EDAMmap

Release 1.1.2-SNAPSHOT

Feb 06, 2023

Contents:

A tool for mapping various text input to EDAM ontology concepts. It is designed to assist not replace a curator.

CHAPTER 1

What is EDAMmap?

A tool for mapping various text input to EDAM ontology concepts. It is designed to assist not replace a curator.

Currently, it is mainly geared towards annotating bio.tools content, hence the structure of input parts: *tool name*, *keywords*, *description*, *publication IDs*, *link* and *documentation URLs*. The content of publications and web pages will be downloaded through the use of the PubFetcher library. However, EDAMmap could also be used on arbitrary text inputs of very different lengths, with results influenceable by a multitude of changeable parameters.

EDAMmap can be run on the command line, but also as a web server. For the latter case, a public web application and *API* are available.

1.1 Background

Longer (but somewhat outdated) background information can be found in the thesis where EDAMmap was initially developed (and the corresponding talk).

1.1.1 bio.tools

In the field of bioinformatics, there are numerous tools, databases and services for solving various biological problems. To provide a common portal for biologists that need to look for a certain tool, several projects gathering a vast amount of tools' metadata available under one web site have been launched. One such project is the ELIXIR tools and services registry bio.tools.

1.1.2 EDAM

Collecting the descriptions of tools into one common place is not enough – to be useful the entries need clear and accurate curation. In addition, to semantically connect meta-data, simplifying the organisation and merging of resources and providing better browse and search capabilities, the tools should be annotated in a standardised way, e.g. using an ontology. In the case of bio.tools, we use the EDAM ontology, which is a simple ontology of well-established concepts that are prevalent in the field of bioinformatics organised into an intuitive hierarchy. EDAM is divided into 4 branches (topic, operation, data and format) and each term has a preferred label, synonyms, longer definition, etc.

1.1.3 EDAMmap

So far, the process of annotating bio.tools entries with EDAM terms had been a manual affair, being both time-consuming and prone to mistakes due to unfamiliarity with the annotated tools or EDAM. The process can partly be automatised by EDAMmap: as input it will take tool description parts (for example name, description, publication IDs, link and documentation URLs from bio.tools), fetch content corresponding to publication IDs and web page URLs, tokenise all parts and find EDAM terms whose parts (label, synonyms, definition, etc) best match with the input parts. Parameters influencing various aspects of the process (like influence of different mapping algorithm parts to the final score or how many terms to suggest to the user) have been tuned for usage with bio.tools content and EDAM ontology, but these can be changed by the user at will. EDAMmap is flexible – different input and ontology parts can be omitted and have very different lengths. But in the end, accurate annotation relies heavily on expert domain knowledge, so EDAMmap is intended to only enhance curation, not replace the curator.

1.2 Outline

In the *Manual*, instructions are given on how to obtain or generate *Setup* files required by EDAMmap, including the *IDF* files. In the *Input* section, the structure of the input submitted as the query is discussed, with the most common file types being *CSV* or biotoolsSchema compatible JSON. The many changeable parameters are discussed in *Parameters* and output results and result formats in *Results*. EDAMmap consists of 3 tools: *EDAMmap-CLI* to run mapping of multiple queries in parallel on the command-line, *EDAMmap-Server* enabling mapping of one query in a web application or through an API, and *EDAMmap-Util* for running many utility operations.

The *EDAMmap API* can consumed through the */api* endpoint, either by sending requests to the public instance https://biit.cs.ut.ee/edammap/api or by sending requests to a local instance set up by following the instructions under *EDAMmap-Server*. *Prefetching* can be used to pre-store the content of webpages, docs and publications for a quicker final mapping call. Possible error situations when using the API are described in *Error handling*.

Lastly, some *Ideas for future* are discussed.

1.3 Install

Installation instructions can be found in the project's GitHub repo at INSTALL.

1.4 Quickstart

Use the public web application at https://biit.cs.ut.ee/edammap/ by filling in the "name" and some other fields, e.g. some "links" and "publications", and by clicking on "MAP".

For command-line usage, some simple examples can be found under *EDAMmap-CLI*.

And for using the API, there are also a few *Examples*.

1.5 Repo

EDAMmap is hosted at https://github.com/edamontology/edammap.

1.6 Support

Should you need help installing or using EDAMmap, please get in touch with Erik Jaaniso (the lead developer) directly via the tracker.

1.7 License

EDAMmap is free and open-source software licensed under the GNU General Public License v3.0, as seen in COPY-ING.

CHAPTER 2

Manual

2.1 Setup

Compilation instructions can be found in INSTALL.

For running the mapping, the latest EDAM ontology is required (in OWL format) – it can be downloaded from http://edamontology.org/page.

The query input to EDAMmap can contain publication IDs and web page URLs, but not the actual content of publications and web pages – this will have to be fetched. This fetched content could be saved in a file for potential later reuse. More information about that file (an on-disk key-value store) can be found in PubFetcher's documentation: Database. To generate an initial empty database:

```
$ java -jar edammap-util-<version>.jar -db-init db.db
```

2.1.1 IDF

For potentially better mapping results, tf–idf weighting could be applied to raise the importance of more meaningful words. To use tf–idf, a file with normalised IDF scores of words is required. This file should be generated based on a large number of entries that are similar (or from a similar domain) to those later input to EDAMmap. As an example, IDF files generated based on all entries of bio.tools (at some point in time) are provided: biotools.idf (stemming has not been applied to words) and biotools.stemmed.idf (stemming has been applied).

If so wished, the IDF files based on bio.tools content can be generated from scratch instead of using the ones provided. This will take several hours.

First, all content from bio.tools can be downloaded with:

```
$ java -jar edammap-util-<version>.jar -biotools-full biotools.json
```

Next, content for publications, webpages and docs is fetched (`-db-fetch` is documented in PubFetcher's documentation at Get content):

```
$ java -jar edammap-util-<version>.jar -pub-query biotools.json --query-type biotools␣
↪-db-fetch db.db --log pub.log
$ java -jar edammap-util-<version>.jar -web-query biotools.json --query-type biotools␣
↪-db-fetch db.db --log web.log
$ java -jar edammap-util-<version>.jar -doc-query biotools.json --query-type biotools␣
↪-db-fetch db.db --log doc.log
```

or alternatively, with a single command:

```
$ java -jar edammap-util-<version>.jar -all-query biotools.json --query-type biotools␣
↪-db-fetch db.db --log all.log
```

Note: Fetching of content could be repeated multiple times in the span of a few days to get more complete content of publications, webpages and docs, as some absent information could be filled in subsequent fetches, when for example resources that were temporarily unavailable will be up again (while on the other hand, `-db-fetch` will not try to re-fetch content that is already deemed to be final in the database, thus saving time and resources).

And as last step, the wanted IDF files are generated:

```
$ java -jar edammap-util-<version>.jar -make-idf biotools.json db.db biotools.idf
$ java -jar edammap-util-<version>.jar -make-idf-stemmed biotools.json db.db biotools.
↪stemmed.idf
```

Another reason to generated own IDF files might be, that the inputs to be annotated with EDAMmap are from a different field and not meant for bio.tools. Then, the queries input from `biotools.json` should be replaced with the different collection of entries from that different domain.

2.2 Input

An input query fed to EDAMmap can have the following parts: *id*, *name*, *list of keywords*, *description*, *webpage URLs*, *documentation URLs*, *publication IDs*, *existing EDAM annotations*. The *name*, *keywords* and *description* are strings describing the tool to be annotated. The *id* can be used as an optional identificator for the tool and (unlike *name*) its content will not be fed to the mapping algorithm. Content corresponding to *URLs* and *IDs* will need to be fetched by leveraging the PubFetcher library. *Existing manual annotations* can be specified to do benchmarking of EDAMmap or for example to exclude already existing annotations from results.

How query parts are read from an input file depends on the query type of the input file (specified with `--type`).

For example, using `--type biotools` means that the input file is a JSON file containing entries adhering to the biotoolsSchema (as returned by https://bio.tools/api/tool?format=json). The tool *name* is found from `"name"`, *publication IDs* are picked from `"publication"`, *existing annotation* are found in `"topic"` and `"function"`, etc. Other, non-relevant fields in the JSON are ignored.

2.2.1 CSV

For self-generated input, using a generic CSV file should be easier. This can be specified with `--type generic` (or it can also be omitted, as it is the default).

The field delimiter character in the CSV file is `,`, the character used for escaping values where the field delimiter is part of the value is `"` and the character used for escaping quotes inside an already quoted value is also `"`. Lines are separated with `\n` (Unix end-of-line) and empty lines and lines beginning with `#` are skipped. The maximum number of characters allowed for any given value is 100000. Within fields, multiple *keywords*, *webpage URLs*, *documentation*

URLs, *publication IDs* and *existing annotation* can be separated with `|` (which means this character can't be used as part of the values of these query parts).

The first line of the CSV file must be the header line describing the columns, with the following content: `id,name,keywords,description,webpageUrls,docUrls,publicationIds,annotations`. But columns, along with their corresponding header entries, can be omitted, as not all query parts have to be be used for mapping. For example, to do simple one input string to EDAM term matching, only the *name* part could be filled. Or if only short descriptions are available about tools, only *description* and *name* could be filled (filling the *name* is mandatory). Also, the order of the fields in the header line can be changed (as long as column data matches with its header).

An example generic input CSV file `example.csv`, with only one tool called "g:Profiler", is the following:

```
name,keywords,description,webpageUrls,docUrls,publicationIds,annotations
g:Profiler,gene set enrichment analysis|Gene Ontology,"A web server for functional␣
↪enrichment analysis, and conversions of gene lists.",https://biit.cs.ut.ee/
↪gprofiler/,https://biit.cs.ut.ee/gprofiler/help.cgi,17478515|PMC3125778|10.1093/nar/
↪gkw199,http://edamontology.org/topic_1775|operation_2436|data_3021|http://
↪edamontology.org/format_1964
```

Note: Specifying the prefix `http://edamontology.org/` is optional for existing annotations.

Note: Only one ID can be specified for one publication, either a PubMed ID, a PubMed Central ID or a DOI (in the example, `17478515|PMC3125778|10.1093/nar/gkw199` are three different publications).

2.3 Parameters

Mapping can be influenced by various changeable parameters, which on the command line can be specified as `--parameter value`. Most of these parameters are documented under *EDAMmap API Parameters*.

Preprocessing parameters influence the tokenisation of the input. *Fetching* parameters influence the fetching of publications, webpages and docs. And *Mapping* parameters influence the mapping algorithm and outputting of the results (more about the mapping algorithm can be found in section 3.10 of the thesis).

In addition, there are some parameters that can't be changed through the API, but can be changed on the command line. These are the Fetching private parameters (from PubFetcher) and the Processing parameters, documented in the table below.

2.3.1 Processing

Parameter	Default	Description
`--fetching`	`true`	Fetch publications, webpages and docs (corresponding to given publication IDs, webpage URLs and doc URLs); if `false`, then only the database is used for getting them (if a database is given with `--db`)
`--db`		Use the given database for getting and storing publications, webpages and docs (corresponding to given publication IDs, webpage URLs and doc URLs); if a database is given, then it is queried first even if fetching is enabled with `--fetching` (and fetching is done only if required and possible for found database entry)
`--idf`		Use the given query *IDF* file (when stemming is not enabled); if not specified, weighting of queries with IDF scores will be disabled (when stemming is not enabled)
`--idfStemmed`		Use the given query *IDF* file (when stemming is enabled); if not specified, weighting of queries with IDF scores will be disabled (when stemming is enabled)

2.4 Results

The output results will contain the requested *matches* number (or less, if scores are too low) of best terms (described by their EDAM URI and label) from the requested *branches* ordered by *score* within each branch, output along with intermediate match scores. Depending on the output type, results can additionally contain extra information about *query part to concept part matches* that form the final score and contain also matched parent and child terms, the supplied *query*, the used *parameters* and information about the fetched *webpages*, *docs* and *publications*. Results can also contain benchmarking *measures* which might be helpful in evaluating the performance of EDAMmap and in choosing optimal parameter values (benchmarking results can only make sense if any existing manually added *annotations* were supplied with the query).

Note: Mapping to terms from the data and format branches does not work that well currently, therefore results from these branches are omitted by default.

Results can be output into a JSON file, a directory containing HTML files and/or a plain text file. The content and structure of the JSON output is documented under the *Response* section of the EDAMmap API documentation. If the JSON output is obtained through running EDAMmap on the command-line (instead of querying through the API), then the *type* in the JSON output will be `"cli"` instead of `"core"` or `"full"` and the *api*, *txt*, *html* and *json* fields will be missing, but otherwise the output structure will be the same as for the `"full"` API response. The HTML output will contain the same information as the `"full"` JSON output, but rendered in a nice way in a web browser with clickable links to outside resources.

The plain text output will contain minimal information besides the matched terms. After the initial header line labelling the columns it will contain one line for each matched term with the following tab-separated values:

query_id The *id* of the query

query_name The *name* of the tool in the query

edam_branch The EDAM branch the matched term is from (one of `topic`, `operation`, `data`, `format`)

edam_uri The EDAM URI of the matched term

edam_label The EDAM label of the matched term

edam_obsolete `true`, if the term is obsolete; `false` otherwise

best_one_query Name of the type of the best matched query part

best_one_concept Name of the type of the EDAM term part the best matched query part matched with

best_one_score If *mappingStrategy* is "average", then the match score of *best_one_query* and *best_one_concept* will be stored here. If *mappingStrategy* is not "average", then it will have a negative value.

without_path_score If *parentWeight* and *pathWeight* are above 0, then the non-path-enriched score will be stored here. Otherwise it will have a negative value.

score The final score of the match

test `tp`, if term was matched and also specified as existing annotation in the query; `fp`, if term was matched, but not specified as existing annotation in query; `fn`, if term was not matched, but was specified as existing annotation in query

In addition to these detailed results, when `--type biotools` is used to *input* a bio.tools JSON file (adhering to biotoolsSchema), then there is a supplementary option (`--biotools`) to output this bio.tools JSON file with the matched terms added to it (but without any extra information about the results). All values present in the input JSON will also be present in the output JSON, except for `null` and empty value which will be removed. New annotations from the topic branch will be added to the topic attribute of the output JSON and new annotations from the operation branch will be added under a new function group object. If requested, then new annotations from the data and format branches should be added under the `"input"` and `"output"` attributes of a function group, however EDAMmap can't differentiate between inputs and outputs. Thus, new terms from the data and format branches will be added as strings (in the form `"EDAM URI (label)"`, separated by `" | "`) to the note of the last function group object.

2.5 EDAMmap-CLI

EDAMmap can be run as a command-line tool with the input being a JSON or CSV local file or URL resource (with the file contents described in the *Input* section) and with the *results* being output to the specified JSON, HTML and/or plain text files. The query can consist of many tools and the mapping process will be multi-threaded.

All command-line arguments suppliable to EDAMmap can be seen with:

```
$ java -jar edammap-cli-<version>.jar -h
```

The output will be rather long, as it contains all parameters described in the *Parameters* section. In addition to these parameters, EDAMmap-CLI accepts arguments described in the following table (entries marked with * are mandatory).

Parameter	Parameter args	Default	Description
`--edam` or `-e` *	*<file path>*		Path of the EDAM ontology file
`--query` or `-q` *	*<file path or URL>*		Path or URL of file containing input queries of QueryType `--type`
`--type` or `-t`	*<QueryType>*	`generic`	Specifies the type of the query and how to output the results. Possible values: `generic`, `SEQwiki`, `msutils`, `Bioconductor`, `biotools14`, `biotools`, `server`.
`--output` or `-o`	*<file path>*		Text file to write results to, one per line. If missing (and HTML report also not specified), then results will be written to standard output.
`--report` or `-r`	*<directory path>*		Directory to write a HTML report to. In addition to detailed results, it will contain used parameters, metrics, comparisons to manual mapping, extended information about queries and nice formatting. The specified directory will be created and must not be an existing directory.
`--json` or `-j`	*<file path>*		File to write results to, in JSON format. Will include the same info as the HTML report.
`--biotools` or `-b`	*<file path>*		File to write results to, in bio.tools JSON format, confirming to biotoolsSchema. Available only for `--type biotools`, where the input JSON is copied to the output, but with new annotations found by EDAMmap added to the `"topic"` and `"function"` attributes.
`--reportPageSize`	*<positive integer>*	`100`	Number of results in a HTML report page. Setting to 0 will output all results to a single HTML page.
`--reportPaginationSize`	*<positive integer>*	`11`	Number of pagination links visible before/after the current page link in a HTML report page. Setting to 0 will make all pagination links visible.
`--threads`	*<positive integer>*	`4`	How many threads to use for mapping (one thread processes one query at a time)

So, for example, to map the example tool ("g:Profiler") defined in the *Input* section (in `example.csv`), the following command could be run:

```
$ java -jar edammap-cli-<version>.jar -e EDAM_1.21.owl -q example.csv -r gprofiler --
↪idfStemmed biotools.stemmed.idf -l gprofiler.log
```

Contents for webpages, docs and publications described in the query `example.csv` will be fetched (but not stored for potential later reuse, as no database file is specified), the IDF file `biotools.stemmed.idf` obtained in the *Setup* section (where words are stemmed as by default `--stemming` is `true`) will be used as an input to the mapping algorithm and *results* will be output to the HTML file `gprofiler/index.html`, with log lines of the whole process appended to `gprofiler.log`.

Another example is the mapping of the whole content of bio.tools:

```
$ java -jar edammap-cli-<version>.jar -e EDAM_1.21.owl -q biotools.json -t biotools -
↪o results.txt -r results -j results.json --threads 8 --fetching false --db db.db --
↪idfStemmed biotools.stemmed.idf --branches topic operation data format --matches 6 -
↪-log biotools.log
```

(continues on next page)

(continued from previous page)

```
```

The query `biotools.json` is the whole content of bio.tools as obtained with the `-biotools-full` command of *EDAMmap-Util*. Contents of webpages, docs and publications has been pre-fetched to the database file `db.db` (as described under *IDF*), thus `--fetching` is disabled. Results will be output as plain text to `results.txt`, as HTML files to the directory `results` and as JSON to `results.json`. Results will contain up to 6 term matches from each EDAM branch. As EDAMmap was run on the whole content of bio.tools, then the benchmarking results can be consulted to assess the performance and as webpages, docs and publications have been stored on disk, then EDAMmap can easily be re-run while varying the parameters to tune these results.

Note: The measures in the benchmarking results assume, that the annotations in bio.tools are correct, which is not always the case. The performacne of EDAMmap can still be assumed to be correlated with the benchmarking results, however care should be taken when looking at individual mapping results.

Instead of specifying the parameters as part of the command line, they could be stored in a configuration file. An initial configuration file, with all parameters commented out, can be generated with:

```
$ java -jar edammap-util-<version>.jar -make-options-conf options.conf
```

In the ensuing file, `#` should be removed from the front of all mandatory parameters and all parameters whose default value should be changed. In the configuration file, parameters and parameter values are separated by newline characters (instead of spaces). Now, EDAMmap can be run as:

```
$ java -jar edammap-cli-<version>.jar @options.conf
```

2.6 EDAMmap-Server

EDAMmap can also be run as a web server. A query can then be input with a HTML form in a web application or posted as JSON to an *API*. However, in contrast to *EDAMmap-CLI*, only one query at a time can be submitted this way.

All command-line arguments suppliable to an EDAMmap server can be seen with:

```
$ java -jar edammap-server-<version>.jar -h
```

In addition to *Processing* and Fetching private parameters, EDAMmap Server accepts arguments described in the following table (entries marked with * are mandatory).

Parameter	Parameter args	Default	Description
`--edam` or `-e` *	*<file path>*		Path of the EDAM ontology file
`--txt`	*<boolean>*	`true`	Output results to a plain text file for queries made through the web application. The value can be changed in the web application itself.
`--json`	*<boolean>*	`false`	Output results to a JSON file for queries made through the web application. The value can be changed in the web application itself.
`--baseUri` or `-b`	*<string>*	`http://localhost:8080`	URI where the server will be deployed (as schema://host:port)
`--path` or `-p`	*<string>*	`edammap`	Path where the server will be deployed (only one single path segment supported)
`--httpsProxy`			Use if we are behind a HTTPS proxy
`--files` or `-f` *	*<directory path>*		An existing directory where the results will be output. It must contain required CSS, JavaScript and font resources pre-generated with *EDAMmap-Util*.
`--fetchingThreads`	*<positive integer>*	8	How many threads to create (maximum) for fetching individual database entries of one query

To setup the server version of EDAMmap, a new directory with required CSS, JavaScript and font resources must be created:

```
$ java -jar edammap-util-<version>.jar -make-server-files files
```

If wanted (i.e. if `--db` will be used when running the server), an initial empty database for storing fetched webpages, docs and publications can also be created:

```
$ java -jar edammap-util-<version>.jar -db-init server.db
```

EDAMmap-Server can now be run with:

```
$ java -jar edammap-server-<version>.jar -b http://127.0.0.1:8080 -p edammap -e EDAM_
↪1.21.owl -f files --fetching true --db server.db --idf biotools.idf --idfStemmed␣
↪biotools.stemmed.idf --log serverlogs
```

The web application can now be accessed locally at http://127.0.0.1:8080/edammap and the *API* is at http://127.0.0.1:8080/edammap/api. How to obtain the IDF files `biotools.idf` and `biotools.stemmed.idf` is described in the *Setup* section. In contrast to the other EDAMmap tools, the server will not log to a single log file, but with `-l` or `--log` a directory can be defined where log files, that are rotated daily, will be stored. The log directory will also contain daily rotated access logs compatible with Apache's combined format.

A public instance of EDAMmap-Server is accessible at https://biit.cs.ut.ee/edammap, with the *API* at https://biit.cs.ut.ee/edammap/api.

2.7 EDAMmap-Util

EDAMmap includes a utility program to manage and fill database files with fetched content or otherwise setup prerequisites for other tools, etc. Many of its operations have already been used above, but this section is still included for completeness.

All command-line arguments suppliable to the utility program can be seen with:

```
$ java -jar edammap-util-<version>.jar -h
```

The list of options is very long, as EDAMmap-Util extends the CLI of PubFetcher, which means that the utility program can run all the same operations as PubFetcher-CLI can. In addition to functionality inherited from PubFetcher-CLI, operations described in the following table can be executed.

Parameter	Parameter args	Default	Description
`-pub-query`	*<file path/URL> <file path/URL> …*		Load all publication IDs found in the specified files of *QueryType* specified with `--query-type`. A file can either be local or a URL, in which case –timeout and –userAgent can be used to change parameters used to fetch it.
`-web-query`	*<file path/URL> <file path/URL> …*		Load all webpage URLs found in the specified files of *QueryType* specified with `--query-type`. A file can either be local or a URL, in which case –timeout and –userAgent can be used to change parameters used to fetch it.
`-doc-query`	*<file path/URL> <file path/URL> …*		Load all doc URLs found in the specified files of *QueryType* specified with `--query-type`. A file can either be local or a URL, in which case –timeout and –userAgent can be used to change parameters used to fetch it.
`-all-query`	*<file path/URL> <file path/URL> …*		Load all publication IDs, webpage URLs and doc URLs found in the specified files of *QueryType* specified with `--query-type`. A file can either be local or a URL, in which case –timeout and –userAgent can be used to change parameters used to fetch it.
`--query-type`	*<QueryType>*	`generic`	Specifies the type of the query files loaded using `-pub-query`, `-web-query`, `-doc-query` and `-all-query`. Possible values: `generic`, `SEQwiki`, `msutils`, `Bioconductor`, `biotools14`, `biotools`, `server`.
`-make-idf`	*<query path/URL> <database path> <IDF path>*		Make the specified IDF file from tokens parsed from queries of type `--make-idf-type` loaded from the specified query file. The tokens are not stemmed. Contents for publication IDs, webpage URLs and doc URLs found in queries are loaded from the specified database file. If `--make-idf-webpages-docs` is `true` (the default), then tokens from webpage and doc content will also be used to make the IDF file and if `--make-idf-fulltext` is `true` (the default), then tokens from publication fulltext will also be used to make the IDF file. If the specified query file is a URL, then `--timeout` and `--userAgent` can be used to change parameters used to fetch it. The fetching parameters `--titleMinLength`, `--keywordsMinSize`, `--minedTermsMinSize`, `--abstractMinLength`, `--fulltextMinLength` and `--webpageMinLength` can be used to change the minimum length of a usable corresponding part (parts below that length will not be tokenised, thus will not used to make the specified IDF file).
`-make-idf-nodb`	*<query path/URL> <IDF path>*		Make the specified IDF file from tokens parsed from queries of type `--make-idf-type` loaded from the specified query file. The tokens are not stemmed. Contents for publication IDs, webpage URLs and doc URLs found in queries are are not loaded and thus are not used to make the specified IDF file. If the specified query file is a URL, then `--timeout` and `--userAgent` can be used to change parameters used to fetch it.
`-make-idf-stemmed`	*<query path/URL> <database path> <IDF path>*		Make the specified IDF file from tokens parsed from queries of type `--make-idf-type` loaded from the specified query file. The tokens are stemmed. Contents for publication IDs, webpage URLs and doc URLs found in queries are loaded from the specified database file. If `--make-idf-webpages-docs` is true (the default), then tokens from webpage and doc content will also be used to make the IDF file and if `--make-idf-fulltext` is `true` (the default), then tokens from publication fulltext will also be used to make the IDF file. If the specified query file is a URL, then `--timeout` and `--userAgent` can be used to change parameters used to fetch it. The fetching parameters `--titleMinLength`, `--keywordsMinSize`, `--minedTermsMinSize`, `--abstractMinLength`, `--fulltextMinLength` and `--webpageMinLength` can be used to change the minimum length of a usable corresponding part (parts below that length will not be tokenised, thus will not used to make the specified IDF file).
`-make-idf-stemmed-nodb`	*<query path/URL> <IDF*		Make the specified IDF file from tokens parsed from queries of type `--make-idf-type` loaded from the specified query file. The tokens are stemmed. Contents for publication IDs, webpage URLs and doc URLs found in queries are are not loaded and thus are not used

Note: `-pub-query`, `-web-query`, `-doc-query`, `-all-query` and `--query-type` are not standalone operations, but are meant to be used as part of the Pipeline of operations inherited from PubFetcher, allowing to inject IDs read from formats not supported by PubFetcher itself.

CHAPTER 3

API

The EDAMmap API is consumed by sending a JSON request with HTTP POST. The main endpoint is */api*, which on the public instance translates to https://biit.cs.ut.ee/edammap/api.

JSON numbers and booleans are converted to strings internally. JSON objects are ignored (except under *bio.tools input*), meaning there is no hierarchy in the request JSON structure.

3.1 /api

The main endpoint is used for performing one mapping. The key-value pairs in the request JSON fall under two categories: *query data* and *parameters*.

3.1.1 Query data

The query data to be mapped can be supplied in two different ways: as strings or arrays of strings under field names mirroring the usual *EDAMmap input* names or as a *bio.tools input* JSON object (like a bio.tools entry in JSON format). In case data is specified using both ways, only data under the *bio.tools input* is used.

3.1.1.1 EDAMmap input

The following data can be given, with only the “name” being mandatory.

Key	Type	Description
name	string	Name of tool or service
keywords	array of strings	Keywords, tags, etc
description	string	Short description of tool or service
webpageUrls	array of strings	URLs of homepage, etc
docUrls	array of strings	URLs of documentations
publicationIds	array of strings/objects	PMID/PMCID/DOI of journal article Note: an article ID can be specified as a string `"<PMID>\t<PMCID>\t<DOI>"` or as an object (the only place besides *bio.tools input* where a JSON object is not ignored), wherein keys `"pmid"`, `"pmcid"`, `"doi"` can be used
annotations	array of strings	Existing annotations from EDAM

3.1.1.2 bio.tools input

Under the field name "tool", a JSON object adhering to biotoolsSchema can be specified. All values possible in bio.tools can be specified, but only values relevant to EDAMmap will be used. A few attributes are mandatory: name, description and homepage. The input will be mirrored under *tool* in the *response*, but with found EDAM terms added to it.

3.1.2 Parameters

3.1.2.1 Main

Parameter	Default	Description
version	`"1"`	API version. Currently, only one possible value: `"1"`.
type	`"core"`	Detail level of the *response*. Possible values: `"core"`, `"full"`.
txt	`false`	Also output results to plain text file. The location of the created file can be read from the *response*.
html	`false`	Also output results to HTML file. The location of the created file can be read from the *response*.

3.1.2.2 Preprocessing

Parameter	Default	Min	Description
numbers	`true`		Include/exclude freestanding numbers (i.e., that are not part of a word) in pre-processing
stopwords	`"lucene"`		Do stopwords removal as part of pre-processing, using the chosen stopwords list. Possible values: `"off"`, `"corenlp"`, `"lucene"`, `"mallet"`, `"smart"`, `"snowball"`.
stemming	`true`		Do stemming as part of pre-processing
minLength	`1`	`0`	When all pre-processing steps are done, tokens with length less to this length are removed

3.1.2.3 Fetching

The fetching parameters are implemented in PubFetcher and thus are described in its documentation: Fetching parameters.

3.1.2.4 Mapping

Parameter	Default	Min	Max	Description
branches	`["topic", "operation"]`			Branches to include. Can choose multiple at once from possible values: `"topic"`, `"operation"`, `"data"`, `"format"`.
matches	`5`	`0`		Number of best matches per branch to output. Output amount can be less than requested if not enough match final scores fulfill *score limits* requirement.
obsolete	`false`			Include matched obsolete concepts
replaceObsolete	`true`			Replace matched obsolete concepts with their best matched replacement defined in EDAM (with “replacedBy” or “consider”)
obsoletePenalty	`0.5`	`0.0`	`1.0`	The fraction of the final score that included or replaced obsolete concepts will get
doneAnnotations	`true`			Suggest concepts already used for annotating query. Parents and children of these concepts are not suggested in any case (unless `inferiorParentsChildren` is set to `true`).
inferiorParentsChildren	`false`			Include parents and children of a better matched concept in suggestion results

Mapping algorithm

Parameter	Default	Min	Max	Description
compoundWords	1	0		Try to match words that have accidentally been made compound (given number is maximum number of words in an accidental compound minus one). Not done for tokens from fulltext, doc and webpage. Set to 0 to disable (for a slight speed increase with only slight changes to the results).
mismatchMultiplier	2.0	0.0		Multiplier for score decrease caused by mismatch
matchMinimum	1.0	0.0	1.0	Minimum score allowed for approximate match. Not done for tokens from fulltext, doc and webpage. Set to `1` to disable approximate matching.
positionOffBy1	0.35	0.0	1.0	Multiplier of a position score component for the case when a word is inserted between matched words or matched words are switched
positionOffBy2	0.05	0.0	1.0	Multiplier of a position score component for the case when two words are inserted between matched words or matched words are switched with an additional word between them
positionMatchScaling	0.5	0.0		Set to `0` to not have match score of neighbor influence position score. Setting to `1` means linear influence.
positionLoss	0.4	0.0	1.0	Maximum loss caused by wrong positions of matched words
scoreScaling	0.2	0.0		Score is scaled before applying multiplier and weighting with other direction match. Setting to `0` or `1` means no scaling.
conceptWeight	1.0	0.0		Weight of matching a concept (with a query). Set to `0` to disable matching of concepts.
queryWeight	1.0	0.0		Weight of matching a query (with a concept). Set to `0` to disable matching of queries.
mappingStrategy	`"average"`			Choose the best or take the average of query parts matches. Possible value: `"best"`, `"average"`.
parentWeight	0.5	0.0		Weight of concept's parent when computing path enrichment. Weight of grandparent is `parentWeight` times `parentWeight`, etc. Set to `0` to disable path enrichment.
pathWeight	0.7	0.0		Weight of path enrichment. Weight of concept is `1`. Set to `0` to disable path enrichment.

IDF

Parameter	Default	Min	Description
conceptIdfScaling	`0.5`	`0.0`	Set to `0` to disable concept IDF. Setting to `1` means linear IDF weighting.
queryIdfScaling	`0.5`	`0.0`	Set to `0` to disable query IDF. Setting to `1` means linear IDF weighting.
labelSynonymsIdf	`false`		IDF weighting for concept label and synonyms
nameKeywordsIdf	`true`		IDF weighting for query name and keywords
descriptionIdf	`true`		IDF weighting for query description
titleKeywordsIdf	`true`		IDF weighting for publication title and keywords
abstractIdf	`true`		IDF weighting for publication abstract

Concept multipliers

Parameter	Default	Min	Max	Description
labelMultiplier	`1.0`	`0.0`	`1.0`	Score multiplier for matching a concept label. Set to `0` to disable matching of labels.
exactSynonymMultiplier	`1.0`	`0.0`	`1.0`	Score multiplier for matching a concept exact synonym. Set to `0` to disable matching of exact synonyms.
narrowBroadSynonymMultiplier	`1.0`	`0.0`	`1.0`	Score multiplier for matching a concept narrow or broad synonym. Set to `0` to disable matching of narrow and broad synonyms.
definitionMultiplier	`1.0`	`0.0`	`1.0`	Score multiplier for matching a concept definition. Set to `0` to disable matching of definitions.
commentMultiplier	`1.0`	`0.0`	`1.0`	Score multiplier for matching a concept comment. Set to `0` to disable matching of comments.

Query normalisers

Parameter	Default	Min	Max	Description
nameNormaliser	`0.81`	`0.0`	`1.0`	Score normaliser for matching a query name. Set to `0` to disable matching of names.
keywordNormaliser	`0.77`	`0.0`	`1.0`	Score normaliser for matching a query keyword. Set to `0` to disable matching of keywords.
descriptionNormaliser	`0.92`	`0.0`	`1.0`	Score normaliser for matching a query description. Set to `0` to disable matching of descriptions.
publicationTitleNormaliser	`0.91`	`0.0`	`1.0`	Score normaliser for matching a publication title. Set to `0` to disable matching of titles.
publicationKeywordNormaliser	`0.77`	`0.0`	`1.0`	Score normaliser for matching a publication keyword. Set to `0` to disable matching of keywords.
publicationMeshNormaliser	`0.75`	`0.0`	`1.0`	Score normaliser for matching a publication MeSH term. Set to `0` to disable matching of MeSH terms.
publicationMinedTermNormaliser	`1.0`	`0.0`	`1.0`	Score normaliser for matching a publication mined term (EFO, GO). Set to `0` to disable matching of mined terms.
publicationAbstractNormaliser	`0.985`	`0.0`	`1.0`	Score normaliser for matching a publication abstract. Set to `0` to disable matching of abstracts.
publicationFulltextNormaliser	`1.0`	`0.0`	`1.0`	Score normaliser for matching a publication fulltext. Set to `0` to disable matching of fulltexts.
docNormaliser	`1.0`	`0.0`	`1.0`	Score normaliser for matching a query doc. Set to `0` to disable matching of docs.
webpageNormaliser	`1.0`	`0.0`	`1.0`	Score normaliser for matching a query webpage. Set to `0` to disable matching of webpages.

Query weights

Parameter	Default	Min	Description
averageScaling	`10.0`	`0.0`	Scaling for the average strategy
nameWeight	`1.0`	`0.0`	Weight of query name in average strategy. Set to `0` to disable matching of names in average strategy.
keywordWeight	`1.0`	`0.0`	Weight of query keyword in average strategy. Set to `0` to disable matching of keywords in average strategy.
descriptionWeight	`1.0`	`0.0`	Weight of query description in average strategy. Set to `0` to disable matching of descriptions in average strategy.
publicationTitleWeight	`0.25`	`0.0`	Weight of publication title in average strategy. Set to `0` to disable matching of titles in average strategy.
publicationKeywordWeight	`0.75`	`0.0`	Weight of publication keyword in average strategy. Set to `0` to disable matching of keywords in average strategy.
publicationMeshWeight	`0.25`	`0.0`	Weight of publication MeSH term in average strategy. Set to `0` to disable matching of MeSH terms in average strategy.
publicationMinedTermWeight	`0.25`	`0.0`	Weight of publication mined term (EFO, GO) in average strategy. Set to `0` to disable matching of mined terms in average strategy.
publicationAbstractWeight	`0.75`	`0.0`	Weight of publication abstract in average strategy. Set to `0` to disable matching of abstracts in average strategy.
publicationFulltextWeight	`0.5`	`0.0`	Weight of publication fulltext in average strategy. Set to `0` to disable matching of fulltexts in average strategy.
docWeight	`0.5`	`0.0`	Weight of query doc in average strategy. Set to `0` to disable matching of docs in average strategy.
webpageWeight	`0.5`	`0.0`	Weight of query webpage in average strategy. Set to `0` to disable matching of webpages in average strategy.

Score limits

Parameter	Default	Min	Max	Description
goodScoreTopic	`0.63`	`0.0`	`1.0`	Final scores over this are considered good (in topic branch)
goodScoreOperation	`0.63`	`0.0`	`1.0`	Final scores over this are considered good (in operation branch)
goodScoreData	`0.63`	`0.0`	`1.0`	Final scores over this are considered good (in data branch)
goodScoreFormat	`0.63`	`0.0`	`1.0`	Final scores over this are considered good (in format branch)
badScoreTopic	`0.57`	`0.0`	`1.0`	Final scores under this are considered bad (in topic branch)
badScoreOperation	`0.57`	`0.0`	`1.0`	Final scores under this are considered bad (in operation branch)
badScoreData	`0.57`	`0.0`	`1.0`	Final scores under this are considered bad (in data branch)
badScoreFormat	`0.57`	`0.0`	`1.0`	Final scores under this are considered bad (in format branch)
outputGoodScores	`true`			Output matches with good scores
outputMediumScores	`true`			Output matches with medium scores
outputBadScores	`false`			Output matches with bad scores
passableBadScoreInterval	`0.04`	`0.0`	`1.0`	Defines the passable bad scores (the best bad scores) as scores falling inside a score interval of given length, where the upper bound is fixed to the bad score limit
passableBadScoresInTopN	`3`	`0`		If a match with passable bad score would be among the top given number of matches, then it is included among the suggested matches (note that matches with any bad score are always included if `outputBadScores` is `true`)

3.1.3 Response

The response output can contain more or less information, depending on the specified *type*. The section of most interest is probably *results* in *core*.

3.1.3.1 core

success `true` (if `false`, then the JSON output of *Error handling* applies instead of the one below)

version `"1"`

type `"core"`

api URL of endpoint where request was sent

txt Location of plain text results file (or `null` if not created)

html Location of HTML results directory (or `null` if not created)

json Location of JSON results file

generator Information about the application that generated the response

- **name** Name of the application
- **url** Homepage of the application
- **version** Version of the application

time

- **start** Start time of mapping as UNIX time (in milliseconds)
- **startHuman** Start time of mapping as ISO 8601 combined date and time
- **stop** Stop time of mapping as UNIX time (in milliseconds)
- **stopHuman** Stop time of mapping as ISO 8601 combined date and time
- **duration** Duration of mapping in seconds

mapping

- **query**
  - **id** Unique ID assigned to the query (and by extension, to this response)
  - **name** Name of tool or service (as specified in *query data*, `null` if not specified)
  - **keywords** Array of strings representing keywords, tags, etc (as specified in *query data*, `null` if not specified)
  - **description** Short description of tool or service (as specified in *query data*, `null` if not specified)
  - **webpageUrls** Array of strings representing URLs of homepage, etc (as specified in *query data*, `null` if not specified)
  - **docUrls** Array of strings representing URLs of documentations (as specified in *query data*, `null` if not specified)
  - **publicationIds** Array of objects representing IDs of journal articles (as specified in *query data*, `null` if not specified)
    - **pmid** PMID of article
    - **pmcid** PMCID of article

doi DOI of article

annotations Array of EDAM URI strings representing existing annotations from EDAM (as specified in *query data*, `null` if not specified)

results

topic Array of objects representing a matched term from the topic branch for the given *query*, ordered by score. If no results in topic branch, then empty array. If results in topic branch were not asked for in *mapping* parameters, then `null`.

edamUri EDAM URI of the matched term

edamUriReplaced If *replaceObsolete* is `true` and this is a concept replacing a matched obsolete concept, then this contains the EDAM URI of that obsolete concept (that is replaced with the concept specified in *edamUri*)

label EDAM label of the matched term in *edamUri*

obsolete `true`, if the term in *edamUri* is obsolete; `false` otherwise

childOf Array of objects that are parents of the current matched term in *edamUri* and that *test* `"fp"`. Absent if there are no such parents.

edamUri EDAM URI of a parent described above

label EDAM label of such parent

childOfAnnotation Array of objects that are parents of the current matched term in *edamUri* and that *test* `"tp"`. Same structure as in *childOf*.

childOfExcludedAnnotation Array of objects that are parents of the current matched term in *edamUri* and that *test* `"fn"`. Same structure as in *childOf*.

parentOf Array of objects that are children of the current matched term in *edamUri* and that *test* `"fp"`. Same structure as in *childOf*.

parentOfAnnotation Array of objects that are children of the current matched term in *edamUri* and that *test* `"tp"`. Same structure as in *childOf*.

parentOfExcludedAnnotation Array of objects that are children of the current matched term in *edamUri* and that *test* `"fn"`. Same structure as in *childOf*.

bestOneQuery Best matched query part. Basis for *bestOneScore* calculation and score *class* determination using *Score limits* parameters. Basis for final *score* calculation if *mappingStrategy* is `"best"`. Otherwise (if *mappingStrategy* is `"average"`), all query parts will be used for calculating final score (use *type* `"full"` to see these partial scores). If *replaceObsolete* is `true` and this is a concept replacing a matched obsolete concept, then will contain match information of the obsolete concept specified in *edamUriReplaced* and not the actually suggested concept in *edamUri*.

type Name of the type of the query part

url URL of best matched webpage/doc/publication. Absent, if type is not webpage, doc or some publication type.

value Value of best matched keyword or publication keyword. Absent, if type is not keyword or some publication keyword type.

bestOneConcept Term part the best matched query part (*bestOneQuery*) matched with

type Name of the type of the term part

value Content of the term part. Absent, if type is `"none"`.

score

class One of `"good"`, `"medium"`, `"bad"`. Calculated based on *Score limits* parameters and the match score between *bestOneQuery* and *bestOneConcept*.

bestOneScore If *mappingStrategy* is `"average"`, then the match score between *bestOneQuery* and *bestOneConcept* will be stored here. If *mappingStrategy* is not `"average"`, then will have negative value.

withoutPathScore If *parentWeight* and *pathWeight* are above `0`, then the non path enriched score will be stored here. Otherwise will have negative value.

score Final score of the match (to *edamUriReplaced*, if it exists, or to *edamUri* otherwise)

test `"tp"`, if term was matched and also specified as existing annotation in the query; `"fp"`, if term was matched, but not specified as existing annotation in query; `"fn"`, if term was not matched, but was specified as existing annotation in query

operation Same structure as in *topic*, but for terms matched from the operation branch

data Same structure as in *topic*, but for terms matched from the data branch

format Same structure as in *topic*, but for terms matched from the format branch

args The *Parameters*

mainArgs Main parameters

edam Filename of the used EDAM ontology OWL file

txt `true`, if output of plain text results was requested; `false` otherwise

html `true`, if output of HTML results was requested; `false` otherwise

json Always `true`

processorArgs Processing parameters

fetching Always `true`

db Name of the used database file

idf Name of the used *IDF* file

idfStemmed Name of the used stemmed *IDF* file

preProcessorArgs *Preprocessing* parameters

fetcherArgs *Fetching* parameters (implemented in PubFetcher)

mapperArgs *Mapping* parameters

algorithmArgs *Mapping algorithm* parameters

idfArgs *IDF* parameters

multiplierArgs *Concept multipliers* parameters

normaliserArgs *Query normalisers* parameters

weightArgs *Query weights* parameters

scoreArgs *Score limits* parameters

tool Present, if *query data* was supplied as *bio.tools input*. The structure and content of this object is the same as in the object supplied as part of the query, except that `null` and empty values are removed. In addition, *results* from the *topic* branch are added to the topic attribute and *results* from the *operation* branch are added under a new function group object. Results from the *data* and *format* branches should be added under the `"input"` and `"output"` attributes of a function group, however EDAMmap can't differentiate between inputs and outputs. Thus, new terms from the *data* and *format* branches will be added as strings (in the form `"EDAM URI (label)"`, separated by `" | "`) to the note of the last function group object.

3.1.3.2 full

The *type* `"full"` includes everything from *core*, plus the following:

mapping

queryFetched

webpages Array of metadata objects corresponding to *webpageUrls* in *query*. Webpages are implemented in PubFetcher and thus are described in its documentation: Content of webpages. The structure of webpages here will be the same as described in PubFetcher, except for content which will be missing. The values of startUrl of webpages will be the URLs given in *webpageUrls* in *query*.

docs Array of metadata objects corresponding to *docUrls* in *query*. Structure of objects same as in *webpages*.

publications Array of metadata objects corresponding to *publicationIds* in *query*. Publications are implemented in PubFetcher and thus are described in its documentation: Content of publications. The structure of publications here will be the same as described in PubFetcher, except for fulltext which will be missing.

results

topic/operation/data/format Array of objects defined in *topic*, i.e. the same content as in *core*, plus the field *parts* defined below.

parts Array of objects representing scores from each query part that are used in calculating the final score (using weights from *Query weights* parameters), in case *mappingStrategy* is `"average"`. Absent, if *mappingStrategy* is not `"average"`.

queryMatch

type Name of the type of the query part

url URL of best matched webpage/doc/publication. Absent, if type is not webpage, doc or some publication type.

value Value of best matched keyword or publication keyword. Absent, if type is not keyword or some publication keyword type.

score Intermediate score of matching to query part from all concept term parts

conceptMatch

type Name of the type of the term part

value Content of the term part. Absent, if type is `"definition"`, `"comment"` or `"none"`.

score Intermediate score of matching to concept term part from query part

score Score of the part

counts

conceptsSize Total number of concepts in the used EDAM ontology

topicSize Number of concepts in the topic branch

operationSize Number of concepts in the operation branch

dataSize Number of concepts in the data branch

formatSize Number of concepts in the format branch

queriesSize Number of queries. Always `1`. Can be bigger in output of *EDAMmap-CLI*.

resultsSize Number of results. Always `1`. Can be bigger in output of *EDAMmap-CLI*.

tp

- **topic** Number of matched terms from the topic branch that *test* `"tp"`
- **operation** Number of matched terms from the operation branch that *test* `"tp"`
- **data** Number of matched terms from the data branch that *test* `"tp"`
- **format** Number of matched terms from the format branch that *test* `"tp"`
- **total** Total number of matched terms that *test* `"tp"`

fp Same structure as in *tp*, but for matched terms that *test* `"fp"`

fn Same structure as in *tp*, but for matched terms that *test* `"fn"`

measures Measures of EDAMmap performance against existing *annotations* provided in *query*. Does not make much sense in case of one query-results pair (if *queriesSize* and *resultsSize* are `1`), but included for completeness.

precision The precision

- **topic** Precision in the topic branch
- **operation** Precision in the operation branch
- **data** Precision in the data branch
- **format** Precision in the format branch
- **total** Precision over all branches

recall Recall. Same structure as in *precision*.

f1 F1 score. Same structure as in *precision*.

f2 F2 score. Same structure as in *precision*.

Jaccard Jaccard index. Same structure as in *precision*.

AveP Average precision. Same structure as in *precision*.

RP R-precision. Same structure as in *precision*.

DCG Discounted cumulative gain. Same structure as in *precision*.

DCGa DCG (alternative). Same structure as in *precision*.

3.1.4 Examples

One way to test the API is to send JSON data using `curl`. For example, for sending the input:

```
{
  "name": "aTool"
}
```

issue the command:

```
$ curl -H "Content-Type: application/json" -X POST -d '{"name":"aTool"}' https://biit.
↪cs.ut.ee/edammap/api
```

In the output, no results can be seen:

```
"results" : {
  "topic" : [ ],
  "operation" : [ ],
  "data" : null,
  "format" : null
}
```

Which is not surprising, given only the tool name was supplied ("aTool"), which is too little for EDAMmap to work with.

A more meaningful input might look like this:

```
{
  "name": "g:Profiler",
  "keywords": [ "gene set enrichment analysis", "Gene Ontology" ],
  "description": "A web server for functional enrichment analysis and conversions of␣
↪gene lists.",
  "webpageUrls": [ "https://biit.cs.ut.ee/gprofiler/" ],
  "docUrls": [ "https://biit.cs.ut.ee/gprofiler/help.cgi" ],
  "publicationIds": [
    "17478515\t\t10.1093/nar/gkm226",
    {
      "pmcid": "PMC3125778"
    },
    {
      "pmid": "27098042",
      "doi": "10.1093/nar/gkw199"
    }
  ],
  "annotations": [
    "http://edamontology.org/topic_1775",
    "operation_2436",
    "data_3021",
    "http://edamontology.org/format_1964"
  ],
  "branches": [ "topic", "operation", "data", "format" ],
  "matches": 6,
  "obsolete": true
}
```

For testing, this input could be saved in a file, e.g. `input.json`, and then the following command run:

```
$ curl -H "Content-Type: application/json" -X POST -d '@/path/to/input.json' https://
↪biit.cs.ut.ee/edammap/api
```

To supply the same data (except the "keywords") as *bio.tools input*, the following could be used:

```
{
  "tool": {
    "name": "g:Profiler",
    "description": "A web server for functional enrichment analysis and conversions␣
↪of gene lists.",
    "homepage": "https://biit.cs.ut.ee/gprofiler/",
    "documentation": [{
      "url": "https://biit.cs.ut.ee/gprofiler/help.cgi",
      "type": "General",
```

(continues on next page)

(continued from previous page)

```
      "note": null
    }],
    "publication": [{
      "pmid": "17478515",
      "pmcid": null,
      "doi": "10.1093/nar/gkm226"
    },{
      "pmcid": "PMC3125778"
    },{
      "pmid": "27098042",
      "pmcid": null,
      "doi": "10.1093/nar/gkw199"
    }],
    "topic": [{
      "term": "Function analysis",
      "uri": "http://edamontology.org/topic_1775"
    }],
    "function": [{
      "operation": [{
        "term": "Gene-set enrichment analysis",
        "uri": "http://edamontology.org/operation_2436"
      }],
      "input": [{
        "data": {
          "uri": "http://edamontology.org/data_3021"
        },
        "format": [{
          "uri": "http://edamontology.org/format_1964"
        }]
      }],
      "output": null
    }]
  },
  "branches": [ "topic", "operation", "data", "format" ],
  "matches": 6,
  "obsolete": true
}
```

3.2 Prefetching

Once a query has been received by the API, content corresponding to *webpageUrls*, *docUrls* and *publicationIds* has to be fetched (unless it has been fetched and stored in some previous occurrence), before mapping can take place.

This content could be prefetched and prestored in the database as a separate step, before the mapping query is sent. This is useful in the web application, where content can be fetched as soon as the user has entered the corresponding query details, and thus mapping time could be less when the entire query form is finally submitted. It might be of less use in the API, but has been included nevertheless.

3.2.1 /api/web

3.2.1.1 Request

Links, whose content is to be prefetched, are specified as an array of strings under the JSON key *webpageUrls*.

In addition to *webpageUrls*, parameters from *Fetching* can be used, as these can influence the fetching.

3.2.1.2 Response

The main result of the query is not the content of the response itself, but the fact that the contents of the requested links were stored in the database on the server. However, some informational output is still provided.

success `true` (if `false`, then the JSON output of *Error handling* applies instead of the one below)

webpageUrls Array of objects describing the completeness of the content of each link on the server

- **id** A webpage URL specified in the request
- **status** The status of that webpage. One of "broken", "empty", "non-usable", "non-final", "final".

3.2.2 /api/doc

Analogous to */api/web*, except for documentation and that the JSON key *docUrls* has to be used.

3.2.3 /api/pub

3.2.3.1 Request

Journal articles, whose content is to be prefetched, are specified using a PMID and/or PMCID and/or DOI. This is done as an array of strings and objects under the JSON key *publicationIds*. If the ID is specified as a string, it has to be in the form `"<PMID>\t<PMCID>\t<DOI>"`. If it is specified as an object, the keys `"pmid"`, `"pmcid"`, `"doi"` are to be used.

In addition to *publicationIds*, parameters from *Fetching* can be used, as these can influence the fetching.

3.2.3.2 Response

The main result of the query is not the content of the response itself, but the fact that the contents of the requested articles were stored in the database on the server. However, some informational output is still provided.

success `true` (if `false`, then the JSON output of *Error handling* applies instead of the one below)

publicationIds Array of objects describing the completeness of the content of each article on the server

- **id** IDs describing one publication specified in the request
  - **pmid** The PMID of the publication
  - **pmcid** The PMCID of the publication
  - **doi** The DOI of the publication
- **status** The status of that publication. One of "empty", "non-usable", "non-final", "final", "totally final".

3.2.4 Example

Try to prefetch the publication with PMID "23479348" and PMCID "PMC3654706", increasing connect and read timeout to give the server more time to fetch the whole publication:

```
$ curl -H "Content-Type: application/json" -X POST -d '{"publicationIds":[
↪"23479348\tPMC3654706\t"],"timeout":30000}' https://biit.cs.ut.ee/edammap/api/pub
```

Sample output:

```
{
  "success" : true,
  "publicationIds" : [ {
    "id" : {
      "pmid" : "23479348",
      "pmcid" : "PMC3654706",
      "doi" : "10.1093/BIOINFORMATICS/BTT113"
    },
    "status" : "final"
  } ]
}
```

3.3 Error handling

If `"success"` is `true` in the JSON *response*, then HTTP status code was "200 OK" and the rest of the JSON is in the format described above.

If `"success"` is `false` in the JSON *response*, then something has gone wrong, the HTTP status code is *400 Bad Request* or *500 Internal Server Error* and the rest of the JSON will be in one of the following formats.

3.3.1 400 Bad Request

Status code 400 means something was done wrong on the client side (syntax error, bad parameter value, etc) and the error should be fixed by the client, before another attempt is made.

The output JSON will have the following format:

success `false`

status `400`

message A string describing the error

time Timestamp string (as ISO 8601 combined date and time) when the error occurred

3.3.2 500 Internal Server Error

Status code 500 is a catch all for all other errors. Usually, it should be some problem on the server side. It might be temporary, so another try later might result in success. It might also be an unforeseen problem on the client side. There's a strong chance there is a bug somewhere, so feedback with a timestamp is appreciated (to GitHub issues or by contacting the author).

The output JSON will have the following format:

success `false`

status `500`

time Timestamp string (as ISO 8601 combined date and time) when the error occurred

3.3.3 Examples

3.3.3.1 Syntax error in JSON

```
$ curl -H "Content-Type: application/json" -X POST -d '{"name"}' https://biit.cs.ut.
↪ee/edammap/api
```

```
{
    "success": false,
    "status": 400,
    "message": "Invalid token=CURLYCLOSE at (line no=1, column no=8, offset=7).␣
↪Expected tokens are: [COLON]",
    "time": "2018-05-28T12:59:57.389Z"
}
```

3.3.3.2 Bad parameter value

```
$ curl -H "Content-Type: application/json" -X POST -d '{"name":"test","goodScoreTopic
↪":2}' https://biit.cs.ut.ee/edammap/api
```

```
{
    "success": false,
    "status": 400,
    "message": "Param 'goodScoreTopic=2.0' is above limit 1.0",
    "time": "2018-05-28T13:02:53.616Z"
}
```

3.3.3.3 Some other illegal requests

```
$ curl -H "Content-Type: application/json" -X POST -d '{"name":"test","annotations":[
↪"http://edamontology.org/1775"]}' https://biit.cs.ut.ee/edammap/api
```

```
{
    "success": false,
    "status": 400,
    "message": "Illegal EDAM URI: http://edamontology.org/1775",
    "time": "2018-05-28T14:07:50.164Z"
}
```

```
$ curl -H "Content-Type: application/json" -X POST -d '{"name":"test","publicationIds
↪":["23479348\tPMC3654706"]}' https://biit.cs.ut.ee/edammap/api
```

```
{
    "success": false,
    "status": 400,
    "message": "Publication ID has illegal number of parts (2), first part is 23479348
↪",
    "time": "2018-05-28T14:09:04.032Z"
}
```

```
$ curl -H "Content-Type: application/json" -X POST -d '{"name":"test","webpageUrls":[
↪"biit.cs.ut.ee/gprofiler"]}' https://biit.cs.ut.ee/edammap/api
```

```
{
    "success": false,
    "status": 400,
    "message": "Malformed URL: biit.cs.ut.ee/gprofiler",
    "time": "2018-05-28T14:10:23.651Z"
}
```

CHAPTER 4

Ideas for future

Sometimes ideas are emerging. These are written down here for future reference. A written down idea is not necessarily a good idea, thus not all points here should be implemented.

4.1 General

- In addition to tools, annotate also training materials.
- Generalise to other ontologies besides EDAM (#8). However, optimising specifically for EDAM is one of the goals of EDAMmap.
- Use existing libraries of some other tools, like Maui or Kea, in addition to the current self-made approach.
- Try to use machine learning. Challenges include a large number of EDAM terms and the quality of manual annotations currently in bio.tools. Also, there will be annotations added by previous versions of EDAMmap in bio.tools. Maybe the ontology needs to be simplified, for example more specific terms removed.

4.2 Algorithm

- Currently, scores are not totally comparable across queries. Try to make a score in one query mean the same thing in another query as exactly as possible.
- An extra query part could be tags present in some web pages, like software registries or code repositories. This would require changes in PubFetcher.
- Maybe WordNet could be used as part of the mapping algorithm. For example use lemmatisation instead of stemming.
- In results got from running EDAMmap against existing entries of bio.tools, look at FNs and see if anything can be done to increase their score.

4.2.1 Parameters

- Further investigate the effect of different parameter values, like the *IDF parameters*, stop words removal, stemming, bi-directional matching, path enrichment, etc. Not all used methods necessarily improve results.
- Currently, default values for *query normalisers* and *score limits* have been manually tuned to give good results for the usual input and default values of other parameters. Instead, try to automatically set the values of these normaliser and limit parameters, based on the input queries and parameter values.
- Try to implement automatic parameter tuning to find optimal values for parameters. If parameters of some methods give best results when turned off, then before discarding the methods, one should check that maybe these methods sometimes find correct results that better methods fail to find.
- Some parameters could be changeable on a per EDAM branch basis.

4.2.2 Weights

- In *Query weights*, maybe the publication should have an overall weight instead of each usable publication part influencing the score independently.
- Different publication full text parts (like image captions) could have different weights. This would require changes in PubFetcher.
- Different publication types (primary, etc) and link types could have different weights. This would mean categorising links got through other means than bio.tools.

4.2.3 Measures

- Add measures about scores, like average maximum score of TPs, etc.
- Maybe also take into account the direct parents and children of an automatically found term when deciding if it matches a manual annotation.
- Plot some measures, for example generate a precision-recall curve by varying the parameter *matches*.

4.2.4 Ontology

- EDAM is not a tree, but a DAG. Does this influence path enrichment? Look more into related terms influencing each other in the results.
- Branch specific tweaks, for example terms from the topic branch could be more specific than terms from the operation branch and maybe more terms could be output for topic than operation.
- Individual concept level tweaks, for example currently some terms are suggested too frequently (many FPs) and others not frequently enough (many FNs, possible because of IDF weighting).
- For more homogeneous results, maybe bias EDAMmap towards some terms, that is some terms could be "more recommended for annotation in bio.tools" than others.
- Look into using inter-branch relations ("has_input", "has_output", "has_topic", etc). Have to be careful, as only some terms have these defined.
- Some concepts have "hasRelatedSynonym" defined, it's currently not read as it's quite rare.
- Results for the data and format branch are not that good currently (thus disabled by default), look into improving them.

4.3 Server

- Give progress information (a progress bar or simply some status information) after the MAP button is pressed in the web app.
- Enable batch queries (more than one result per query is currently only possible on the command-line).
- Option to download the HTML report as a ZIP file.
- Option to choose the EDAM ontology version from a dropdown (or supply own file).
- Make the size of the server worker thread pool configurable.

4.4 Maintenance

- Update PubFetcher's scraping rules, by testing the rules and modifying outdated rules in journals.yaml, webpages.yaml and most importantly the hardcoded rules for Europe PMC and other built-in resources.
- Update dependencies in pom.xml (but care should be taken to not cause regressions).
- Check for broken links in the documentation using `make linkcheck`.
- When a new biotoolsSchema is released, some code modifications might be necessary to adhere to it.
- Also, when a new EDAM ontology is released, some modifications might be necessary (for example in blacklist.txt and blacklist_synonyms.txt; also, any running *EDAMmap-Server* instances could be restarted to use the new ontology version).