Lab activity: analysis of transcriptional regulatory networks

1. Introduction

This lab activity consists of six sessions, that will go through the data life cycle (data acquisition, preprocessing, analysis and visualitzation) within the context of biological networks.

Our first step will be data acquisition and preprocessing, illustrating where and how we can obtain the data and what obstacles we may face during this process, as well as how to store the data to later use it. Our second step will be to perform basic analyses of the generated network in order to gain insights into the network structure and function. Our third step will be to analyze how these networks are organized internally, identifying recurrent patterns of interconnection known as network motifs. Finally, we will work on generating graphical network representations that adequately convey the results we have obtained through our analyses.

The lab activity will be carried out in groups of four students.

Grades for the lab activity will take into account the code you program to solve specific tasks, your understanding of the problem and proposed solutions, as well as your participation in class presentations and discussions. Strong emphasis will be placed on the reasoning behind your solutions and the implications of your results.

The final lab grade will be calculated as follows:

- [P] Presentations and discussions (40%): Includes in-class presentations, defense of your solutions, and engagement in discussions.
- [C] Code (20%): Evaluation of your submitted code for functionality, structure, and clarity.
- [E] Lab Exam (40%): An individual exam conducted together with the second partial exam.

The lab sessions are designed to be highly interactive, encouraging discussion, collaboration, and in-class learning. You are expected to explain, justify, and defend your work during class. Due to the participatory nature of these activities, missed in-class components cannot be made up or recovered. Additionally, because each session builds on the previous one, code deliveries cannot be recovered neither.

For each code deliverable, only one submission per group is required. Submissions must be made through the Virtual Campus, in the designated lab submission section.

Date	Lab activity	Submission
12/11/2026	Session 1: graph generation	18/11/2025
19/11/2026	Session 2: operon detection	25/11/2025
26/11/2026	Session 3: graph analysis	02/12/2025
03/12/2026	Session 4: perturbation analysis	09/12/2025
10/12/2026	Session 5: network motifs	TBD
17/12/2026	Session 6: final project	TBD

As you will see from the statement of each session, lab sessions have two components: coding and analyzing results.

Coding needs to be done in Python (3.7 or higher). A template plain python file (.py) will be provided for each session. You need to implement all the code of each part in the given template file[1] (indications of where you need to write your code are included as comments in the template). You can define additional global functions (and import any libraries) if needed, but do not include any other code in the global scope. The algorithm that implements the work for each session (the calls to the functions) needs to be included inside the main scope.

The other component of the project requires you to analyze the information obtained using the code you have developed and reflect on some issues. These discussions will take place during class hours.

At the beginning of each class session, we will provide the statements for the session and, in some cases, present some concepts to be able to carry out the proposed work. The rest of the class session will be used to perform the work requested in the lab session description and to discuss either the implementation approaches or the implications of the obtained results.

[1] Do not convert the file to a Jupyter Notebook. Because of the length of this activity, plain Python files, which can be easily included from other files and debugged with IDEs, need to be used.

2. Session 1: Data acquisition and storage

2.1. Biological background

In this lab activity we will analyze the transcriptional regulatory network of bacterial species. What follows is a short introduction to the molecular biology concepts required to undertake this lab activity.

2.1.1. The master molecule: DNA

Each cell (whether a bacterium or one of the cells in your body) contains all the necessary information to create new copies of itself (and to keep itself alive). We call this type of information *genetic information*. Within the cell, genetic information is stored in a specific type of molecule called DNA. We refer to the totality of the genetic information stored in each cell as the *genome*.

DNA is a polymer made of four different nucleotides (or bases), which we denote as A, C, G and T. For instance, a short DNA sequence could be ACCTGGTTACATC. DNA is set up as a double strand[2], in which each base has a complementary base (A-T, G-C). Hence, our short sequence is in reality a molecule composed of two *strands*:

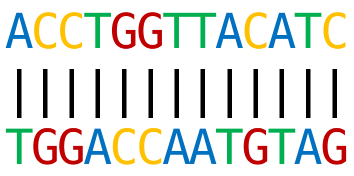


Figure 1 – Double strand representation of the short DNA sequence ACCTGGTTACATC. Since it is a double helix, with two strands, the length of a DNA sequence is conventionally measured in base *pairs* (bp). By convention, the top strand is referred to as the *forward* strand, and the bottom one as the *reverse* strand. One is read left-to-right, and the other one right-to-left. Converting one (ACCTGGTTACATC) into the other (GATGTAACCAGGT) is known as *reverse complementing*, since it involves flipping the sequence as well as complementing it (A→T, T→A, C→G, G→C).

2.1.2. Genes and genomes: transcription and translation

Information is only useful insofar as it can be read, and DNA is read all the time. It is read in chunks, which we call *genes*. Genes code for *proteins*, which perform tasks in the cell. In humans, for instance, hemoglobin is a protein used to transport oxygen (and other stuff) in the blood, and the lactase enzyme is a protein used to digest lactose. Hence, you can think of genes as cell microinstructions (e.g. *transport oxygen*, *digest lactose*).

The process by which genes are read in order to become proteins (which will carry out those instructions) is known as the Central Dogma of molecular biology, and it is a two-step process:

[2] This is used to detect and correct errors during the lifetime of a cell, and especially when DNA needs to be replicated (i.e. when the cell divides). Errors happen very frequently, and cells constantly repair their DNA (just for reference, each day, in each of the cells of your body, about 5,000 bases fall off the backbone of your DNA due to thermal noise).

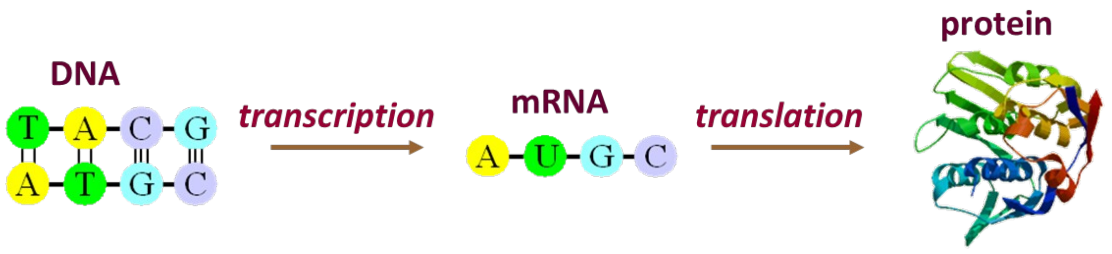


Figure 2 – Schematic representation of the Central Dogma of molecular biology.

Genes are first read (*transcribed*) from DNA into a similar molecule (RNA), which is then read (*translated*) into a protein (this is done using the Genetic Code, because proteins use an alphabet of 20 amino acids, instead of 4 nucleotides as DNA and RNA).

In order to be read, a gene must have a set of signals that enable its *reading* (transcription and then translation). For this lab activity, we will only consider one of them, the *promoter*, which is where a specialized enzyme, RNA-polymerase, binds to initiate transcription.

'Gene'

- Promoter [RNA-polymerase binding]
- Ribosome-binding site (RBS) [ribosome binding]
- Open Reading Frame (ORF) [translated region]
- Terminator [RNA-polymerase detachment]

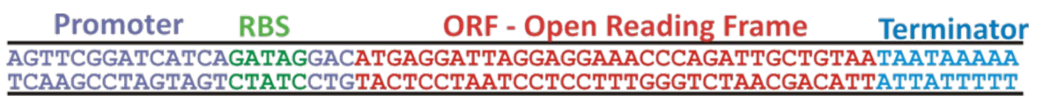

Promoter
RBS
ORF - Open Reading Frame
Terminator
AGTTCGGATCATCAGATAGGACATGAGGATTAGGAGGAAACCCAGATTGCTGTAATAATAAAAA
TCAAGCCTAGTAGTCTATCCTGTACTCCTAATCCTCCTTTGGGTCTAACGACATTATTATTTTT

Figure 3 – Schematic representation of a *gene* with its different components: promoter and terminator are used to initiate and terminate *transcription* by the RNA-polymerase enzyme. The RBS is the location where the ribosome binds to the transcribed RNA to initiate *translation*. The ORF is the actual "message" that gets translated into a protein, defined by start (ATG) and stop (TAA, TGA or TAG) signals.

We can model the genome as a long DNA sequence, which contains genes. Because the *promoter* and other signals on the DNA have directionality, genes can be read on one or the other strand.

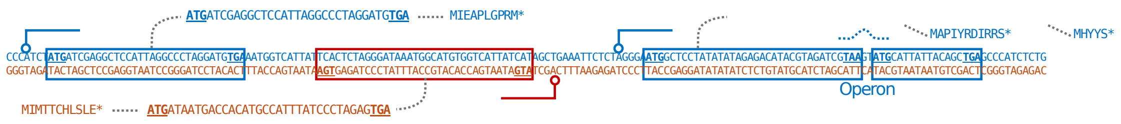

ATGATCGAGGCTCCATTAGGCCCTAGGATGTGA
MIEAPLGPRM*
MAPIYRDIRRS*
MHYYS*
CCCATCTATGATCGAGGCTCCATTAGGCCCTAGGATGTGAAATGGTCATTATTCACTCTAGGGATAAATGGCATGTGGTCATTATCATAGCTGAAATTCTCTAGGGAATGGCTCCTATATATAGAGACATACGTAGATCGTAAGTATGCATTATTACAGCTGAGCCCATCTCTG
GGGTAGATACTAGCTCCGAGGTAATCCGGGATCCTACACTTTACCAGTAATAAGTGAGATCCCTATTTACCGTACACCAGTAATAGTATCGACTTTAAGAGATCCCTTACCGAGGATATATATCTCTGTATGCATCTAGCATTCATACGTAATAATGTCGACTCGGGTAGAGAC
Operon
MIMTTCHLSLE*
ATGATAATGACCACATGCCATTTATCCCTAGAGTGA

Figure 4 – Schematic representation of a segment of a genome, containing several *genes*, illustrating their transcription and then translation. The directionality of the promoter (arrow) determines the strand the gene is read (transcribed) on. Some genes don't have a promoter; they simply are very close to the previous gene and are transcribed together with it. These genes are said to be in an *operon*.

2.1.3. Genetic programming: transcriptional regulation

In a cell, genes are akin to the microinstructions of a processor, and they can be read (i.e. executed). But having microinstructions is of little use if we cannot program with them. Programming, in a cell, is performed by a process known as *transcriptional regulation*.

Like conditional jumps in computer code (e.g. JZ, JNE), some of the microinstructions (proteins) in a cell have *programming* functions. These proteins, known as *transcription factors*, usually have two components: one that detects some signal, and another one that binds to the DNA molecule. By binding to the DNA, transcription factors facilitate or hinder the binding of RNA-polymerase to the promoter of some genes, essentially turning them ON or OFF.

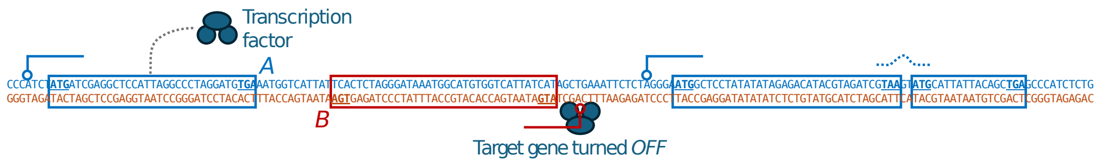


Figure 5 – Schematic representation of transcriptional regulation. Gene *A* codes for a transcription factor. Once “read” into a protein, the transcription factor binds to the promoter region of target gene *B*, shutting off its transcription.

2.2. Analyzing transcriptional regulatory networks (TRNs)

Bacterial species have between 80 and 300 transcription factors, and each transcription factor can regulate from very few to hundreds of genes. Some of these genes may code for other transcription factors, and may be regulated by other transcription factors as well. This leads to a complex network of regulatory interactions that we call the transcriptional regulatory network of the cell, and which we will analyze in this lab activity using graph theory approaches.

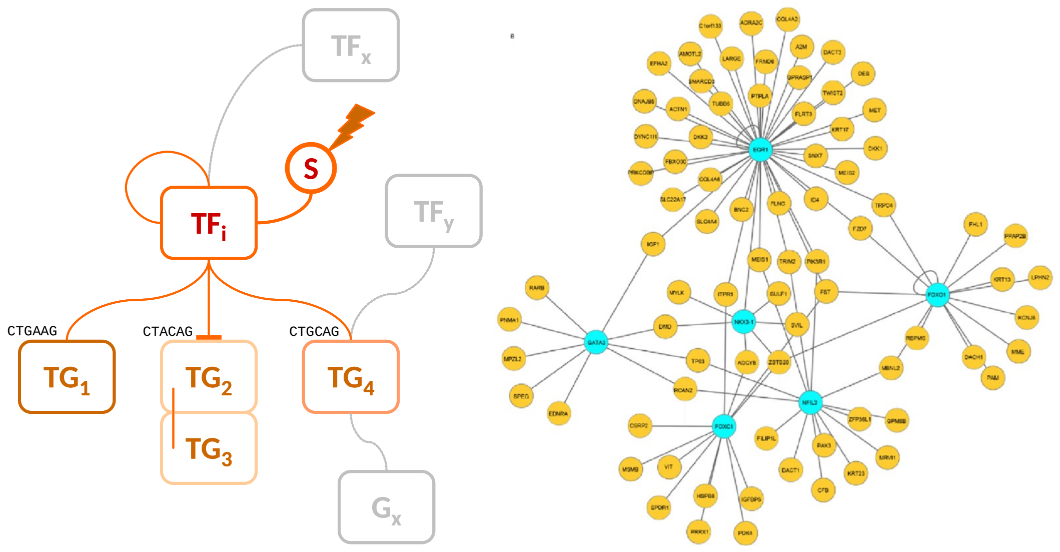
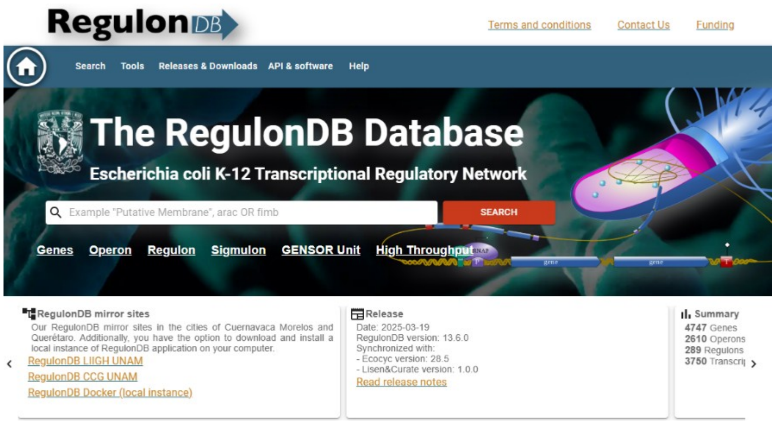


Figure 6 – [Left] Local structure of the cell’s TRN, with a TF regulating several genes (TG_2 and TG_3 are in an operon), including itself. Other TFs (TF_Y) can regulate the same target genes, or (TF_X) regulate the TF itself, leading to [Right] a complex network structure (TFs shown in blue, TGs in yellow).

2.3. Primary data sources

2.3.1. The RegulonDB database

The RegulonDB database is a database of experimentally-validated TF-binding sites (TFBS) of the bacterium *Escherichia coli*, a human commensal. The database is maintained by Julio Collado-Vides' team at the *Centro de Ciencias Genómicas* of the *Universidad Nacional Autónoma de México* (UNAM), and contains information on the binding sites and regulatory activity of 184 *E. coli* transcription factors.


RegulonDB

Terms and conditions Contact Us Funding

Search Tools Releases & Downloads API & software Help

The RegulonDB Database

Escherichia coli K-12 Transcriptional Regulatory Network

Example "Putative Membrane", arac OR fimb SEARCH

Genes Operon Regulon Sigmulon GENSOR Unit High Throughput

RegulonDB mirror sites

Our RegulonDB mirror sites in the cities of Cuernavaca Morelos and Querétaro. Additionally, you have the option to download and install a local instance of RegulonDB application on your computer.

RegulonDB LIIGH UNAM

RegulonDB CCG UNAM

RegulonDB Docker (local instance)

Release

Date: 2025-03-19

RegulonDB version: 13.6.0

Synchronized with:

- Ecocyc version: 28.5

- Lisen&Curate version: 1.0.0

Read release notes

Summary

4747 Genes

2610 Operons

289 Regulons

3750 Transcri

Figure 7 – The RegulonDB landing page at https://regulondb.ccg.unam.mx/.

2.3.2. The NCBI GenBank database

The National Center for Biotechnology Information (NCBI) of the National Library of Medicine (NLM) is one of the largest genetic data repositories in the world. Among others, the NCBI hosts the GenBank database, which contains genetic sequences for thousands of species and is used by biomedical researchers across the world.

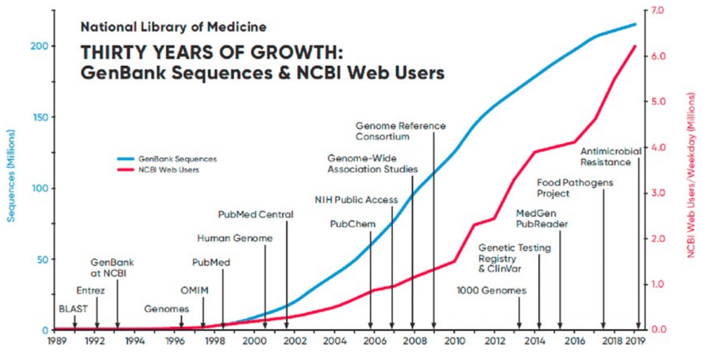


Figure 8 – Growth in sequence number and web user activity of the NCBI and its GenBank database (source: NLM 2021 Congressional Justification; Gaffney *et al. Global Food Security*, 26, 2020, doi: 10.1016/j.gfs.2020.100411).

2.4. Dataset acquisition and storage

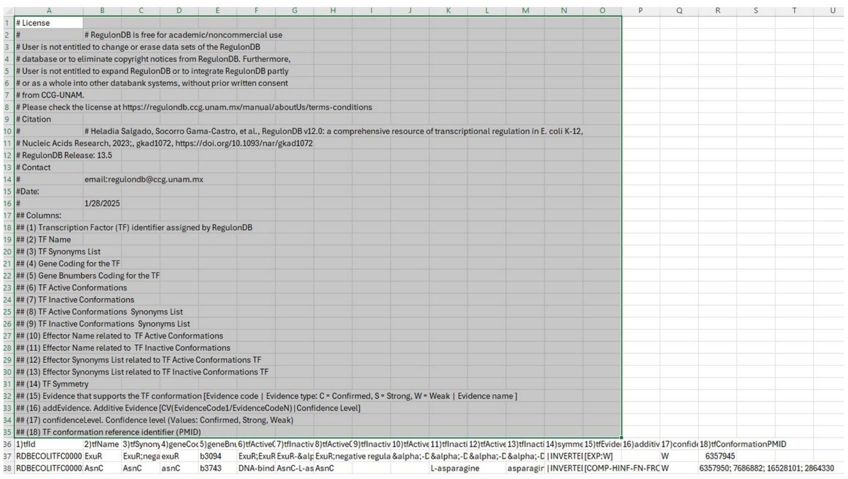
In this first lab activity we will acquire data from RegulonDB and from the NCBI GenBank database to generate TF-binding motif models. Together with the statement of this first session, a skeleton file where the functions to be implemented in each activity are explicitly stated is attached (`Lab AGX 202425 P1 skeleton.py`).

2.4.1. File download

We will first navigate to the Releases & Download section of RegulonDB and we will download two files: `TF-RISet` and `TF-Set`.

- `TF-RISet` contains information on the experimentally-validated regulatory interactions between transcription factors and their target genes. Among other nuggets of information, for each regulatory interaction we have the TFBS sequence, its genomic location, the experimental evidence for the regulatory activity and the scientific article where it was reported.
- `TF-Set` contains information on *E. coli* transcription factors, including the identifier (*locus_tag*) of the gene that codes for the TF.

These are tab-separated (TSV) files that, inconveniently, contain an explanation header. We will edit out this header, so that the files start on the column headers row, in order to facilitate their automated processing.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	# License																	
2	#	# RegulonDB is free for academic/noncommercial use																
3	# User is not entitled to change or erase data sets of the RegulonDB																	
4	# database or to eliminate copyright notices from RegulonDB. Furthermore,																	
5	# User is not entitled to expand RegulonDB or to integrate RegulonDB partly																	
6	# or as a whole into other databank systems, without prior written consent																	
7	# from CCG-UNAM.																	
8	# Please check the license at https://regulondb.ccg.unam.mx/manual/aboutUs/terms-conditions																	
9	# Citation																	
10	#	# Heladia Salgado, Socorro Gama-Castro, et al., RegulonDB v12.0: a comprehensive resource of transcriptional regulation in E. coli K-12,																
11	# Nucleic Acids Research, 2023;, gkad1072, https://doi.org/10.1093/nar/gkad1072																	
12	# RegulonDB Release: 13.5																	
13	# Contact																	
14	#	email:regulondb@ccg.unam.mx																
15	#Date:																	
16	#	1/28/2025																
17	## Columns:																	
18	## (1) Transcription Factor (TF) identifier assigned by RegulonDB																	
19	## (2) TF Name																	
20	## (3) TF Synonyms List																	
21	## (4) Gene Coding for the TF																	
22	## (5) Gene Bnumbers Coding for the TF																	
23	## (6) TF Active Conformations																	
24	## (7) TF Inactive Conformations																	
25	## (8) TF Active Conformations Synonyms List																	
26	## (9) TF Inactive Conformations Synonyms List																	
27	## (10) Effector Name related to TF Active Conformations																	
28	## (11) Effector Name related to TF Inactive Conformations																	
29	## (12) Effector Synonyms List related to TF Active Conformations TF																	
30	## (13) Effector Synonyms List related to TF Inactive Conformations TF																	
31	## (14) TF Symmetry																	
32	## (15) Evidence that supports the TF conformation [Evidence code | Evidence type: C = Confirmed, S = Strong, W = Weak | Evidence name]																	
33	## (16) addEvidence. Additive Evidence [CV(EvidenceCode1/EvidenceCodeN) | Confidence Level]																	
34	## (17) confidenceLevel. Confidence level (Values: Confirmed, Strong, Weak)																	
35	## (18) TF conformation reference identifier (PMID)																	
36	1)tfId	2)tfName	3)tfSynony	4)geneCoc	5)geneBnu	6)tfActive(	7)tfInactiv	8)tfActive(	9)tfInactiv	10)tfActive	11)tfInacti	12)tfActive	13)tfInacti	14)symme	15)tfEvide	16)additiv	17)confid	18)tfConformationPMID
37	RDBECOLITFC0000	ExuR	ExuR;nega	exuR	b3094	ExuR;ExuR	ExuR-&alp	ExuR;negative regula		&alpha;-D	&alpha;-D	&alpha;-D	&alpha;-D	|INVERTEI	[EXP:W]		W	6357945
38	RDBECOLITFC0000:	AsnC	AsnC	asnC	b3743	DNA-bind	AsnC-L-as	AsnC			L-asparagine		asparagir	|INVERTEI	[COMP-HINF-FN-FRC		W	6357950; 7686882; 16528101; 2864330

Figure 9 – Rows to be removed from the TF-Set file.

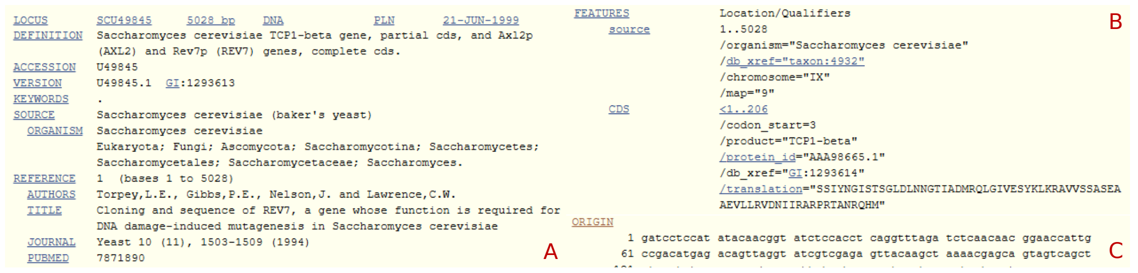
The complete genome of the *E. coli* reference strain (*Escherichia coli* str. K-12 substr. MG1655) is available on the NCBI GenBank Nucleotide database (https://www.ncbi.nlm.nih.gov/nuccore/) under accession number `U00096.3`. We will search for this accession number and download the nucleotide sequence in *GenBank (full)* format through the Send to File link. We will save the file as "`U00096.3.gb`".

2.4.2. Biology in Python: the `biopython` package

Bioinformatics is a scientific discipline focused on the analysis of large amounts of biological data to infer new biological knowledge. Although many programming languages are used in bioinformatics, the discipline has gradually adopted Python as a standard, propelled in large part by the development of the biopython package (https://biopython.org/). This package contains libraries defining classes for most biological data elements (e.g. genetic sequences) as well as parsers to read and write from and to common bioinformatics tools and pipelines.

2.4.3. Genome processing

A central object in bioinformatics is the genome. Genome files are most often distributed using the GenBank format, which contains the DNA sequence of the genome, annotations (e.g. the species name) and features (e.g. from position *X* to position *Y*, in the forward strand, there is gene *Z*).



```
LOCUS       SCU49845     5028 bp    DNA             PLN       21-JUN-1999
DEFINITION  Saccharomyces cerevisiae TCP1-beta gene, partial cds, and Axl2p
            (AXL2) and Rev7p (REV7) genes, complete cds.
ACCESSION   U49845
VERSION     U49845.1  GI:1293613
KEYWORDS    .
SOURCE      Saccharomyces cerevisiae (baker's yeast)
  ORGANISM  Saccharomyces cerevisiae
            Eukaryota; Fungi; Ascomycota; Saccharomycotina; Saccharomycetes;
            Saccharomycetales; Saccharomycetaceae; Saccharomyces.
REFERENCE   1  (bases 1 to 5028)
  AUTHORS   Torpey,L.E., Gibbs,P.E., Nelson,J. and Lawrence,C.W.
  TITLE     Cloning and sequence of REV7, a gene whose function is required for
            DNA damage-induced mutagenesis in Saccharomyces cerevisiae
  JOURNAL   Yeast 10 (11), 1503-1509 (1994)
  PUBMED    7871890
```

```
FEATURES             Location/Qualifiers
     source          1..5028
                     /organism="Saccharomyces cerevisiae"
                     /db_xref="taxon:4932"
                     /chromosome="IX"
                     /map="9"
     CDS             <1..206
                     /codon_start=3
                     /product="TCP1-beta"
                     /protein_id="AAA98665.1"
                     /db_xref="GI:1293614"
                     /translation="SSIYNGISTSGLDLNNGTIADMRQLGIVESYKLKRAVVSSASEA
                     AEVLLRVDNIIRARPRTANRQHM"
ORIGIN
        1 gatcctccat atacaacggt atctccacct caggtttaga tctcaacaac ggaaccattg
       61 ccgacatgag acagttaggt atcgtcgaga gttacaagct aaaacgagca gtagtcagct
```

Figure 10 – The three main elements of a GenBank-formatted file. [A] Annotations. [B] Features. [C] Sequence. See https://www.ncbi.nlm.nih.gov/genbank/samplerecord/ for details.

When you read in a genome file with `biopython` (using the `SeqIO` read command and specifying that you are reading “genbank” format) the three central elements of the GenBank file become attributes of the SeqRecord object that is generated. Hence, `record.features` is a list with all the features in the genome.

Each feature in a genome has a `location` (start, end and strand) and a `type`. The most common feature types are *genes* and *CDS* (which stands for ‘coding sequence’; the part of the gene that is translated into protein). A feature will also have a number of `qualifiers`, organized as a dictionary. Hence, `record.features[N].qualifiers` is a dictionary with keys such as `‘gene’` or `‘product’`.

We will be working primarily with the following qualifiers:

- gene: the *name* of the gene the feature corresponds to (e.g. *dapB*)
- `locus_tag`: a unique *identifier* for that feature in the genome (e.g. *b0031*)
- `protein_id`: a unique *identifier* for the protein coded by the CDS feature (e.g. AAC73142.1)
- `product`: the *function* (if known) of the protein coded by the CDS feature (e.g. 4-hydroxy-tetrahydrodipicolinate reductase)

The skeleton code for this session contains to auxiliary functions that you can use in your implementation.

The feature_list function returns a list of tuples (`locus_tag`, `protein_id`) with all the CDS features in a genome sequence record object whose `product` qualifier contains the query string.

```
def feature_list(genome : SeqRecord, query : str) -> list:
    '''
    Extract CDS features with specific feature description.
    - param genome : SeqRecord
        genome SeqRecord object to be analyzed.
    - param query : str
        feature descriptor.
    - return list
        list of tuples (locus_tag, protein_id) matching descriptor.
    '''
```

The gene_qualifier function receives a gene qualifier value (e.g. *insL1*) and its type (e.g. gene) as parameters, and returns the index and the specified qualifier (e.g. `locus_tag`) for the corresponding gene (type) feature in the genome.

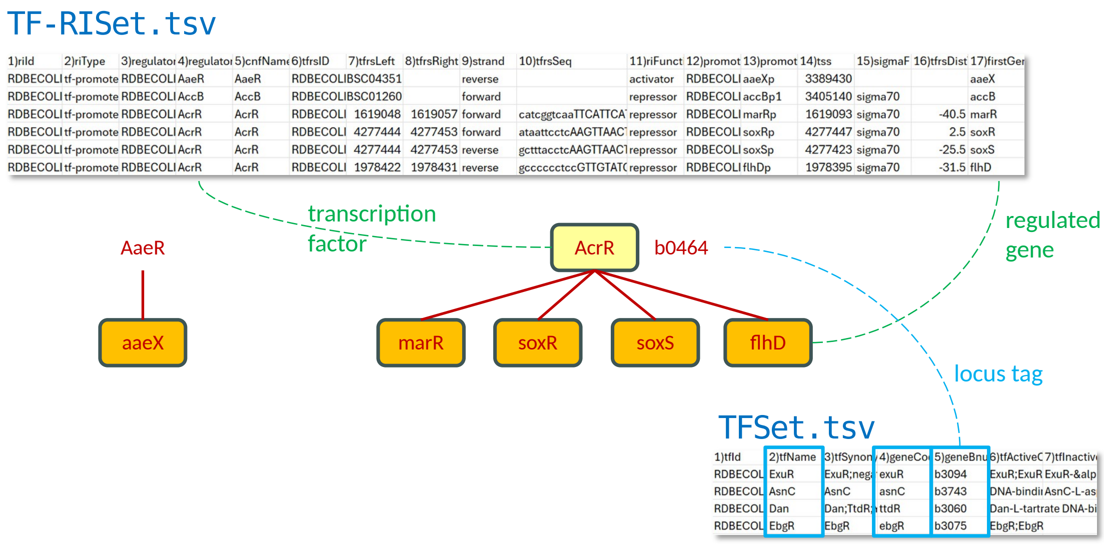
```
def gene_qualifier(query : str, query_field : str,
                   target_field : str, genome : SeqRecord) -> tuple:
    '''
    Obtain the specified qualifier identifier for a given gene qualifier
    - param: query: str
        gene name/locus_tag to map to corresponding locus_tag/name
    - param: query_field: str
        query type indicator (gene/locus_tag/protein_id/product)
    - param: target_field: str
        target type indicator (gene/locus_tag/protein_id/product)
    - param: genome : SeqRecord
        genome SeqRecord object containing features
    - return: tuple
        int : feature index
        str : specified qualifier for gene (empty string if no match)
    '''
```

2.4.4. Network generation

The `TF-RISet` and `TFSet` files we downloaded from RegulonDB contain the data we need to recreate the transcriptional regulatory network of *E. coli*. The resulting network will be a directed graph.

The `TFSet` file contains information on the different *E. coli* transcription factors, including the name and locus tag of the genes that code for these proteins. The `TF-RISet` file contains information on the regulatory interaction between a TF and its target gene (TG), which can be the first gene in an operon. This information includes the name of the TF, the start, end, strand and length of the TF-binding site, and the name and other information of the target gene.

Using the data in the `TFSet` file we will build a TF dictionary, with the TF name (`2)tfName`) as key, and the gene (`4)geneCodingForTF`) and `locus_tag` (`5)geneBnumberCodingForTF`) information for the TF gene as values. We will use this dictionary to get the TF gene information as we process the `TF-RISet` file.



TF-RISet.tsv

1)riId	2)riType	3)regulator	4)regulator	5)cnfName	6)tfrsID	7)tfrsLeft	8)tfrsRight	9)strand	10)tfrsSeq	11)riFunct	12)promot	13)promot	14)tss	15)sigmaF	16)tfrsDist	17)firstGer
RDBECOLI	tf-promote	RDBECOLI	AaeR	AaeR	RDBECOLIBSC04351			reverse		activator	RDBECOLI	aaeXp	3389430			aaeX
RDBECOLI	tf-promote	RDBECOLI	AccB	AccB	RDBECOLIBSC01260			forward		repressor	RDBECOLI	accBp1	3405140	sigma70		accB
RDBECOLI	tf-promote	RDBECOLI	AcrR	AcrR	RDBECOLI	1619048	1619057	forward	catcggtcaaTTCATTCA	repressor	RDBECOLI	marRp	1619093	sigma70	-40.5	marR
RDBECOLI	tf-promote	RDBECOLI	AcrR	AcrR	RDBECOLI	4277444	4277453	forward	ataattcctcAAGTTAAC	repressor	RDBECOLI	soxRp	4277447	sigma70	2.5	soxR
RDBECOLI	tf-promote	RDBECOLI	AcrR	AcrR	RDBECOLI	4277444	4277453	reverse	gctttacctcAAGTTAAC	repressor	RDBECOLI	soxSp	4277423	sigma70	-25.5	soxS
RDBECOLI	tf-promote	RDBECOLI	AcrR	AcrR	RDBECOLI	1978422	1978431	reverse	gcccccctccGTTGTAT(	repressor	RDBECOLI	flhDp	1978395	sigma70	-31.5	flhD

TFSet.tsv

1)tfId	2)tfName	3)tfSynony	4)geneCo	5)geneBnu	6)tfActiveC	7)tfInactive
RDBECOL	ExuR	ExuR;nega	exuR	b3094	ExuR;ExuR	ExuR-&alp
RDBECOL	AsnC	AsnC	asnC	b3743	DNA-bindi	AsnC-L-as
RDBECOL	Dan	Dan;TtdR;	ttdR	b3060	Dan-L-tartrate	DNA-bi
RDBECOL	EbgR	EbgR	ebgR	b3075	EbgR;EbgR	

Figure 11 – Extraction of data from RegulonDB tab-separated files.

For the `TF-RISet` file, we will process each row (one regulatory interaction), adding the TF (`4)regulatorName`) and target genes (`17)firstGene`) as nodes to the network, and establishing a link (directed edge) between them. We will obtain `locus_tag` information for the target genes from the *E. coli* genome file, using the `gene_qualifier` function. We will assess whether the target gene is the first gene of an operon and, if so, we will also add links from the TF to all the operon genes. We will add a `ntype` field to designate transcription factor ('TF') and target gene ('TG') nodes. A TF can be a target gene for another TF. We should make sure that we do not overwrite the `ntype` attribute of an existing TF node with a 'TG' qualifier.

Hence, we will store the following information for nodes:

- locus tag
  - ntype ('TF' or 'TG')
  - name (gene name)

2.5. Implementation

1. Implement a function that, given two file names (TF-RISet and TFSet) and a genome record, returns a directed graph with the reconstructed transcriptional regulatory network.

```
def TF_RISet_parse(tf_riset_filename : str, tf_set_filename : str, \
                   detect_operons : bool, max_intergenic_dist : int, \
                   genome : SeqRecord) -> nx.DiGraph:
    """
    Parse TF-RISet file to obtain a TRN graph.
     The TFSet file will be used to extract information on the gene coding
for
    each transcription factor [4)geneCodingForTF, 5)geneBnumberCodingForTF]
    The TF-RISet file will be used to extract genes regulated by each TF.
     We will create nodes using their locus_tag identifier, and save the
gene
    name as a 'name' attribute.
     If selected, operons will be predicted for each of these genes to
determine
    the entire set of genes regulated by the TF.

    - param: tf_riset_filename : str
        name/path of the TF_RISet file
    - param: tf_set_filename : str
        name/path of the TFSet file
    - param: detect_operons : bool
        whether we run operon detection
    - param: max_intergenic_dist : int
        maximum distance between consecutive, same strand genes
    - param: genome : SeqRecord
        genome SeqRecord object to extract information from
    - return: TF dictionary
    """
    # ------- IMPLEMENT HERE THE BODY OF THE FUNCTION ------- #
    pass
    # ---------------- END OF FUNCTION -------------------- #
```

For this session, do not use the detect_operons nor the max_intergenic_dist arguments of the function (your code can just ignore them).

2. Using the previous function:

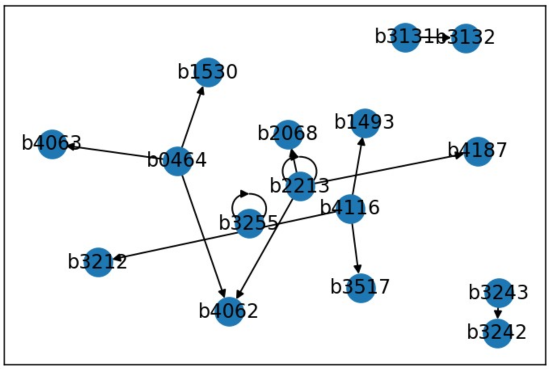
   a. Load the *E. coli* genome from the `U00096.3.gb` file into a biopython genome object.

   b. Generate a graph with the reconstructed transcriptional regulatory network of *E. coli*. For this session, set the detect_operons parameter to False and the max_intergenic_dist to 0 (your code should ignore both parameters).

   c. Save the network as a GraphML file (`Ecoli_TRN.graphml`).

   d. Familiarize yourself with the obtained network.

You can use the provided `miniTF-RISet.tsv` file to check that your code is working as expected. You should obtain a network like the one shown below.



3. Evaluation and submission

As explained in the introduction, the evaluation of the lab activities will consider in-class presentations, defense of your solutions, and engagement in discussions; code; and a lab exam.

The code part of this session needs to be delivered via the virtual campus the day before the next lab session (that is, at most on November, 18th). The submission has to be a single zip file with:

- The `AGICI_lab_S1.py` file with your implementation of the TF_RISet_parse function.
- The E.coli network (`.graphml` file) generated from the execution of your code.