GENOME COMPLEXITY BROWSER

General considerations

Motivation

GCB tool enables the graph-based visual representation of gene context and calculation/visualization of genome variability profile. It is mainly suited for the analysis of prokaryotic and viral genomes.

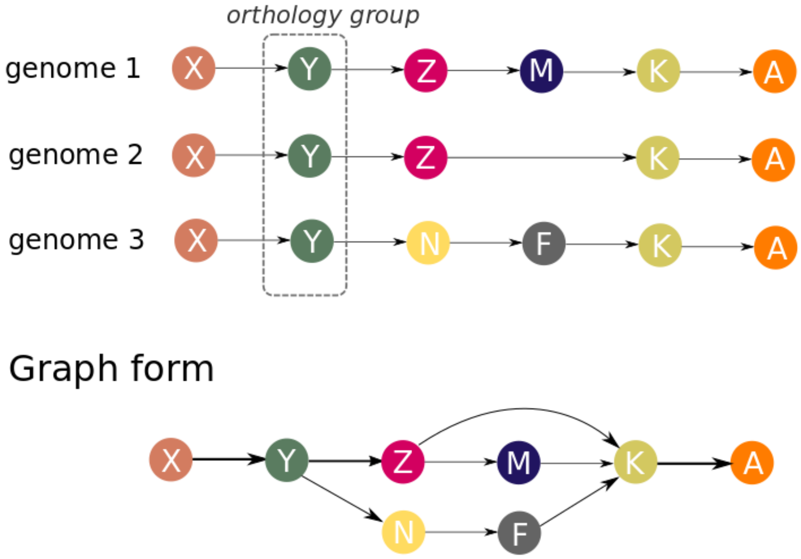
Graph representation of gene contexts facilitates answering such questions as:

- Is a gene (operon) located in all considered genomes in the same gene context? If not, then how many alternatives are present?
- Which parts of a gene set (operon) are conservative and which are variable? Which variants are present in different genomes?
- Which genomes contain this particular combination of gene order?

Genome variability (complexity) profile allows to detect hot spots of gene rearrangements - regions of the genome in which insertions, deletions or permutation of the gene(s) occur frequently, and cold spots - regions of the genome with almost no changes in the considered set of organisms.

Graph representation of gene order in a set of genomes

Let's consider three genomes in which orthology groups were inferred. To make a graph representation of the genome order in these genomes we will consider each orthology group 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.



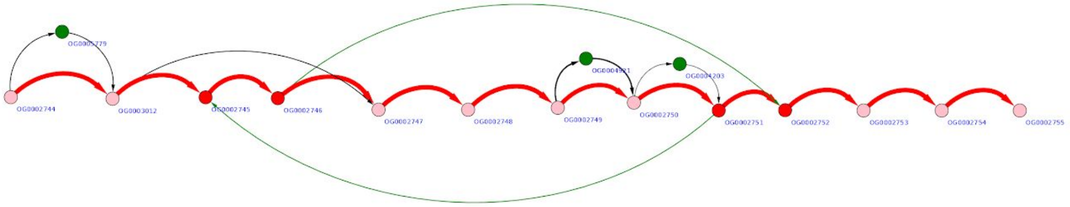
Genes are considered to be located on the same strand and are ordered by their centered position: (start + end)/2 on a replicon (contig). Each replicon (contig) is treated separately. Because gene order in a set of genomes is represented by a directed graph, genomes are aligned against each other prior to graph construction in order to optimize their orientation.

Paralogous genes can be orthologized and be incorporated in 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 .

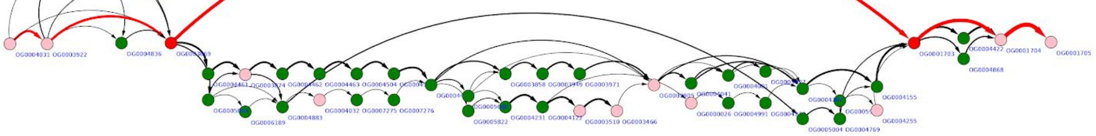
Complexity profile

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.

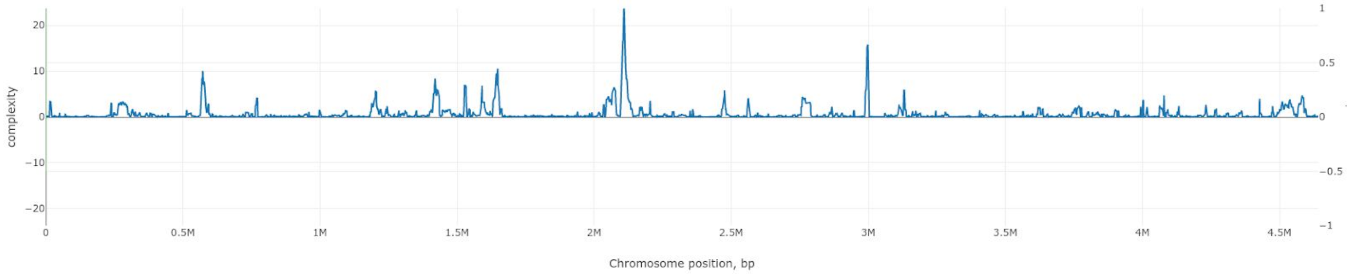
Example of a rather "quiet" region is below. Here three possible gene insertions, one deletion, and one gene transposition can be seen.



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 profile for *Escherichia coli K12 MG1655* genome is given below.

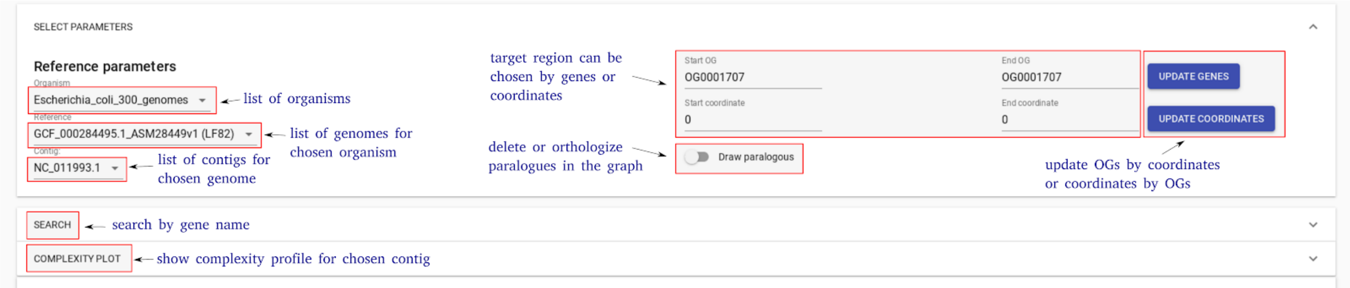


Regions of low and high complexity can be seen which corresponds to cold and hot spots of genome rearrangements, mainly due to the frequency of fixed horizontal gene transfer (HGT) events.

Interface elements and settings

GCB page consists of three main parts: 1) top panel to select genome and region to work with, 2) complexity plot which shows complexity profile for selected genome and contig, 3) subgraph visualization form.

The **top panel** allows selecting one of the precalculated organisms, reference genome and contig (replicon for finished assemblies). The right side of the panel allows selecting the region of the genome for the subgraph visualization. User can specify the start and end coordinates of the region. OG identifiers (which can be taken from the subgraph visualization) can also be used to define a range. Draw paralogues option changes the default program behavior which is to ignore genes which have paralogues. When this option is switched ON, orhologization of paralogous genes is applied to draw subgraph and calculate complexity. By default, paralogous genes are not displayed and not contribute to complexity value.



Organism: Escherichia_coli_300_genomes

Reference: GCF_000284495.1_ASM28449v1 (LF82)

Contig: NC_011993.1

Start OG: OG0001707

End OG: OG0001707

Start coordinate: 0

End coordinate: 0

list of organisms

list of genomes for chosen organism

list of contigs for chosen genome

target region can be chosen by genes or coordinates

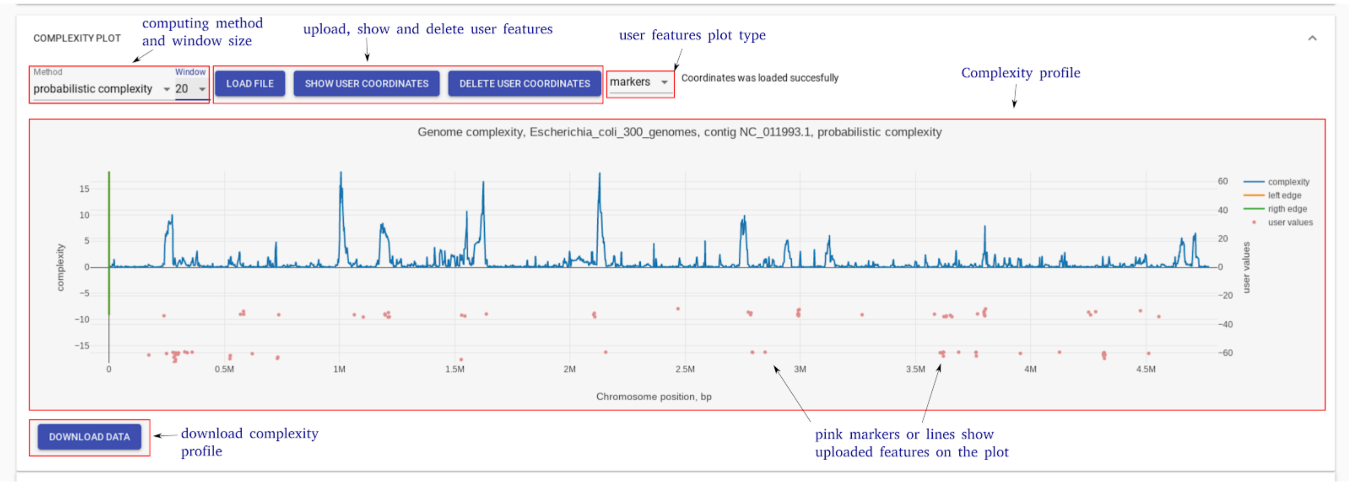
delete or orthologize paralogues in the graph

update OGs by coordinates or coordinates by OGs

search by gene name

show complexity profile for chosen contig

Complexity plot panel shows visualization of complexity profile and allows a user to visualize custom data (GC content, pathogenicity islands, prophage regions, sequence motifs, etc.). Complexity profile can be downloaded as a text file. User can add custom features file, in which each line should be in the format: <genome position> <numeric value>



Method: probabilistic complexity

Window: 20

markers

Coordinates was loaded succesfully

computing method and window size

upload, show and delete user features

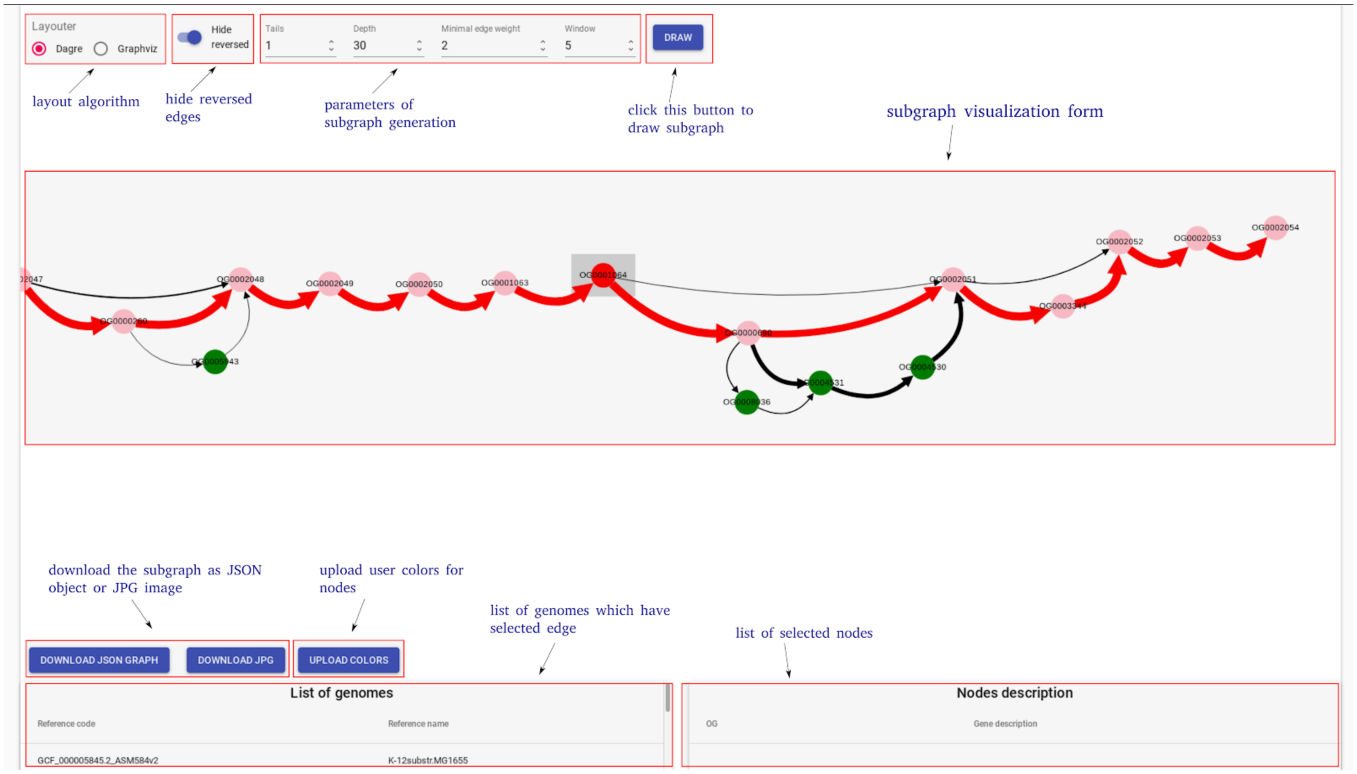
user features plot type

Complexity profile

download complexity profile

pink markers or lines show uploaded features on the plot

Subgraph visualization form contains a number of settings to simplify and customize subgraph. On the bottom, there is information about edges (which genomes contain the corresponding edge) and nodes (gene product).



Layouter: ● Dagre ○ Graphviz

Hide reversed: on

Tails: 1 | Depth: 30 | Minimal edge weight: 2 | Window: 5

layout algorithm

hide reversed edges

parameters of subgraph generation

click this button to draw subgraph

subgraph visualization form

download the subgraph as JSON object or JPG image

upload user colors for nodes

list of genomes which have selected edge

list of selected nodes

List of genomes

Reference code	Reference name
GCF_000005845.2_ASM584v2	K-12substr.MG1655

Nodes description

OG	Gene description

To explain parameters of visualization let's consider subgraph construction procedure: 1) nodes of the reference genome in selected region (+/- *window*) are added, this is called base chain; 2) all paths which connect with the base chain are added to subgraph; 3) paths which start and end on the base chain and which have length <= *Depth* remain unchanged; 4) all other paths are cropped to the *Tails* length; 5) edges with number of genomes (weight) lower than *minimal edge weight* are removed from the subgraph.

By clicking on the edge user selects this edge. List of genomes which contain gene pair corresponding to the selected edge is shown on the information panel. Edges which have at least one of the genomes from this list are colored blue.

User can upload colors for the nodes from the current subgraph. Each line of this file should be in the format:

<OG id> <hex color code>

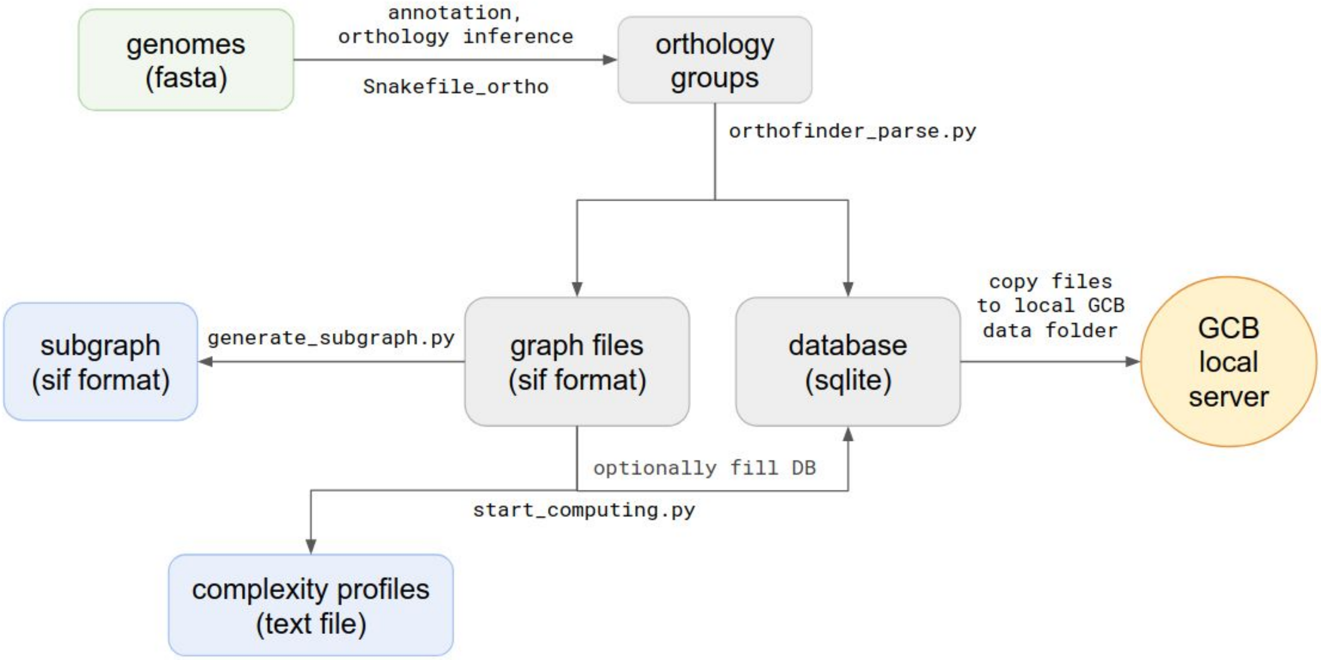
Example:

OG000004 #ff0000

OG000005 #777777

Standalone usage to add custom genome sets

GCB can be used in three main ways: 1) as a web server with precalculated set of genomes; 2) standalone application with browser based GUI; 3) set of command line scripts. Standalone version should be used when a custom set of genomes should be added. User can use command line scripts 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.



1. Orthology group inference

To add a custom set of genomes orthology groups should be inferred first. We provide snakemake script [] to accomplish this task. It takes fasta formatted genome sequences as input. Than gene annotation with prokka [github.com/tseemann/prokka] tool of each genome is performed. Genebank files than converted to fasta formatted amino acid protein sequences with a custom python3 script. This script is used to include special information about genes in fasta headers, namely: genome name, numeric id, product name, contig, start, end (for example, `>GCF_000007445|4|Threonine_synthase|NC_004431.1|4445|5731`). Then these files are used to infer orthology groups with OrthoFinder tool [github.com/davidemms/OrthoFinder]. The resulting file with orthology groups (OG) contains information about each OG in the format: <og id>: <gene1> <gene2> ... For example:

```
OG0008594: GCF_001618325|2406|Small_toxic_polypeptide_LdrD|NZ_CP015069.1|2607133|2607240
GCF_001663475|366|Small_toxic_polypeptide_LdrD|NZ_CP015159.1|380042|380149.
```

2. Generating of the graph structure

When orthology groups are calculated, the next step is parsing of Orthofinder outputs. To do this you should open `source` directory and type in terminal:

```
python orthofinder_parse.py -i [path to txt file with orthogroups] -o [path and name prefix for output files]
```

For example:

```
python orthofinder_parse.py -i ~/data/Mycoplasma/Results/Orthogroups.txt -o ~/data/outputs/Mycoplasma/graph
```

Output files:

- graph.sif - all edges list of the genomes graph
- graph.db - SQLite database with all parsed information
- graph_context.sif - number of unique contexts, computed for each node in the graph
- graph_genes.sif - list of all genes (nodes) from all genomes, with coordinates and Prokka annotations

The main graph structure is stored in a text graph.sif file, where each one line describes one edge, with its source and target nodes, genome id and contig id, to which this edge belongs. This file is used to create a graph object, which is used in all graph processing procedures.

3. Complexity computing

The next step is the computing of genome complexity. To do this type in terminal:

```
python start_computing.py -i graph.sif -o [path to output folder] --reference [name of reference genome]
```

Additional parameters:

- `--window` - sliding window size (default 20)
- `--iterations` - number of iterations in probabilistic method (default 500)
- `--genomes_list` - path to file with a list of names which will be used to create a graph (default all strains from *.sif will be used)
- `--min_depth, --max_depth` - minimum and maximum depth of generated paths in the graph (default from 0 to inf)

- `--save_db` - path to the database, created by orthfinder_parse.py (default data will not be saved to db, only to txt). It's necessary to use this parameter if you want to observe this complexity profile in the stand-alone GCB.

Output files for each contig in the reference genome:

- all_bridges_contig_n.txt

This file contains information about the number of deviating paths between each pair of nodes in the reference genome

- window_complexity_contig_n.txt

Table with window complexity values for each node in the reference genome

- IO_complexity_table_contig_n.txt

Table with number of deviating paths which start or end in this node

- prob_all_bridges_contig_n.txt
- prob_window_complexity_contig_n.txt
- prob_IO_complexity_table_contig_n.txt

These are the same files provided by the probabilistic algorithm to generate deviating paths

- main_chain_contig_n.txt

Just chain of nodes in the reference genome

- params.txt

Parameters that were used for computing (reference genome, iterations, window)

4. Generating of subgraph

So, we computed complexity for each gene in the reference genome. Let's suppose that we found some interesting node and we want to observe its context. We developed a script which allows us to draw genomes graph structure. But the current `graph.sif` file contains information about the full graph with too many nodes and edges to draw. So, to generate a small part of the graph you should use `generate_subgraph.py` script.

Common usage is

```
python generate_subgraph.py -i graph.sif -o subgraph --reference [name of reference genome] --start [name of start node] --end [name of end node]
```

This command generates subgraph `subgraph.sif`, which is connected with START......END simple chain of nodes in the reference genome.

Additional parameters:

- `--window` - number of nodes, added to the left and right side of reference chain (default 20)
- `--depth` - the maximum length of deviating paths, which will be added to the subgraph {default is the length of the reference chain)
- `--tails` - if deviating path too long, it will be replaced by the left and right "tails". This parameter is tails length (default 5)
- `--names_list` - path to file with a list of names for subgraph generating (default all names from *.sif will be used)

5. Graph drawing

Now we can run a drawing of our mini-graph. Let's go to the recombinatin_draw directory in the geneGraph folder and type in terminal:

```
python run_drawing.py -i subgraph.sif -o subgrah_img
```

This script generates:

- subgraph_img.ps file as image and
- subgraph_img.dot file with DOT description of subgraph (DOT is popular graph structure description language)

Additional parameters:

- `--freq_min` - minimal edge weight to draw. Edge weight is a number of genomes containing this edge.
- `--da` - legacy parameter, it's not recommended to use. Draws all subgraph edges in any case, but edges with weight < freq_min do not influence to subgraph layout.

6. Add user genomes group and complexity profiles to local GCB server.

After complexity computing, you can add your graph and complexity profiles added to DB (see `-save_db` parameter in `start_computing.py` script) to local GCB server. To do this create `data` folder in the GCB root directory if it doesn't exist and create a new directory in data with a name the same as the name of you sif file. Finally, you need to dump the graph object.

Suppose that we generated `Escherichia_coli_user.sif` file and computed complexity for a number of strains. To upload them to local GCB we need to:

- Create `GCB-master/data/` folder if it doesn't exist
- Create `GCB-master/data/Escherichia_coli_user` folder
- Copy or move `Escherichia_coli_user.sif` and `Escherichia coli_user.db` files to this folder
- Run in the terminal `python dump_scipt.py`
- Start or reload GCB server