PyCon PL
2018-08-24

Living with Python

how unravel the mystery of life

Jacek Śmietański

About me

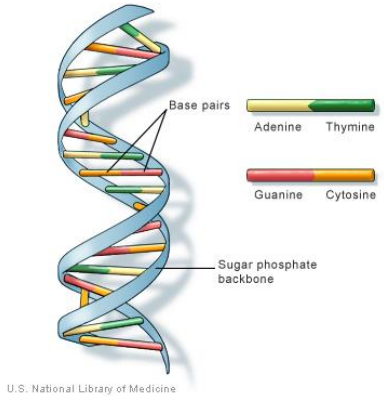
* Data scientists
* Bioinformatitian
* Python developer

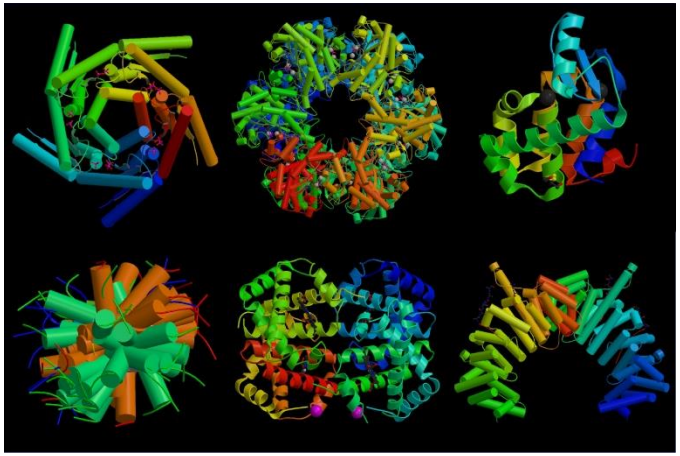
Epam Systems – data science for industry
Jagiellonian University – bioinformatics, lectures, classes

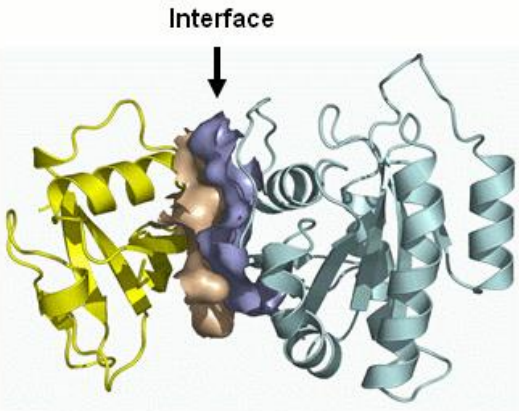
Agenda

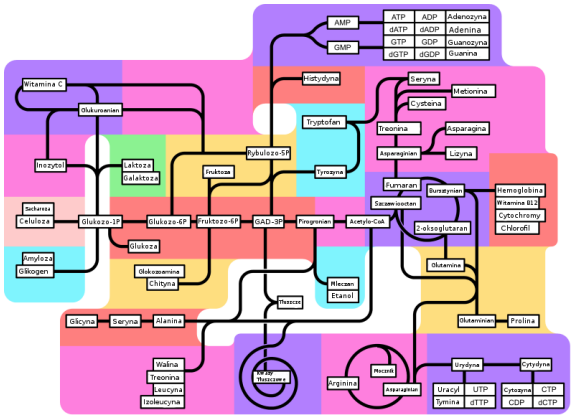
- Bioinformatics
  - data (types, sources, availability, reliability)
  - main problems and common tasks
- Biopython
  - introduction to library
  - examples: sequence analysis, pairwise alignment, working with databases
- Rosalind
  - idea
  - examples: DNA composition, Hamming distance, reverse complement

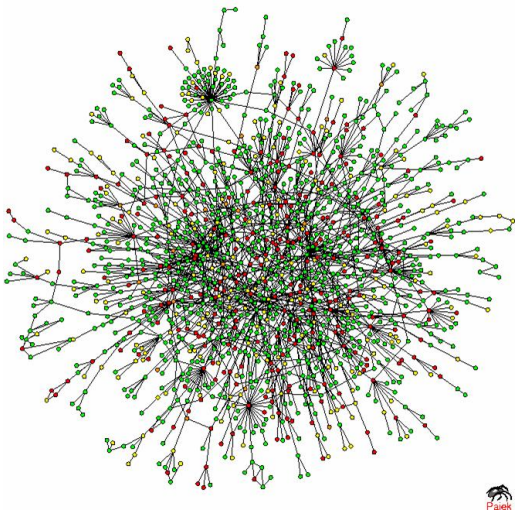
Biological data

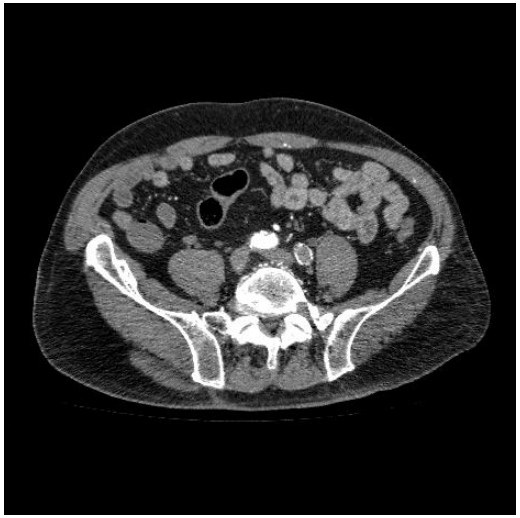




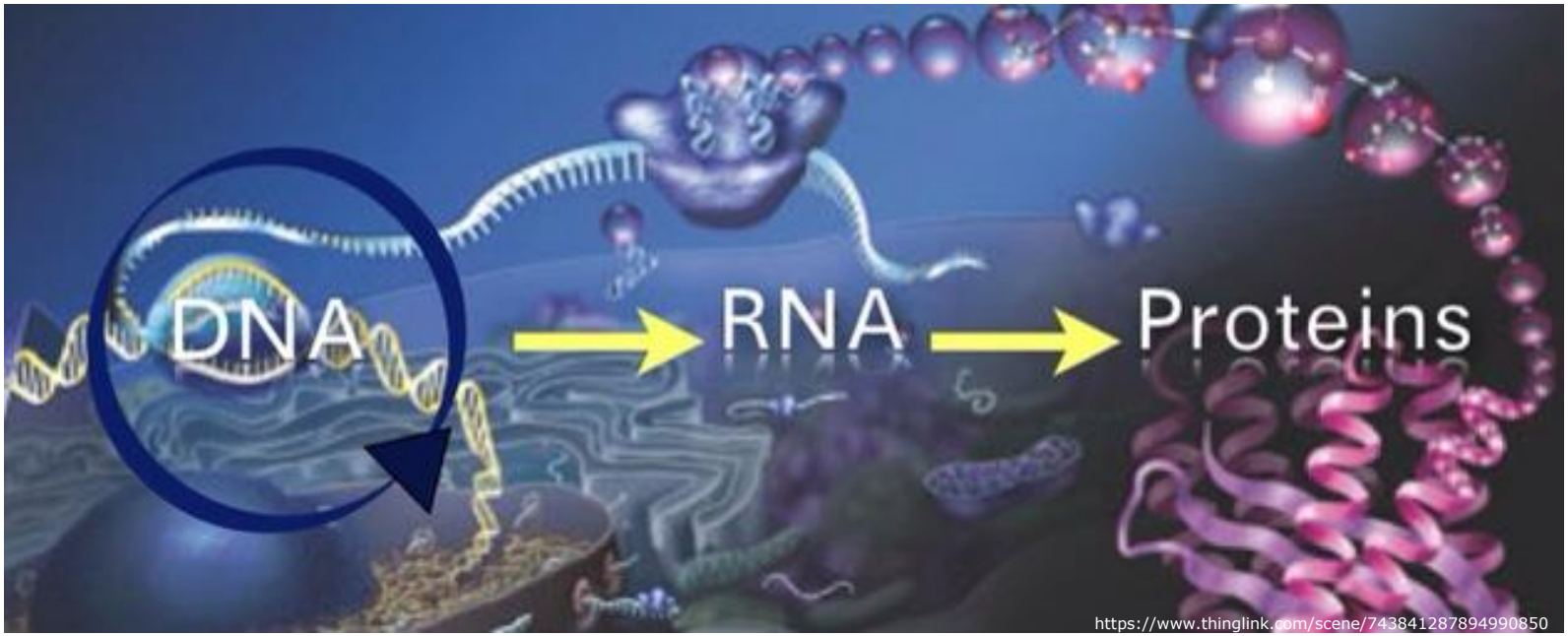








Information flow in cell



Sequence → Structure → Function

Sequences

> AY169899.1 Morelia viridis strain ABTC66386 cytochrome b gene, partial cds; mitochondrial gene for mitochondrial product

TTCGGCTCAATATTATTAACATGTTTAGCCCTACAAGTACTA
CCGGCTTCTTCTTAGCCGTCCACTACACAGCAAACATCAAC
CTAGCATTCTCATCCATTATCCATATCACTCGAGATGTCCCA
TACGGCTGAATAATACAAAACCTACACGCCATCGGAGCATC
CATATTCTTCATTTGCATTTACATCCACATCGCACGAGGACT
ATACTACGGATCCTACCTCAACAAAGAGACTTGAATATCCG
GTATCACCCTACTCATCACATTAATAGCAACCGCCTTCTTTG

Data availability

NCBI Entrez:

https://www.ncbi.nlm.nih.gov/search

GenBank

Swiss Prot

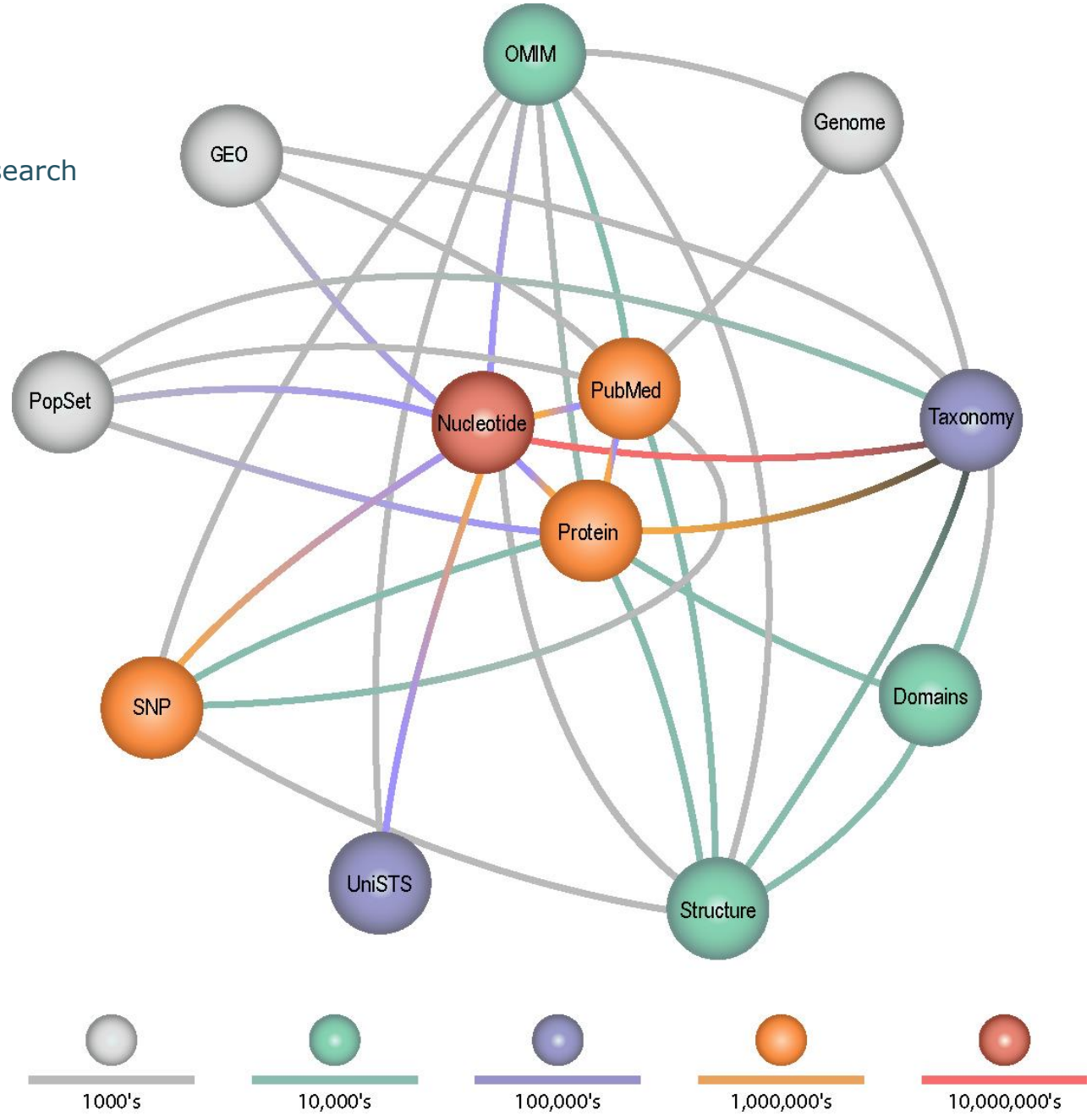
Protein Data Bank

PubMed

OMIM

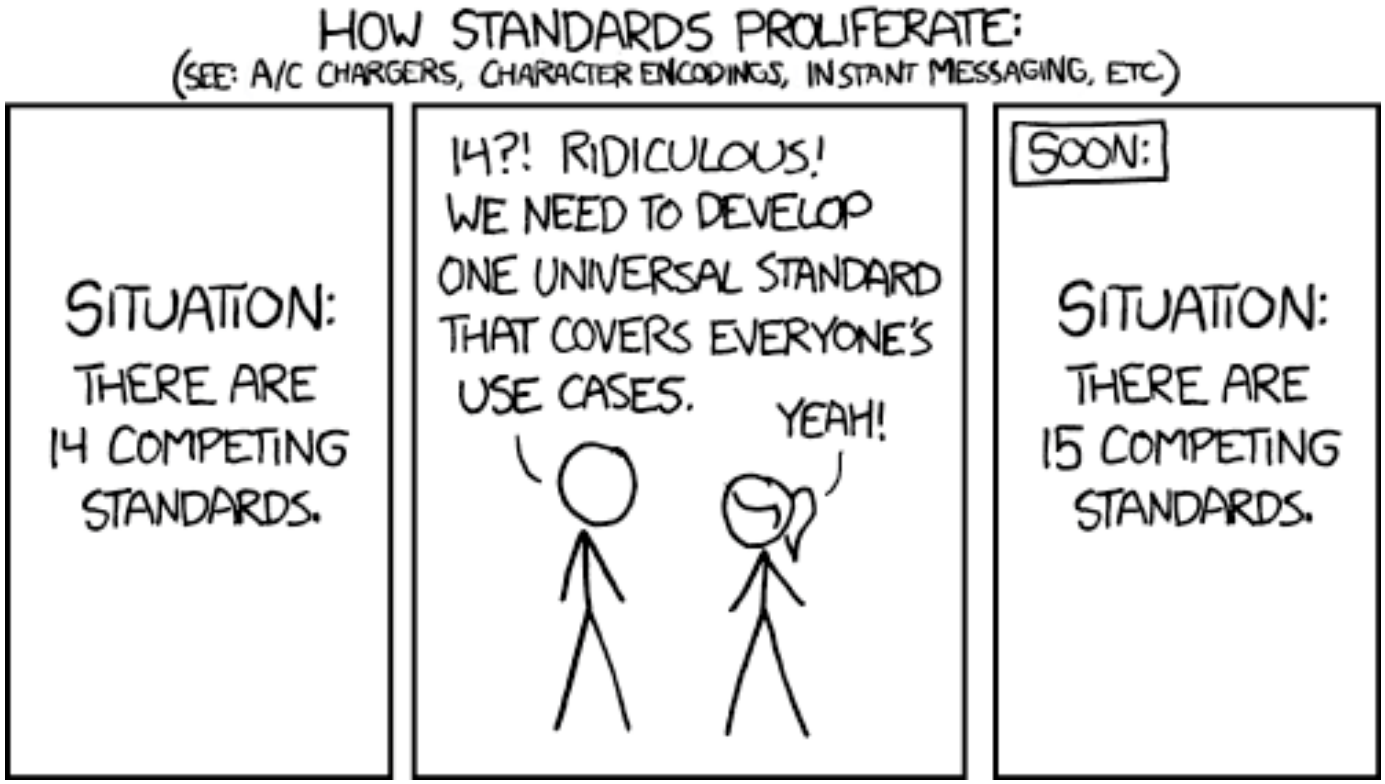
SNP

(...)



Main issues

- Data corectness and reliability.
- Inconsistency of data formats.
- Computational complexity.



Open-source library for computational biology and bioinformatics.

https://biopython.org/
https://github.com/biopython/biopython

Biopython – main functions

- Parse bioinformatics files, including the following formats:
  Blast output, ClustalW, FASTA, GenBank, PubMed
  and Medline, ExPASy files, SCOP, UniGene, SwissProt, PDB
- Files in the supported formats can be iterated over record by record
  or indexed and accessed via a dictionary interface.
- Deal with popular on-line bioinformatics databases and tools:
  NCBI (Blast, Entrez, PubMed), ExPASy (Swiss-Prot, Prosite).
- Interfaces to common bioinformatics programs:
  Standalone Blast, Clustalw, EMBOSS.
- A standard sequence class that deals with sequences.

Biopython – main functions (2)

- Tools for performing common operations on sequences (translation, transcription, weight calculations).
- Perform data classification using k-Nearest Neighbors, Naive Bayes and Support Vector Machines.
- Deal with alignments, including a standard way to create and deal with substitution matrices.
- Split up parallelizable tasks into separate processes.
- GUI-based programs to do basic sequence manipulations, translations, BLASTing, etc.
- Integration with BioSQL, a sequence database schema also supported by the BioPerl and BioJava projects.
- Deal with structural data.

Sequence and alphabet

```
from Bio.Seq import Seq
from Bio.Alphabet import IUPAC

sequenceN = Seq("GATCGATGGGCCTATATAGGATCGAAAATCGC",
                IUPAC.unambiguous_dna)
print(sequenceN.alphabet)
IUPACUnambiguousDNA()

 sequenceAA = Seq("GATCGATGGGCCTATATAGGATCGAAAATCGC",
                  IUPAC.protein)
 print(sequenceAA.alphabet)
IUPACProtein()

 # You cannot join different sequence types:
 print(sequenceN + sequenceAA)
 TypeError: Incompatible alphabets IUPACUnambiguousDNA()
 and IUPACProtein()
```

Translation

```
from Bio.Seq import Seq
from Bio.Alphabet import IUPAC
coding_dna = Seq("ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCG",
                 IUPAC.unambiguous_dna)
print(coding_dna, coding_dna.alphabet)

# Transcription
messenger_rna = coding_dna.transcribe()

# Translation from RNA
protein = messenger_rna.translate()

# Translation from DNA
protein2 = coding_dna.translate()

print(messenger_rna, messenger_rna.alphabet)
print(protein, protein.alphabet)
print(protein2, protein2.alphabet)
```

Working with on-line databases

```
# Which databases are available via Entrez?

from Bio import Entrez
Entrez.email = your@email
handle = Entrez.einfo()
result_xml = handle.read()
print(result_xml)  # XML

handle = Entrez.einfo()
result_dict = Entrez.read(handle)
print(result_dict)  # python dictionary
```

Working with on-line databases

```
# Search in database
handle = Entrez.esearch(db="nucleotide",
                        term="swine influenza",
                        retmax="10")
result = Entrez.read(handle)
print(result["IdList"])


# Fetch record
handle = Entrez.efetch(db="nucleotide",
                       id="326579398",
                       rettype="gb",
                       retmode="text")
print(handle.read())
```

Pairwise alignment

```
from Bio import pairwise2
from Bio import SeqIO

sequences = list(SeqIO.parse("MHC.fa", "fasta"))
alignments = pairwise2.align.globalxx(sequences[0].seq,
                                      sequences[1].seq)
print(pairwise2.format_alignment(*alignments[0]))
```

```
MVDGTLL---LL-LSEAL--ALTQ--TWAG-S--HSLK-YFHTS--VSRPGR-GEPRFIS-VGYVDDTQFVRFDN
|    ||   || |   |  | |   |    |  |||  || |   |||||  ||||||  ||||||||||||
M----LLFAHLLQL---LVSA-T-VPT---QSSPHSL-RYF-T-TAVSRPG-LGEPRFI-IVGYVDDTQFVRFD- (...)
Score=245
```

```
from Bio import SeqIO

result_handle = NCBIWWW.qblast("blastn", "nr", "23527284")
print(result_handle.read())
s = SeqIO.read(open("sequence.fa"), format="fasta")
result_handle = NCBIWWW.qblast("blastn", "nt", s.seq)
```

Databases:

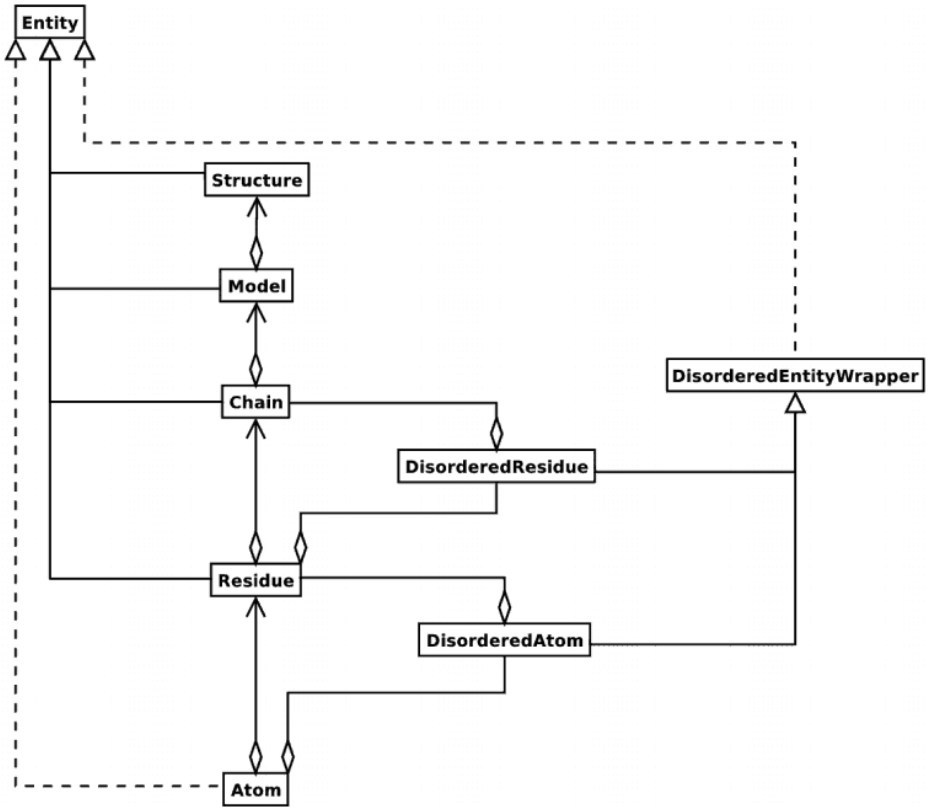
nr – non-redundant, protein sequences

nt – non-redundant, nucleotide sequences

```
# Parse XML
from Bio.Blast import NCBIXML
blast_records = NCBIXML.parse(result_handle)
blast_record = next(blast_records)
```

```
from Bio.PDB.PDBParser import PDBParser
parser = PDBParser()
structure = parser.get_structure("PHYTOHEMAGGLUTININ-L",
                                 "1fat.pdb")

model = structure[0]
chain = model["A"]
residue = chain[1]
atom = residue["CA"]
```



„An Addictive Bioinformatics Learning Site”

http://rosalind.info/

Features

- Learning bioinformatics through problem solving.
- Short problems with biological background.
- Real challenges from molecular biology.
- Solutions are checked automatically.
- Quick gratification (badges).
- Disscusion and solutions compare.

Python Village

If you are completely new to programming, try these initial problems to learn a few basics about the Python programming language. You'll get familiar with the operations needed to start solving bioinformatics challenges in the Stronghold.

Bioinformatics Stronghold

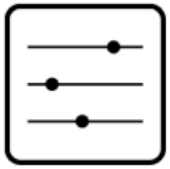
Discover the algorithms underlying a variety of bioinformatics topics: computational mass spectrometry, alignment, dynamic programming, genome assembly, genome rearrangements, phylogeny, probability, string algorithms and others.

Bioinformatics Armory

Ready-to-use software tools abound for bioinformatics analysis. Whereas in the Stronghold you implement algorithms on your own, in the Armory you solve similar problems by using existing tools.

Badges

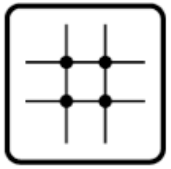
Alignment



Combinatorics

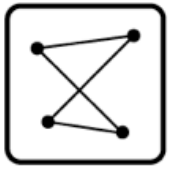
Computational Mass Spectrometry

Dynamic Programming

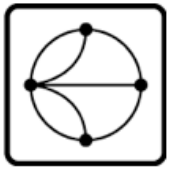


Genome Assembly

Genome Rearrangement



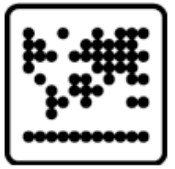
Graph Algorithms



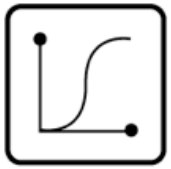
Heredity

Phylogeny

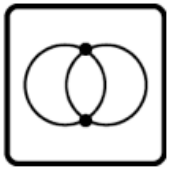
Population Dynamics



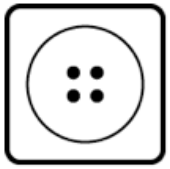
Probability



Set Theory



String Algorithms



Problems

Bioinformatics Stronghold ▾ | List | Tree

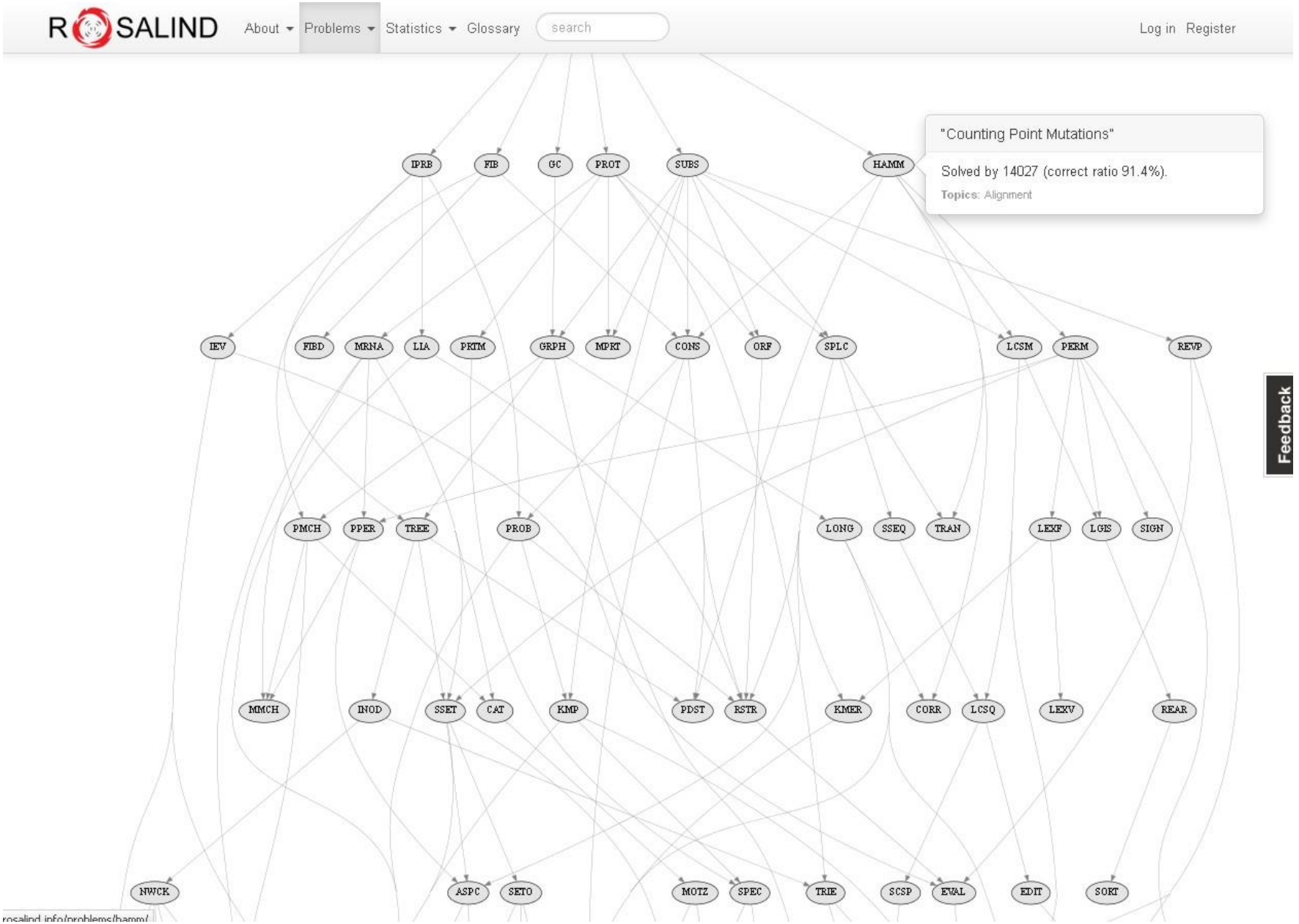
Rosalind is a platform for learning bioinformatics and programming through problem solving. Take a tour to get the hang of how Rosalind works.

Last win: markazhang vs. "Complementing a Strand of DNA", 7 minutes ago

Problems: 285 (total), users: 56855, attempts: 957794, correct: 537592

ID	Title	Solved By	Correct Ratio
DNA	Counting DNA Nucleotides	33297	
RNA	Transcribing DNA into RNA	29786	
REVC	Complementing a Strand of DNA	26984	
FIB	Rabbits and Recurrence Relations	15348	
GC	Computing GC Content	15909	
HAMM	Counting Point Mutations	17996	
IPRB	Mendel's First Law	10207	
PROT	Translating RNA into Protein	13949	
SUBS	Finding a Motif in DNA	14329	
CONS	Consensus and Profile	8023	
FIBD	Mortal Fibonacci Rabbits	6641	
GRPH	Overlap Graphs	6642	
IEV	Calculating Expected Offspring	6039	
LCSM	Finding a Shared Motif	5600	
LIA	Independent Alleles	3199	
MPRT	Finding a Protein Motif	3505	
MRNA	Inferring mRNA from Protein	5450	
ORF	Open Reading Frames	4182	
PERM	Enumerating Gene Orders	7533	
PRTM	Calculating Protein Mass	6910	
REVP	Locating Restriction Sites	4462	
SPLC	RNA Splicing	4924	
LEXF	Enumerating k-mers Lexicographically	4207	
LGIS	Longest Increasing Subsequence	1844	
LONG	Genome Assembly as Shortest Superstring	2113	

Problems tree



Problem: DNA Composition

Computing GC Content solved by 15909

3 sierpnia 2012 00:00:00 by Rosalind Team

Topics: String Algorithms

←

→

> **Identifying Unknown DNA Quickly** click to expand

Problem

The GC-content of a DNA string is given by the percentage of symbols in the string that are 'C' or 'G'. For example, the GC-content of "AGCTATAG" is 37.5%. Note that the reverse complement of any DNA string has the same GC-content.

DNA strings must be labeled when they are consolidated into a database. A commonly used method of string labeling is called **FASTA format**. In this format, the string is introduced by a line that begins with '>', followed by some labeling information. Subsequent lines contain the string itself; the first line to begin with '>' indicates the label of the next string.

In Rosalind's implementation, a string in FASTA format will be labeled by the ID "Rosalind_xxxx", where "xxxx" denotes a four-digit code between 0000 and 9999.

Given: At most 10 DNA strings in FASTA format (of length at most 1 kbp each).

Return: The ID of the string having the highest GC-content, followed by the GC-content of that string. Rosalind allows for a default error of 0.001 in all decimal answers unless otherwise stated; please see the note on **absolute error** below.

Sample Dataset

```
>Rosalind_6404
CCTGCGGAAGATCGGCACTAGAATAGCCAGAACCGTTTCTCTGAGGCTTCCGGCCTTCCC
TCCCACTAATAATTCTGAGG
>Rosalind_5959
CCATCGGTAGCGCATCCTTAGTCCAATTAAGTCCCTATCCAGGCGCTCCGCCGAAGGTCT
ATATCCATTTGTCAGCAGACACGC
>Rosalind_0808
CCACCCTCGTGGTATGGCTAGGCATTCAGGAACCGGAGAACGCTTCAGACCAGCCCGGAC
TGGGAACCTGCGGGCAGTAGGTGGAAT
```

Sample Output

```
Rosalind_0808
60.919540
```

Problem: Hamming distance

Given two sequences, calculate the number of differences.

GAGCCTACTAACGGGAT
CATCGTAATGACGGCCT

Example: hamming distance = 7

```
def hamming_distance(seq1, seq2):
    diffs = 0
    for ch1, ch2 in zip(seq1, seq2):
        if ch1 != ch2:
            diffs += 1
    return diffs
```

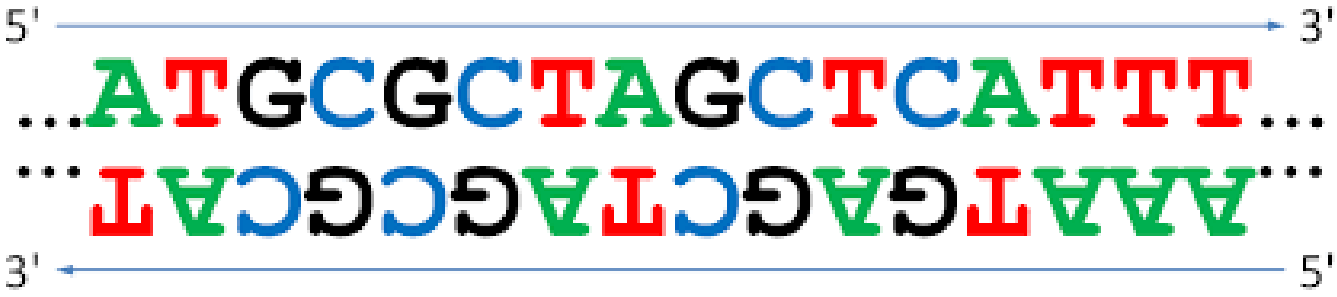
Problem: Reverse complement

For given sequence, find complementary sequence in reversed order.

Complementarity rule:

A – T

G – C



Reverse complement, solution 1

If-else clause

```
def revc_1(sequence):
    seq_temp = ''
    for letter in sequence:
        if letter == 'A':
            seq_temp += 'T'
        elif letter == 'G':
            seq_temp += 'C'
        elif letter == 'C':
            seq_temp += 'G'
        elif letter == 'T':
            seq_temp += 'A'
        else:
            seq_temp += 'N'
    return seq_temp[::-1]
```

Reverse complement, solution 2

If-then-else - lists

```
def revc_2(sequence):
    list_seq = list(sequence)
    for i in range(0,
len(sequence)):
        if list_seq[i] == 'A':
            list_seq[i] = 'T'
        elif list_seq[i] == 'T':
            list_seq[i] = 'A'
        elif list_seq[i] == 'C':
            list_seq[i] = 'G'
        elif list_seq[i] == 'G':
            list_seq[i] = 'C'
        else:
            list_seq[i] = 'N'
    return ''.join(list_seq[::-1])
```

Reverse complement, solution 3

Replacement

```
def revc_3(sequence):
    return (sequence.upper().replace('A', 't')
                            .replace('T', 'a')
                            .replace('C', 'g')
                            .replace('G', 'c')
                            .upper()[::-1])
```

Reverse complement, solution 4

Maketrans

```
def revc_4(sequence):
    return (sequence[::-1]
            .translate(str.maketrans('ACGTN', 'TGCAN')))

# """
# Code for python 2:

# from string import maketrans
# return (sequence[::-1]
#         .translate(maketrans('ACGTN', 'TGCAN')))
# """
```

Reverse complement, solution 5

With dict and `reversed` function

```
def revc_5(sequence):
    complement = {'A': ,T',
                  'C': 'G',
                  'T': 'A',
                  'G': 'C',
                  'N': 'N'}
    result = []
    for i in reversed(sequence):
        result.append(complement[i])
    return ''.join(result)
```

Reverse complement, solution 6

With dict and sequence slicing

```
def revc_6(sequence):
    complement = {'A': 'T',
                  'C': 'G',
                  'T': 'A',
                  'G': 'C',
                  'N': 'N'}
    result = ""
    for i in sequence[::-1]:
        result += complement[i]
    return result
```

Reverse complement, solution 7

With dict and list comprehention

```
def revc_7(sequence):
    complement = {'A': 'T',
                  'C': 'G',
                  'T': 'A',
                  'G': 'C',
                  'N': 'N'}
    return (''.join(
        [str(complement[sequence[i]])
         for _ in range(len(sequence))]))[::-1]
```

Reverse complement, solution 8

With sets

```
def revc_8(sequence):
    result = ""
    for x in sequence[::-1]:
        for pair in ["GC", "AT"]:
            if x in pair:
                result += "".join(set(x) ^ set(pair))
    return result
```

Reverse complement, solution 9

With zip

```
def revc_9(sequence):
    complement = dict(zip('ACGTN', 'TGCAN'))
    return ''.join(complement[x] for x in sequence[::-1])
```

Reverse complement, solution 10

With lambda

```
def revc_10(sequence):
    complement = {'A': 'T',
                  'C': 'G',
                  'T': 'A',
                  'G': 'C',
                  'N': 'N'}
    return ''.join(list(map(
        lambda x: complement[x], sequence)))[::-1]
```

Reverse complement, solution 11

With biopython

```
from Bio.Seq import Seq
def revc_11(sequence):
    seq = Seq(sequence)
    return seq.reverse_complement()
```

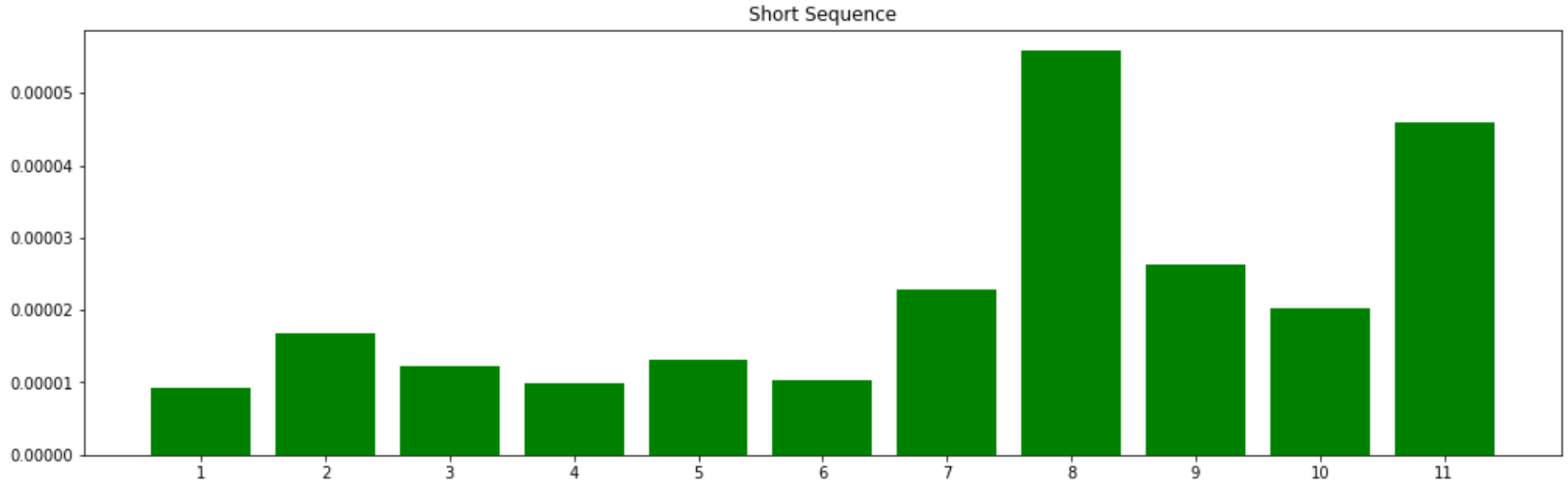
Reverse complement, tests

Performance tests for three sequences that significantly differs in length:

- really short sequence created manually
- gene sequence, about 1k (human hemoglobin beta subunit)
- long sequence, about 1M (fragment of human chromosome Y)

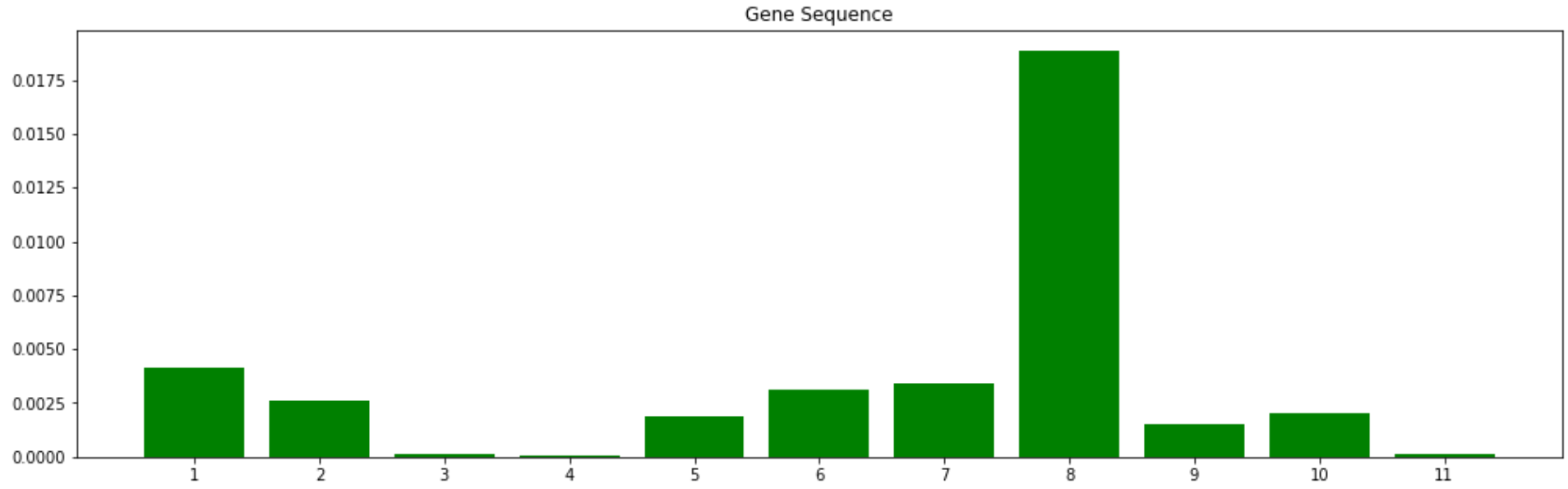
Reverse complement, results

Short sequence, 5nt



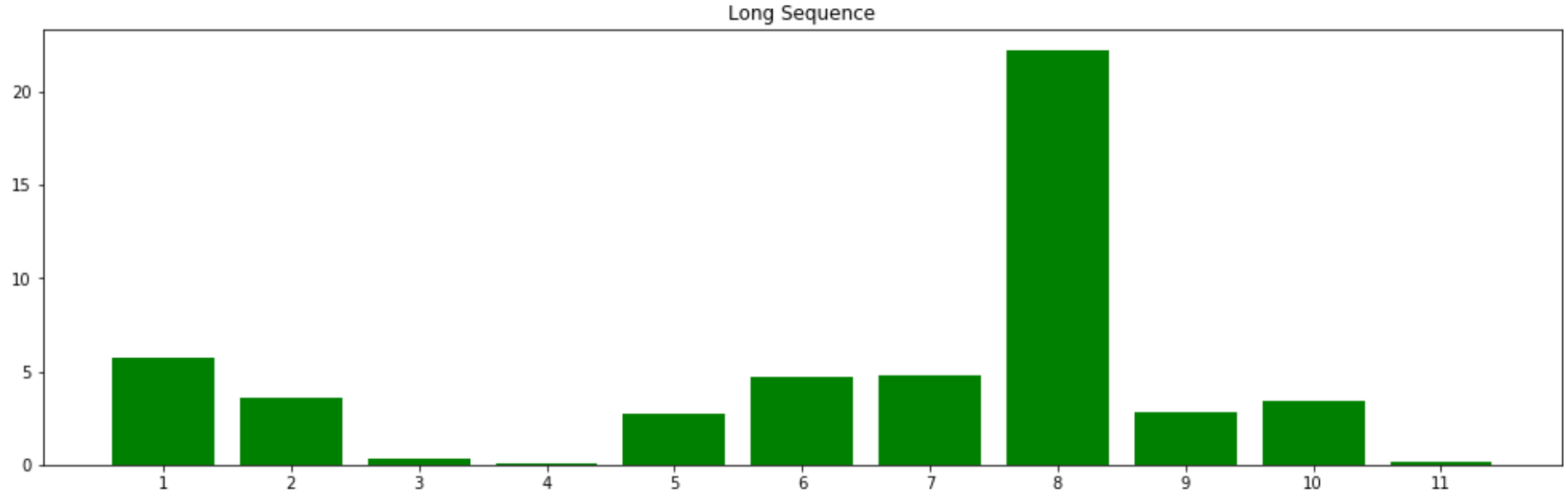
Reverse complement, results

Gene sequence, 1606nt



Reverse complement, results

Long sequence, 1939345nt



Thank you

jacek.smietanski@ii.uj.edu.pl