UNIVERSIDADE FEDERAL DE JUIZ DE FORA

Programa de Pós Graduação em Modelagem Computacional

CGPGRN - User Guide 2.2

José Eduardo Henriques da Silva

July 3, 2023

Contents

1 Introduction **1**

1.1 CGPGRN pipeline 1

1.2 What's New? 2

1.3 CGPGRN structure 3

2 CGPGRN Usage **5**

2.1 Input Datasets 5

2.2 Arguments 6

2.3 Clustering Methods 7

2.3.1 Discretization Approaches 8

2.3.2 Spline Files 9

2.4 CGP 9

2.5 Multiple Executions 10

3 Post Processing **12**

4 Recommendations **13**

5 Installation Guide **14**

5.1 Windows 14

5.1.1 Bash 15

5.1.2 G++ Compiler 15

5.1.3 Make 15

5.1.4 OpenCL 15

5.2 Linux 15

6 Problem Solving **17**

6.1 ModuleNotFoundError 17

6.2 INVALID KERNEL 17

6.3 valueError 18

1 Introduction

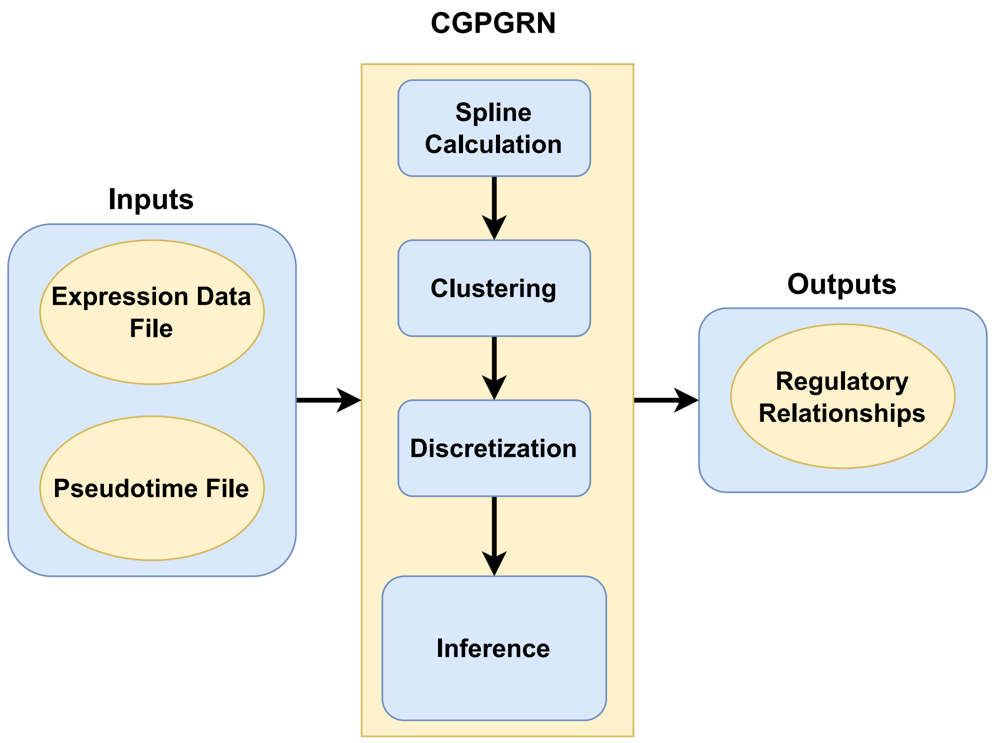
CGPGRN is a framework for inferring Gene Regulatory Networks (GRNs) using time-series data which is the result of a Ph.D. Thesis. Initially developed for single-cell data, CGPGRN can be used for any data-type technology.

This user guide contains information about the usage of the CGPGRN framework for inferring Gene Regulatory Networks from time-series data.

1.1 CGPGRN pipeline

The general CGPGRN framework pipeline is presented in Figure 1.

Figure 1: General CGPGRN framework pipeline.



Given expression data and pseudotime files, CGPGRN first calculates smoothing splines for each gene in order to obtain a single curve. This curve represents the expression level of the gene over time. Also, smoothing spline helps in removing or softening technical and biological variations. Then, it is possible to group the genes, according to a given similarity measure. Grouping genes helps in reducing the possible regulators for each gene. This step is called clustering. The user is able to choose between several grouping approaches and can also ignore the clustering step. After that, each gene expression is discretized in order to obtain transition state diagrams and, consequently, truth tables.

The truth tables are given to CGPGRN inference algorithm to evolve digital circuits. The evolved digital circuits represent the interactions (activation and inhibition) between genes. It is possible to choose between evolving one circuit for each gene or evolving the entire gene regulatory network depending on the selected clustering approach (KMeans, Agglomerative, and No Clustering). For the other clustering approaches, one circuit is evolved for each gene. Finally, the evolved circuits are analyzed to obtain the regulators for each target gene. The final results are presented in the form of a table with the regulator, target gene, and interaction strength of each regulatory relationship.

1.2 What's New?

The complete Change Log can be found at `https://github.com/jeduaardo/cgpgrn`.

- New general requirements: G++ Compiler, Make and Bash.
- added support for multiple outputs and don't care recognizing;
- -kd default changed to False
- -run default changed to False
- added -gsl argument for automatically generating spline list files (default = False)
- added makeFile.py in includes for automatically generating Makefile
- Added CGPGRN inference algorithm parametrization:
    - -cgpn defines the number of nodes
    - -cgpg defines the number of generations
- added timeStatistics for counting time in the entire framework
    - inference (mean and total)
    - complete framework (total)
- Minor fixes and more information on Log
- added mkdir function in utils
- all functions are now documented in utils

- noClustering now can be used with -fullTT
- fullBashScript is now generated in utils
    - automatically defines the number of nodes and generations of CGPGRN inference algorithm. However, the user still can define
- objects added for performing the CGPGRN framework main steps
    - spline
    - clustering
    - discretization
- DSSPD discretization implementation
- Support to multiple executions

1.3 CGPGRN structure

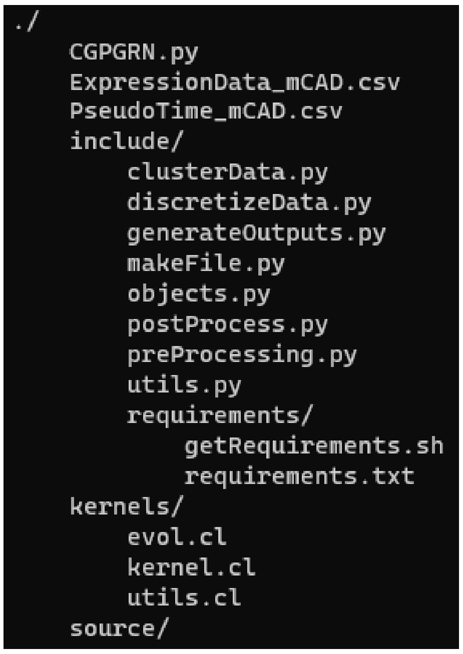
The folder and files structure of CGPGRN framework is presented in Figure 2a. Also, the source directory tree view is presented in Figure 2b.

The main path contains 3 files and 3 directories, as presented in Table 1. The description of the files in the folder *include* is presented in Table 2.

Name	Type	Description
CGPGRN.py	File	Main file of the CGPGRN framework
example	Dir	folder containing one expression data and one pseudotime file
include	Dir	folder with all scripts used with CGPGRN framework
kernels	Dir	folder with OpenCL kernels for CGP inference procedure
source	Dir	folder with CGPGRN inference algorithm source code

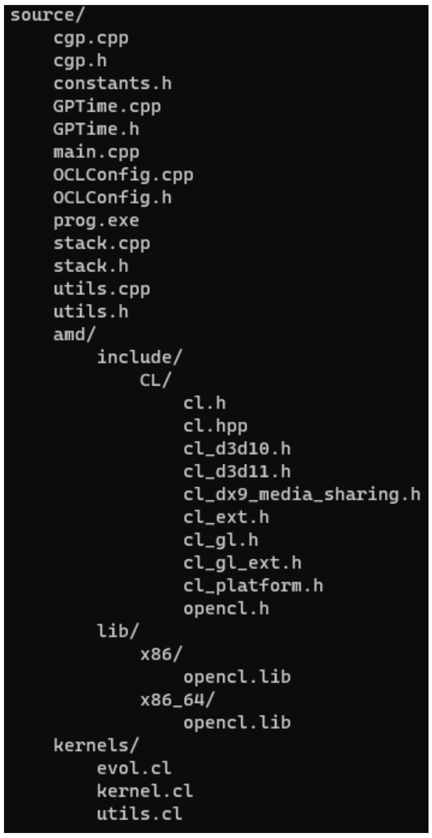
Table 1: Files and folders description of the main path of CGPGRN.

Figure 2: CGPGRN structure.



```
./
    CGPGRN.py
    ExpressionData_mCAD.csv
    PseudoTime_mCAD.csv
    include/
        clusterData.py
        discretizeData.py
        generateOutputs.py
        makeFile.py
        objects.py
        postProcess.py
        preProcessing.py
        utils.py
        requirements/
            getRequirements.sh
            requirements.txt
    kernels/
        evol.cl
        kernel.cl
        utils.cl
    source/
```

(a) Main (root) directory structure.



```
source/
    cgp.cpp
    cgp.h
    constants.h
    GPTime.cpp
    GPTime.h
    main.cpp
    OCLConfig.cpp
    OCLConfig.h
    prog.exe
    stack.cpp
    stack.h
    utils.cpp
    utils.h
    amd/
        include/
            CL/
                cl.h
                cl.hpp
                cl_d3d10.h
                cl_d3d11.h
                cl_dx9_media_sharing.h
                cl_ext.h
                cl_gl.h
                cl_gl_ext.h
                cl_platform.h
                opencl.h
        lib/
            x86/
                opencl.lib
            x86_64/
                opencl.lib
    kernels/
        evol.cl
        kernel.cl
        utils.cl
```

(b) Source directory structure.

Name	Description
clusterData.py	python script for clustering the data
discretizeData.py	python script for discretizing the data
generateOutputs.py	python script for generating the truth tables
makeFile.py	python script for generating the make file
objects.py	python script contaning the main objects of CGPGRN framework
postProcess.py	python script for post processing the data
utils.py	python script with utilities
requirements/getRequirements.sh	bash script for get/update the CGPGRN's required python libraries
requirements/requirements.txt	txt file with the name and version of python's required libraries

Table 2: Files in the folder include.

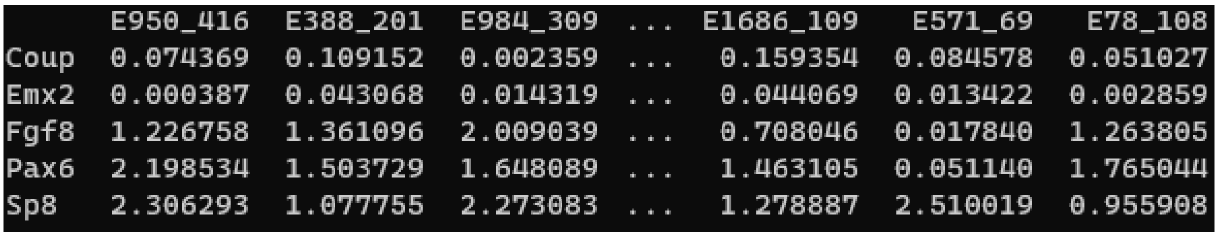
2 CGPGRN Usage

In the following subsections, the CGPGRN framework usage is explained. Furthermore, Section 4 presents recommendations of the arguments and parameters to be used with CGPGRN framework.

2.1 Input Datasets

The expression data must be in *csv* format with genes as rows and cell IDs as columns. An example of a simple expression data file is presented in Figure 3. This file example is also present in the main path of CGPGRN framework as an example (Expression-Data_mCAD.csv).

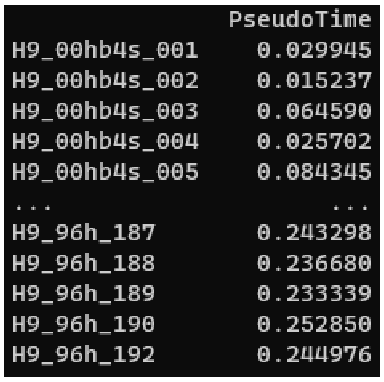
Figure 3: Expression Data file example. The file must contain genes as rows and cell IDs as columns.



```
      E950_416  E388_201  E984_309  ...  E1686_109   E571_69   E78_108
Coup  0.074369  0.109152  0.002359  ...   0.159354  0.084578  0.051027
Emx2  0.000387  0.043068  0.014319  ...   0.044069  0.013422  0.002859
Fgf8  1.226758  1.361096  2.009039  ...   0.708046  0.017840  1.263805
Pax6  2.198534  1.503729  1.648089  ...   1.463105  0.051140  1.765044
Sp8   2.306293  1.077755  2.273083  ...   1.278887  2.510019  0.955908
```

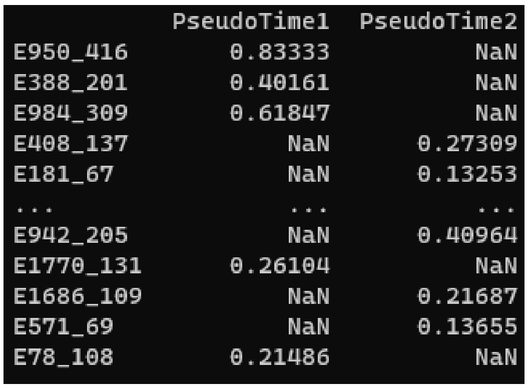
For Pseudotime files, csv format is also needed and columns are pseudotimes and rows, cell IDs. Figure 4 shows examples of pseudotime files with one pseudotime (a) and two pseudotimes (b). The PseudoTime_mCAD.csv present in the main path of CGPGRN framework is an example of pseudotime file with two pseudotimes.

Figure 4: Pseudotime file examples. Rows are cell IDs and columns are pseudotimes. NaN values are empty comma-separated fields.



```
                PseudoTime
H9_00hb4s_001     0.029945
H9_00hb4s_002     0.015237
H9_00hb4s_003     0.064590
H9_00hb4s_004     0.025702
H9_00hb4s_005     0.084345
...                    ...
H9_96h_187        0.243298
H9_96h_188        0.236680
H9_96h_189        0.233339
H9_96h_190        0.252850
H9_96h_192        0.244976
```

(a) Pseudotime file example with one pseudo-time.



```
            PseudoTime1  PseudoTime2
E950_416        0.83333          NaN
E388_201        0.40161          NaN
E984_309        0.61847          NaN
E408_137            NaN      0.27309
E181_67             NaN      0.13253
...                 ...          ...
E942_205            NaN      0.40964
E1770_131       0.26104          NaN
E1686_109           NaN      0.21687
E571_69             NaN      0.13655
E78_108         0.21486          NaN
```

(b) Pseudotime file example with two pseudo-times.

2.2 Arguments

All possible arguments for CGPGRN framework are presented in Table 3. Further explanation about some arguments is presented in the following subsections.

For example, to perform CGPGRN framework on a dataset with expression data file ExpressionData.csv, pseudotime file PseudoTime.csv using KMeans as clustering method testing number of clusters at a maximum of 10. The problem name is mCAD and the suffix is 500cells. The discretization is made through BiKMeans and CGPGRN inference algorithm will perform 5 independent runs.

python CGPGRN.py -e ExpressionData.csv -p PseudoTime.csv -cm KMeans -nc 10 -pn mCAD -s 500cells -d BiKMeans -r 5 -run

Argument	Description
-cm	The cluster method used for grouping genes. (See Section 2.3)
-nc	The number of maximum clusters to be tested within -cm. (See Section 2.3)
-t	The minimum correlation coefficient to be used within -cm. (See Section 2.3)
-e	Expression data file.
-p	PseudoTime file.
-pn	Problem Name (string).
-s	Suffix: an identificator of the current problem. (e.g. 2000cells)
-d	Discretization approach used for discretizing spline data. (See Section 2.3.1)
-r	Number of independent runs of CGPGRN inference algorithm.
-run	Run or not the CGPGRN inference algorithm (default = False).
-kd	Keep or not the source and spline data (default = False).
-fd	To use or not full discretization. (See Section 2.3.1)
-sf	To use or not a user given spline data file. (See Section 2.3.2)
-sl	Identificator for multiple given spline data files. (See Section 2.3.2)
-gsl	Generate spline list file. (See Section 2.3.2)
-fullTT	To use full truth table (one GRN for all genes). (See Section 2.3).
-cgpn	Number of CGPGRN inference algorithm nodes. (See Section 2.4)
-cgpg	Number of maximum generations for CGPGRN inference algorithm. (See Section 2.4)
-me	Perform multiple problem execution (See Section 2.5)

Table 3: All possible arguments for CGPGRN framework.

2.3 Clustering Methods

All implemented clustering methods are present in include/clusterData.py script. The available clustering approaches are presented in Table 4. Note that correlation methods require -t argument and clustering methods require -cm argument. Furthermore, when considering clustering methods, the maximum number of clusters to be tested (-nc) must be lower than the total number of genes present in the expression data file. If -nc is greater than the number of genes, CGPGRN framework will inform you.

It is possible to generate one GRN containing all genes or several partial GRNs for each gene and the final one is obtained by merging the partial GRNs. When using KMeans, Agglomerative, or None, one can generate one GRN, using -fullTT. If -fullTT is not

Method	Type	Description
Spearman	Correlation	Perform spearman correlation between the expression data of genes.
Pearson	Correlation	Perform pearson correlation between the expression data of genes.
KendallTau	Correlation	Perform kendall tau correlation between the expression data of genes.
KMeans	Clustering	Perform kmeans between the expression data of genes.
Agglomerative	Clustering	Perform agglomerative clustering between the expression data of genes.
None	-	Do not use grouping genes step. Default value.

Table 4: Possible values for -cm argument.

considered, the partial GRNs are generated, and then, the final merged GRN. However, only partial GRNs a final one obtained by merging these partial GRNs is possible for all other clustering approaches.

2.3.1 Discretization Approaches

All implemented discretization approaches are present in include/discretizeData.py script. The available discretization approaches are presented in Table 5. For further information about each discretization approach, we refer to [1]. For DSSPD, we refer to BRACIS se aprovado.

Method	Discretization
Mean	The expression data are divided into two groups based on the mean of expression data.
Median	The expression data are divided into two groups based on the median of expression data.
EFD	The expression data is divided into two groups with the same number of data points.
TSD	Discretization based on the difference between successive time points.
BiKMeans	KMeans is performed on both rows and columns to determine the discrete value.
DSSPD	Discretization based on distribution and successive spline points.

Table 5: Possible values for -d argument.

Note: The CGPGRN framework generates several files for each chosen clustering method. Some of them, such as correlation ones, generate one file per gene, containing the possible regulators for a given target gene. The discretization can be performed only on the spline files generated for each pseudotime or one can discretize every spline file per gene (computationally expensive). If you want to discretize every spline file you must use

-fd argument.

2.3.2 Spline Files

Spline files are generated through the pre-processing step of CGPGRN framework. However, you can give the framework your own spline files or store the spline files generated once to reuse them using different parameters.

If you have only one spline file, you must use -sf and pass the spline file name as an argument. Example:

python CGPGRN.py -cm Pearson -t 0.7 -e ExpressionData.csv -p PseudoTime.csv -pn ProblemName -s Suffix -d TSD -r 5 -sf splineFile.csv -run

However, if you want to use several spline files, for example when having multiple pseudotimes, you must pass a txt file containing one spline filename per line to -sf and use -sl to indicate that you are using a list of spline files. Example:

python CGPGRN.py -cm Pearson -t 0.7 -e ExpressionData.csv -p PseudoTime.csv -pn ProblemName -s Suffix -d TSD -r 5 -sf splineFileNames.txt -sl -run

Furthermore, this spline list file can be automatically generated if you use -gsl. Example:

python CGPGRN.py -cm KendallTau -t 0.7 -e ExpressionData.csv -p PseudoTime.csv -pn ProblemName -s Suffix -d BiKMeans -r 5 -run -gsl

Note: **It is important to notice that spline files must be at the same format as expression data files. Furthermore, if you have multiple pseudotimes, each given spline file must have an indentificator of pseudotime, separated by an underscore (e.g. splineData_pt0.csv, splineData_pt1.csv).**

2.4 CGP

CGPGRN framework uses Cartesian Genetic Programming (CGP) as evolutionary search technique for evolving combinational logic circuits. In CGP, programs are represented by a directed acyclic graph, encoded by a matrix of processing nodes. Each node contains inputs and a function that is performed on its inputs. Here, functions are boolean ones. A CGP's individual is presented in Figure 5.

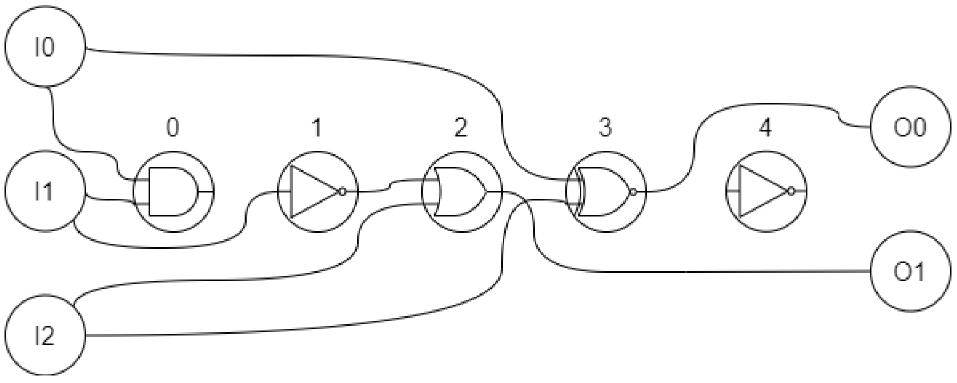


Figure 5: Representation of CGP individual with 3 inputs (I0, I1, and I2), 5 nodes (0, 1, 2, 3 and 4), and 2 outputs (O0 and O1).

Furthermore, a common stop criterion for evolutionary algorithms is a given maximum number of generations or the number of fitness function evaluations.

These parameters can be defined in CGPGRN framework by setting -cgpn for the number of nodes and -cgpg for the number of generations.

Note: **If the user doesn't give these arguments values, CGPGRN framework will automatically define these parameters.**

2.5 Multiple Executions

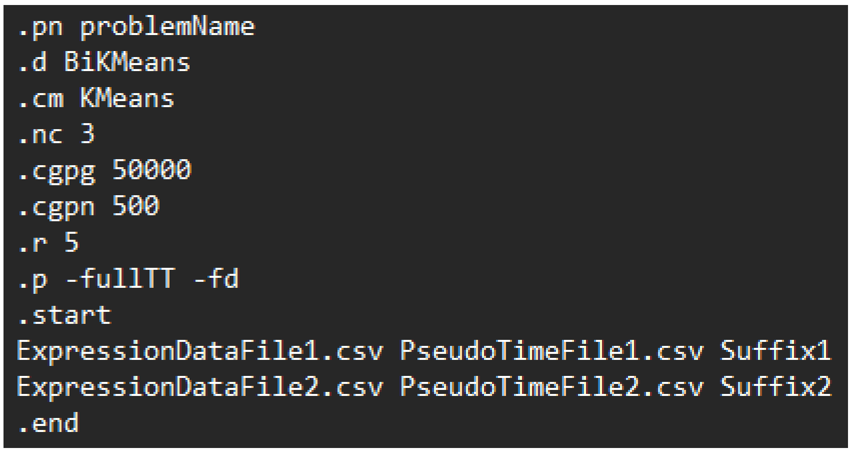
CGPGRN framework supports the execution of multiple problems. This feature generates a shell script file named *multipleExecutions.sh* and automatically executes the script. This is achieved by using the *-me* parameter. The usage is given as follows:

```
python CGPGRN.py -me multipleExecutionsFileName.txt
```

where *multipleExecutionsFileName.txt* is the file with the information of the problems.

This file must be formatted containing several information for automatically run the CGPGRN framework for each problem. An example of this file is presented in Figure 6. The list of parameters and their purposes is given in Table 6. All parameters must be identified by a dot and its name. The value of the parameter is given immediately after the declaration, with one empty space. Also, each parameter must be given in one separate line. The expression data and pseudotime files and the suffix must be given, for each problem, between *.start* and *.end* syntax. Furthermore, each expression data, pseudotime file and suffix must be separated by one empty space. Note that all problems will be run using the general parameters, given in the head of multiple execution file, before *.start*. If you pass an invalid argument, it will be held when performing the CGPGRN framework, which is automatically called after the multiple executions file processing. It is also possible to use the *.p* parameter to pass all boolean parameters presented in Table 3.

Figure 6: An example of multiple executions file.



```
.pn problemName
.d BiKMeans
.cm KMeans
.nc 3
.cgpg 50000
.cgpn 500
.r 5
.p -fullTT -fd
.start
ExpressionDataFile1.csv PseudoTimeFile1.csv Suffix1
ExpressionDataFile2.csv PseudoTimeFile2.csv Suffix2
.end
```

Parameter	Description	Required
.pn	Problem Name	True
.d	Discretization Approach	True
.r	Number of Independent Runs	True
.start	Defines the start of filenames and suffix	True
.end	Defines the end of filenames and suffix	True
.p	Other Parameters	False
.cm	Cluster Method	False
.nc	Number of Clusters	False
.t	Clustering Threshold	False
.cgpg	Number of CGP generations	False
.cgpn	Number of CGP nodes	False

Table 6: Multiple Executions File arguments.

3 Post Processing

CGPGRN framework performs a post-processing step on the inferred GRNs. In post process, three main steps occur:

- Unify the different pseudotime and independent runs ranked edges,
- Regulatory relationships duplicate removal, and
- Generate output folder for the current problem containing all files generated during the CGPGRN framework operation and cleaning the main path.

Ranked edges are files containing the strength of each regulatory relationship (Gene2 is regulated by Gene1 with the strength of EdgeWeight). This standard structure is used on [2] for comparing different GRN inference algorithms.

All results are stored in the folder named *finalResults_problemName*. If the folder already exists, a new folder with an index (e.g. finalResults_problemName_1) is created. The structure of directories in this folder is as follows:

- discretizedData: contains all discretized data files,
- executions_parallel: contains all evolved circuits from CGPGRN inference procedure,
- kmeans,pearson,spearman,kendalltau: one of these names (-cm) is created an contains the spline files for each clustering approach,
- shellScripts: contains all the shell scripts used within CGPGRN framework,
- sourceData: contains the source expression data and pseudotime files,
- splineData: contains the general spline files generated from source expression data and pseudotime files,
- time_counting: contains information about the spent time for evolving circuits in CGPGRN inference procedure,
- CGPGRN_parameters.txt: log file containing information of the parameters used for inferring the GRN,

- CGPGRN_times.txt: log file containing information of the time spent per step of the framework,
- LogFile_PROBLEMNAME.txt: log file of the steps performed by the framework,
- rankedEdges_problemName.csv: the final file containing the inferred regulatory relationships, and
- unified_rankedEdges_problemName.csv: a file containing the unified ranked edges inferred for each pseudotime and independent runs.

4 Recommendations

From all experiments performed using CGPGRN framework, we recommend the following parameters:

- -cm KMeans
- -nc 10
- -d BiKMeans
- -fullTT
- -r 1

Note that -nc 10 requires at least 11 genes. Using -fullTT, CGPGRN framework will generate one GRN containing all genes. Also, -fullTT automatically calculates the best parameters for CGP inference algorithm (number of nodes and generations). As standard, CGPGRN inference algorithm is run 5 times on each gene. Then, the full command line is given as:

```
python CGPGRN.py -cm KMeans -nc 10 -e ExpressionData.csv -p
PseudoTime.csv -pn ProblemName -s Suffix -d BiKMeans -r 1 -fullTT -run
```

5 Installation Guide

CGPGRN framework uses several python libraries, presented in Table 7. All these requirements can be found in *include/requirements/requirements.txt* and can be automatically installed using the getRequirements.sh bash script, present in the same folder. If you don't know how to run bash scripts on Windows, see Section 5.1.1. In Linux systems, usually, the third version of python is accessible by calling python3. For this reason, if your python is python3 in PATH, you must edit *getRequirements.sh*, replacing python with python3.

Furthermore, the CGPGRN inference method uses GPU OpenCL $\geq$ 1.2. We recommend a GPU with 2GB+ of dedicated video memory.

Library	Version
csaps	1.1.0
matplotlib	3.6.2
numpy	1.23.5
pandas	1.5.2
psutil	5.9.4
scikit_learn	1.2.0
scipy	1.10.0

Table 7: CGPGRN python libraries requirements.

Also, pandas==1.5.2 requires python-dateutils≥2.8.1, which is automatically upgraded with the bash script.

The following sections explain in detail how to meet all the requirements for using CGPGRN framework.

5.1 Windows

This section gives information about how to meet the requirements for running CGPGRN on Windows.

5.1.1 Bash

We recommend using Git Bash once you can use Git tools for cloning the CGPGRN repository. All Git Tools for Windows can be downloaded through the following link:

`https://gitforwindows.org/`

5.1.2 G++ Compiler

For compiling the CGPGRN inference procedure you must have G++ compiler. For Windows, we recommend MinGW. You can use MSYS2, downloadable via:

`https://www.mingw-w64.org/downloads/`

5.1.3 Make

For automatically compiling modifications of CGPGRN inference algorithm, CGPGRN framework uses make. For Windows, we recommend using Chocolatey. It can be downloaded through the link:

`https://chocolatey.org/install`

Once you have chocolatey installed, you must install make. For install make, open your command prompt and type:

choco install make

5.1.4 OpenCL

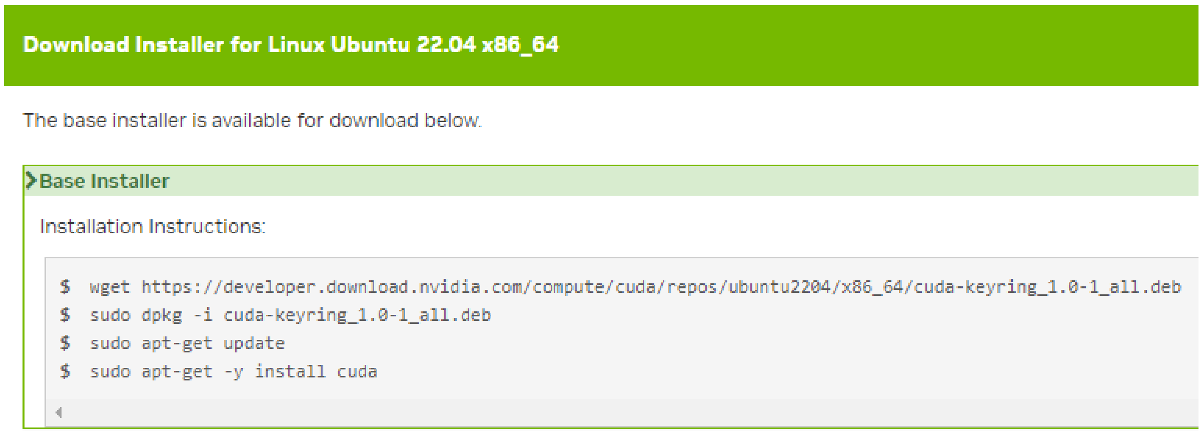
On Windows, you must download and install OpenCL, available in:

`https://github.com/GPUOpen-LibrariesAndSDKs/OCL-SDK/releases`

5.2 Linux

If you are using NVIDIA GPU, access `https://developer.nvidia.com/cuda-downloads?target_os=Linux` and properly select your Linux configurations. After selecting everything, the website will return to you the prompt commands needed to install the CUDA Toolkit. An example of Linux with architecture x86_64, using Ubuntu 22.04 to be installed with deb(network) is shown in Figure 7.

Figure 7: Example of the installation of CUDA Toolkit on Ubuntu 22.04 x86_64 deb(network).


Download Installer for Linux Ubuntu 22.04 x86_64

The base installer is available for download below.

Base Installer

Installation Instructions:

```
$ wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64/cuda-keyring_1.0-1_all.deb
$ sudo dpkg -i cuda-keyring_1.0-1_all.deb
$ sudo apt-get update
$ sudo apt-get -y install cuda
```

Note: When using -y you are accepting every prompt command question. It is highly recommended to not install drivers directly on the kernel because it can result in operating system initialization problems.

After that, just type the following command in your prompt to install nvidia open-cl.

sudo apt-get install nvidia-opencl-dev

For OpenCL on AMD processors and GPUs, type the following command in your prompt.

sudo apt-get install amd-opencl-dev

For more information, we refer to `https://subscription.packtpub.com/book/application-development/9781849692342/1/ch01lvl1sec12/an-example-of-opencl-program`.

To verify if OpenCL was properly installed, type *clinfo* in your prompt command. It will list your OpenCL available devices. If you don't have *clinfo* install it by typing the following command in your command.

sudo apt install clinfo

Note: if your *clinfo* still returns 0 as available devices, try to restart your computer.

If you further want to edit and compile the source code, you need to install the OpenCL headers by typing the following command in your prompt:

sudo apt-get install opencl-headers

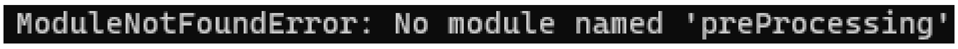
6 Problem Solving

This section provides detailed information on problem-solving when running the CGP-GRN framework.

6.1 ModuleNotFoundError

If your prompt return a value similar to that one presented in Figure 8, verify if your *include* folder is present in the main directory of CGPGRN.

Figure 8: An occurance of ModuleNotFoundError.



```
ModuleNotFoundError: No module named 'preProcessing'
```

This error commonly presents the name of the not found module. The *include* folder must contain all the scripts presented in Table 8.

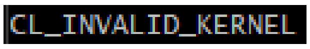
File Name
clusterData.py
discretizeData.py
generateOutputs.py
makeFile.py
objects.py
postProcess.py
preProcessing.py
utils.py

Table 8: Files that must be in *include* folder.

6.2 INVALID KERNEL

If your prompt return a value similar to that one presented in Figure 9, verify if your *kernels* folder is present in the main directory of CGPGRN.

Figure 9: An occurance of CL INVALID KERNEL error.



```
CL_INVALID_KERNEL
```

This error refers to not finding the OpenCL kernel. The *kernels* folder must contain all the files presented in Table 9.

File Name
evol.cl
kernel.cl
utils.cl

Table 9: Files that must be in *kernels* folder.

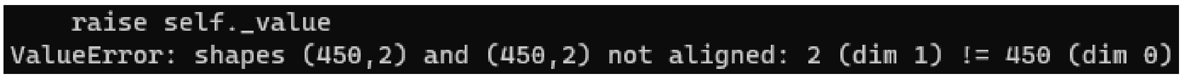
6.3 valueError

If your prompt return a value similar to that one presented in Figure 10, verify if there are any generated spline file in the main directory.

For performance reasons, spline is calculated in parallel. Then, the spline file is kept open during the spline procedure. If you keep a previous spline file, this one will have appended new spline values, generating a valueError.

For removing this error, just remove any previously generated spline file.

Figure 10: An occurance of valueError.



```
    raise self._value
ValueError: shapes (450,2) and (450,2) not aligned: 2 (dim 1) != 450 (dim 0)
```

References

[1] C. A. Gallo, R. L. Cecchini, J. A. Carballido, S. Micheletto, and I. Ponzoni, "Discretization of gene expression data revised," *Briefings in bioinformatics*, vol. 17, no. 5, pp. 758–770, 2016.

[2] A. Pratapa, A. P. Jalihal, J. N. Law, A. Bharadwaj, and T. Murali, "Benchmarking algorithms for gene regulatory network inference from single-cell transcriptomic data," *Nature methods*, vol. 17, no. 2, pp. 147–154, 2020.