become more prevalent because they are more likely to be selected.