Motif Finding

CMSC 423

Motif Finding

Transcription factor

1. ttgccacaaaataatccgccttcgcaaattgaccTACCTCAATAGCGGTAgaaaaacgcaccactgcctgacag
2. gtaagtacctgaaagttacggtctgcgaacgctattccacTGCTCCTTTATAGGTAcaacagtatagtctgatgga
3. ccacacggcaaataaggagTAACTCTTTCCGGGTAtgggtatacttcagccaatagccgagaatactgccattccag
4. ccatacccggaaagagttactccttatttgccgtgtggttagtcgcttTACATCGGTAAGGGTAgggattttacagca
5. aaactattaagatttttatgcagatgggtattaaggaGTATTCCCCATGGGTAacatattaatggctctta
6. ttacagtctgttatgtggtggctgttaaTTATCCTAAAGGGGTAtcttaggaatttactt

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

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

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.

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

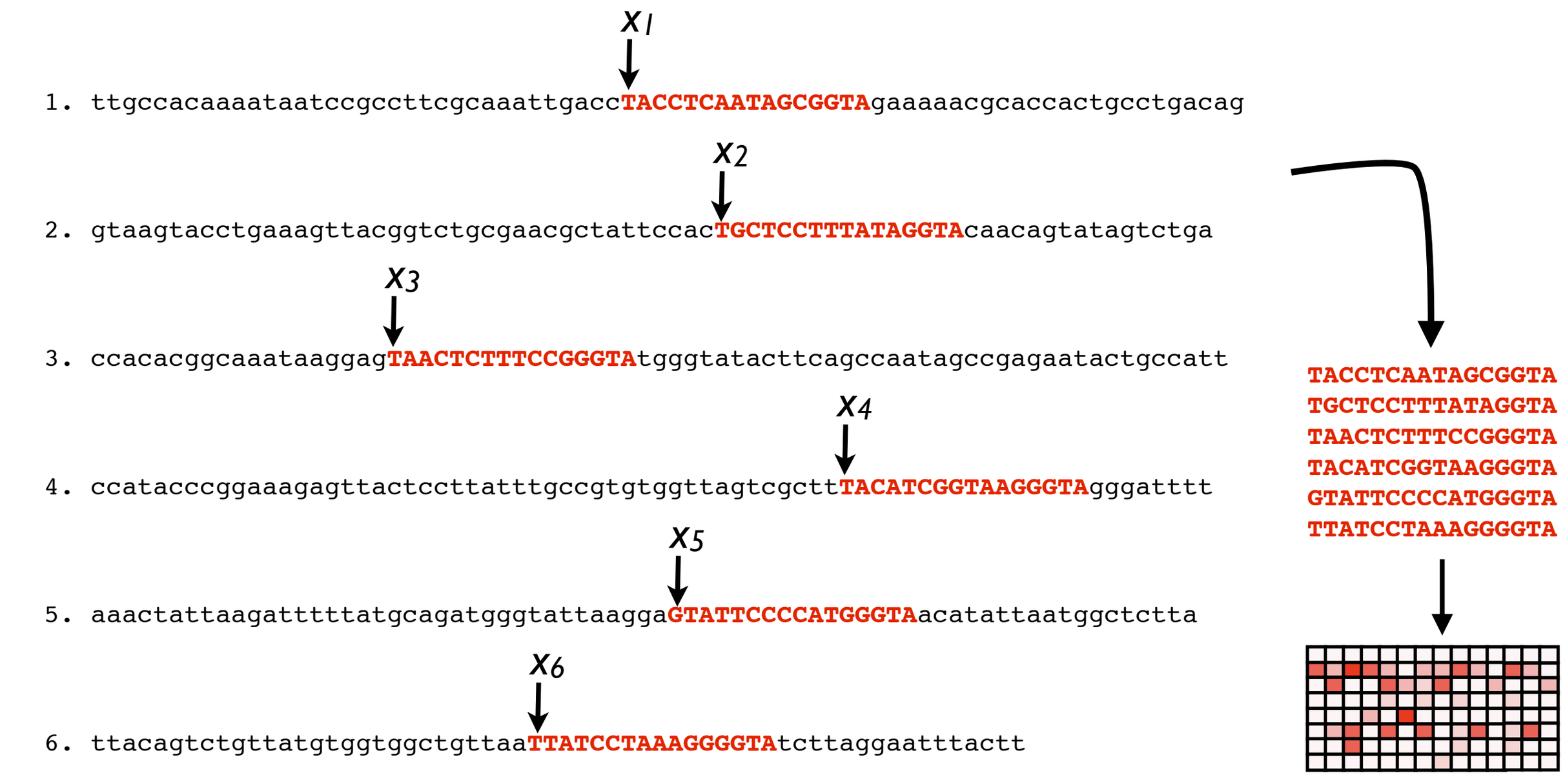
It doesn't guarantee good performance, but often works well in practice.

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:



Gibbs Sampling, Version 1: Pseudocode

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

repeat until the answer $(x_1, x_2, ..., x_p)$ doesn't change

for i = 1 ... p:

Build a profile Q using sequences at $(x_1, x_2, ..., x_p)$ except x_i

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

```
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)]
```

Gibbs Example

```
gibbs(["thequickdog", "browndog", "dogwood"], k=3)
1: [8, 1, 2] ['dog', 'row', 'gwo']
2: [8, 5, 0] ['dog', 'dog', 'dog']
F: [8, 5, 0] ['dog', 'dog', 'dog']
```

random starting positions

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.

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 $(x_1, x_2, ..., x_p)$ to random positions in each input string.

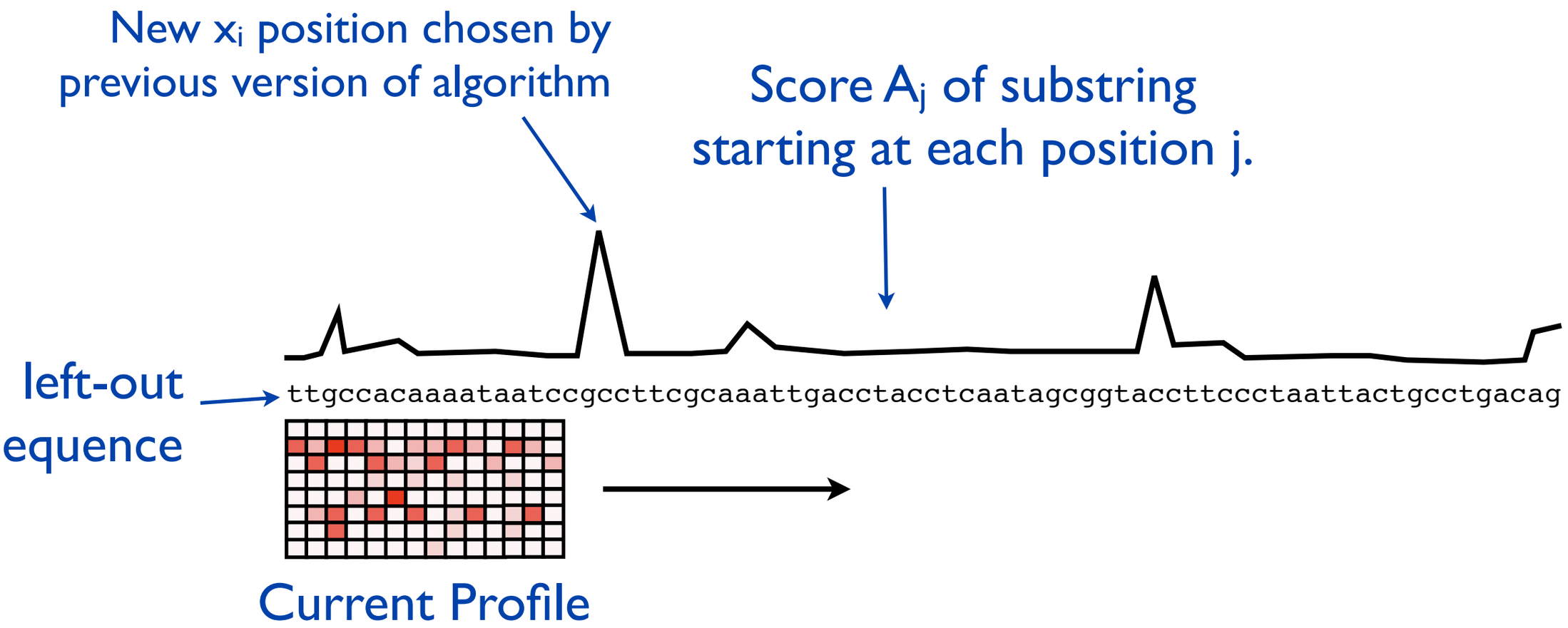
repeat until the best $(x_1, x_2, ..., x_p)$ doesn't change too often

    for i = 1 ... p:

        Build a profile Q using sequences at $(x_1, x_2, ..., x_p)$ 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.