Genome Complexity Browser

Apr 21, 2021

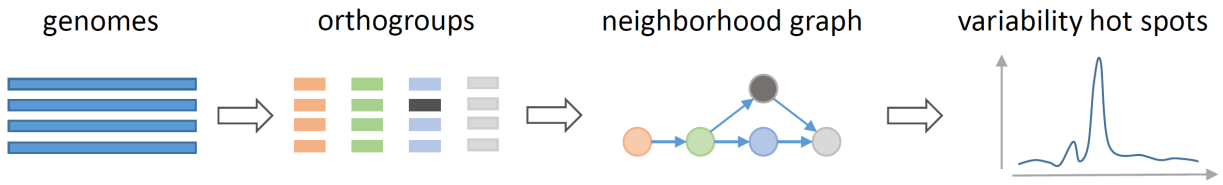
Contents:

Welcome to the Genome Complexity Browser (GCB) documentation.

GCB is created to analyse variability in hundreds of prokaryote or viral genomes simultaniously.

GCB performs:

- Visualization of gene neighborhoods in a graph-based format.
- Quantification of local genome variability.



The graph-based visualization may facilitate answering questions:

- Is a gene (operon) located in the same location in all genomes? If not, then what alternative genes are present?
- Which parts of a gene set (operon) are conserved and which are variable?
- Which genomes contain some particular combination of genes?

Local genome variability profile may be used to identify:

- Hotspots of horizontal gene transfer or other local gene rearrangement events.
- Cold spots, regions of the genome with virtually no changes in the considered set of genomes.

The GCB is available as a web server and as a standalone application. Web server contains precalculated data for 143 prokaryote species. Standalone version allows to analyze a custom set of genomes (basic command-line skills will be needed).

You can find screen recordings of GCB here (Youtube).

GCB paper is published in:

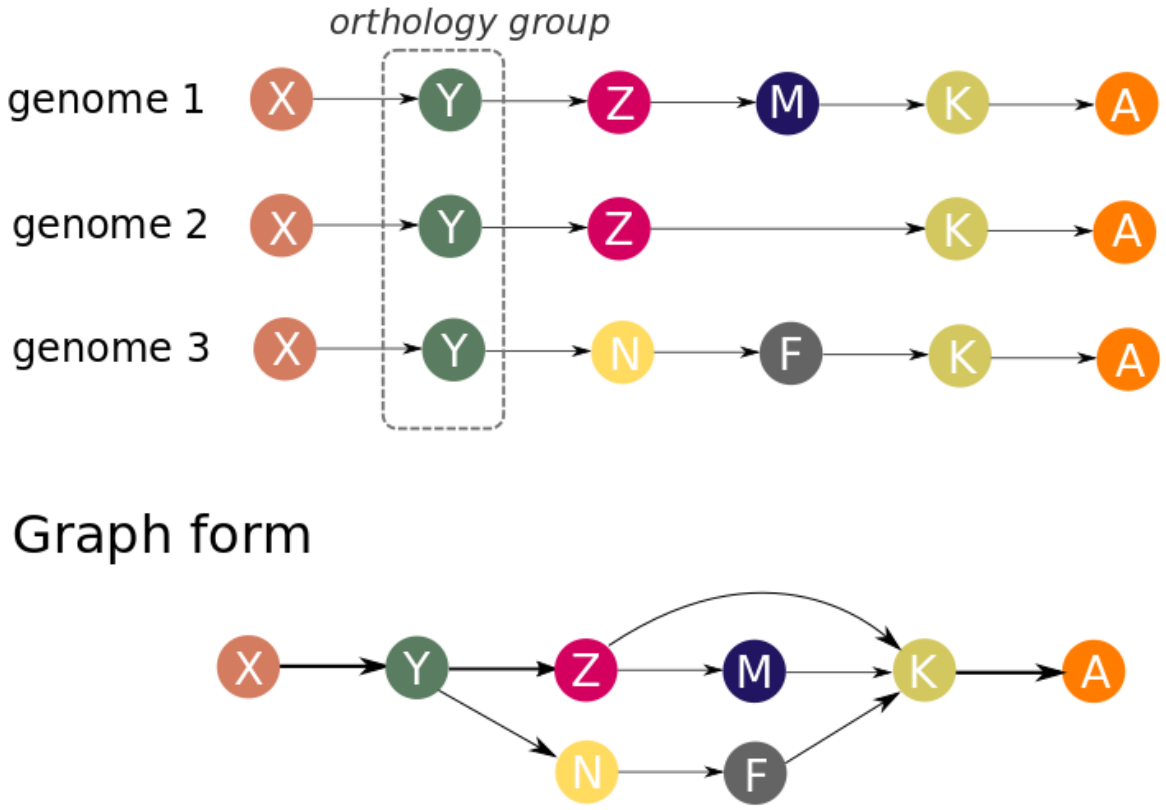
Manolov, A., Konanov, D., Fedorov, D., Osmolovsky, I., Vereshchagin, R., & Ilina, E. (2020). Genome Complexity Browser: Visualization and quantification of genome variability. PLOS Computational Biology, 16(10), e1008222. https://doi.org/10.1371/journal.pcbi.1008222

CHAPTER 1

Main concepts

1.1 Graph representation of genomes

Let's consider three genomes in which orthogroups were inferred. To make a graph representation of the gene neighbourhoods in these genomes we will consider each orthogroup as a node. Nodes are connected by an edge if there is at least one genome in which corresponding genes (orthology groups representatives) are adjacent.



In our approach, genes are ordered by their centered position: (start + end)/2 on a replicon (contig). Each replicon (contig) is treated separately. Strand information is ignored in graph building procedure. Because gene order in a set of genomes is represented by a directed graph, genomes may be co-aligned against each other before graph construction to optimize their orientation, but this does not significantly affect the result.

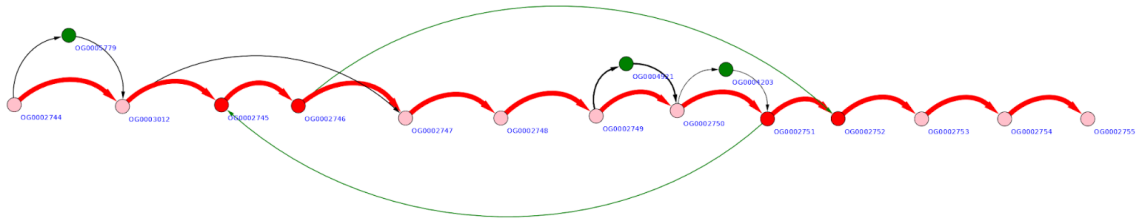
Several genes from one genome may be present in same orthogroup (paralogous genes). This makes inapplicable the above described scheme, and we propose two possible way-outs. The first approach is to skip such orthogroups in the graph for genomes containing paralogs (however, the same orthogroup will be included in the graph for genomes containing no paralogs). Or paralogous genes can be orthologized and be incorporated in the graph. By orthologization we mean procedure, which delineate paralogues genes by adding unique suffix to each unique paralogous gene context. For example, sequence of genes: A -> B -> C -> D -> B -> E, after orthologization becomes A -> B_1 -> C -> D -> B_2 -> E.

The graph representation allows compactly represent possible variants of the repertoire and the neigbourhood of genes in genomes. However, complete graph of all genomes will be too complicated for visualization, unless you are working with viruses. To explore a certain region of interest (gene, operon, genomic island), subgraphs can be generated. Subgraph is part of a total graph that represents some specific region of the reference genome.

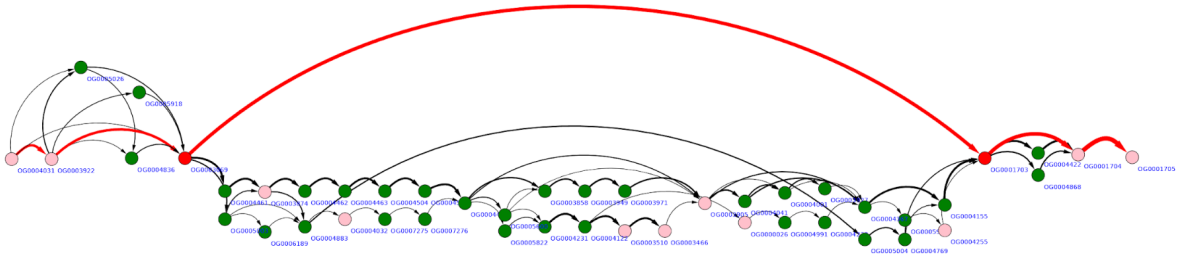
1.2 Complexity profile

By complexity of a genomic region, we mean the number of paths in the graph representing this region. The rationale here is the following. If no genome rearrangements resulting in gene order changes are observed in this region than its graph representation will be a simple chain. If some changes are observed (insertion, deletion, transposition, inversion of regions longer than a single gene) then the graph will contain additional edges. The more frequently gene rearrangements occur in a particular genomic region the more edges (and paths) the corresponding graph will contain.

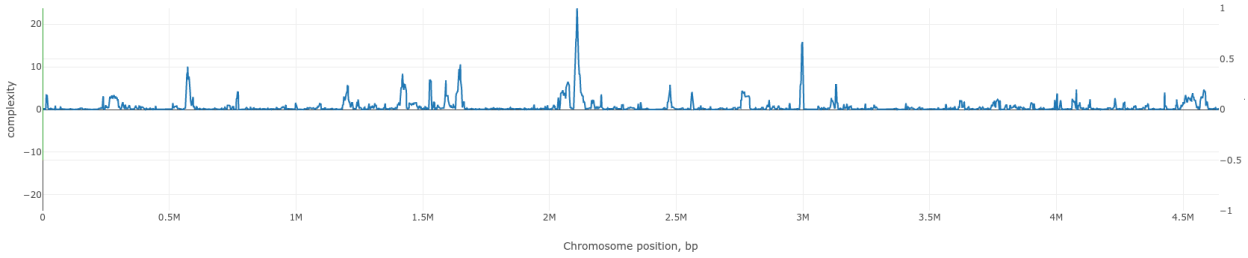
Example of a rather “quiet” region is below. Here three possible gene insertions, one deletion, and one gene transposition can be seen.



More “complex” region may look like the graph below, with multiple overlapping gene replacements and insertions.



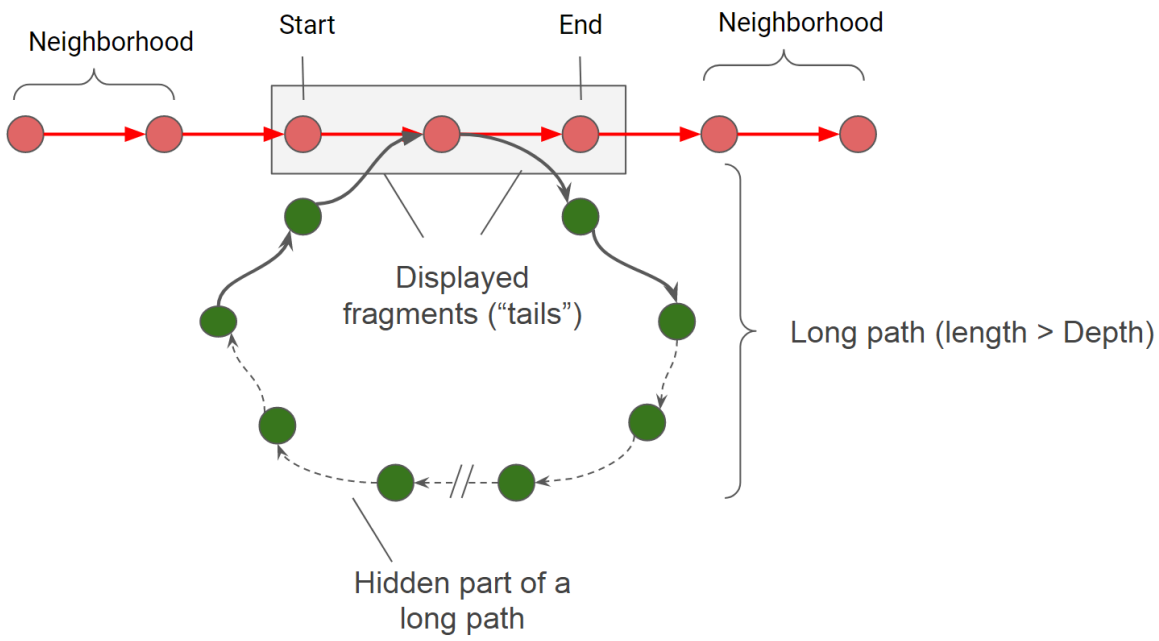
To quantify this visual expression we implement an algorithm which counts the number of distinct paths in a (sub)graph representing some genome region - the value we will further call **complexity**. Complexity value is calculated for a given gene in a reference genome in a window of defined width (5, 10,20 are typical values). Scanning the genome with the sliding window we obtain complexity profile of that genome. Example of such a profile for *Escherichia coli K12 MG1655* genome is given below.



Regions of low and high complexity can be seen which corresponds to cold and hot spots of genome rearrangements, mainly due to the frequency of fixed horizontal gene transfer (HGT) events.

1.3 Subgraph generation

When generating a subgraph, the nodes of the reference genome which are located between the `sart` and `end` coordinates (set in b.p.). It is sometimes useful to expand the region of the genome under consideration, in this case, you can increase the value of the parameter `Neighborhood` (set in genes).



Even for small fragments of the genome (especially when they are highly variable), subgraphs may be too large for visualization and efficient analysis. We have implemented two filters to simplify subgraphs: a long path filter and a low weight edge filter.

If the path begins and returns in the considered region of the genome, but its length is greater than the `Depth` parameter, then it is cropped up to fragments of length `tails`. The same happens with all paths, which have only a beginning or only an end in the considered area. After applying the filter of long paths, a filter of low weight edges is applied, that is, such combinations of genes that occur in a small number of genomes. Edges weighing less than `Minimal edge weight` are excluded from the subgraph.

1.4 Genome Complexity browsing

We created GCB to analyze genome variability on two scales simultanously. Globally, the complexity profile of the whole genome is shown. Localy, some genome region is represented in a graph-based format. This combination creates a unique opportunity to “browse” genome variability: to find hotspots, to analyze gene contents and patterns in this hotspots, or to investigate the level of variability in the region in which the genes of interest are located.

Web browser alows surfing the Internet, our browser allows surfing genomes.

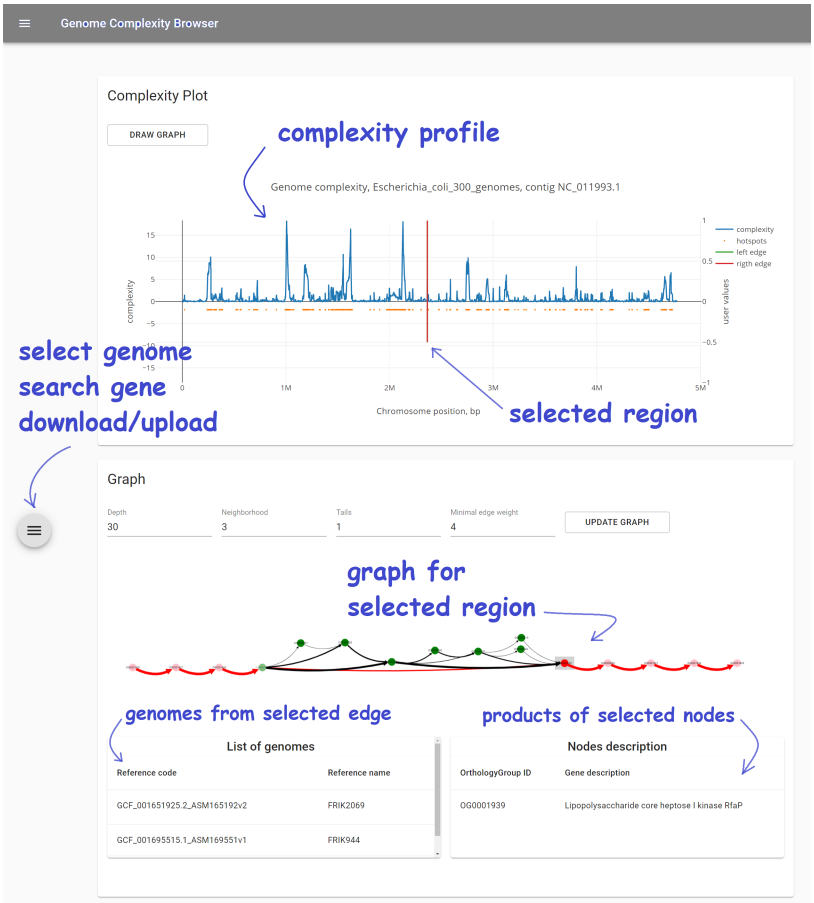
CHAPTER 2

GUI description

2.1 GUI elements

GCB page consists of three main parts:

- Complexity plot which shows complexity profile for selected genome and contig
- Subgraph visualization of selected genome region
- Left sliding panel to:
  - select genome and region to work with
  - search gene product names
  - download complexity values and graph.



2.2 Tutorials

2.2.1 Complexity analysis

Complexity is the measure of local genome variability. It is calculated for a set of genomes, one of which is selected as a reference. The greater the number of local changes in a certain neighborhood, the greater the value of complexity. Locality is set by the parameter `Window`, the larger it is, the more large-scale changes influences complexity value.

Thus, the three parameters that you need to select to calculate the complexity profile are: 1) set of genomes, 2) reference genome, 3) window size. All of them can be set in the left sliding panel.

At gcb.rcpcm.org we precalculated complexity profiles for over 140 prokaryotic species. Any set of genomes may be used in a stand-alone version. Window size equals 20 by default, and can be changed to other precalculated values (20, 50 and 100 values are available at gcb.rcpcm.org, any value may be used in a standalone version).

To obtain the complexity profile for a certain set of genomes (we will consider the set of *Bacillus Subtilus* available at gcb.rcpcm.org), follow these steps:

(1) Open gcb.rcpcm.org in you web-browser.

Left panel with a number of settings will open automatically.

(2) Click on the `Organism` selector and choose `Bacillus_subtilis` item.

(3) Click on the `Reference` selector and choose one of the available genomes. Let's choose for example `GCF_001889385.1_ASM188938v1_genomic (J-5)` genome.

Genome name is constructed based on its filename (NCBI RefSeq filenames are used in the web version, genomes were downloaded from NCBI FTP server). In the web version strain name from the RefSeq is added to this name in brackets.

(4) Click on the `Reference` selector.

The contigs of the selected genome will be shown. For complete (finished) genomes they corresponds to replicones (chromosome(s) and plasmids), for draft genomes they are some fragments of genome. You cannot see the complexity profile for all contigs at the same time, they must be selected in turn.

In case of `J-5` genome, only one contig is present (`NZ_CP018295.1`), which means that it is a complete (finished) genome, with single chromosome and without plasmids.

(5) Close left sliding panel by clicking on the some point outside it (e.g., on the middle of the page).

To open this panel once again, click on the icon with three horizontal lines on the left side of the page.

(6) Complexity profile wile appear in the *Complexity Plot* panel.

A number of peaks and flat areas are visible in the complexity profile. Peaks corresponds to the genome regions in which many changes were fixed during evolution.

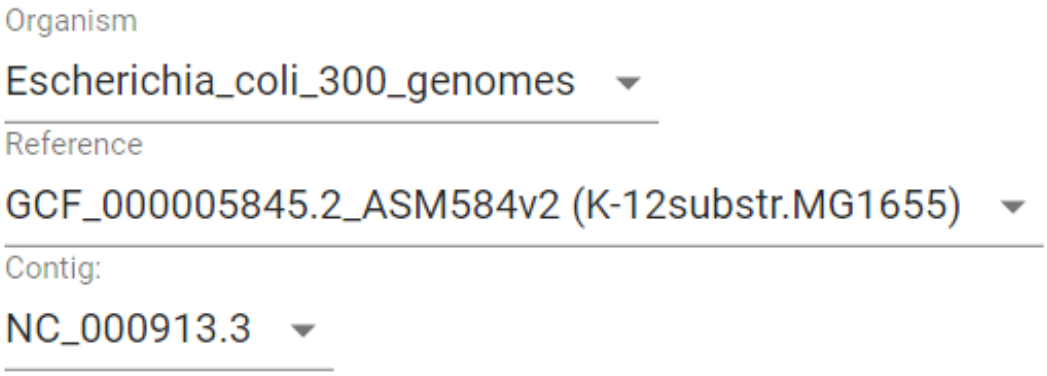
(7) To get the numerical values of the complexity profile, open left sliding panel, select the *File* tab at the top of the panel, click `Download complexity values` button.

Screen recording with these steps performed is available here.

2.2.2 Gene neigborhood analysis

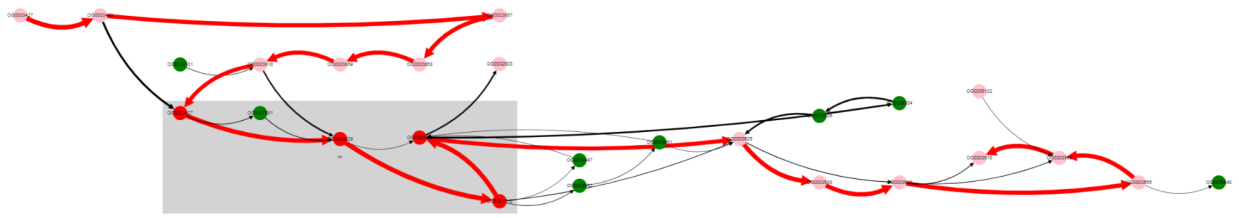
Consider we are interested in a particular operon and we want to know more about its representation in a particular group of organisms. Let's take lactose operon with its regulator in the Escherichia coli as an example (we will call it lac operon further).

First, we need to determine genome position of genes of interest. We can do it in multiple ways, e.g., from EcoCyc database or from NCBI Refseq. According to EcoCyc, the operon is located at 361249-366305 in *K-12 substr. MG1655*. Let's first select *Escherichia coli* as `Organism`, in a dataset containing 300 *E. coli* genomes. Then, select K-12 MG1655 as `Reference`. Only one contig is present (finished genome, no plasmids), so we do not need to change `Contig` value.



Then we should put the operon's margin coordinates to text fields: `Start coordinate` and `End coordinate`.

Now click `DRAW GRAPH` button and graph-based representation of the operon will appear in *Graph* panel.

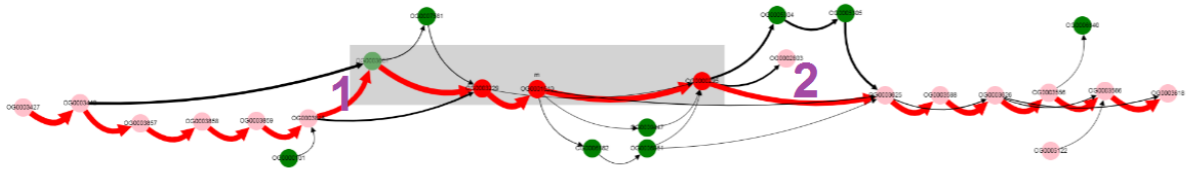


Gray rectangle is drawn around nodes located between `Start coordinate` and `End coordinate` (lac operon genes in this case).

Note that edges are of different colors. Red edges are from the reference genome (and also from genomes in which genes are located in the same order), black edges correspond to variants not present in the reference genome.

Nodes also have different colors. Nodes of genes from the reference genome are colored red, if they are located inbetween `Start coordinate - neigbhourhood` and `End coordinate + neigbhourhood`; they are colored pink, if present in the reference genome, but located outside the mentioned region. Nodes of genes, that are absent in a reference genome, are colored green.

More clear layout can be obtained manually by left clicking and dragging nodes with mouse.



By looking at thick edges designated with 1 and 2, we can tell that operon is located in a conserved context, in most of the genomes.

One of the operon's genes is absent in some set of genomes. By left clicking on that gene, we can select it, and know its gene product (gene products are assigned with Prokka).

Missing gene is Galactoside acetyltransferase. This gene is not in operons of a number of genomes. what are that genomes? Let's click on bypassing edge.

Click on the edge, selects it.

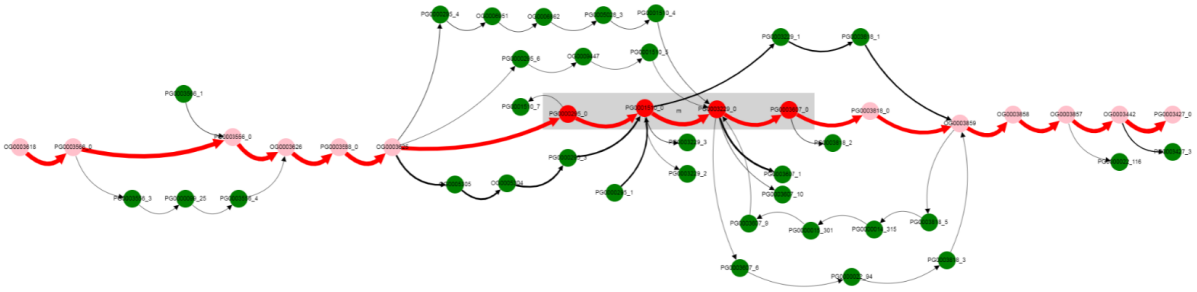
Selection of the edge results in two effects:

1) names of genomes corresponding to this edge appears in the `List of genomes` section below the graph
2) other edges, that contain at least one of the genomes from the selected edge, are colored blue. In this way, one can determine possible variants of gene neigbhorhoods, and in which genomes they are present.

For now we have determined, that a number of genomes does not contain Galactoside acetyltransferase. We can also notice nodes connected by a thin edges, which seems to represent other alternative variants of the operon. Let's click on that nodes and on the nodes from the reference, to see their products.

We see that their names are the same, but their length differs a lot: 263 b.p. for an outlier gene and 1253 b.p. for a reference gene. Often, and also in this particular case, it comes from frameshit splitting some genes into parts, some of which may become part of homology groups representing original gene.

Finaly to verify our findings let's switch to paralogues orthologization mode. To do it you should toggle `Draw paralogous` switcher on the top panel and click `Draw` button once more (be careful, your current graph layout will be lost, so consider opening new page). After clicking and dragging nodes it should be looking like this. A little bit scary.



This more complicated graph comes from not ignoring paralogous genes as it done by default, but instead showing all of them.

2.2.3 Combined analysis

With GCB, you can find which genes are in the hot spots of genome variability.

To do this, first select an organism, strain and replicon (chromosome or plasmids, complete genomes are reccomended to be used as reference).

Then, in complexity profile panel, click on some of the hot spots to set the current position. Current position is marked with vertical line in the complexity plot and also in `Start` and `End` coordinates in the left sliding panel).

Before proceding to the graph visualization, we recommend adjusting graph rendering options: set `Minimal edge value` to 10 (the more intense the hostpot, the bigger this value should be), `Window` to 10-20, depending on the hotspot width. Now press the `DRAW GRAH` button in the upper left corner of the *Complexity* panel. Changing colors will be visible above the graph draw buttons while it is being built, and then graph will apear in the *Grpah* panel.

To select some genes, for example, located at the variable region, press the left mouse button and while holding it move the cursor to surround the desired genes. Their products will apear in the bottom right part of the **Graph** panel.

Screen recording with these steps, and medium hot spot, is available here.

2.2.4 Publication-ready graph rendering

A graph-based representation of genome region can be exported in the form of JPEG image or a JSON file. To do this, fist draw some graph and then go to the left sliding panel, select *File* tab, select *Graph* subtab, click `DOWNLOAD JPEG` or `DOWNLOAD JSON` buttons.

JPEG file stores only a bitmap image, JSON file contains all infromation regarding the current graph, including its layout.

JSON file can be imported in Cytoscape for complete visualization control (customize the look of nodes, edges, do manual or one of the automatic layouts). Cytoscape graph renderings can be exported in a number of bitmap and vector formats (e.g., pdf, svg).

To import JSON graph file into Cytoscape, select `File` - `Import` - `Network from File` and select file, that was downloaded from GCB. Now you may arange graph nodes and adjust style.

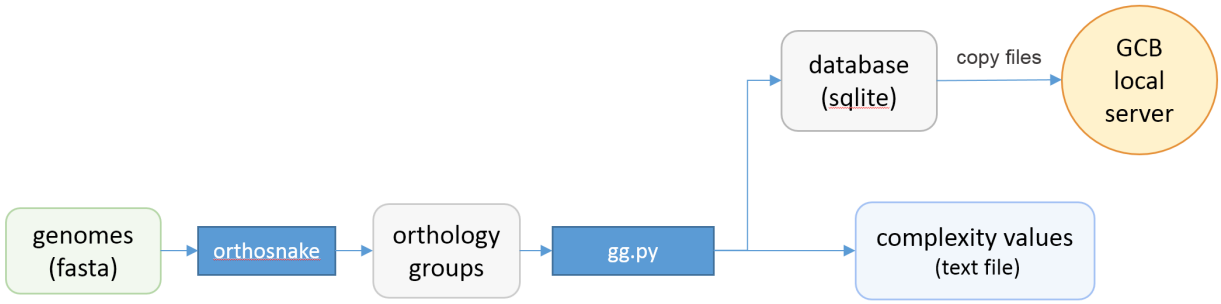
CHAPTER 3

Standalone version

3.1 Command-line tool

3.1.1 Roadmap

Standalone version should be used when the user wants to work with a custom set of genomes. Command-line scripts are provided to: calculate complexity profile, generate subgraphs, generate a database which can be imported to browser-based GUI application. Scheme of actions and scripts is shown below.



3.1.2 Prerequisites

Conda

Please refer to the installation guide. Select Python3 version.

Snakemake

Installation instrctions are here.

Python 3.6 or later

Comes with conda.

Graphviz and pygraphviz libraries

Can be installed via conda:

```
conda install -c anaconda graphviz
conda install -c conda-forge pygraphviz==1.7
```

gene_graph_lib Python3 module

Can be installed by running:

```
pip3 install gene_graph_lib
```

3.1.3 Orthogroup inference

Orthogroup inference is the first step in the standalone analysis. We recomend using our orthosnake pipeline for inference of orthogroups, because GCB requires some special formating of the files.

Orthosnake pipeline

INPUT: Fasta-formated files with .fna extension, one file per genome

OUTPUT: orthogroups file 'Orthogroups.txt' in OrthoFinder format

Steps:

1. Clone or download orthosnake GIT repository: https://github.com/paraslonic/orthosnake
2. Put fasta-formated genome files in `fna` folder of the orthosnake folder.
3. Rut snakemake with `--use-conda` and other appropriate options

The following code will run analysis of the test dataset (three plasmids):

```
git clone https://github.com/paraslonic/orthosnake.git
cd orthosnake
cp test_fna/* fna # copy test fasta files with three plasmids
snakemake -j 4 --use-conda
```

Here, we used snakemake with parameters specifying the number of available cores (`-j 4`), and using conda environments (`--use-conda`). During the first start of the pipeline, it will take about ten minutes to install the necessary programs into the conda environments.

If genome files have extension, other than `.fna` (e.g., `*.fasta`) , please rename them. For example, to change the file extension from `.fasta` to `.fna`, run:

```
for i in fna/*.fasta; do mv $i fna/$(basename $i .fasta).fna; done
```

Orthosnake pipeline performs the following steps:

- Fasta headers are modified, to satisfy the requirement of the Prokka
  - symbols other than alphanumericals and '_' are converted to '_'
  - if header is longer than 20 symbols, it is cropped to the first 18 symbols, and dots are added to the end (e.g., `gi|15829254|ref|NC_002695.1` becomes `gi|15829254|ref|NC..`)
- Annotation with Prokka.
- Genebank files converted to amino acid fasta files.
- Orthogroups are inferred with OrthoFinder.

When amino acid fasta files are generated from genebank files, information regarding location and product is included in headers in the following format: genome_name|numerical_id|start|end|product. Please note this if you want to perform orthogroups inference without using orthosnake.

3.1.4 Building a graph and complexity estimation

INPUT: orthogroups file 'Orthogroups.txt' in OrthoFinder format

OUTPUT: graph-based representation of genomes (sqlite database), complexity values (text file)

To perform the basic analysis, you should install geneGraph:

```
git clone https://github.com/DNKonanov/geneGraph
cd geneGraph
```

and run:

```
python3 gg.py -i [orthogroups file] -o [output] --reference [name of the reference␣
↪genome]
```

Parameters:

`-i` – path to orthogroups file in OrthoFinder format.

`-o` – path and name prefix for output files

`--reference` – name of the reference genome (its filename without extension), used for complexity estimation. This parameter can be skipped, in this case, the complexity values for all genomes in the set will be estimated (this can take a lot of time).

Optional parameters:

`--window` – size of the window used in the complexity estimation (default is 20). Only the paths inside the window increase the value of complexity.

`--genomes_list` – a text file containing the names of the genomes that will be used to calculate the complexity. Useful for analysis of parts of genomes, such as phylogenetic tree clades, for their subsequent comparison.

`--coalign` – a binary value (True/False) that determines whether to perform the step of selecting the optimal genomes orientation. It can take a lot of time when using a large number of draft genomes (5 hours for 1000 draft genomes).

Advanced complexity estimation algorithm settings (practically not used):

`--iterations` – number of iterations in algorithm (default is 500)

`--min_depth` – minimum length of deviating path (default is 0)

`--max_depth` – maximum length of deviating path (default is inf)

What this command does:

- parses *OrthoGroups.txt*
- creates graph file in sif-format and additional information about genes, their contextes, etc.
- creates SQLite database used in GCB
- computes complexity profiles for ALL genomes in the dataset and fills the DB
- dumps graph object for fast access
- all operations are executed by two ways: with deletion of paralogues, and with artificial orthologization

Main output files are:

- `<output>.db` - SQLite database containing graph and complexity values, paralogues genes are skipped.
- `<output>_pars.db` - SQLite database conatining graph and complexity values, paralogues genes are orthologized.
- `[reference genome]/prob_window_complexity_contig_[contig].txt` - text file containing complexity values for each contig in the reference genome.
- `<output>_context.sif` - number of unique contexts, computed for each node in graph
- `<output>_genes.sif` - list of all genes (nodes) from all genomes, with coordinates and Prokka annotations

Where, `<output>` is what was specifiend in `-o` option, while running *gg.py*.

3.2 Local GCB server

3.2.1 Installation

First, clone or download git repository:

```
git clone https://github.com/DNKonanov/GCB.git
```

Then, install dependencies:

```
pip3 install -r requirements.txt

conda install -c conda-forge graphviz pygraphviz==1.7

git clone https://github.com/paraslonic/orthosnake

git clone https://github.com/DNKonanov/geneGraph
```

3.2.2 Usage

Add data

To generate a dataset for GCB you need to run GeneGraph, a command-line tool to generate graphs and complexity profiles.

Suppose, we have a number of `*.fna` files for 100 different genomes of Mycoplasma.

First, orthogroup file `OrthoGroups.txt` should be generated with orthosnake. `OrthoGroups.txt` will be in `orthosnake/Results` folder.

Next, use `gg.py` script from geneGraph to generate GCB databases:

```
python gg.py -i [path to OrthoGroups.txt] -o Mycoplasma
```

Now move the created `Mycoplasma` folder to `GCB/data`. If there is no `data` folder in `GCB` root folder, just create it with `mkdir data`.

Start server

To start GCB server on your computer type in GCB_package folder this:

```
python3 gcb_server.py
```

Open **127.0.0.1:8000** or **localhost:8000** in your web-browser and use GCB.

Restart the sever after adding new datasets.

3.3 Complete step-by-step example

Here we will install all prerequisites and software needed for the analysis, and will add a new dataset to a local GCB server.

Following folders will be created:

`orthosnake` - Snakemake pipeline to infere orthogroups. We will download genomes from the RefSeq here.

`geneGraph` - console applications needed to build the graph-based represenatation of genomes.

`mpneumoniae` - a project folder, here the graph representation will be build for 5 *Mycoplasma pneumoniae* genomes.

`GCB` - local server application, this folder contains all datasets.

Note: we will work in the home directory, consider using other folders according to your preference.

Install conda

Install conda if you have not done it previously, for example by following miniconda installation guide.

Create conda environment

To create conda environment for GCB and install *snakemake* into this environment, run:

```
conda install -c conda-forge mamba
mamba create -c conda-forge -c bioconda -n gcb snakemake
conda activate gcb
```

Download orthosnake

Orthosnake will be needed to infere orthogroups:

```
cd ~
git clone https://github.com/paraslonic/orthosnake.git
cd orthosnake
```

Download genomes

Get file with RefSeq genomes information:

```
wget ftp://ftp.ncbi.nlm.nih.gov/genomes/refseq/assembly_summary_refseq.txt
```

Download 5 complete genomes of *M. pneumoniae* from the RefSeq database:

```
grep "Mycoplasma pneumoniae" assembly_summary_refseq.txt | grep "Complete" | awk -F
↪"\t" '{print $20}' | awk 'BEGIN{FS=OFS="/";filesuffix="genomic.fna.gz"}{ftpdir=$0;
↪asm=$10;file=asm"_"filesuffix;print ftpdir,file}' > complete_genomes.url
head -5 complete_genomes.url > selected_genomes.url
wget $(cat selected_genomes.url)
```

Move this genomes to `fna` folder (inside *orthofinder* folder):

```
gunzip *.gz
mv *.fna fna
```

Run orthofinder

Note: Installation of prokka and orthofinder will be performed during the first run.

Run orthofinder in 4 threads (will take around 20-30 minuttes):

```
snakemake -j 4 --use-conda
```

We have inferred orthogroups, they are located in `Results/Orthogroups.txt` file.

Download geneGraph console tools

geneGraph is needed to build a grpah-based representation of genomes:

```
cd ~
git clone https://github.com/DNKonanov/geneGraph.git
pip3 install gene_graph_lib
```

Build a graph-based representation of genomes:

```
cd ~
mkdir mpneumoniae
cd mpneumoniae
cp ~/orthosnake/Results/Orthogroups.txt .
python ~/geneGraph/gg.py -i Orthogroups.txt -o mycoplasma_pneumoniae
```

Analysis will take about a minute.

Install GCB local web server

GCB application is needed to visualize genomes in a graph-based form. All datasets are located in its `data` subfolder:

```
cd ~
git clone https://github.com/DNKonanov/GCB.git
cd GCB
pip3 install -r requirements.txt
conda install -c conda-forge graphviz pygraphviz==1.7
```

Copy results of geneGraph run into ‘‘data‘‘ folder:

```
mkdir data
cp -r ~/mpneumoniae/mycoplasma_pneumoniae/ data
```

Run local server:

```
python3 gcb_server.py
```

Open http://127.0.0.1:8000/ in a web browser, select *mycoplasma_pneumoniae* in `Organism` selecter in left panel, work with GCB.

CHAPTER 4

Frequently Asked Questions

4.1 What is this curvulin in the Complexity panel?

- It is a local genome variability profile, which we call complexity profile. The higher the values, the more often changes occurs in this region of genome. By change we mean: gene insertions/deletions/translocations. Large-scale changes (e.g., large inversions) have little effect on complexity value; large scale means larger than the *window* parameter. Complexity profiles calculated with window values 20, 50, 100 are available at gcb.rcpcm.org, to obtain complexity profiles with other windows sizes you need to use stand-alone version.

4.2 Why do you call it complexity, not variability?

- For two reasons. Firstly, only fixed variants are observed, so we measure not variability *per se*, but combination of genome variability and fitness cost of the genome changes. Secondly, not only the quantity, but also the pattern of the changes contributes to the complexity value. The more overlapping changes occur, the greater their contribution.

4.3 What are those circles and lines in the Graph panel?

- Circles correspond to the genes, or more precisely, to the orthogroups, inferred from the set of genomes. Lines (graph edges) connects genes, which are adjacent in at least one of the genomes (the larger the number of genomes these genes are located next to each other, the thicker the line).

4.4 How node name is selected?

- By majority rule. We consider how many times the name of a gene occurs in the names of genes within the same orthogroup, and we choose the most common one.

4.5 Does all variants that present in considered set of genomes are represented in the graph visualization form?

- Not always. GCB by default skips paralogues genes, but they can still be analysed by switching `Draw paralogues` toggle in the left slide bar.

4.6 Where should I take coordinates of genes of interest?

- from genome annotations (i.e. Genebank or RefSeq database) by searching for particular gene.
- from DOOR database of operons [Mao, Xizeng, et al., Nucleic acids research, 2014] (currently available here).
- from papers, in which genome coordinates are specified.

4.7 How can I obtain graph image in vector format for publication?

- Export graph to JSON (left panel->File->Graph->Download JSON), and then import it in a Cytoscape (File->Import->Network from File), or other graph editing software.

4.8 What if genome of interest is absent at gcb.rcpcm.org?

- You should use standalone application in this case. Command-line tools are available here, local webserver is available here.

4.9 Can I run standalone version on Windows?

- Most likely yes, via Windows Subsystem for Linux, but we only tested GCB on Linux (Ubuntu, CentOS).

4.10 What genome sets can be used in a standalone version?

- We recomend using phylogenetically narrow groups of genomes as input to the GCB analysis (species, and even better - clades of the phylogenetic tree consisting of several tens of genomes). In this case local variability of genomes is not confused by large-scale changes in the genome (e.g., inversions). If you have a genome of interest, it's a good idea to take in addition to it 50-100 closest genomes.

4.11 Can I use PC for the standalone analyisis or do I need a computational cluster?

- In case of bacterial (not viral) genomes, it is recommended to run orthology inference on computational cluster. However, you may try it, and if orthogroups inference will be completed, then it is OK to continue the analysis. Local server application can be run on a standart PC.

4.12 How many genomes may be analyzed in a standalone version?

- It depends on computational resources you have. If you have a computational cluster, then hundreds of genomes will be OK in most cases. If you have problems running analysis for your project feel free to contact us, we will try to help you (e.g., run orthology inference on our side).

4.13 Can I adress someone to help me with my organisms?

- Please, write to the google group.

CHAPTER 5

Contacts

If you have any question regarding the GCB please contact us via GCB Google Group.

Or directly, via e-mail:

Aleksander Manolov, paraslonic@gmail.com

Dmitriy Konanov, konanovdmitriy@gmail.com

- search