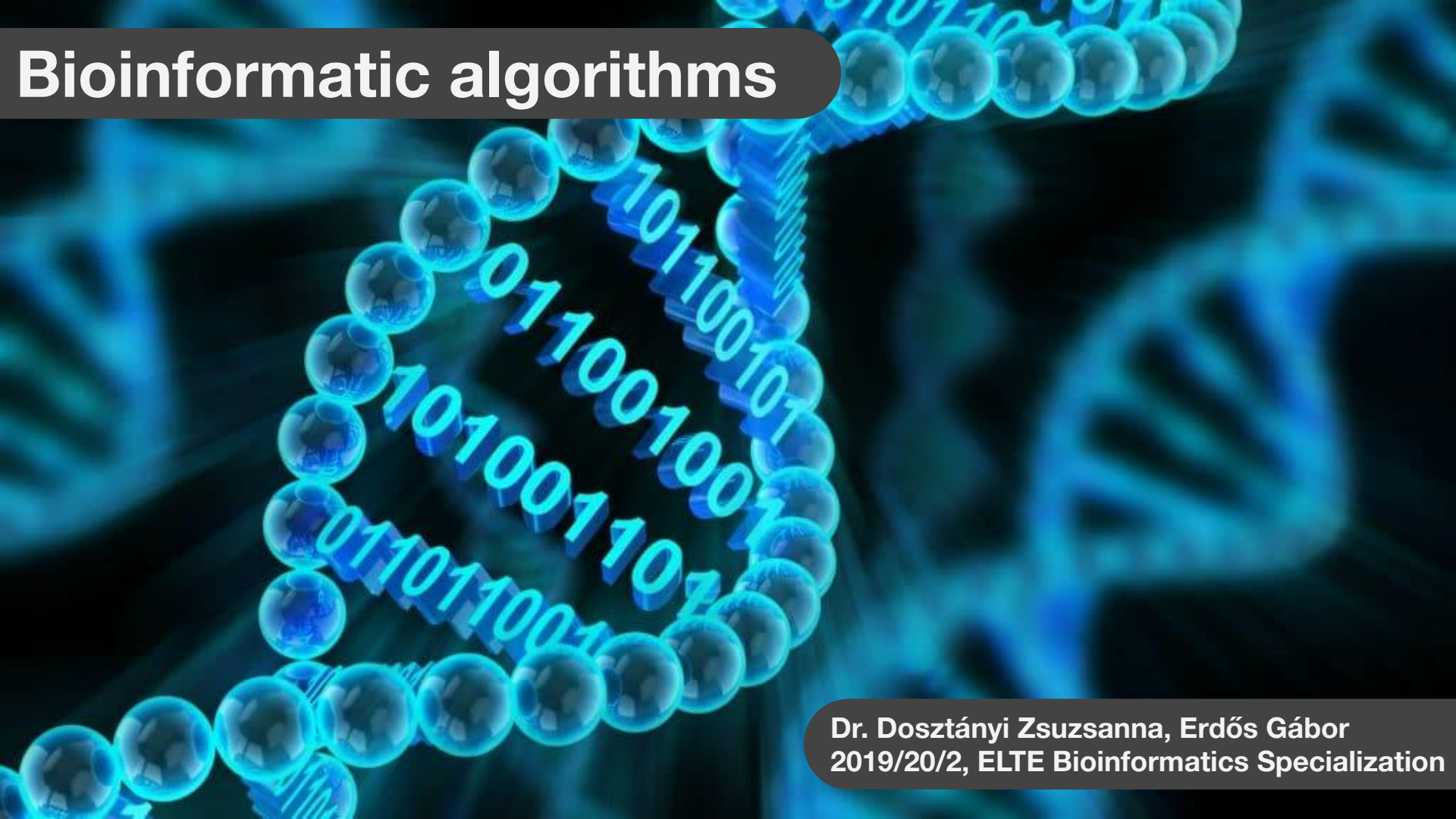
Bioinformatic algorithms

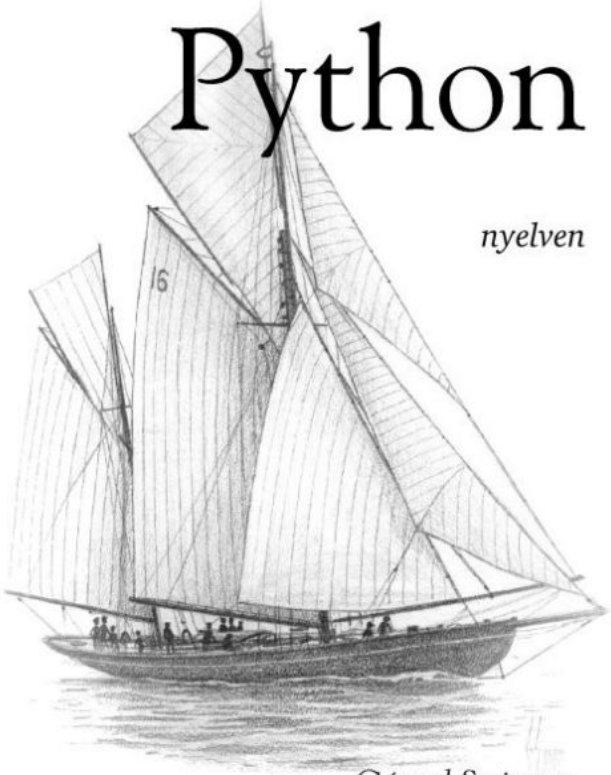


Dr. Dosztányi Zsuzsanna, Erdős Gábor
2019/20/2, ELTE Bioinformatics Specialization

Tanuljunk meg programozni

Python

nyelven



Gérard Swinnen

Allen B. Downey, Jeffrey Elkner & Chris Meyers
"How to think like a computer scientist" művének szabad adaptációja

Profile matrices

Lets consider the second and the last column. They both contribute 4 to the final score.

Do they have equal contribution?

Entropy is a measure of the uncertainty of a probability distribution ($p1$, ..., pN), and is defined as follows:

$$H(p_1, \ldots, p_N) = -\sum_{i=1}^{N} p_i \cdot \log_2 p_i$$

Note: Technically, $\log_2(0)$ is undefined, but in the computation of entropy, we assume that $0 \cdot \log_2(0)$ is equal to 0.

Calculate the entropy of each column

Motifs	T	C	G	G	G	G	g	T	T	T	t	t
	c	C	G	G	t	G	A	c	T	T	a	C
	a	C	G	G	G	G	A	T	T	T	t	C
	T	t	G	G	G	G	A	c	T	T	t	t
	a	a	G	G	G	G	A	c	T	T	C	C
	T	t	G	G	G	G	A	c	T	T	C	C
	T	C	G	G	G	G	A	T	T	c	a	t
	T	C	G	G	G	G	A	T	T	c	C	t
	T	a	G	G	G	G	A	a	c	T	a	C
	T	C	G	G	G	t	A	T	a	a	C	C
Score(Motifs)	3	+ 4	+ 0	+ 0	+ 1	+ 1	+ 1	+ 5	+ 2	+ 3	+ 6	+ 4 = 30
Count(Motifs)	A:	2	2	0	0	0	0	9	1	1	1	3	0
	C:	1	6	0	0	0	0	0	4	1	2	4	6
	G:	0	0	10	10	9	9	1	0	0	0	0	0
	T:	7	2	0	0	1	1	0	5	8	7	3	4
Profile(Motifs)	A:	.2	.2	0	0	0	0	.9	.1	.1	.1	.3	0
	C:	.1	.6	0	0	0	0	0	.4	.1	.2	.4	.6
	G:	0	0	1	1	.9	.9	.1	0	0	0	0	0
	T:	.7	.2	0	0	.1	.1	0	.5	.8	.7	.3	.4
Consensus(Motifs)	T	C	G	G	G	G	A	T	T	T	C	C

Profile matrices

Given a profile matrix *Profile*, we can evaluate the probability of every *k*-mer in a string *Text* and find a ***Profile*-most probable** *k*-mer in *Text*, i.e., a *k*-mer that was most likely to have been generated by *Profile* among all *k*-mers in *Text*. For example, **ACGGGGATTACC** is the *Profile*-most probable 12-mer in GGT**ACGGGGATTACC**T. Indeed, every other 12-mer in this string has probability 0. In general, if there are multiple *Profile*-most probable *k*-mers in *Text*, then we select the first such *k*-mer occurring in *Text*.

***Profile*-most Probable *k*-mer Problem:** *Find a Profile-most probable k-mer in a string.*

- **Input:** A string *Text*, an integer *k*, and a 4 × *k* matrix *Profile*.
- **Output:** A *Profile*-most probable *k*-mer in *Text*.

Motifs	T	C	G	G	G	G	g	T	T	T	t	t
	c	C	G	G	t	G	A	c	T	T	a	C
	a	C	G	G	G	G	A	T	T	T	t	C
	T	t	G	G	G	G	A	c	T	T	t	t
	a	a	G	G	G	G	A	c	T	T	C	C
	T	t	G	G	G	G	A	c	T	T	C	C
	T	C	G	G	G	G	A	T	T	c	a	t
	T	C	G	G	G	G	A	T	T	c	C	t
	T	a	G	G	G	G	A	a	c	T	a	C
	T	C	G	G	G	t	A	T	a	a	C	C
Score(Motifs)	3	+ 4	+ 0	+ 0	+ 1	+ 1	+ 1	+ 5	+ 2	+ 3	+ 6	+ 4 = 30
Count(Motifs)	A:	2	2	0	0	0	0	9	1	1	1	3	0
	C:	1	6	0	0	0	0	0	4	1	2	4	6
	G:	0	0	10	10	9	9	1	0	0	0	0	0
	T:	7	2	0	0	1	1	0	5	8	7	3	4
Profile(Motifs)	A:	.2	.2	0	0	0	0	.9	.1	.1	.1	.3	0
	C:	.1	.6	0	0	0	0	0	.4	.1	.2	.4	.6
	G:	0	0	1	1	.9	.9	.1	0	0	0	0	0
	T:	.7	.2	0	0	.1	.1	0	.5	.8	.7	.3	.4
Consensus(Motifs)	T	C	G	G	G	G	A	T	T	T	C	C

Laplace's Rule of Succession

Profile

A:	.2	.2	.0	.0	.0	.0	.9	.1	.1	.1	.3	.0
C:	.1	.6	.0	0	.0	.0	.0	.4	.1	.2	.4	.6
G:	.0	.0	1	1	.9	.9	.1	.0	.0	.0	.0	.0
T:	.7	.2	.0	.0	.1	.1	.0	.5	.8	.7	.3	.4

Pr("TCGTGGATTTCC", Profile) = .7 · .6 · 1 · .0 · .9 · .9 · .9 · .5 · .8 · .7 · .4 · .6 = 0

The fourth symbol of TCGTGGATTTCC causes Pr(TCGTGGATTTCC, *Profile*) to be equal to zero. As a result, the entire string is assigned a zero probability, even though TCGTGGATTTCC differs from the consensus string at only one position. For that matter, TCGTGGATTTCC has the same low probability as AAATCTTGGAA, which is very different from the consensus string.

Motifs

T	A	A	C
G	T	C	T
A	C	T	A
A	G	G	T

Count(Motifs)

A:	2	1	1	1
C:	0	1	1	1
G:	1	1	1	0
T:	1	1	1	2

Profile(Motifs)

2/4	1/4	1/4	1/4
0	1/4	1/4	1/4
1/4	1/4	1/4	0
1/4	1/4	1/4	2/4

Count(Motifs)

A:	2+1	1+1	1+1	1+1
C:	0+1	1+1	1+1	1+1
G:	1+1	1+1	1+1	0+1
T:	1+1	1+1	1+1	2+1

Profile(Motifs)

3/8	2/8	2/8	2/8
1/8	2/8	2/8	2/8
2/8	2/8	2/8	1/8
2/8	2/8	2/8	3/8

Randomized motif search

- **Randomized algorithms** that flip coins and roll dice in order to search for solutions
- Randomized algorithms may be non intuitive because they lack the control of traditional algorithms.
    - **Las Vegas algorithms:** deliver solutions that are guaranteed to be exact
    - **Monte Carlo algorithms:** not guaranteed to return exact solutions, but they do quickly find *approximate* solutions. Because of their speed, they can be run many times, allowing us to choose the best approximation from thousands of runs.

Randomized motif search

Let's create two helper functions first

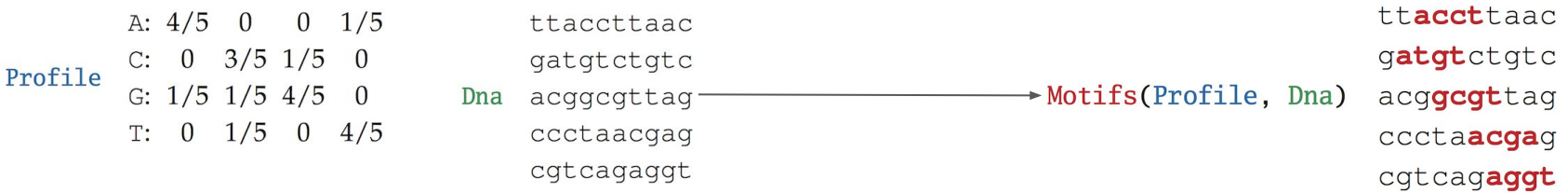
Consensus: The consensus motif (most frequent nucleotide in each column)

Score: The sum of Hamming distances from consensus for each motif in the alignment

Motifs	T C G G G G g T T T t t c C G G t G A c T T a C a C G G G G A T T T t C T t G G G G A c T T t t a a G G G G A c T T C C T t G G G G A c T T C C T C G G G G A T T c a t T C G G G G A T T c C t T a G G G G A a c T a C T C G G G t A T a a C C
Score(Motifs)	3 + 4 + 0 + 0 + 1 + 1 + 1 + 5 + 2 + 3 + 6 + 4 = 30
Consensus(Motifs)	T C G G G G A T T T C C

Randomized motif search

We define *Motifs*(*Profile*, *Dna*) as the collection of *k*-mers formed by the *Profile*-most probable *k*-mers in each string from *Dna*.



We can begin from a collection of randomly chosen k-mers *Motifs* in *Dna*, construct *Profile*(*Motifs*), and use this profile to generate a new collection of *k*-mers:

Motifs(*Profile*(*Motifs*), *Dna*).

We hope that *Motifs*(*Profile*(*Motifs*), *Dna*) has a better score than the original collection of k-mers *Motifs*. Then form the profile matrix of these *k*-mers,

Profile(*Motifs*(*Profile*(*Motifs*), *Dna*))

and use it to form the most probable *k*-mers,

Motifs(*Profile*(*Motifs*(*Profile*(*Motifs*), *Dna*)), *Dna*).

We can continue to iterate. . .

...*Profile*(*Motifs*(*Profile*(*Motifs*(*Profile*(*Motifs*), *Dna*)), *Dna*))...

Randomized motif search

Why should this work?

If the nucleotides are random, their distribution will be uniform, thus the profile we generate is a uniform profile

In uniformly distributed nucleotides every segment has the same score, so this will lead us nowhere. However if there is some bias in the distribution (motifs) the profile will also show some bias and hopefully will lead us iteratively to the source of the bias, the motif

A:	0.25	0.25	0.25	0.25
C:	0.25	0.25	0.25	0.25
G:	0.25	0.25	0.25	0.25
T:	0.25	0.25	0.25	0.25

The 4-mer **AGGT** in the last string happened to capture the implanted motif simply by chance. In fact, the profile formed from the remaining 4-mers (**taac**, **GTct**, **ccgG**, and **acta**) is uniform. Note that only completely captured motifs (like **AGGT**) rather than partially captured motifs (like **GTct** or **ccgG**) contribute to the statistical bias in the profile matrix.

	ttACCT**taac**
	gAT**GTct**gtc
Dna	**ccgG**CGTtag
	c**acta**ACGAg
	cgtcag**AGGT**

What is the probability that at least one of ten randomly selected 15-mers from a 600-nucleotide long strings matches an implanted motif?

Randomized motif search

```
RandomizedMotifSearch(Dna, k)
        randomly select k-mers Motifs = (Motif1, …, Motift) in each string from Dna
        BestMotifs ← Motifs
        while forever
            Profile ← Profile(Motifs)
            Motifs ←ProfileMostProbableKmer(Profile, Dna) for each string in Dna
            if Score(Motifs) < Score(BestMotifs)
                BestMotifs ← Motifs
            else
                return BestMotifs
```

Sample Input:

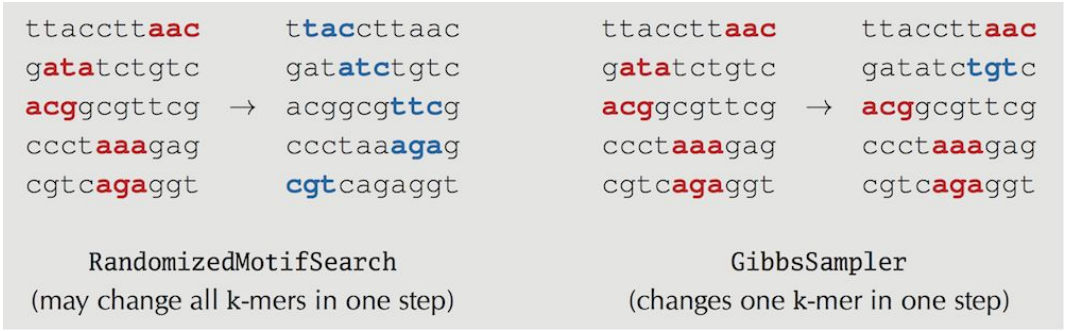
```
8
CGCCCCTCTCGGGGGTGTTCAGTAAACGGCCA
GGGCGAGGTATGTGTAAGTGCCAAGGTGCCAG
TAGTACCGAGACCGAAAGAAGTATACAGGCGT
TAGATCAAGTTTCAGGTGCACGTCGGTGAACC
AATCCACCAGCTCCACGTGCAATGTTGGCCTA
```

Sample Output:

```
TCTCGGGG
CCAAGGTG
TACAGGCG
TTCAGGTG
TCCACGTG
```

Gibbs sampling

RandomizedMotifSearch may change all *t* strings *Motifs* in a single iteration. This strategy may prove reckless, since some correct motifs (captured in *Motifs*) may potentially be discarded at the next iteration. **GibbsSampler** is a more cautious iterative algorithm that discards a single *k*-mer from the current set of motifs at each iteration and decides to either keep it or replace it with a new one.




```
GibbsSampler(Dna, k, t, N)
        randomly select k-mers Motifs = (Motif1, …, Motift) in each string from Dna
        BestMotifs ← Motifs
        for j ← 1 to N
            i ← Random(t)
            Profile ← profile matrix constructed from all strings in Motifs except for Motifi
            Motifi ← Profile-most-probable k-mer in the i-th sequence
            if Score(Motifs) < Score(BestMotifs)
                BestMotifs ← Motifs
        return BestMotifs
```

Genome sequencing

Multiple identical copies of a genome

Shatter the genome into reads

Sequence the reads

Assemble the genome using overlapping reads

AGAATATCA
GAGAATATC
TGAGAATAT
...TGAGAATATCA...

Genome assembly

One of the (partially) unsolved problems of modern biology. We are able to discover the entire genome various species, but for example *Amoeba dubia* a creature with one of the longest genome is still a mystery.

It is “easy” experimentally, we are able to generate the sequence reads

Genome assembly is extremely difficult (computationally)

First, DNA is double-stranded, and we have no way of knowing *a priori* which strand a given read derives from. Second, modern sequencing machines are not perfect, and the reads that they generate often contain errors. Sequencing errors complicate genome assembly because they prevent us from identifying all overlapping reads. Third, some regions of the genome may not be covered by any reads, making it impossible to reconstruct the entire genome.

Since the reads generated by modern sequencers often have the same length, we may safely assume that reads are all *k*-mers for some value of *k*. The first part will assume an ideal situation in which all reads come from the same strand, have no errors, and exhibit **perfect coverage**.

Genome assembly

String Composition Problem: *Generate the k-mer composition of a string.*
Input: An integer *k* and a string *Text*.
Output: *Composition*$_k$(*Text*), where the *k*-mers are arranged in lexicographic order.

In order to solve the genome assembly problem we need the inverse of this straightforward function

AAT ATG GTT TAA TGT

```
TAA
 AAT
  ATG
   TGT
    GTT
TAATGTT
```

~~AAT~~ ATG ~~ATG~~ ATG CAT CCA GAT GCC
GGA GGG ~~GTT~~ ~~TAA~~ TGC TGG ~~TGT~~

```
TAA
 AAT
  ATG
   TGT
    GTT
TAATGTT
```

Repeats

The difficulty in assembling this simulated genome arises because ATG is *repeated* three times in the 3-mer composition, which causes us to have the three choices TGG, TGC, and TGT by which to extend ATG. Repeated substrings in the genome are a serious problem when we have millions of reads

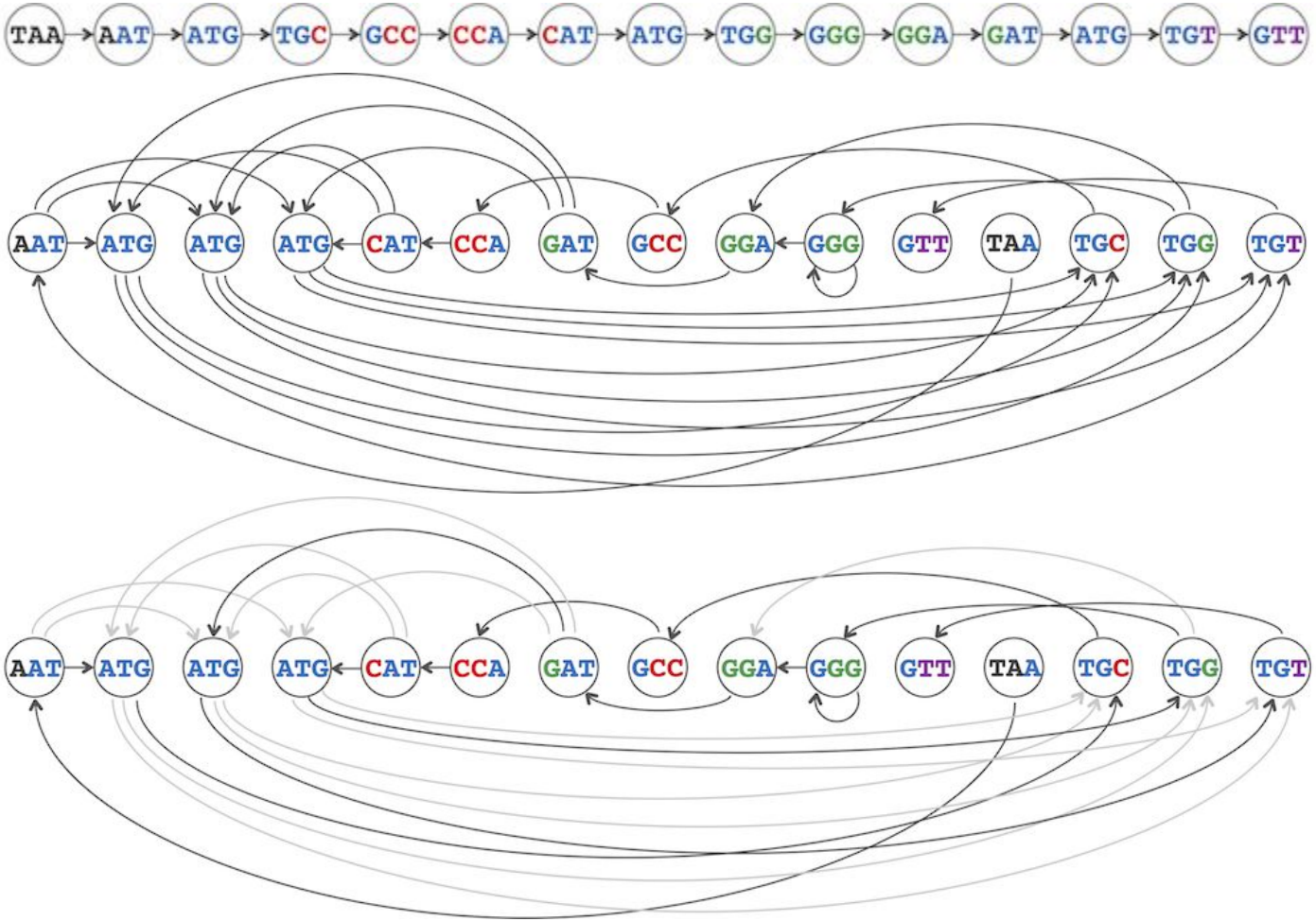
Let's take a step back for a second and see the investigate the reverse problem

String Spelled by a Genome Path Problem. *Reconstruct a string from its genome path.*

Input: A sequence *path* of *k*-mers $Pattern_1, \ldots, Pattern_n$ such that the last k - 1 symbols of $Pattern_i$ are equal to the first k-1 symbols of $Pattern_{i+1}$ for $1 \le i \le n$-1.

Output: A string *Text* of length $k+n-1$ such that the i-th k-mer in *Text* is equal to $Pattern_i$ (for $1 \le i \le n$).

Graphs


TAA AAT ATG TGC GCC CCA CAT ATG TGG GGG GGA GAT ATG TGT GTT
AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT
AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT

The Overlap Graph Problem

Solve The Overlap Graph Problem

Input: A collection *Patterns* of *k*-mers.

Output: The overlap graph *Overlap*(*Patterns*), in the form of an adjacency list.

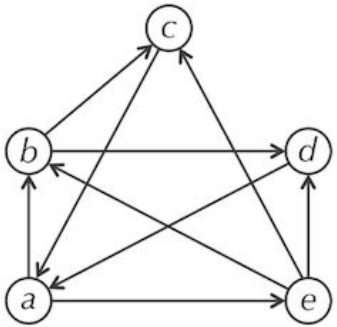
Sample Input:

```
ATGCG
GCATG
CATGC
AGGCA
GGCAT
GGCAC
```

Sample Output:

```
CATGC -> ATGCG
GCATG -> CATGC
GGCAT -> GCATG
AGGCA -> GGCAC,GGCAT
```

Graph



Adjacency Matrix

	a	*b*	*c*	*d*	*e*
a	0	1	0	0	1
b	0	0	1	1	0
c	1	0	0	0	0
d	1	0	0	0	0
e	0	1	1	1	0

Adjacency List

a is adjacent to *b* and *e*

b is adjacent to *c* and *d*

c is adjacent to *a*

d is adjacent to *a*

e is adjacent to *b*, *c*, and *d*