git2net

Release 1.7.1

Christoph Gote

Jul 25, 2023

CONTENTS:

CHAPTER

ONE

OVERVIEW AND INSTALLATION

git2net is an Open Source Python package that facilitates the extraction of co-editing networks from git repositories.

1.1 Requirements

git2net is pure Python code. It has no platform-specific dependencies and thus works on all platforms. The only requirement is a version of *Git >= 2.0*.

1.2 Installing git2net

Assuming you are using pip, you can install latest version of *git2net* from the command-line by running:

```
$ pip install git2net
```

This command also installs the necessary dependencies. Among other dependencies, which are listed as *install_requires* in git2net's setup file, git2net depends on the python-Levenshtein package to compute Levenshtein distances for edited lines of code. On sytems running Windows, automatically compiling this C based module might fail during installation. In this case, unofficial Windows binaries can be found here, which might help you get started.

1.3 Contributing to git2net

The source code for git2net is available in a repository on GitHub which can be browsed at:

- https://github.com/gotec/git2net

If you find any bugs related to git2net please report them as issues there.

git2net is developed as an Open Source project. This means that your ideas and inputs are highly welcome. Feel free to share the project and contribute yourself. To get started, you can clone git2net's repository as follows:

```
$ git clone git@github.com:gotec/git2net.git
```

Now uninstall any existing version of git2net and install a local version based on the cloned repository:

```
$ pip uninstall git2net
$ cd git2net
$ pip install -e .
```

This will also install git2net's dependencies.

git2net provides a set of tests that you should run before creating a pull request. To do so, you will first need to unzip the test resitory they are based on:

```
$ unzip test_repos/test_repo_1.zip -d test_repos/
```

Then, you can run the tests with:

```
$ pytest
```

1.4 Citing git2net

```
@inproceedings{gote2019git2net,
    title={git2net: {M}ining time-stamped co-editing networks from large git␣
↪repositories},
    author={Gote, Christoph and Scholtes, Ingo and Schweitzer, Frank},
    booktitle={Proceedings of the 16th International Conference on Mining Software␣
↪Repositories},
    pages={433--444},
    year={2019},
    organization={IEEE Press}
}

@article{gote2021analysing,
    title={Analysing time-stamped co-editing networks in software development teams␣
↪using git2net},
    author={Gote, Christoph and Scholtes, Ingo and Schweitzer, Frank},
    journal={Empirical Software Engineering},
    volume={26},
    number={4},
    pages={1--41},
    year={2021},
    publisher={Springer}
}
```

1.5 License

This software is licensed under the GNU Affero General Public License v3 (AGPL-3.0).

CHAPTER

TWO

GETTING STARTED

A central aim of git2net is allowing you to conveniently obtain and visualise network projections of editing activity in git repositories. Let's have a look at an example on how you can achieve this:

```
import git2net
import pathpy as pp

github_url = 'gotec/git2net'
git_repo_dir = 'git2net4analysis'
sqlite_db_file = 'git2net4analysis.db'

# Clone and mine repository from GitHub
git2net.mine_github(github_url, git_repo_dir, sqlite_db_file)

# Disambiguate author aliases in the resulting database
git2net.disambiguate_aliases_db(sqlite_db_file)

# Obtain temporal bipartite network representing authors editing files over time
t, node_info, edge_info = git2net.get_bipartite_network(sqlite_db_file, time_from=time_
↪from)

# Aggregate to a static network
n = pp.Network.from_temporal_network(t)

# Visualise the resulting network
colour_map = {'author': '#73D2DE', 'file': '#2E5EAA'}
node_color = {node: colour_map[node_info['class'][node]] for node in n.nodes}
pp.visualisation.plot(n, node_color=node_color)
```

In the example above, we used three functions of git2net. First, we extract edits from the repository using *mine_github*. Then, we disambiguate author identities using *disambiguate_aliases_db*. Finally, we visualise the bipartite author-file network with *get_bipartite_network*.

Corresponding to the calls above, git2net's functionality is partitionied into three modules: *extraction*, *disambiguation*, *visualisation*, and *complexity*. We outline the most important functions of each module here. For a comprehensive details on all functions of git2net we refer to the API reference.

2.1 Tutorials

To help you get started, we provide an extensive set of tutorials covering different aspects of analysing your repository with *git2net*. You can directly interact with the notebooks in *Binder*, or view them in *NBViewer* via the links below.

- Open all tutorials in Binder
- Open all tutorials in NBViewer

In addition, we provide links to the individual tutorial notebooks in the tabs below:

Cloning

We show how to clone and prepare a git repository for analysis with git2net.

- Open the cloning tutorial in Binder
- Open the cloning tutorial in Google Colab
- Open the cloning tutorial in NBViewer

Mining

We introduce the mining options git2net provides and discuss the resulting SQLite database.

- Open the mining tutoral in Binder
- Open the mining tutoral in Google Colab
- Open the mining tutoral in NBViewer

Disambiguation

We explain how you can use git2net to disambiguate author aliases in a mined repository using gambit.

- Open the dismabiguation tutorial in Binder
- Open the dismabiguation tutorial in Google Colab
- Open the dismabiguation tutorial in NBViewer

Networks

We show how you can use git2net to generate various network projects of the edits in a mined repository.

- Open the network analysis tutorial in Binder
- Open the network analysis tutorial in Google Colab
- Open the network analysis tutorial in NBViewer

Database

We show how you can the database mined by git2net to answer elaborate questions on the activity in git repositories.

- Open the database analysis tutorial in Binder
- Open the database analysis tutorial in Google Colab
- Open the database analysis tutorial in NBViewer

Complexity

We show how you can use git2net to compute changes in files' complexity which can be used as a proxy for productivity.

- Open the database analysis tutorial in Binder
- Open the database analysis tutorial in Google Colab
- Open the database analysis tutorial in NBViewer

2.2 Usage Examples

We have published some motivating results as well as details on the mining algorithm in "git2net - Mining Time-Stamped Co-Editing Networks from Large git Repositories".

In "Analysing Time-Stamped Co-Editing Networks in Software Development Teams using git2net", we use *git2net* to mine more than 1.2 million commits of over 25,000 developers. We use this data to test a hypothesis on the relation between developer productivity and co-editing patterns in software teams.

Finally, in "Big Data = Big Insights? Operationalising Brooks' Law in a Massive GitHub Data Set", we mine a corpus containing over 200 GitHub repositories using *git2net*. Based on the resulting data, we study the relationship between team size and productivity in OSS development teams. If you want to use this extensive data set for your own study, we made it publicly available on zenodo.org.

CHAPTER

THREE

MODULES

git2net provides four modules—*extraction*, *disambiguation*, *visualisation*, and *complexity*.

3.1 Extraction

The module *extraction* contains all the functions that operate directly on a git repository. The most important functions in this module are:

- **mine_git_repo**: mines edits from a locally cloned git repositry to an SQLite database.
- **mine_github**: creates a local clone and mines edits from repository on GitHub to an SQLite database.
- **check_mining_complete**: checks if a repository has been fully mined.
- **mining_state_summary**: provides information on any commits that have not been fully mined.

Checkout the API reference for information on the complete list of available functions.

3.2 Disambiguation

The module *disambiguation* only contains a single function which allows you to disambiguate author identities. The disambiguation is based on the algorithm gambit.

- **disambiguate_aliases_db**: disambiguates author aliases in a database mined with git2net.

3.3 Visualisation

The *visualisation* module provides functions to generate various network projections based on the SQLite database created during the mining process.

- **get_coediting_network**: creates a co-editing network where nodes are authors who are connected by a directed link if they consecutively edited the same line of code. Links are directed from the previous to the subsequent author.
- **get_coauthorship_network**: creates a co-authorship network were nodes are authors who are connected by an undirected link if the edited the same file.
- **get_bipartite_network**: creates a bipartite network with nodes representing authors and files. Undirected links exist between an author and all files the author edited.

- **get_line_editing_paths**: creates paths for all lines in a repository. The paths contain ordered sequences of authors who subsequently edited a line. The number of paths generated for a line depends on the number of forks and merges the line was involved in.
- **get_commit_editing_dag**: creates a directed acyclic graph where nodes are commits. Commits are connected by a directed link if a one commit modifies lines last editied in another commit. Links are directed from the editing commit to the edited commit.

3.4 Complexity

The module *complexity* provides the functionality to compute a variety of complexity measures for the commits and files in a git repository. Specifically, for all commits, we compute the number of editing events (*events*) and the total Levenshtein edit distance (*levenshtein_distance*) for all modified files. In addition, we compute the Halstead effort (*HE*), the cyclomatic complexity (*CCN*), the number of lines of code (*NLOC*), the number of tokens (*TOK*), and the number of functions (*FUN*) in all modified files before (**_pre*) and after (**_post*) each commit. We further compute the change (**_delta*) for all complexity measures. As we show in this publication, the absolute value of the change in complexity can be used as a proxy for the productivity of developers in Open Source software projects.

- **compute_complexity**: computes complexity measures for all mined commit/file combinations in a database mined with git2net.

CHAPTER

FOUR

API REFERENCE

4.1 Extraction

git2net.extraction.**check_mining_complete**(*git_repo_dir*, *sqlite_db_file*, *commits=[]*, *all_branches=False*, *return_number_missing=False*)

Checks status of a mining operation

Parameters

- **git_repo_dir** (*str*) – path to the git repository that is mined
- **sqlite_db_file** (*str*) – path (including database name) where with sqlite database
- **commits** (*List[str]*) – only consider specific set of commits, considers all if empty

Returns

bool – True if all commits are included in the database, otherwise False

git2net.extraction.**get_commit_dag**(*git_repo_dir*)

Extracts commit dag from given path to git repository.

Parameters

git_repo_dir (*str*) – path to the git repository that is mined

Returns

pathpy.DAG – dag linking successive commits in the same branch

git2net.extraction.**get_unified_changes**(*git_repo_dir*, *commit_hash*, *file_path*)

Returns dataframe with github-like unified diff representation of the content of a file before and after a commit for a given git repository, commit hash and file path.

Parameters

- **git_repo_dir** (*str*) – path to the git repository that is mined
- **commit_hash** (*str*) – commit hash for which the changes are computed
- **file_path** (*str*) – path to file (within the repository) for which the changes are computed

Returns

pandas.DataFrame – pandas dataframe listing changes made to file in commit

git2net.extraction.**identify_file_renaming**(*git_repo_dir*)

Identifies all names and locations different files in a repository have had.

Parameters

git_repo_dir (*str*) – path to the git repository that is mined

Returns

- *pathpy.DAG* – pathpy DAG object depicting the renaming process
- *dict* – dictionary containing all aliases for all files

git2net.extraction.**is_binary_file**(*filename*, *file_content*)

Detects if a file with given content is a binary file.

Parameters

- **filename** (*str*) – name of the file including its file extension
- **file_content** (*str*) – content of the file

Returns

bool – True if binary file is detected, otherwise False

git2net.extraction.**mine_git_repo**(*git_repo_dir*, *sqlite_db_file*, *commits=[]*, *use_blocks=False*, *no_of_processes=2*, *chunksize=1*, *exclude=[]*, *blame_C=''*, *blame_w=False*, *max_modifications=0*, *timeout=0*, *extract_text=False*, *extract_merges=True*, *extract_merge_deletions=False*, *all_branches=False*)

Creates sqlite database with details on commits and edits for a given git repository.

Parameters

- **git_repo_dir** (*str*) – path to the git repository that is mined
- **sqlite_db_file** (*str*) – path (including database name) where the sqlite database will be created
- **commits** (*List[str]*) – only consider specific set of commits, considers all if empty
- **use_blocks** (*bool*) – determins if analysis is performed on block or line basis
- **no_of_processes** (*int*) – number of parallel processes that are spawned
- **chunksize** (*int*) – number of tasks that are assigned to a process at a time
- **exclude** (*List[str]*) – file paths that are excluded from the analysis
- **blame_C** (*str*) – string for the blame C option following the pattern "-C[<num>]" (computationally expensive)
- **blame_w** (*bool*) – bool, ignore whitespaces in git blame (-w option)
- **max_modifications** (*int*) – ignore commit if there are more modifications
- **timeout** (*int*) – stop processing commit after given time in seconds
- **extract_text** (*bool*) – extract the commit message and line texts
- **extract_merges** (*bool*) – process merges
- **extract_merge_deletions** (*bool*) – extract lines that are not accepted during a merge as 'deletions'

Returns

SQLite database will be written at specified location

git2net.extraction.**mine_github**(*github_url*, *git_repo_dir*, *sqlite_db_file*, *branch=None*, ***kwargs*)

Clones a repository from github and starts the mining process.

Parameters

- **github_url** (*str*) – url to the publicly accessible github project that will be mined can be priovided as full url or <OWNER>/<REPOSITORY>
- **git_repo_dir** (*str*) – path to the git repository that is mined if path ends with '/' an additional folder will be created
- **sqlite_db_file** (*str*) – path (including database name) where the sqlite database will be created
- **branch** (*str*) – The branch of the github project that will be checked out and mined. If no branch is provided the default branch of the repository is used.
- ****kwargs** – arguments that will be passed on to mine_git_repo

Returns

- git repository will be cloned to specified location
- SQLite database will be written at specified location

git2net.extraction.**mining_state_summary**(*git_repo_dir*, *sqlite_db_file*, *all_branches=False*)

Prints mining progress of database and returns dataframe with details on missing commits.

Parameters

- **git_repo_dir** (*str*) – path to the git repository that is mined
- **sqlite_db_file** (*str*) – path (including database name) where with sqlite database

Returns

pandas.DataFrame – dataframe with details on missing commits

git2net.extraction.**text_entropy**(*text*)

Computes entropy for a given text based on UTF8 alphabet.

Parameters

text (*str*) – string to compute the text entropy for

Returns

float – text entropy of the given string

4.2 Disambiguation

git2net.disambiguation.**disambiguate_aliases_db**(*sqlite_db_file*, *method='gambit'*, ***quargs*)

Disambiguates author aliases in a given SQLite database mined with *git2net*. The disambiguation is performed using the Python package *gambit*. Internally, *disambiguate_aliases_db* calls the function gambit.disambiguate_aliases.

Parameters

- **sqlite_db_file** (*str*) – path to SQLite database
- **method** (*str*) – disambiguation method from {"gambit", "bird", "simple"}
- ****quargs** – hyperparameters for the gambit and bird algorithms; **gambit**: thresh (*float*) – similarity threshold from interval 0 to 1, sim (*str*) – similarity measure from {'lev', 'jw'}, **bird**: thresh (*float*) – similarity threshold from interval 0 to 1

Returns

creates new column with unique *author_id* in the *commits* table of the provided database

4.3 Visualisation

git2net.visualisation.**get_bipartite_network**(*sqlite_db_file*, *author_identifier='author_id'*, *time_from=None*, *time_to=None*)

Returns temporal bipartite network containing time-stamped file-author relationships for given time window.

Parameters

- **sqlite_db_file** (*str*) – path to SQLite database
- **time_from** (*datetime.datetime*) – start time of time window filter, datetime object
- **time_to** (*datetime.datetime*) – end time of time window filter, datetime object

Returns

- *pathpy.TemporalNetwork* – bipartite network
- *dict* – info on node charactaristics, e.g. membership in bipartite class
- *dict* – info on edge characteristics

git2net.visualisation.**get_coauthorship_network**(*sqlite_db_file*, *author_identifier='author_id'*, *time_from=None*, *time_to=None*)

Returns coauthorship network containing links between authors who coedited at least one code file within a given time window.

Parameters

- **sqlite_db_file** (*str*) – path to SQLite database
- **time_from** (*datetime.datetime*) – start time of time window filter
- **time_to** (*datetime.datetime*) – end time of time window filter

Returns

- *pathpy.Network* – coauthorship network
- *dict* – info on node charactaristics
- *dict* – info on edge characteristics

git2net.visualisation.**get_coediting_network**(*sqlite_db_file*, *author_identifier='author_id'*, *time_from=None*, *time_to=None*, *engine='pathpy'*)

Returns coediting network containing links between authors who coedited at least one line of code within a given time window.

Parameters

- **sqlite_db_file** (*str*) – path to SQLite database
- **time_from** (*datetime.datetime*) – start time of time window filter
- **time_to** (*datetime.datetime*) – end time of time window filter

Returns

- *pathpy.TemporalNetwork* – coediting network
- *dict* – info on node charactaristics
- *dict* – info on edge characteristics

git2net.visualisation.**get_commit_editing_dag**(*sqlite_db_file*, *time_from=None*, *time_to=None*, *filename=None*)

Returns DAG of commits where an edge between commit A and B indicates that lines written in commit A were changed in commit B. Further outputs editing paths extracted from the DAG.

Parameters

- **sqlite_db_file** (*str*) – path to SQLite database
- **time_from** (*datetime.datetime*) – start time of time window filter, datetime object
- **time_to** (*datetime.datetime*) – end time of time window filter, datetime object
- **filename** (*str*) – filter to obtain only commits editing a certain file

Returns

- *pathpy.DAG* – commit editing dag
- *dict* – info on node charactaristics
- *dict* – info on edge characteristics

git2net.visualisation.**get_line_editing_paths**(*sqlite_db_file*, *git_repo_dir*, *author_identifier='author_id'*, *commit_hashes=None*, *file_paths=None*, *with_start=False*, *merge_renaming=False*)

Returns line editing DAG as well as line editing paths.

Parameters

- **sqlite_db_file** (*str*) – path to SQLite database mined with git2net line method
- **git_repo_dir** (*str*) – path to the git repository that is mined
- **commit_hashes** (*List[str]*) – list of commits to consider, by default all commits are considered
- **file_paths** (*List[str]*) – list of files to consider, by default all files are considered
- **with_start** (*bool*) – determines if node for filename is included as start for all editing paths
- **merge_renaming** (*bool*) – determines if file renaming is considered

Returns

- *pathpy.Paths* – line editing paths
- *pathpy.DAG* – line editing directed acyclic graph
- *dict* – info on node charactaristics
- *dict* – info on edge characteristics

4.4 Complexity

git2net.complexity.**compute_complexity**(*git_repo_dir*, *sqlite_db_file*, *no_of_processes=2*, *read_chunksize=1000000.0*, *write_chunksize=100*)

Computes complexity measures for all mined commit/file combinations in a given database. Computing complexities for merge commits is currently not supported.

Parameters

- **git_repo_dir** (*str*) – path to the git repository that is analysed
- **sqlite_db_file** (*str*) – path to the SQLite database containing the mined commits
- **no_of_processes** (*str*) – number of parallel processes that are spawned
- **read_chunksize** (*str*) – number of commit/file combinations that are processed at once
- **write_chunksize** (*str*) – number of commit/file combinations for which complexities are written at once

Returns

adds table *complexity* containing computed complexity measures for all commit/file combinations.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

g

INDEX