UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Erik Jaaniso

Automatic mapping of free texts to bioinformatics ontology terms

Master's Thesis (30 ECTS)

Supervisor: Hedi Peterson, PhD

Tartu 2016

Bioinformaatika ontoloogia terminite automaatne leidmine vabatekstist

Lühikokkuvõte: Bioinformaatika valdkonnas on olemas palju tööriistu ja teenuseid, mille hulk kasvab üha kiirenevas tempos. Et informatsioon nende ressursside kohta oleks kättesaadav võimalikult kasulikul viisil, annoteerime me need ontoloogia terminitega. Hetkel toimub annoteerimine käsitsi, mis on aeganõudev ja veaohtlik protsess. Antud töös seame eesmärgiks luua tööriist, mis aitab annotaatorit, pakkudes talle annoteerimissoovitusi. Me loome programmi, mis loeb sisse vabatekstilise tööriistade ja teenuste kirjelduse, lisab neile seotud veebilehtede ja teadusartiklite sisu ja sellel põhinevalt annab välja parimad leitud ontoloogia terminite vasted. Seejärel, optimimeerime programmi parameetreid käsitsi tehtud annotatsioonide põhjal. Esmased tulemused on paljulubavad – paljud leitud soovitused on kooskõlas käsitsi tehtud annotatsioonidega. Veelgi enam, kogenud annotaatorite väitel on mitmed teised soovitused samuti korrektsed.

Võtmesõnad: tekstianalüüs, bioinformaatika, ontoloogiad

CERCS: B110, P170

Automatic mapping of free texts to bioinformatics ontology terms

Abstract: In the field of bioinformatics, the number of tools and services is ever-increasing. In order to make information about these resources available in a useful was, we annotate them with ontology terms. This is currently done manually – which is time-consuming and error-prone. In this thesis, we set out to make a tool that helps the annotator by providing annotation suggestions. We developed a program, that reads in free text descriptions of tools and services, adds content of web pages and publications related to the tool and based on this outputs best matches to ontology terms. Then, we optimised the parameters of the program on manually done annotation sets. Initial results look promising, as when comparing performance against these manual annotations, we see that many suggestions are agreeing with them. Moreover, according to experienced annotators, many of the other suggestions are also correct.

Keywords: text analysis, bioinformatics, ontologies

CERCS: B110, P170

Contents

1 Introduction

In the field of bioinformatics, the number of tools and services is ever-increasing. Which means, that a problem a scientist might have, can already have a solution, or at least tools available to reach it. But how does he find this valuable work of others and not waste his resources on duplication efforts?

One option is to collect the descriptions of these resources – tools and services – in a common place. But even there, it will be difficult to find the relevant tools from thousands of entries. Hence, we should annotate these resources in a standardised way.

One possibility is to use ontologies for this task. Ontologies allow to semantically connect meta-data – this simplifies the organisation and merging of resources and provides better browse and search capabilities. For example, if standard annotations are available, we could query the database and get a list of tools, that convert from a set of inputs we have to a set of outputs we want, and additionally get a description of operations necessary to achieve this.

But how to annotate these resources? One way is to do it manually. However, this is very time-consuming – the resource spent on annotating could be better allocated elsewhere. Moreover, the current distributed way of annotating involves three potential problems – carelessness (mistakes due to lack of motivation), mistakes due to lack of knowledge about the tool/service and mistakes due to lack of knowledge of the used ontology.

So, in this thesis we look at ways to partially automate this process. We set out to make a tool, that gives suggestions to the annotator – this can help both to reduce mistakes and free up the annotator's time, as he doesn't have to look deeply into each tool or service anymore.

2 Background information

2.1 EDAM ontology

EDAM is a simple ontology for bioinformatics purposes [1].

It has 4 main sub-ontologies or *branches*:

topic describes a general concept or field, e.g., "Data handling" and "Proteomics"

operation describes a function that processes a set of inputs to a set of outputs, e.g., "Classification" and "Sequence alignment"

data describes the type of input or output, e.g., "Image" and "Ontology"

format describes the format of the input or output, e.g., "HTML" and "FASTA"

Generally, only concepts and data types from the fields of bioinformatics and computational biology have been included.

EDAM concepts can be related to other concepts, e.g., an *operation* might have a *topic* and *format* might be a format of a concrete *data* concept. Concepts are organised in a hierarchy structure, e.g., "Multiple sequence alignment" is a child of "Sequence alignment", which is a child of "Alignment". Concepts can be deprecated be marking them as *obsolete*.

2.2 Mapping of free text

Free text is any text not annotated with meta-data or any additional information. For example, the content of a book is free text, but also the abstract of a journal article or every word in a web page.

There are different ways to categorise and annotate a collection of free texts with terms from an ontology. A person familiar with the ontology might read the content of this collection and based on his interpretation of the texts and his knowledge of the annotation possibilities of the ontology, manually assign concepts to the collection.

In this work, this collection of free texts is the description of a bioinformatics tool or service: its name, reference manual, an abstract of the corresponding journal article, etc. As for the annotator part, we will try to do it automatically.

One way to algorithmically find annotations, is by comparing words in the free texts with words in the ontology concepts. If we find matching words

between a free text and a concept, then there is a chance that the concept is describing one of the ideas expressed in the free text.

However, what to do, if matches are found to many concepts, which ones are more descriptive of the full text? What if the text contains spelling mistakes or non-relevant content to its main ideas?

In this thesis we develop techniques for finding terms from an ontology to describe a bioinformatics tool or service, using word-to-word matching, while trying to minimise the concerns raised in the previous paragraph.

2.3 Existing algorithms

2.3.1 Initial mapper

The initial mapper was written by Rabie Saidi [2]. It allows comparing keywords, consisting of one or a couple of words, against concept descriptors in EDAM. Scores based on the similarity of a keyword string and strings describing the concepts are calculated. Then, concepts corresponding to the highest scores are suggested as annotations for the keyword.

While working well, it does so only in case of very short input text, such as a tool or service name or for keywords attached to the tool or service. The reason is, that the full input text is compared against concept descriptors – if input text is very long, then two strings with very different lengths will be compared.

To enable more useful results from the automatic mapper, we would also like to use all parts available in the tool or service description. So the approach described in this section needs to be extended, by enabling individual word-to-word comparisons as first step.

2.3.2 ZOOMA

One existing tool finding possible ontology mappings for free text terms is ZOOMA [3]. It works with not only one, but a combination of ontologies. In addition to string comparisons, it uses a repository of previous mappings to use existing knowledge in deciding the best mapping. In doing so it may be able to avoid many pitfalls associated with basing decisions only on the results of simple word-to-word comparisons.

However, it is mainly concerned with biological entities, rather than bioinformatics tools and services. But more importantly, it does 1-to-1 mappings, while we are interested in annotating tools and services with potentially many concepts, and, like the mapper described in the previous paragraph, it works

only for short phrases. So, it serves a different purpose and is not usable for the task we set out to solve.

3 Methods

3.1 Architecture

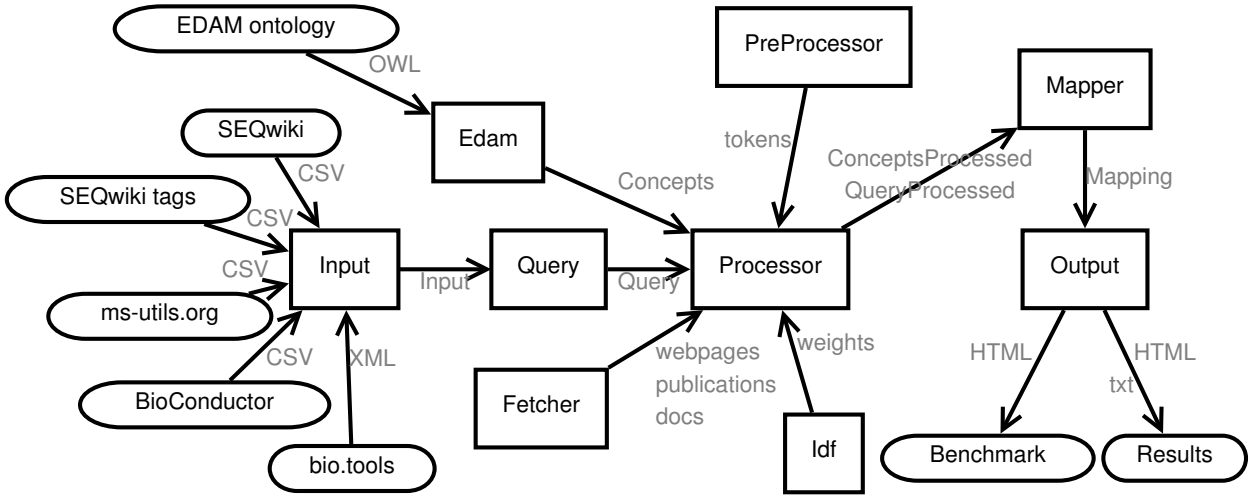


Figure 1: Architecture

The data processing pipeline of the automatic mapper can be seen on Figure 1. The round boxes illustrate inputs and outputs, which will be described in sections 3.2, 3.11 and 3.12. Square boxes encapsulate steps along the pipeline – in the following sections, we will describe each of those.

The program has been implemented as a command-line program in the Java programming language. Argument parsing is done by the *JCommander* library (listed in Appendix A). A listing of all options is given in Appendix B.

3.2 Input: The free texts

In this section, we describe the different databases about bioinformatics tools and services and the metadata these contain.

3.2.1 SEQwiki

The SEQanswers wiki (SEQwiki from now on) is a catalogue of bioinformatics tools for high-throughput sequencing (HTS) analysis [4]. It was born out of pressing need for a structured knowledge sharing platform that could keep up with the rapid emergence of HTS analysis tools. Wiki pages for each tool include a short free-text summary, an optional longer description, input and output formats used by the tool, licence, features and so on. It

is a volunteer based initiative, so the level of annotations might fluctuate based on the person annotating the resources. A data dump of all the tools can be obtained from the SEQanswers website [5]. We are interested in the following fields:

Name Tool name, which might contain its purpose or used formats in it

Summary A short, usually one-two sentence description about the tool

Biological domain Corresponding to EDAM topic

Bioinformatics method Corresponding to EDAM operation

In addition, the full description available at each individual tool page and properties like "PubMed id" and "URL type" (which includes Homepage, Manual, etc) could be exploited for the mapping. As of writing, there are 700 tools in SEQwiki.

3.2.2 SEQwiki tags

Tools in SEQanswers have been annotated with tags from "Biological application domain(s)" and "Principal bioinformatics method(s)", which correspond to EDAM topic and operation respectively. Manual mapping of these tags to EDAM terms has been done and many tags have been renamed to the matching EDAM label or synonym. The manual mapping took several weeks to complete.

One report of such mapping is available on SEQanswers [6]. While out-of-date, it still offers a valuable comparison target for the automatic mapper. Thus, it could be used to test and tune the performance of the automatic mapper in case of simple keyword-to-keyword matches. The report contains 172 mappings from SEQanswers tag to EDAM term.

While the up-to-date tags should directly correspond to an EDAM term or synonym, there are still mistakes or omissions. One example is mix-ups, where a SEQwiki domain has been erroneously mapped to an EDAM operation, or a method mapped to a topic. So, in addition to suggesting annotations for SEQwiki tools, based on input described in the previous section, the interoperability between EDAM terms and existing SEQwiki tags can be improved.

3.2.3 ms-utils.org

Ms-utils.org is a list of free software directed to mass spectrometry experts [7]. Emphasis is on tools for visualisation and analysis of mass spectrometry data

and automated methods for proteomics and protein analysis. As of writing, 235 tools are available in the software list.

We are interested in the following readily available metadata:

Name Tool name, which might contain its purpose or used formats in it

Description A very short, one-sentence summary of the tool

Link Homepage of the tool

Paper Optional PubMed ID(s) of related articles attached to the tool

In addition, ongoing manual curation to EDAM topics, operations and formats is being performed. This can be used as benchmark for the automatic mapper.

3.2.4 BioConductor

BioConductor is an open bioinformatics project, creating free software for the analysis and comprehension of high-throughput genomic data [8]. It is primarily based on the R programming language. 1085 software packages are listed in the BioConductor file used as input.

Package metadata includes the following:

Name Tool name

Title A short title (a few words)

Description A short description (a few sentences)

biocViews One or more hierarchical tags, analogous to EDAM term labels

Also, each tool includes a reference manual in PDF format that can be used as additional source for extracting EDAM terms. It can be obtained using an URL pattern where the tool name is placed in for each tool. Each tool can contain additional documentation (vignettes) that can also be fetched for additional input.

Manual curation is being done for EDAM topics and operations. Like for ms-utils.org, this could be used as benchmark.

3.2.5 bio.tools

The Tools and Data Services Registry (hereinafter Registry) is a community-driven curation effort for bioinformatics, developed under the European Bioinformatics for Life Science infrastructure (ELIXIR). resources [9]. It strives to become a standard portal for analytical tools and data services in bioinformatics. Aiming for comprehensiveness, it should help scientists find the best tools for their needs. The BIO.TOOLS input data used in this thesis contains 2402 entries of tools and services.

Consistency of tool and service description is achieved using the EDAM ontology. In fact, the previously described tool databases are scheduled to be merged in the Registry, and annotations using a common ontology would help to ensure consistency. For example, some BioConductor packages have already been imported.

The motivation to use Registry entries database as input to the automatic mapper is threefold:

1. Improve the manual curation of existing entries to EDAM terms
2. Conversely, improve the automatic mapping algorithm and tune parameters to suggest terms more similarly to manual mapping
3. Lay the groundwork for an on-line term suggester, used at a time when a new tool is being added to the Registry

The Registry uses the BiotoolsXSD schema as a resource description model [10]. Among other attributes it defines, the following are useful for the automatic mapper:

name Canonical resource name (1–50 characters)

homepage Resource homepage (URL)

mirror Mirror homepage (URL) (optional)

description Textual description of the resource (up to 1000 characters)

topic EDAM topic term(s)

functionName EDAM operation term(s)

dataType EDAM data term(s), input or output (optional)

dataFormat EDAM format term(s), input or output (optional)

docsHome Link to documentation main page (optional)

docsGithub Link to Github page (optional)

publicationsPrimaryID PMCID, PMID or DOI (these IDs are explained in section 3.6.1) of the primary publication (optional)

publicationsOtherID PMCID, PMID or DOI of other relevant publications (optional)

Webpages, docs and publications contain only URLs and IDs pointing to their content. The content itself has to be fetched from these external sources. This is the topic of section 3.6.

3.2.6 Obtaining the data

For MS-UTILS.ORG and BIOCONDUCTOR, the manual curation is being worked on in Google Docs. From there, we could easily obtain CSV files of the data. For SEQWIKI, a CSV download link was provided. In case of the mapping report, converting the HTML table to CSV is also trivial. CSV files are read using the *opencsv* software library (listed in Appendix A).

Obtaining all resources in the Registry for off-line analysis is possible through its REST API [11]. The resulting XML file can be parsed using capabilities built-in to the standard Java distribution.

SEQWIKI and BIO.TOOLS input can also be obtained in JSON format, in which case new input classes could be defined to parse it.

3.3 Query: Abstracted input

Different inputs have different fields describing the same concepts. In order to work with the input parts' content in an input type independent way, we define a new structure abstracting away these differences.

A Query, to be mapped to EDAM concepts, is defined as follows:

name Name of the tool

webpageUrls Links to homepage and other web resources

description Short free-text description of the tool

keywords Tags attached to the tool

publicationIds Publications related to the tool

docUrls Links to documentation resources

annotations EDAM URIs of the manual curation

How different inputs relate to the abstract query can be seen in Table 1. For SEQwiki, the webpage URL is obtained by appending the tool name to the general SEQanswers URL. This gives the wiki page of the tool, from where a longer description of it can be read. Additional webpages of the tool and potential PubMed IDs are not being used currently. For Bioconductor, package Vignettes are looked for from a few standard locations.

Table 1: Mapping of Input to Query

Query part	SEQwiki	SEQwiki tags	ms-utils.org	BioConductor	bio.tools
name	Name	-	Name	Name + Title	name
webpageUrls	from Name	-	Link	-	homepage, mirror
description	Summary	-	Description	Description	description
keywords	Domain, Method	Domain/ Method	-	biocViews	-
publicationIds	-	-	Paper	-	publications PrimaryID, publications OtherID
docUrls	-	-	-	Reference Manual, Vignettes	docsHome, docsGithub
annotations	-	topic/ operation	topic, operation, format (in, out)	topic, operation	topic, functionName, dataType, dataFormat

3.4 Edam: The ontology concepts

The EDAM ontology is available in Web Ontology Language (OWL) format, which is based on XML and Resource Description Framework (RDF). EDAM version 1.14 contains 3218 concepts (425 in *topic*, 728 in *operation*, 1480 in *data* and 585 in *format* branch). The EDAM ontology homepage contains a download link and additional information [12].

Concepts are loaded using the *OWL API* library [13]. Currently, the hierarchy within EDAM and any relations between concepts are ignored;

any semantic reasoning thus also not done. As we get a clearer picture on how EDAM terms are found and suggested, we might want to take these hierarchical relations into account. In essence, each concept is regarded as an independent mapping target, with the following structure for string matching:

label Preferred label (short name or phrase in common use)

exactSynonyms List of optional exact synonyms of the preferred term

narrowSynonyms List of optional narrow synonyms

broadSynonym List of optional broad synonyms

definition Concise description of the concept

comment Optional peripheral but important information

Concepts are stored in a map data structure, with their unique EDAM URI as key. Each concept contains also a boolean flag, indicating whether the concept is obsolete.

3.5 Processor: Making inputs ready for mapping

Before queries can be mapped to concepts, both need to be processed. The following steps are done:

- For possible homepage and documentation URLs in the query, the content of the corresponding web pages is obtained; for possible publication IDs present in the query, the corresponding publications will be obtained and the found content will be stored in a structure (publication title, abstract, ...) (section 3.6)
- Strings in all query and concept parts will be stripped of punctuation and tokenized at word borders, with optional stopwords removal and stemming (section 3.7)
- For term weighting, the inverse document frequency (IDF) for all tokens is obtained (section 3.8)

Content from the Internet can be fetched directly or obtained from a local database where it has been pre-saved (section 3.6.6). In case of query tokens, IDF values are obtained from a pre-generated file. Managing the web content database and the IDF file is done using a utility program (section 3.9).

For each query, a new structure for storing the results of processing is made. Same for each concept. These will consist of the same parts as the original structure, but with each string replaced with a list of tokens. And for each list of tokens there is a list of IDF scores, with entries corresponding to the tokens. In the query, URLs will be replaced with tokens of the content of the corresponding web resources. Publication IDs will be replaced with two structures: a publication containing strings of the various parts of a publication (described in section 3.6.1) and a processed publication containing tokens and IDF scores of the same publication.

Note, that we could also not store the list of IDF scores and just take the value from the map representation of the IDF file each time it is required during the mapping. However, we prefer small gains in speed over small gains in memory usage.

3.6 Fetcher: Getting web page and publication content

This section describes, how getting a web page or publication content from the Internet is done, given an URL or publication ID. For fetching HTML and XML pages, we use the *jsoup* library (listed in Appendix A).

3.6.1 Publications

A publication can have different parts, like title and abstract. Not all sources where publication information is pulled from have all parts available. Thus, if all publication parts can't be filled using the first tried source, then a next source is tried, until we have a complete publication or we run out of sources. If the publication part fetched from the next source is smaller in content than we currently have, the current content is not replaced. Also, we define a minimum length or size for the publication parts. This can helped if the fetched content is only partial or erroneous. For example, at least two keywords per publication should be retrieved. If this is the case, then the keywords part is final and no more sources are probed for keywords. The source can still be fetched from to complete other parts. However, if all parts the source can potentially provide are already final, fetching from it is not done.

A publication is defined as follows:

pmid PubMed ID of the article. It is a unique identifier assigned to each record in PubMed (described in next paragraph). The format of the ID is a positive integer. This ID is set in case the original publication ID in the query was in PMID format. But sources can also include other IDs

associated with the article in the article's metadata. Thus, we could fill in this ID, and IDs mentioned below, after the initial fetching. If all IDs have been set, the article's ID is considered final. We try to get all IDs, as knowing more IDs enables us to query more sources.

pmcid PubMed Central ID of the article. A unique identifier assigned to each record in PubMed Central (described in next paragraph). The format is "PMC" with a positive integer appended to it (e.g., PMC1234).

doi Digital object identifier (DOI) is a persistent interoperable identifier by the International DOI Foundation for uniquely identifying objects [14]. Once a DOI ID has been registered, the related resource can be accessed by appending the ID to http://doi.org/. If the resource is moved, only redirection metadata has to be updated by the owner, the URL for accessing the resource remains the same Format is as follows: "10." + registrant code + "/" + suffix chosen by registrant (e.g., 10.1000/j42). A DOI ID may be specified using some prefixes (like "http://doi.org" or "doi:"), we remove this prefix part to avoid duplicates. Also, DOI names are case-insensitive (but only for ASCII characters, that is, only characters "A" to "Z" and "a" to "z" are treated case-insensitively) However, some on-line tools treat these characters case-sensitively. So, for better interoperability, we save the ID preserving case and give letters cased as in the original publication ID when fetching from sources. The DOI system is often used for journal articles.

title Title of the article. Minimum length to be considered final has been set to 10 characters.

keywords User-defined keywords the article has been tagged with. Usually presented along with the abstract, but not available for all articles. They are not from any standard vocabulary or any standard form, just written as the author sees fit. If a good match between a keyword and an EDAM concept is found, this is valuable additional input for the mapping algorithm. We require at least 2 keywords from a source for this part to be considered final.

meshTerms Medical Subject Headings (MeSH) thesaurus is a hierarchical controlled vocabulary by the United States National Library of Medicine [15]. It is used for annotating and classifying journal articles and books in biomedical and health-related fields. Each journal article in PubMed is indexed with MeSH terms. Terms can be narrowed down using qualifiers and some terms can be marked as being major topics

of the article. Some information may be lost when article themes are translated to MeSH terms. As with keywords, we require at least 2 terms per article.

efoTerms The Experimental Factor Ontology (EFO) is an ontology used at the European Bioinformatics Institute [16]. It combines parts of several other ontologies, including mainly biomedical data, such as anatomy, disease and chemical compounds. EFO terms have been mapped from the full text of articles (when available) by Europe PMC (described in next paragraph). The count of how many times the term was encountered is also included. As with MeSH terms, some information may have been lost while mapping to this controlled vocabulary. We require 2 EFO terms for this part to be final.

goTerms The Gene Ontology (GO) is a controlled vocabulary by the Gene Ontology Consortium [17]. Its terms define gene product properties in three domains: cellular component, biological process and molecular function. GO terms have also be mined by Europe PMC. While, as with MeSH and EFO terms, information might have been lost by representing article content with GO terms, and, its domains don't necessarily overlap with EDAM, it can still be small valuable additional input to the mapper. We require 2 GO terms for this part to be final.

abstract Abstract of the article. After the title, this is the part that has the highest potential to be fetched. As the abstract should contain the main concepts and ideas of the article and not much noise, then it is a valuable resource for mapping to EDAM. We require the abstract to be at least 500 characters to be considered final.

fulltext The full content of the article (including the title and abstract). Article abstracts are rather limited in size, so searching for matches in the full-text of the article can improve results. However, we can more easily pick up noise from the full content, especially if the article is not directly related to the tool it is attached to, and the full content of an article might not be as readily available. We require it to be 5000 characters in order to not probe next sources for it.

3.6.2 Publication sources

Different web services can answer with metadata, or even full text, about an article, when given a publication ID as parameter in the query. Usually XML format is used, from which we can extract the information we need.

However, in many cases some or any data is not available in XML form. Then the article is usually available in HTML format, for viewing in the browser. We can still extract information then, however, this can be more error-prone. Namely, an HTML view of the article may not be so well defined and be geared more towards the presentational, rather than structural side of the content. Also, web pages could be more susceptible to changes and redesign.

What follows, is a description of sources we have used to fetch articles from.

PubMed PubMed is a search engine for accessing references and abstracts, mainly in life sciences and biomedicine more particularly [18]. The main bibliographical database used is MEDLINE. Both are administered, like MeSH terms, by the United States National Library of Medicine. By using a PubMed ID as query, we get an HTML view of the corresponding article. From the HTML, we can extract the publication abstract (which is between "<abstracttext>" tags), but also title, MeSH terms and sometimes keywords. PubMed does not include the full text of journal articles, but only links to them.

PubMed Central PubMed Central is, in a sense, the companion service to PubMed [19]. It is a digital archive of the actual articles along with their full content. However, it only includes publicly accessible literature, thus only a subset of entries indexed in PubMed are available. As of time of writing, 3.9 million articles are available. Using the PMCID of a publication, we can thus extract the full text of an article from the PubMed Central website. Publication title, abstract and optional keywords (but not MeSH terms) can also be extracted. If the full text is not available in HTML format for some reason (some old articles for example), then a PDF link is extracted from the page (if available) and the corresponding PDF file fetched.

Entrez Entrez is an indexer and search engine for the many databases of National Center for Biotechnology Information (part of United States National Library of Medicine). This includes PubMed and PubMed Central. However, the Entrez programming utilities [20] enable us to get the PubMed and PMC entries in a structurally better format, like XML. For example, for PubMed, article title is between <ArticleTitle> tags, keywords between <Keyword> tags, MeSH term name between <DescriptorName> tags and abstract between <AbstractText>. In addition, the XML output contains

all possible IDs (that is, PMID, PMCID and DOI) that the article has been registered with.

Europe PMC Europe PubMed Central (formerly UK PubMed Central) is a mirror service of both PubMed and PubMed Central, but with some additional content and features [21]. It is managed by the European Bioinformatics Institute (like EFO terms). As of writing, it contains 31.1 million abstracts (including 26 million from PubMed) and 3.7 million full text articles. Using its RESTful API web service for articles, we can get query for articles using either PMID, PMCID or DOI for it. From the resulting XML file, we can extract the title, optional keywords, MeSH terms, abstract and any missing IDs. In addition, we can see from the output, if Europe PMC has the full text available (in either XML or HTML format) and if terms mined from it are available (in XML format). If so, then these additional resources are fetched. The EFO and GO ontology concepts are mined from publication full texts by the Europe PMC project itself.

DOI DOI links usually point to the article on the publisher's website. So, here, we are not dealing with a single source for publication content, but each journal website might need different rules to extract content relevant to us. As not all articles are available through PubMed and every bit of extra information in the query helps when mapping to EDAM concepts, we invest some effort to define these rules. This is described in the next section.

Given all these sources, we have to decide on the order these are queried. As Europe PMC can potentially give us all the parts of the publication and we can receive well structured and uniform output for our publications, we query it first. If Europe PMC fails to provide us with some parts, we query the PubMed, PubMed Central or follow http://doi.org/DOI, depending if the original specified ID was a PMID, PMCID or DOI. In case of PubMed and PubMed Central, the XML output received through Entrez is parsed first and if it fails to provide some expected parts, we try the HTML version on PubMed or PubMed Central website. If we still have some publication parts missing and have sources that could potentially try to fill these, we query those, unless we don't know the IDs to query the source, because we have not received them from previously queried sources. In our data fetching pipeline, PubMed is the most preferred option and DOI the least preferred.

3.6.3 Extracting content from DOI links

For extracting relevant content into the publication structure from the various websites that DOI links point to, we need a way to define different extraction rules for these sites. For this, we can exploit the CSS-like element selector syntax provided by the *jsoup* library [22], which can execute on a fetched and parsed HTML document. For example, ".fulltext > p" selects all paragraph tags that are direct descendants of an element of class "fulltext". This can return zero to many elements whose content we can convert to plain text then. In case more than one matching elements are returned, we can separate the text content of these with newlines — this does not affect text processing, but provides a nicer text output we can use to check that rules are working.

Actually, we use the same technique for extracting the title, abstract, etc from HTML and XML documents from PubMed, PubMed Central and Europe PMC. However, in case of DOI, the number of possible websites is greater, with their content more prone to change. So we need a configuration file for the rules. For the format, we have chosen YAML (YAML Ain't Markup Language [23]), which is a human-readable data-serialisation standard. It can be read using the *SnakeYAML* library (listed in Appendix A).

Each rule set should have a unique identifier, which we can use to get the rules we need. As the rules are per website, then a natural choice is the domain name of the on-line journal. Usually, an on-line journal has claimed a certain registrant code for the DOI links it uses. For example, links beginning with "10.1234" could point to "www.example.com". But sometimes, a journal has many registrant codes, i.e., "10.4321" could also point to "www.example.com". Thus, to avoid duplication, we defined a second structure, where we map registrant codes to domain names. Thus, "1234" and "4321" will both map to "www.example.com", which we can use as key to retrieve the rules. Also, sometimes journals use a common platform, so it may be possible to use common rules for these. Which means we could merge "www.example.com" and "www.example.org" to a common rules set with id "platform-X".

An example configuration file with only two registrant codes follows:

```
'1234': platform-X
'4321': platform-X
---
platform-X:
  title: '#article-title'
  keywords: li.kwd
  keywords_split: .Abstract p:matches((?i)^\s*keywords\s*:)
  abstract: .abstract_area > h4 ~ p
```

```
  fulltext:
   '.fulltext-view > p,
    .fulltext-view > .section > h2,
    .fulltext-view .subsection > .fig'
  fulltext_src: /*$
  fulltext_dst: \.full
  fulltext_a: .viewFullText a
  pdf_src: /full/(.*)\.html$
  pdf_dst: /pdf/$1\.pdf
  pdf_a: 'a[name=FullTextPDF]'
```

The example shows all possible rule types, but only those that can be used have to be defined, so in reality a subset of those will be defined for a given ruleset. The meaning of the rules is as follows:

title Selector for article title. If many elements are returned, only the text of the first element is used. In the example, the element with id "article-title" is selected.

keywords Selector for keywords. In the example, all list elements with class "kwd" will be selected.

keywords_split If keywords are not selectable as separate elements, we need to break down the string we get from the one element containing all keywords. In the example, keywords are contained within a paragraph, that is a descendant of an "Abstract" class element, and the paragraph begins with the string "Keywords:" (ignoring case and allowing whitespace). Selecting this paragraph, we get one string, from which we remove "Keywords:" from the front and split the remaining string on comma to get the individual keywords. Other separators can't be specified currently.

abstract Selector for the article abstract. Can consist of one or more paragraphs. In the example, selects all paragraphs of same level that follow the "h4" tag that is a direct descendant of a .abstract_area element.

fulltext Selector for full text. If the full text is on a separate page and title and abstract should be extracted differently on that separate page, then selectors for title and abstract have to also be included here. Otherwise, the previously specified title and abstract selectors will be reused. In the example, the following will be selected: all "p" tags directly descending from fulltext-view class, all "h2" tags directly descending from

section class that directly descends from fulltext-view class and all elements of fig class that are direct descendants of elements of subscetion class who are descending from (not necessarily directly) from fulltext-view class.

fulltext_src The full text of the article may be on the same page as redirected to by the DOI link, or it may be on a separated page. In case it is on a separate page, we need to know the URL of that page. Often, we can construct this URL by simply changing a few things in the current URL. We can use regular expressions for this, this rule specifies the source pattern for the replacement. In the example, all potential slashes are remove from the end of the URL.

fulltext_dst This rule specifies the destination pattern of the replacement. In the example, the string ".full" will be appended to the current URL. After fetching the resulting URL, we use the rule in fulltext to extract the full text from resulting HTML document.

fulltext_a In some cases, a simple regular expression string replacement might not be possible or might be too complicated or fragile for getting the URL of the full-text HTML page. In that case, the current page might contain a link to the full-text page. We can specify here the selector to get the tag containing this link. The URL can then be read from the "href" attribute of this tag. Only the first "a" tag found is used. In the example, we select the "a" tag that is the descendant of a viewFullText class element.

pdf_src If getting the fulltext from HTML fails or the full-text is not available in HTML format, we try to get it from a PDF file, if available. This specifies the source pattern for regex URL replacement, as in the fulltext_src rule. In the example, the end of the URL is in form "/full/article42.html".

pdf_dst The destination pattern for regular expression URL replacement for PDF files. In the example, the end of the URL will be transformed to "/pdf/article42.pdf".

pdf_a For PDF files, this simple URL replacement often does not work and we use the option to select the element containing the link to the file instead. In the example, we select the first "a" tag whose name attribute is "FullTextPDF". The PDF file will be fetched from the URL pointed to by the "href" attribute of this "a" tag.

Specifying these rules for all on-line journals is a lot of work. Also, basing the rules on just one article might not cover all possible scenarios. However, this system described here is meant as a backup, rather than primary mean to get publication content. For many articles, most or all content can be received through the Europe PMC API or through Entrez utilities. So, the configuration file and the rulesets do not have to be comprehensive and perfect. Just using the publications used in the current input files and from these choosing the ones used multiple times will give good enough results. Any further work in defining rules will only yield small improvements. In case rules are missing for a journal, the entire content of the webpage containing the article is set as publication fulltext.

Currently, we have specified rules for all DOI registrants that can be extracted from publications present in BIO.TOOLS, which makes, in total, 42 rulesets.

3.6.4 Web pages

In addition to publications, which are fetched based on ID, we are also interested in the content of web pages and documentation pages specified in the input as URLs. The variety of these pages is however even greater than in case of DOI links. It would be very time-consuming to find out for each page, what is relevant content and what is not (navigation header, sidebar, etc). So in case of these resources, we currently just take the whole text content of the HTML document for processing. So this does include noise and we may pick up keywords that cause false positives, but depending on how this resource is used during the automatic mapping this noise could be filtered out.

However, for one case we do extract relevant content from the web page. Namely, in case of SEQWIKI input, we get the longer description from each individual tool wiki page. So we have defined a rule to select this description in case the webpage is from the "seqanswers.com" domain. It may be worth exploring, if this could be done for some other web pages that occur frequently, like GitHub, and make a YAML configuration file similar to the DOI one.

3.6.5 Getting content of PDF files

In addition to HTML and XML documents, we also encounter documents in PDF format. This can happen, when homepage and documentation links point directly to PDF files or we try to fetch the publication fulltext from a PDF or a DOI link points directly to a PDF file. It is hard to get any

structured content from a PDF file, so we just get the full plain text content of the PDF (parsing is done by the PDFBox library, listed in Appendix A).

As with HTML web pages, the PDF file might contain non-relevant content. But in addition, parsing from it is not perfect and thus the return text may contain different errors, like spelling mistakes, words broken into many parts, random symbols.

In case of publications, we are mainly interested to fetch the PDF file as it contains the full-text of the article. But in addition to setting the fulltext of publication, PDF files might contain different metadata, which might be relevant to us. The basic PDF info dictionary contains among others the Title, Subject and Keywords fields, which we can use as publication title, abstract and keywords, respectively.

PDF files might contain additional metadata in XMP (Extensible Metadata Platform) format, which is, like the OWL format, based on RDF. We use the *XMPBox* library (part of *Apache PDFBox* listed in Appendix A) to read this metadata from PDF files. The metadata can be found in different XMP schemas. Currently, we take title, keywords and abstract from the Dublin Core schema [24] properties *dc:title*, *dc:subject* and *dc:description* and keywords from the *pdf:Keywords* property of the Adobe PDF Schema [25].

However, PDF files do not contain metadata, especially in XMP format, that often. Also, by the time we fetch the PDF file, the properties PDF metadata could potentially fill are already final, or at least their content is already longer that could be obtained from the PDF. Although the PDF file could potentially deliver us keywords about the publications, we determined this rarely happens, so as an optimisation, if we already have the full-text of the publication and are only missing keywords, then the PDF is not fetched. PDF metadata has definitely its use however, when the PDF file is the first and only file we are going to fetch for a given publication.

3.6.6 Saving content to local database

If we are analysing the same inputs multiple times (to test different parameters or algorithms for example), then we need the web page and publication contents each time. Fetching them each time would take many seconds per resource and would put unnecessary load on the resource provider. Therefore, we need a way to cache the fetched text.

We are using the *MapDB* library for this (listed in Appendix A). It allows storing structures in an on-disk database. We define three maps for this: webpages, publications and docs. The unique key for each map entry will be an URL or a publication ID. As for the content, in case of webapges and docs, the value stored will be the string representing the plain text content of

a website, and in case of publications, the stored value will be the serialized publication structure we have filled in during fetching.

Note, that in case of webpages and docs the same URL might be present in both, thus some duplication is possible. However, in the future, we might want to describe different rules when extracting content from URLs that contain a homepage and URLs that contain documentation for example.

If a database file is specified to the automatic mapper, then contents of web pages and publications are read from there. If some are not present there and fetching is not disabled, they will be fetched from the web and put to the database. The database can also be filled beforehand, as described in section 3.9.

3.7 PreProcessor: Cleaning and tokenising

As the mapper works at word level, we need to break the strings making up the concept and query parts to a list of tokens at word boundaries. During this, punctuation (like commas, quotation marks, periods) will also be removed, as it is mostly not a part of the word and can make matching of words fail. In addition, some words might be removed (e.g., because they are very common and don't carry a meaning) and words may be manipulated to only include their stem.

The following will be done:

1. First, the special case of hyphenation is considered, which is mostly a problem only in some PDF files. For example, if we have "ex-" at the end of the line and "ample" one the next line. So, in case a hyphenation symbol is encountered before a newline symbol, then both symbols, and any extra whitespace, are removed to make the word "example". As hyphen, we consider the ASCII hyphen-minus ("-") and the Unicode character HYPHEN (U+2010). As newline, we consider different symbols, including the ASCII line feed ("\n") and the Unicode character LINE SEPARATOR (U+2028). In some unfortunate case, like "nineteenth-<newline> and twentieth-century", this will do the wrong thing (make the word "nineteenthand"), however we assume these are quite rare, compared to legitimate cases anyway.

2. When removing punctuation, we have two choices about what to do with punctuation that is between two letters or numbers (like "sequence(s)" or "yes/no"): just remove the punctuation, effectively making the two parts on both sides of it into one word ("sequences" and "yesno"), or replace it with space, making two separate words ("sequence s" and "yes no"). We choose the first approach, assuming, that

if no whitespace has been left before and after the punctuation symbol, this is intentional and means the punctuation symbol is part of the one word. However, there are some exceptions to this, so as second step, replace with a space (“ ”) the following symbols: en dash (“–”), em dash (“—”) and slash (“/”).

3. There is another special punctuation we need to consider: the apostrophe (“ ’ ”). Namely, stopword lists, discussed in the next section, contain only letters, and the apostrophe symbol. If we remove the apostroph in the processed string, we should also remove it in the stopword list. This can however result words that actually should not be removed in the stopword list, such as “he’ll” is transformed to “hell”. This means the apostrophe will remain also in other case, like in case of possessive (“bacteria’s”) or otherwise (“3’utr”). There are other apostrophe characters besides the ASCII apostrophe character, for example the Unicode character MODIFIER LETTER APOSTROPHE (code point U+02BC). So, in this step we replace all the different apostrophe character representations with the ASCII one (“ ’ ”).

4. Now we are ready to remove punctuation. Replace all punctuation symbols, except the apostrophe, with an empty string. As punctuation, we include all symbols from the Unicode categories Punctuation and Symbol.

5. We need a clear way to define word boundaries. At this stage, what is remaining is words made up of letters, numbers and apostrophe, which are separated by one or more whitespace characters. So replace all sequences of one or more characters from the Unicode separator and control character categories with a space character.

6. Next, we remove some of the apostrophe characters. Namely, we are actually not interested in apostrophe characters at the beginning and end of words, where they are often used as single quotation marks. Also remove words that are made of only apostrophe characters.

7. So far, we have left numbers untouched. If numbers are part of a word, like “ 3’utr ”, then they should not be removed. For free-standing numbers, like “ 3 ”, it’s more debatable. At this step, allow free-standing numbers to be removed. This is an optional operation, that can be turned on by using the corresponding program parameter.

8. Removing a freestanding apostrophe or a freestanding number leaves

two subsequent space characters in the input. So we replace all sequences of two or more space characters with one space character.

9. We use one space character to separate two words. In the beginning and end of the input string, they separate one word and the empty string, which we are not interested in. Therefore, remove potential space characters from the beginning and end of the input string.

10. When comparing words, we are not interested in case, e.g., "Virus" in the beginning of a sentence should be equal to "virus" in the middle of another sentence. To achieve this equality, we could convert all letters to upper or lower case. We choose to convert all letters to lower case here.

11. Now we can break the input string to a list of tokens (i.e., words in our case). Words are separated by one space character in the input, so just split the input string at space.

12. Next, stop words removal is done, by comparing each word from the list we got in the previous step with a list of words that should be filtered out. Stop words are the most common words in a language, so they are often filtered out before processing, e.g., because they could cause a lot of false positive matches or because we might want to increase performance by making our input smaller. Often, stop words lists mostly include function words, which are words that carry little meaning themselves, but are needed to grammatically connect other words. Examples of function words include articles ("a", "the"), pronouns ("he", "him"), particles ("if", "from"). There are many English stop words lists of differing size and content available, we have chosen to include a few of them. They are summarized in Table 2. In addition to choosing a list from the table, we could also choose to not do stop words removal, as the benefit of removing stop words might not be clear in all cases.

13. Next, stemming is considered. Stemming is the process of removing any suffixes from a word to reduce it to its root. For example, the words "sequence", "sequences", "sequenced" and "sequencing" have a very similar meaning and we might want to treat them equivalently. Reducing them to the root means that a string comparison between any two of them will result in equality. Actually, stemming does not necessarily reduce words to a valid root, as it is not necessary that related words map to a valid root, but just to the same string, that we call

Table 2: Stop words lists

List name	Source	Word count
corenlp	Stanford CoreNLP http://stanfordnlp.github.io/CoreNLP/	257
lucene	Apache Lucene https://lucene.apache.org/	33
mallet	MALLET http://mallet.cs.umass.edu/	524
smart	SMART Information Retrieval System of Cornell University	571
snowball	Snowball stemmer project http://snowball.tartarus.org/	174

a stem. So in the example, the words will be reduced to the stem "sequenc", not to the actual root "sequence". We use the Porter stemming algorithm [26], for which we borrowed the original Java implementation (listed in Appendix A). As we have not removed the apostrophe from inside the words, we needed to do a small modification: in the first step of the algorithm, remove also the possessive form from the end of words, e.g., remove "'s" from "sequence's". As stemming can have both positive (group words of similar meaning) and negative (cause words of different meaning to be grouped) effects, we allow it to be turned off.

14. As last step, we may want to remove words that are shorter than some length. For example, we may want to remove all words of length one, such as "x", "y", "z". By default, the minimum required length is 0, but we can increase it to evaluate its effect on performance.

3.8 Idf: Inverse document frequency weights

Not all words are created equal. For example, the mentioned stop words occur very frequently. But even if we remove these words that carry little meaning, we are still left with meaningful words, that occur much more frequently than other meaningful words. Take "sequence" for example — a large number of bioinformatics articles will contain this words. This does not mean, that "sequence" is one of the most important concepts describing all these articles. Words describing other concepts more meaningful for an article at hand might occur less frequently in the article than the word "sequence".

So we need a way to raise the importance of these words and lower the importance of "sequence".

To solve this, we can weight terms using inverse document frequency (IDF) [27]. Its value shows, how common a term is in a collection of documents. In a sense, it is a measure of how much information a word provides. The value is proportional to the number of documents in a corpus and inverse proportional to the number of documents the term occurs in. After logarithmically scaling this division, we obtain the following formula:

$$\mathrm{idf}(t) = \log(\frac{N}{\mathrm{df}(t)}), \tag{1}$$

where idf_t is the inverse document frequency of a word t, N is the number of documents in a collection of documents and df_t is the number of documents containing word t in that collection.

If IDF value is required for a word that does not occur in any of the documents, a division-by-zero will happen. We can thus also adjust the denominator and use the following formula:

$$\mathrm{idf}(t) = \log(\frac{N}{\mathrm{df}(t)+1}). \tag{2}$$

For our purposes, we want the IDF weight to be a value between 0 and 1, with 0 meaning that the term occurs in all documents and 1 meaning that it occurs in only one (Equation 1) or in no documents (Equation 2). Thus, we normalise the weights with the maximum possible value $\log(N)$:

$$\mathrm{idf}_{\mathrm{norm}}(t) = \frac{\log(\frac{N}{\mathrm{df}(t)})}{\log(N)}, \tag{3}$$

$$\mathrm{idf}_{\mathrm{norm}}(t) = \frac{\log(\frac{N}{\mathrm{df}(t)+1})}{\log(N)}. \tag{4}$$

Note, that if in the second case the value of $\mathrm{df}(t)$ is equal to N, the value of $\mathrm{idf}_{\mathrm{norm}}(t)$ will be slightly less than 0. In such case, we set the value to 0.

One important question when computing IDF scores for terms in a document, is the choice of the document collection. Different approaches could be tried. We could choose a very large corpus, such as all articles from the English Wikipedia, in which case scores will be approximately based on the frequency of the words in the language itself. Or, we could choose the document collection to be representative of the documents we are interested to compute the scores for, like all available journal articles, or all articles from a certain field. For the tokens in the concepts and queries we need IDF weights for, we have chosen to try the second approach.

3.8.1 IDF for concepts

In case of EDAM concepts, we choose as the document corpus all concepts that we read from the EDAM ontology file. Thus, for any word in an EDAM concept we are currently calculating IDF score for, we have at least one document in the corpus containing this word. That is, $\mathrm{df}(t)$ is at least 1 and we can use Equation 3. In case of EDAM version 1.14, N is 3218.

As we can compute IDF weights based on the data available in the ontology file alone, we don't need any additional input for processing the map of EDAM concepts. Processing is done as follows:

1. For each concept (i.e., document), we pre-process all its parts (i.e., labels, exact synonyms, etc), using the steps from section 3.7, and store the resulting lists of tokens as part of the processed concept structure.

2. From these lists of tokens, we collect all distinct tokens and increase their count by 1 in a global map, where the key is a token and the value is its count.

3. When all concepts have been processed, we make a new global map, where the key is a token and the value is its IDF score computed from Equation 3.

4. Now we can do a second pass: for each list of tokens in each concept, compute IDF scores for the tokens and store the list of scores alongside their corresponding list of tokens. The global maps can now be discarded.

We can see the most frequent terms (i.e., the terms with the smallest IDF scores) for concepts in Table 3. Most of these terms are not occurring frequently in general usage, but on the other hand we can expect to frequently see them in relation to bioinformatics tools or journal articles. Some frequent words in general usage, which were not eliminated by the rather small "lucene" stop words list, have also made it top the top (like "from", "other"). These words were probably mostly picked up from concept definitions and comments, which contain sentences rather than concise phrases of mostly only 2–3 words like labels and synonyms contain.

3.8.2 IDF for queries

In case of queries, we have different options for choosing the document corpus. If we are working with entries from BIO.TOOLS, we could choose as document corpus all entries from BIO.TOOLS. So in case of the BIO.TOOLS input file

Table 3: Top 30 most frequent stemmed terms for concepts from EDAM version 1.14 (3218 documents) and queries from bio.tools (2402 documents). In both cases, the lucene stop words list was used, which is rather small, so many function words can be seen at the top. Count shows the number of documents the term occurs in.

Concepts			Queries		
Term	Count	IDF score	Term	Count	IDF score
sequenc	1011	0.143	us	2215	0.0104
from	765	0.178	from	2046	0.0205
databas	657	0.197	all	2022	0.0221
protein	608	0.206	data	1876	0.0317
identifi	598	0.208	help	1853	0.0333
format	584	0.211	you	1833	0.0347
data	582	0.212	search	1820	0.0356
structur	457	0.242	can	1796	0.0373
us	359	0.272	new	1780	0.0384
molecular	343	0.277	http	1770	0.0392
id	333	0.281	pleas	1754	0.0403
gene	330	0.282	about	1753	0.0404
report	323	0.285	refer	1739	0.0414
entri	314	0.288	sequenc	1739	0.0414
includ	294	0.296	on	1725	0.0425
typic	294	0.296	version	1714	0.0433
align	272	0.306	develop	1701	0.0443
inform	252	0.315	inform	1696	0.0446
acid	240	0.321	more	1689	0.0452
other	208	0.339	avail	1674	0.0463
name	200	0.344	provid	1672	0.0465
more	198	0.345	softwar	1660	0.0474
annot	191	0.350	list	1654	0.0479
analysi	188	0.352	sourc	1632	0.0496
specif	188	0.352	support	1619	0.0506
gener	187	0.352	1	1615	0.0509
dna	183	0.355	download	1611	0.0512
featur	182	0.356	tool	1610	0.0513
predict	173	0.362	file	1602	0.0520
exampl	172	0.363	need	1601	0.0520

used in this thesis, N is 2402. However, in addition to running the automatic mapper on entries already present in the BIO.TOOLS input, we could be doing mapping for new entries to be included to the http://bio.tools Registry. Or, we could work on another input type, which is too small to be a document corpus for meaningful IDF calculations. Then, we could use the BIO.TOOLS entries as document corpus instead, especially if the other input is meant to be merged into the Registry eventually. In such case, we might need IDF scores for words, which are not present in any of the BIO.TOOLS entries. Therefore, for IDF calculations in case of queries, we use Equation 4.

The list of queries given as input for the mapper might not constitute the document corpus the IDF scores should be based on. Therefore, we need an external file we can read IDF scores for given terms from. We generate this external file using an external utility program (described in section 3.9).

In the IDF generation step, we take a list of queries as input (such as all entries from BIO.TOOLS) and proceed like in steps 1–3 from the previous section about concepts. Except we use the query parts (name, description, etc), we don't need to store the processed tokens for later use and we use Equation 4. We write the resulting maps to a file in the following form:

```
term<tab>term_count<tab>term_idf
```

When processing new queries, we use this generated file as source to compute and store all IDF weights for all tokens in the queries. As we used Equation 4, if a token is not present in the file, its IDF score will be 1. One thing to remember, however, is that the used preprocessing options (stopwords list used, whether stemming was done, etc) must be the same as were used when generating the IDF file. Which means, if we want to experiment with different pre-processing options, while also doing IDF scaling, we must generate and use different IDF files for each case.

In Table 3, we see the most frequent terms for queries from BIO.TOOLS. A lot of terms seem to concern navigating a web page or getting software or data ("help", "search", "http", "about", "version", "more", "software", "sourc", "download", "tool", "file", etc). The top entry — "us" — is both the personal pronoun not removed by the lucene stopwords list, but probably more frequently the result of stemming "use", "using", etc. A number ("1") is also at the top.

Scores for the top frequent queries are an order of magnitude lower than for top frequent concepts. This can be caused by the repetitive nature of web pages and documentation, especially when contrasted with a concise controlled vocabulary.

Some entries, like "sequenc" and "data", are near the top in both lists. This means that when matching these terms, their scores will be reduced

considerably. Which is why we should support disabling IDF for either concepts or queries, or disabling IDF for just some concept or query parts, or weakening the effects of the IDF weight scores.

3.9 Processor utility program

In sections 3.6.6 and 3.8.2 we defined a database to store fetched web pages and publications and a file to store IDF scores of all terms in a document corpus based on a large list of queries. To create, manipulate and query these files, we have created a separate utility program.

3.9.1 Database management

In this section, we refer to the content of a web page or documentation page and to the publication structure as *resource*, and to a URL or publication ID identifying this resource as *ID*. Except if there are differences in handling these resources, in which case these differences will be brought out.

The following operations concerning the database file have currently been implemented:

- Fetch and print a resource, given its ID
- Print a list of all IDs present in a file, output can be plain text or HTML with clickable links of the IDs
  - in case of publications, we can also try to convert all publication IDs to DOI and print the resulting list of DOIs
  - and more usefully, print all DOIs directly or indirectly obtainable from an input file for which no rules currently exist in the DOI YAML configuration file
- Initialize a new database, given a file name
- Fetch all resources present in a file and add them to the specified database, overwriting existing entries
- Fetch and store only those resources from a given file which are not already present in the database
- Given an ID, fetch and store its corresponding resource to the database
- Print all IDs present in the database, giving output in plain text or HTML

  - print all IDs of publications, whose fulltext part is not yet final

- Given an ID, print the corresponding resource in the database
- Given an ID, remove the corresponding resource from the database
- Remove all resources (i.e., remove all web pages or remove all documentation or remove all publications) present in the database
- Remove all resources whose ID matches given regular expression from the database
- Given an ID, refresh the corresponding resource in the database, i.e., if there is a resource with the given ID available, then re-fetch and re-store it, if no such resource if available, nothing is done
- Refresh all resources in the database
  - refresh all publications, whose ID was given as a PMID
  - refresh all publications, whose ID was given as a PMCID
  - refresh all publications, whose ID was given as a DOI
  - refresh all publications, whose ID was given as a DOI and whose registrant code matches the given registrant code; this is useful if we have just changed the rules for the given registrant in the YAML configuration file
  - refresh all publications, whose fulltext part is not yet final
- Refresh all resources in the database matching the given regular expression
- Commit changes to the database (this is usually not needed, as committing is done automatically for all operations defined here)
- Compact the database

3.9.2 IDF management

We have implemented the following operations concerning the IDF file:

- Generate a new IDF file based on the given input file of queries, getting the content for web pages and publications from the given database and using given pre-processing options

- If no database is available, we may still generate an IDF file, but in such case web page and publication content will not be available
- Even if a database is available, we may decide that the content of web pages or publications is too noisy and separately disable their usage in IDF calculations
- Print a list of most frequent terms, given an IDF file

3.10 Mapper: The mapping algorithm

We have obtained a set of concepts, each concept consisting of lists of tokens representing the pre-processed words of each of its parts (label, an exact synonym, etc). Similarly, we have obtained a list of queries, each query consisting of lists of tokens, each such list representing one query part, like name or one publication abstract. We want to give the best annotation suggestions for each input we receive. This means, for each query, finding out, scoring and ordering the concepts, whose parts match best the parts of the query in question. The next sections explain in a "bottom-up" fashion how this matching is done.

3.10.1 Approximate matching

We start by matching one word in the query to one word in a concept. To give a score to the match between a list of tokens in the query and a list of tokens in the concept, we should score each word-to-word match. One possibility would be to give a score of 1, if the words are equal, and score it 0, if they are not equal. However, sometimes there are small differences in words that otherwise should be considered equal. This can happen because of spelling differences between dialects, spelling mistakes, errors extracting text from a PDF, errors in optical character recognition, etc. We may still want to consider the words in equal in such cases, but giving some score between 0 and 1 to differentiate them from perfect matches.

To do such approximate matching, we use the Levenshtein distance metric between two strings. The Levenshtein distance between strings a and b is defined as the minimum number of edit operations to change a into b. We allow the following operations: inserting a character, deleting a character and changing one character into another. We may want to assign a cost to each operation, but we choose the simple case of all costs being 1, thus the distance is equal to the number of operations.

The Levenshtein distance for strings a and b can be recursively defined as:

$$\mathrm{d}_{a,b}(i,j) = \min \begin{cases} \mathrm{d}_{a,b}(i-1,j)+1 \\ \mathrm{d}_{a,b}(i,j-1)+1 \\ \mathrm{d}_{a,b}(i-1,j-1)+(a_i = b_i \ ? \ 1 : 0), \end{cases} \tag{5}$$

where $\mathrm{d}_{a,b}(i,j)$ is the distance between the first i characters of a and the first j characters of b. If length of a is m and length of b is n, then the distance between a and b is the value at $\mathrm{d}_{a,b}(m,n)$. We find this value using the standard dynamic programming technique that employs a $m+1$ times $n+1$ matrix, illustrated in Figure 2.

		a	b	r	a	c	a
	0	1	2	3	4	5	6
d	1	1	2	3	4	5	6
a	2	1	2	3	3	4	5
b	3	2	1	2	3	4	5
r	4	3	2	1	2	3	4
a	5	4	3	2	1	2	3

Figure 2: Calculating the Levenshtein distance between "abraca" and "dabra"

We denote the found Levenshtein distance as d and define the score between a and b as:

$$s_w = \frac{l - k \cdot d}{l}, \tag{6}$$

where $l = \max m, n$. We call k the `mismatch multiplier` parameter. It is essentially the real-valued cost of all three edit operations. If it is equal to 1, then s_w is the relative edit distance. If we decide, that an approximate match should get a lower score than its relative edit distance suggests, we increase the value of k.

In addition, a low enough score can very likely be the result of matching two unrelated but somewhat similarly spelled words, rather than the result of matching two related words that were accidentally spelled differently. We may decide, that such low scores should not raise the total score between two lists of tokens. Hence, introduce also the parameter `match minimum` —

if score s_w is below its value, then s_w is set to 0. Setting `match minimum` to 1 will mean that no approximate matching is allowed.

As an example, we take the words "optimize" and "optimise". Their Levenstein distance is 1 — from the one operation of changing "z" to "s". If k is 2, then the final score will be $\frac{8-2\cdot1}{8} = 0.75$. However, if stemming is done, then we would be comparing "optim" and "optimis". In such case, the distance is 2 — from inserting "i" and "s". The length of the longer word — l — is equal to 7. So total score is $\frac{7-2\cdot2}{7} \approx 0.43$. If we have set `match minimum` to 0.5, then the final score will be 0.

As another example, we can take "center" and "centre". Changing one into the other requires only the transposition "er" to "re". However, as we have not defined such operation, then we do a removal operation of "e" and insertion operation of "r", and get as distance 2.

These examples illustrated some problems our current approach might have. However, disabling stemming or defining a transposition operation with a cost smaller than 2 can have their own problems. The performance of these different approaches, and the performance of varying the parameter values of `mismatch multiplier` and `match minimum`, could be compared.

Another source of problems could be decisions made in the pre-processing phase when splitting the input string into words. For example, "pre-processing" would be pre-processed to the word "preprocess", while "pre processing" would be split to two words: "pre" and "process". Thus, comparing "pre-processing" to "pre processing" would mean comparing "preprocess" to "pre" and comparing "preprocess" to "process". Both would yield scores lower (or even 0), than it should be.

To compensate for this mistake, we could do the following: in addition to comparing every word from the source list with every world in the destination list, compare every two words with every world and compare every word with every two words. Which means, that we would also try to match "pre-processing" to "pre processing", which has an edit distance of 1. If we decide to enable this approach, then we allow one mismatch for free (so it will work even if `match minimum` is 1 for example).

Enabling this approach would mean, that a triple amount of comparisons will be done. So we may want to disable it, if performance becomes an issue. Also, we may want to expand this approach to enable comparisons of three words to one and so on. Hence, we introduce the parameter `compound words` — a value of 0 means that the described approach will be disabled, 1 means that two words to one word comparison will be enabled, 2 means that three words to one word comparisons will be enabled additionally, etc.

3.10.2 Proximity matching

We have a source list of words that we try to match to a destination list of words. A word in the destination list can be matched more than once by words in the source list. To get the final score, we could just independently sum up the maximum scores of the words in the destination list. However, we may also want to have a higher score, if not only words, but also phrases match.

For example, given the sentence "The quick brown fox jumps over the lazy dog", we may want that "brown fox" matches it better than "brown clever fox" which matches it better than "lazy fox". All three source lists have two exact matches to the destination list, so based on this alone, we can't differentiate between them. In the subsequent examples, we will use the following example destination sentence:

- "**quick brown fox jumps over lazy dog**"

To enable proximity matching, we define a penalty for the matched word in the destination list not having the expected words from the source list on its left and right side. If it has the correct words on its sides, there is no penalty and we set the proximity matching score s_p to 1; if it hasn't any of the corresponding words from the source list in its neighbourhood, then s_p is 0. For example, in matching the destination word list with "quick brown fox", the position score at "brown" will be 1. However, when matching it with "lazy fox", the position score at "fox" will be 0.

Naturally, there can be cases between 0 and 1. When matching the sentence with "slow brown fox", then there is a match on the right side of "brown", but no match on the left side. Therefore, to get the position score, we take the arithmetic average over two best position scores — 0 and 1 in this case, getting a score of $\frac{0+1}{2} = 0.5$. There are two special cases to this:

1. If the matched word is the first word of the destination list and also the first word of the source list, and analogously the last word of the lists, then we only take one position score into account. For example, matching the sentence with "quick brown" means that the position score at "quick" will be 1, as we require only "brown" to be at its right side.

2. If the destination list consists of only one word, then the position score is always 1 in case of a match.

In addition, we want to give a position score greater than zero to cases when the words are almost, but not exactly at the same relative positions in

the destination and source lists. For example, if the sentence is matched with "quick fox", then at position "quick" in the destination we expect to also have "fox" on its right, but it is actually one step further to the right. In this case, the position score will be o_1, which is set by the program parameter `position off by 1`. As a more complicated example, we match the sentence with "brown very quick fox". In this case, we have two position scores at position "fox": "brown" is off by 2 and "quick" is off by 1. The final score is taken as the arithmetic average: $\frac{o_1+o_2}{2}$, where o_2 is set by the parameter `position off by 2`. If we try with sentence "brown very quick fox jumps", then the position score will be $\frac{1+o_1}{2}$, as we now have the extra word "jumps" in the right position.

Words can also be switched, for example when matching with "brown quick". In this case, at position "brown" in the destination sentence, we have "quick" at the left side instead of being on the right side. We set the penalty of such case equivalent to o_1, that is `position off by 1`. Similarly, "brown very quick" will have a penalty of o_2.

We can also have the case, when the match on the right or left side is from the same word as the currently matched one. For example, when matching destination "foo buffalo buffalo bar" with "zoo buffalo zap". In the destination at the first "buffalo" position we have no match on the left side, but have a match on the right side, which originates from the same word from the source list as the current match. We decide to set the penalty of such case to o_1.

Additionally, we should decide what to do in case the words are not besides each other in neither the source list and destination list. For example, when matching the sentence with "quick grey fox", then "quick" is two positions to the left of "fox" in both the source and destination sentences. In such case, an off-by-1 error will happen because "quick" is off by one position in the destination list and an off-by-1 error will happen, because it is off by one position in the source list, so in total we get a score of o_2 at position "fox" in the destination. That way, we don't have to look more than two positions to the left and right for matches in the destination list when computing the proximity score. The words immediately next to the matched word in the source or destination are more important for the context and should contribute a higher score.

If position is off by more than 2, then the position score is 0. We may also be tempted to define o_3 and so on, or to define the score decrease as a decay function. However, for simplicity and performance, we can also say, that if position of two words in the source list differ by more than 2 in the destination list, that it is not a meaningful proximity and we should not take this into account any more.

In order to implement this, we do the following: for each match we find in the destination list (score not less than `match minimum`), remember also the position index of the word in the source list that matched. Previous examples can be seen in Table 4.

Table 4: Proximity matching

	quick	brown	fox	jumps	over	lazy	@pos :	score
quick brown fox	1	2	3	-	-	-	brown :	1
lazy fox	-	-	2	-	-	1	fox :	0
slow brown fox	-	2	3	-	-	-	brown :	$\frac{1}{2}$
quick brown	1	2	-	-	-	-	quick :	1
quick fox	1	-	2	-	-	-	quick :	o_1
brown very quick fox	3	1	4	-	-	-	fox :	$\frac{o_1+o_2}{2}$
brown very quick fox jumps	3	1	4	5	-	-	fox :	$\frac{1+o_1}{2}$
brown quick	2	1	-	-	-	-	brown :	$o_1/2$
brown very quick	3	1	-	-	-	-	brown :	$o_2/2$
quick grey fox	1	-	3	-	-	-	fox :	$o_2/2$
quick fox brown fox	1	3	2, 4	-	-	-	fox :	max

The last entry in the table is also interesting: the sentence is matched twice by "fox". In this case, we do proximity matching for both matches independently, compute the final score for them and assign the maximum of these two scores to position "fox" in the destination. How this is done is explained further down.

For a general summary of proximity matching, see Table 5.

In addition, we may decide that if a word from a context in the source list fits well into a context in the destination list, but the contexts have only been approximately matched, then this fitting score should be reduced. That is, when computing an individual position score, we reduce it if the match score at the position we found a relation to is less than 1.

3.10.3 Best scores at a destination position

Putting it together, we get the formula:

$$s_{ps_{j_l,k}} = s_{p_{j_l,k}} \cdot s_{w_k}^{pms}, \quad \text{where} \tag{7}$$

- k is a match from the neighbourhood N, which includes all matches at and around our current position j not farther than $j-3$ and $j+3$,
- j_l is the l-th match at our current position j,

Table 5: Word at position i in source matched a word at position j in destination. We look at all matches at positions $j-3, \ldots, j+3$ in the destination. For every such match we look at the position it occurred (column) and the value of the position index in the source list. If the value occurs in the table, we get a score from the "score" column. Otherwise, the score is 0. As result, we take the average of two highest scores (except in the special cases).

score	$j-3$	$j-2$	$j-1$	j	$j+1$	$j+2$	$j+3$
1			$i-1$		$i+1$		
o_1		$i-1$	$i-2$, $i+1$, i	$i-1$, $i+1$	$i+2$, $i-1$, i	$i+1$	
o_2	$i-1$	$i-2$, $i+1$, i	$i-3$, $i+2$	$i-2$, $i+2$	$i+3$	$i+2$, $i-1$, i	$i+1$

- $s_{p_{j_l,k}}$ is the individual position score from match j_l to k, computed using explanations given above
- s_{w_k} is the match score for match k
- pms is a scaling factor, controlled by the `position match scaling` parameter:
  - if set to 0, match score of linked position does not influence the position score
  - if set to 1, match score of linked position influences the position score linearly
  - if set to e.g., 0.5, influence will be according to square root, which is somewhere between no influence and linear influence

Next, we find the two maximum values of $s_{p_{j_l,k}}$ over all $k \in N$ and take the average of these two values as $s_{p_{j_l}}$. Except for the two special cases mentioned above, when

1. $s_{p_{j_l}}$ will be equal to the one maximum value
2. $s_{p_{j_l}}$ will be equal to 1

The final score for the l-th match at position j is as follows:

$$s_{j_l} = (s_{w_{j_l}} - \alpha \cdot (1 - s_{p_{j_l}})) \cdot \mathrm{idf}_l^{is}, \quad \text{where} \tag{8}$$

- $s_{w_{j_l}}$ is the match score between the word at position j in the destination and the word corresponding to the l-th match in the source list
- α is a parameter controlled by `position loss`, with a value between 0 and 1:
  - if set to 0, then the proximity score will be ignored
  - if set to 1, then the final score will be 0 (or less), if proximity score is 0
  - if set to a value between 0 and 1, its effect will be between these 2 extreme cases
- idf_l is the IDF weight of the word from the source list that corresponds to the l-th match
- is is the IDF scaling factor:
  - if set to 0, then the IDF scores of the words in the source list will be ignored
  - if set to 1, then the influence of these scores will be linear
  - if set to greater than 1, then the effect of IDF scores will be magnified
  - if set to a value between 0 and 1, then the effect of IDF scores will be lessened

Note, that we have decided to make the match score and proximity score independent, in the sense, that position loss is not scaled by the match quality, but is given as fraction of the full perfect score of 1. This means, that $s_{w_{j_l}} - \alpha \cdot (1 - s_{p_{j_l}})$ may be less than 0, in which case it is set to 0. This independence does not matter in case of exact matches that are not in the right place and approximate matches that are in the right place, but only for of out-of-context approximate matches, who we will be penalising more by choosing such form of calculating the final score.

3.10.4 Score between source and destination lists

Let's fix a destination list of words, for example, a query description. When we try to match it with one concept, we try to match it with the label of the concept, with all exact synonyms of the concept, etc. While we don't know yet, how to get the overall score for a match between a list of words and another list of words, let's just say that we get a score with a value between 0 and 1 for all the described matchings. The question is now, how do we aggregate these scores?

One option would be to take the average. But then, if we add a few synonyms to the concept, which don't match at all the query description, the average score will decrease. We would probably like it to remain the same.

We could also take the maximum score. But let's assume two sentences in the description: one talks about "High-throughput sequencing" and the other "Next-generation sequencing". We have a concept with label "High-throughput sequencing" and with an exact synonym to it "Next-generation sequencing". Both match the phrase in one of the sentences and we take whichever has the higher match score (if we have decided that matching a label is more important, we take its score). However, as the description mentions basically the same concept twice, then it seems to be an important feature of the tool or service it describes. So we may want to exploit the semantic richness of the concept more and base the score on matching both sentences.

We do this in the following way. When matching label to description, we get the values s_{j_l} for each word j in the description. Unless there is word repetition in the label, there will usually be only one such score per word j, or zero such scores in case of no match. We multiply these with $m_{label}^{1/\beta}$ m_{label} is set by the program parameter `label multiplier` and is a value between 0 and 1 enabling to decrease the importance of the label part. β is set by `score scaling` and will be explained later. As the label of a concept should be the most important and normative part of a concept, we usually set it to 1. Now, we define $s_{j_{max}}$ which is equal to the maximum over all l match scores s_{j_l} from label to word j of description, or to 0 if there are no matches.

Next, for an exact synonym of a concept, we get the scores $m_{exactsynonym}^{1/\beta} \cdot s_{j_l}$ for each word j of destination. If any of these scores is higher than the current maximum score $s_{j_{max}}$ at that position, then $s_{j_{max}}$ is set to that higher score. We repeat the procedure with all other exact synonyms and also with all other parts of the concept, including definition and comment, if present.

Like for the label, $m_{exactsynonym}$ is set by parameter `exact synonym`, which we can set to be a bit less than 1, if we want matches by `exact synonyms` matter a bit less than matches by `labels`. Analogously for other concept

parts.

In the end, we have a score $s_{j_{max}}$ for each word j of the description. Many of the scores will be 0, because no match from any concept part to a word in description was found. The other scores can be a result of a match between one or another concept part, that is, we have a mix of score sources for the score set of the description. Note, however, that when these scores were computed, then in proximity scoring, matches of only the same source were taken into account.

We get the final score of matching the description as:

$$s_{description} = \left(\frac{\sum_{j=1}^{n} s_{j_{max}} \text{idf}_j^{qis}}{n} \right)^{\beta} \cdot m_{description}, \quad \text{where} \tag{9}$$

- n is the number of words in the description
- idf_j is the IDF weight of the word j in the description
- qis is the query IDF scaling factor, which has value 0 or greater, its effects being explained at the end the previous section
- β is score scaling, set by parameter `score scaling`
  - calculating the score with the scaling value as 0 does not make sense, so we allow disabling score scaling by setting it to 0
  - setting it to 1 also effectively disables score scaling
  - a value greater than 1 makes bad scores even worse compared to good score
  - a value between 0 and 1 makes good and bad scores more equal and is the reason we have actually introduced this parameter
- $m_{description}$ is the description multiplier, similar to the multipliers for concept parts

Here, we weighted the scores with IDF weights of words in the destination list (the description sentence). These scores have already been weighted with IDF weights of words from the source list, this being done in the previous section. So each match score has been multiplied with $\text{idf}_c^{cis} \cdot \text{idf}_q^{qis}$, where idf_c is an IDF score for a word from a concept and idf_q is an IDF score for a word from a query; cis is the concept IDF scaling factor, set by the `concept IDF scaling` parameter, and qis is the query IDF scaling factor, set by the `query IDF scaling` parameter. If we do concept to query matching, then in Equation 8 we will have idf_l^{cis} and in Equation 9 idf_j^{qis}, and if we do query to

concept matching, then in Equation 8 we will have idf_l^{qis} and in Equation 9 idf_j^{cis}.

To elaborate on β, let's consider the following example. Let's have two descriptions – one 10 words long and the other 1000 words long. Let's say, that we match the first one perfectly with one word and nothing else, with proximity scoring disabled, thus getting the score $\frac{1}{10} = 0.1$. Similarly, let's match the same word 10 times in the second description, thus getting the score $\frac{10}{1000} = 0.01$. Should the second match be scored 10 times worse? To score equally, the second description would have to contain the word 100 times, which is not a reasonable requirement. To make the scores not depend linearly on the number of words, we use the `score scaling` parameter. Let's set β to 0.2. Then the score in the first case will be $0.1^{0.2} \approx 0.63$ and in the second case will be $0.01^{0.2} \approx 0.40$.

The same way we computed $s_{description}$, we will also find $s_{keyword_1}$, $s_{keyword_2}$, $s_{publication_{1_{title}}}$, $s_{publication_{1_{keyword_1}}}$, $s_{publication_{2_{keyword_1}}}$, $s_{publication_{2_{keyword_2}}}$, $s_{publication_{3_{efoTerm_1}}}$, etc. There is an additional step done in case of mined terms (EFO and GO). Namely, after finding the score of matching these, this score is additionally multiplied by $\left(\frac{count}{f}\right)^{\beta}$, where *count* is the number of times the term was mined (see section 3.6.1), f is the number of words in the full text the term was mined from and β is `score scaling`.

3.10.5 Bi-directional matching

We can also match the other way: from query to concept. However, we will do it with a slight difference, namely, we will not calculate the score by matching all query parts to the concept, but we will calculate it by matching only all query parts of the same type to the concepts. The reason being, that in case of concepts we have a well-defined controlled vocabulary, where each part of a concept is strongly connected to the main idea of the concept. But in case of queries, we have parts of varying quality. For example, a web page that can contain a lot of noise and contain words only weakly related to the tool or service it is attached to in the query. So it would be better not to mix the noise picked up from this web page with a better quality source, like query keywords.

Thus, we calculate the scores of matching concept parts separately for each query part type and look at these scores at a later stage. Among others, we will get the following scores:

- $s_{label,name}$ – score of matching the label of a concept with the name of a query

- $s_{exactsynonym_1,keywords}$ – score of matching an exact synonym of a concept with all keywords of a query

- $s_{narrowsynonym_2,publication_{mesh}}$ – score of matching a narrow synonym of a concept with all meshTerms of all publications of a query

Like mentioned in the previous section, we have also multipliers for the query parts, such as $m_{description}$ and $m_{publication_{mesh}}$. However, as we are not mixing description with meshTerms yet, we have currently no need to differentiate between the query part types. So we set all these multipliers to 1. We may have a need for them in the future however. Namely, we may decide to differentiate between query parts within one query type, for example between a primary and other publication. But until then, there are no program parameters available to set these multipliers to some other value than 1.

So now, we can match from query to concept and from concept to query. But which one to choose?

Consider the one-word sentence "ideas". We may match to it from "green ideas". Or we may match to it from "colorless green ideas sleep furiously". In both cases, the match score will be the same. However, we may argue, that the second case contains more ideas than the first case, and as the first case manages to more succinctly describe the one-word sentence, we should prefer the first case. So we could also do an opposite matching: from "ideas" to the two cases. In which case, the first case will get a higher score indeed. However, we don't know if "ideas" is part of concepts or part of a query. To solve this situation, we could take the average of the two scores as the final score.

Taking the average makes also better the situation, where we have a much longer sentence, containing thousands of words, but the word "ideas" only once. If we match this sentence to "ideas" we get a perfect score, but if we match the other way, we get a very low score. Both scores could be considered to be wrong, as a very low score should also not be indicative of successfully extracting an idea from a long text. Or, on the other hand, if "ideas" was just noise in this long text (illustrated by the fact that it occurred only once), then taking an average means that this very low score will bring the score down from being a perfect score. So, in a way, taking the average of the two scores is a way to reduce errors and noise.

However, we have more than two scores currently. We decide to combine our current scores the following way: we take $s_q = \max(s_{name}, s_{keywords_1}, \dots)$ and $s_{c,name} = \max(s_{label,name}, s_{exactsynonym_1,name}, \dots)$. Then, we calculate the

weighted average of s_q and $s_{c,name}$ as follows:

$$S_{name} = \frac{w_q s_q + w_c s_{c,name}}{w_q + w_c}, \tag{10}$$

where w_q and w_c are `query weight` and `concept weight`, respectively. If w_q is set to 0, then matching from concept to query is not done at all. If w_q is set to a value larger than w_c, then matching from concept to query is more important than doing the opposite.

Analogously, we calculate also the values $S_{keywords}, S_{publication_{mesh}}, \ldots$. In case of mined terms, we consider all of them to be of the same query part type, which is why we will get $S_{minedTerms}$ instead of $S_{efoTerms}$ and $S_{gotTerms}$.

3.10.6 Final score between a query and a concept

We have now the following 11 scores to describe a match between a query and a concept: S_{name}, $S_{webpages}$, $S_{description}$, $S_{keywords}$, $S_{publication_{title}}$, $S_{publication_{keywords}}$, $S_{publication_{meshTerms}}$, $S_{publication_{minedTerms}}$, $S_{publication_{abstract}}$, $S_{publication_{fulltext}}$, S_{docs}. The question now is again: how do we combine them to a single match score?

We could again take the maximum value. Before we do that, however, we should normalise the values to a common range. Namely, matching a query part made of long text (such as a publication full-text) is harder than matching short text (such as a keyword made of two words). Even if we find very good matches for both, the score will still be noticeably smaller for the former match. As, in case of a very good match against a keyword, we can expect to match the entire keyword (both words), but against a full-text, we are not expecting to match the entire full-text (all words), even matching a few percents is good.

So, to normalise these values, we take as new scores $S_{publication_{fulltext}} \cdot n_{publication_{fulltext}}$ and $S_{keywords} \cdot n_{keyword}$, choosing the normalisers `publication fulltext normaliser` and `keyword normaliser` such, that scores of an excellent, good and mediocre match in case of publication full-text would be of comparable value to scores of an excellent, good and mediocre match in case of keywords. Similarly, we calibrate the rest of the normalisers n_{name}, $n_{description}, \ldots$.

Now, with the scores normalised, we can take the maximum score as our final score S_{max}. This score is made up of two part – s_q and s_c – so we could also remember, which query part was matched the best (with score s_q) and which concept part was matched the best (with score s_c), and indicate these best matching parts in the output.

We could also indicate in the output, which scores we consider good, which mediocre and which bad. It might be possible to set these decision

limits automatically, currently however, we are setting these manually via program parameters. The limits depend on the normaliser values, but can also depend on the input type, concept branch, etc. For example, we could set that scores over 0.63 are good and that scores under 0.57 are bad. Parameters to do so are `good score topic`, `bad score topic`, `good score operation`, etc.

But coming back to our decision to choose the maximum of the S scores as our final score. Let's assume that we get the maximum score from a match against a meshTerm. This could well be a mistake, there could have been some approximations made when translating an idea describing the query part to the meshTerm or maybe the query has over 10 terms attached to it and the meshTerm we got an excellent match for is only tangentially related to the query, with some other concepts describing the query better. Let's say there is such other concept, and we get good scores when matching the query description, a query publication title and publication abstract with this concept. But if all three are smaller than the score of matching the meshTerm, we still choose the concept matching the meshTerm, even if it didn't match any other part of the query.

So, we could take the approach, that the more parts of the query have a good match to a concept, the stronger is the connection between the query and the concept. To implement it, we can take an average over all scores S of all query parts that are present in the query. This will also even out noise, such as meshTerm from the previous example.

Thus, we get the formula:

$$S_{avg} = \frac{w_{name}\,(S_{name}n_{name})^{\gamma} + \cdots + w_{doc}\,(S_{docs}n_{doc})^{\gamma}}{w_{name} + \cdots + w_{doc}}, \tag{11}$$

where the weights w_{name} and w_{doc} are set by `name weight` and `doc weight`, weights for query parts not present in the query are 0 and γ is the `average scaling` parameter.

Weights are used to make the scores S of some query parts more important or less important than others. For example, the description of a tool should contain, on average, more pertinent information than, say, the web page of a tool, which might contain more noise. So, we could decide to take into account the score of matching a concept to a description with twice the weight as matching it to a web page. But on the other hand, the web page has potentially more content and may contain ideas not present in the short description, therefore we don't set its weight to a very small value or 0 either. Just like for other parameters, we could optimise the values of the weights by measuring the accuracy of the automatic mapper over a large query set while varying the weight values.

The `average scaling` parameter serves an opposite function to the `score scaling` parameter described before. Namely, when calculating S_{avg}, we may want to enlarge the difference between good, mediocre and bad matches. For example, we may not want two bad matches, having only half the score of a good match, to result in an equal or better match. So mediocre and bad matches should boost the final score less, than their score numbers would suggest. Note, that the scores have actually been scaled up by β previously. Thus, we should set γ to at least $\frac{1}{beta}$ so that the scores would again be approximately equal to the number of matched words divided by the total number of words. If we set γ bigger than that, this will start to emphasis good matches and marginalise bad ones.

This scaling makes scores a lot smaller. While previously – depending on `score scaling` and other parameters of course – good scores would be in the range 0.7 (70%) for example, then now – if $\beta \cdot \gamma$ is 2 for example – already 0.01 (1%) could be considered a good score. And while good scores would be in the range of a few percent, then the theoretical maximum, although extremely unlikely, would still be 100%.

While this is not a problem, what might constitute an annoyance, is that scores S_{avg} between different queries might not be comparable between queries anymore. This could happen for instance, when one query has only keywords available, which could be considered a good source of good matches, while for another query we have many other parts (like webpage, doc, etc) available. All these other parts actually help to differentiate between concepts better for the query, but they will also result in the score number being smaller (as we can't expect all these parts to have a good match to a concept, even in case of a good match). This means, that we can't set limits to differentiate good, mediocre and bad scores anymore. So, as a workaround, we still set the limits based on S_{max} and also decide the match quality based on S_{max}. Note, that ordering matches by S_{avg} and ordering matches by S_{max} does not give the same result. So, we actually still select the best match according to S_{avg} and use the match quality as an additional hint.

As even when using S_{avg}, we still would want to find S_{max}, then we still have to normalise scores. Using the normalised scores when calculating S_{avg} (in Equation 11) has the added benefit of being able to work with the weights under the assumption, that setting all weights to 1 would mean that all query parts have equal importance.

Sometimes, we may not want to use S_{avg} at all and just limit ourselves with S_{max}. The program parameter `mapping strategy` can be set to either "average" or "best" to select the required strategy.

3.10.7 Final output

As final output we would like to give out for each query a list of concepts, ordered from best matching to least matching. Therefore, for a query, we calculate the S_{avg} or S_{max} score, depending on strategy, against all concepts in the ontology given as input, and order the concepts based on this score. But there are some additional things to consider.

First, as mentioned in the background, a concept in the EDAM ontology is from one of the following branches: *topic*, *operation*, *data*, *format*. We would like to get results for each of these branches, therefore, we should not mix the results of matching different branches. Therefore, instead of one ordered list of results, we get four lists. Or actually, we may want to disable matching concepts from some branch. This can be done using the program parameter `branches`, which is a list of values describing the branches where matching is enabled (e.g., "topic", "operation").

As also mentioned in the background, some concepts can be obsolete. With the boolean parameter `obsolete` we can decide if obsolete concepts should be matched against or not.

We are not interested in the full list of concepts, ordered by score, but only in some limited number of best matches. The `match` parameter is used to set the number of top matches we are interested in. So, while matching concepts, if a score of the concept makes it the top, it gets inserted into the list to the correct position, otherwise it is discarded.

So, as final output we can expect the following: a list of queries that were matched and for each query, a list of top matched concepts per branch. For each match, we can output the label of the matched concept, the concept part corresponding to s_c (and potentially its text content), the query part corresponding to s_q and the final match score (with the hint about match quality described above).

3.10.8 Optimisation

Before starting to optimise the parameters and getting the results, we should see if we can optimise the speed of the mapper. This can be important, when the number of entries in the input is very large. In addition, this can also be important when using the automatic mapper in an on-line situation, when we would expect that the annotation suggestions don't appear more than a few seconds after specifying the description of a tool or service.

One way to decrease the run time of the mapper, is to disable or modify some parameters. For example, setting `compound words` from 1 to 0 would cause roughly 3 times less comparisons to happen. The speed gain can be

even bigger – for example, if we have approximate match disabled and set `compound words` from 0 to 1, then this will automatically allow 1 mistake for the compound comparisons and this can increase the run time by an order of magnitude. We could also disable bi-directional matching by setting either `concept weight` or `query weight` to 0 – this will cut run time more or less by half. We could also disable matching for some query parts, especially for web pages, docs and publication full-text, as these can contain the longest texts. Or disable matching in some concept branch we are not interested in – run time will decrease by roughly how many concepts were disabled compared to the time it took when they were enabled.

However, we also looked at optimising performance without manipulating parameters. We found, that unsurprisingly, the majority of time is spent doing string comparisons between words. The first approach we had taken, was quite naive: for every word pair, compute the Levenshtein distance, then calculate the score given by Equation 6 and if that score was not smaller than `match minimum`, then count it as a match. This can be optimised quite easily. In case `match minimum` is 1, we have disabled approximate matching, and in such case just use string equality check for finding a match. Otherwise, compute the maximum number of allowed errors, such that the score will still be greater than `match minimum`:

$$d_{max} = \left\lfloor \frac{l \cdot (1 - s_{w_{min}})}{k} \right\rfloor \tag{12}$$

Then, we notice that the edit distance is equal to at least the difference in length between the compared strings. Thus, if the string lengths differ by more than d_{max}, then the score will be less than `match minimum`, so we don't need to know the exact score and can skip calculating the Levenshtein distance. Using these optimisations, we get the following results: if `match minimum` is disabled, then total mapping run time is decreased more than 20 times; if `match minimum` is set to 0.35, then run time is decreased a bit over 1.5 times. These improvement factors are just indicative, as they can depend on various parameters.

Next, we will try to improve the time of approximate matching even further. Computing the Levenshtein matrix takes $\mathcal{O}(mn)$ in time, where m and n are the lengths of the strings whose edit distance we are computing. From Figure 2 we notice, that not all values in the matrix need to be computed to get the lowest right value that represents the edit distance. We use improvements made to the basic algorithm by Ukkonen [28]. On the figure, we have coloured diagonal: on the green one, the error is at least 0, on the yellow ones, the error is at least 1, etc. Ukkoken noticed, that a number of diagonals far away from the central one can be discarded outright. Also, if the

cost of operations is 1, then in each diagonal we only need to remember the position, where one number changes to the next one. Thus we can translate the Levenshtein matrix coordinates to a smaller matrix, with which we will actually be working. We work by iterating over the error – first find places where 0 changes to 1, then places where 1 changes to 2, 2 to 3 etc. If we arrive at the end of the diagonal containing the edit distance (the upper yellow diagonal on the figure), we have an answer. Berghel and Roach improved the algorithm even further [29]. They noticed, that the number of values calculated can be reduced even further. For example, not all places where 2 changes to 3 have to be found — this is often the case on diagonals farther away from the centre diagonals, where have basically reached a dead-end concerning the optimal path. We implemented the algorithm given in the "Afterword" section of their article. Notice, that the algorithm will calculate the edit distance for two strings. However, if it will be greater than d_{max}, then we are not interested in it. As the algorithm works over a loop with d as index, we can stop executing it if d grows larger than d_{max} and we haven't an answer yet, returning an invalid distance.

This new algorithm is $\mathcal{O}(d_{max} \cdot \min(m, n))$, where d_{max} depends on k and $s_{w_{min}}$, which are set to 2 and 0.35, like in the simple optimisation case. Compared to the simple optimisation case, the increase in speed is a bit over 3 and compared to the naive case, run time has decreased roughly 5 times (for used parameters). As m and n are rather small (stemmed English language words, so usually less than 10), then speed gains are not overly dramatic. However, the speed is increased 5 times for the whole mapper run (as other operations are at least an order of magnitude less costly than calculating the edit distance), then this could still be considered an important gain.

Of course, another option is to disable approximate matching all-together. In this case, the built-in Java string comparison will be done, which just compares strings character-by-character until a difference is found (if string lengths were not equal, then this is not done). In such case, we get a speed increase of about 4.5 times over the improved edit distance algorithm (when `match minimum` is 0.35 and `mismatch multiplier` 2). This a difference between the overall mapper run times, the actual difference between the options is probably a bit higher, as now other operation costs besides comparing words to words also start to become more important (although still many times less costly).

One such operation, which is also at the core of the mapper, is proximity matching. A naive algorithm for it could be considered as such: for a destination word list, make an array with size equal to the number of words, and for each array position make a list to save the matches. However, as the number of matches will be rather small, compared to the number of words in the list,

then this array would usually be quite sparse. So it might be better to make just one list of matches for the destination list: for each match, we save it to the list with the match score and source list index, as before, and adding the destination list index. If matches are added in destination list index order into the match list, then we can just move upwards and downwards in the list for a position to get the nearest matches on the right and left of that position. If we make `match minimum` smaller, then there will be many more matches and our position matching will get slower. However, approximate matching will also get slower and position matching will still be an order of magnitude slower from it.

Other optimisations, like replacing some variable length list structure with arrays, could yield even more speed gains. But these will most likely increase performance a few percent instead of a few times. One algorithmic optimisation we could try: stop calculations once it has been determined, that the match we are currently calculating a score for can't theoretically make it to the top any more.

However, if we have multiple processor cores available, then we can still decrease run time multiple times by parallelising the algorithm. We have chosen the following way: as calculating matches for a query is independent from calculating matches for another query, then each query can be handled by a separate processor thread. We have a list of queries, from which threads can pick a query, and a list of results, where the mapping is put. As we want to preserve the order of queries, each thread also remembers the index of the query it took. As the synchronisation cost is minimal (the get and put operations for the lists), then we can expect speed to increase linearly with the number of threads. Increasing thread count from 1 to 8 on an Intel Core i7 system with advertised thread count of 8, increased speed by 4.0. This is expected, as the processor employs Intel's Hyper-Threading: each actual processor core present two virtual cores to the operating system.

This enables us to do processing of multiple queries, for example during parameter tuning, multiple times faster. If we want to get speed gains for one query this way, parallelisation should be done deeper in the mapper.

3.11 Output: Getting the results

We can output the mapping results in two formats: plain text and HTML. The plain text output can be used as input for other tools for displaying and doing analysis of results. The HTML output is for displaying the results in a more intuitive way than just text file lines, with some added conveniences, such as displaying the content of queries or proving links to tool homepages or matched concept URLs.

3.11.1 As plain text

The plain text output can be written to standard output or a text file. An example result for a tool from the BIO.TOOLS input, when mapping was enabled for all branches and top 3 matches per branch are output, follows:

```
 1 PASTA | Genetic variation | http://edamontology.org/topic_0199 |
   false | topic | narrow_synonym | webpage | 0.0036608106763123733
 2 PASTA | Small molecules | http://edamontology.org/topic_0154 |
   false | topic | narrow_synonym | publication_mesh | 0.0035594688293943627
 3 PASTA | Pathology | http://edamontology.org/topic_0634 |
   false | topic | label | publication_abstract | 0.0032547220482351384
 4 PASTA | Aggregation | http://edamontology.org/operation_3436 |
   false | operation | label | description | 0.011032163882532895
 5 PASTA | Protein secondary structure prediction |
   http://edamontology.org/operation_0267 | false | operation |
   exact_synonym | publication_fulltext | 0.0032746269399028236
 6 PASTA | Protein secondary structure prediction (coils) |
   http://edamontology.org/operation_0470 | false | operation |
   label | publication_fulltext | 0.002947160256308562
 7 PASTA | Protein sequence | http://edamontology.org/data_2976 |
   false | data | label | description | 0.02983743484576472
 8 PASTA | Sequence | http://edamontology.org/data_2044 |
   false | data | label | webpage | 0.027199031526033222
 9 PASTA | Protein residue | http://edamontology.org/data_1756 |
   false | data | exact_synonym | webpage | 0.024387215023225817
10 PASTA | protein | http://edamontology.org/format_1208 |
   false | format | label | webpage | 0.030948125267984246
11 PASTA | Protein secondary structure format |
   http://edamontology.org/format_2077 | false | format |
   label | publication_abstract | 0.017590781403984544
12 PASTA | FASTA | http://edamontology.org/format_1929 |
   false | format | label | webpage | 0.016265366321642064
```

Lines have been wrapped to fit the page. Line numbers are not present in the output.

Each line consists of the following, separated by "|":

1. query name (a tool or service name)
2. label of matched concept
3. URI of matched EDAM term
4. whether the concept is obsolete
5. branch of the concept
6. best matched concept part (corresponding to s_c)

7. best matched query part (corresponding to s_q)

8. match score

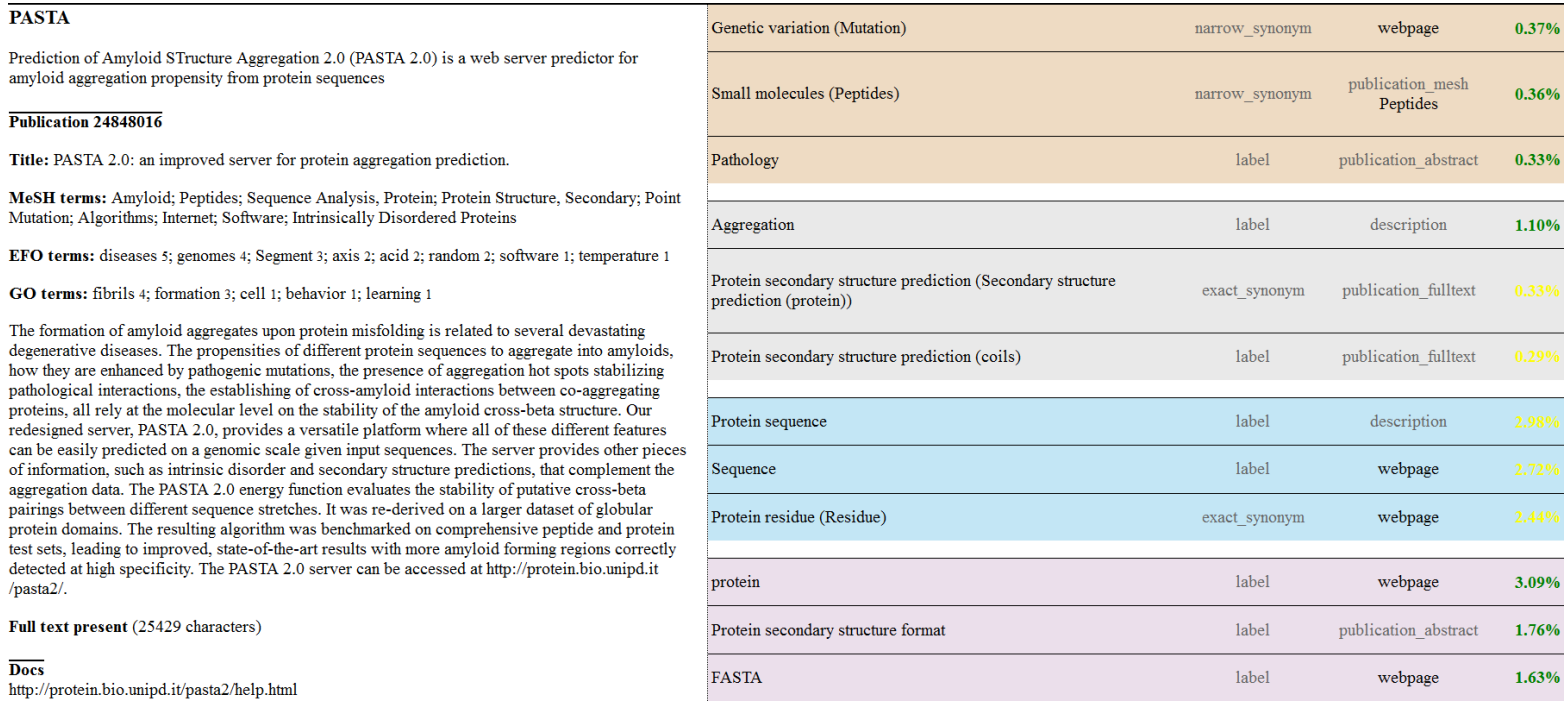
The output could be customised (entried grouped, order change, etc), per input type for example, depending on annotator needs. We have done so for SEQWIKI for example.

3.11.2 As HTML

An example HTML output, equivalent to the plain text output of the previous section, can be seen on Figure 3.

On the left hand side, we can see the query content. First, the tool name (PASTA) followed by a short description. It has one publication, with PMID 24848016, attached to it. We can see the publication title, MeSH, EFO, GO terms and abstract. The full text is not presented, only its character count is printed. The tool has also one documentation URL attached, which is also brought out. The homepage can be accessed by clicking on the name (PASTA). Also, clicking on the Publication header goes to the corresponding journal article and clicking on terms goes to the individual term web pages. For EFO and GO terms, their count (how many times they were found in the full-text) is also brought out. The publication is missing one part, namely, author assigned keywords.

On the right hand side, we can see found matches, grouped by branch and ordered by score. From top to bottom, the branches are *topic*, *operation*, *data* and *format*. For each branch we can see the top 3 matches. First, we have the matched concept label, followed by the best matched concept part, followed by the best matched query part. In case the best matched concept part is not a label, the content of the concept part is brought out in parentheses after the label. Such as the narrow synonym "Mutation" for label "Genetic variation". In case the description of the best matched query part is ambiguous, it is made more specific. For example, "Peptides" is added for publication_mesh. Clicking on the matched concept label takes to the corresponding EDAM URI. If the matched label text is stricken through, this means the concept is obsolete (we did not match any obsolete concepts in the top 3). In the last column scores can be seen. Here, S_{avg} has been used with large `average scaling`, so numbers are low. Colours are set by the score limits given as parameters and mean the following: green for good match, yellow for mediocre match and red for bad match. As the colours are based on S_{max} and not S_{avg}, then it may happen in some cases (like in the case of a match composed of only one excellent match between parts and another



PASTA

Prediction of Amyloid STructure Aggregation 2.0 (PASTA 2.0) is a web server predictor for amyloid aggregation propensity from protein sequences

Publication 24848016

Title: PASTA 2.0: an improved server for protein aggregation prediction.

MeSH terms: Amyloid; Peptides; Sequence Analysis, Protein; Protein Structure, Secondary; Point Mutation; Algorithms; Internet; Software; Intrinsically Disordered Proteins

EFO terms: diseases 5; genomes 4; Segment 3; axis 2; acid 2; random 2; software 1; temperature 1

GO terms: fibrils 4; formation 3; cell 1; behavior 1; learning 1

The formation of amyloid aggregates upon protein misfolding is related to several devastating degenerative diseases. The propensities of different protein sequences to aggregate into amyloids, how they are enhanced by pathogenic mutations, the presence of aggregation hot spots stabilizing pathological interactions, the establishing of cross-amyloid interactions between co-aggregating proteins, all rely at the molecular level on the stability of the amyloid cross-beta structure. Our redesigned server, PASTA 2.0, provides a versatile platform where all of these different features can be easily predicted on a genomic scale given input sequences. The server provides other pieces of information, such as intrinsic disorder and secondary structure predictions, that complement the aggregation data. The PASTA 2.0 energy function evaluates the stability of putative cross-beta pairings between different sequence stretches. It was re-derived on a larger dataset of globular protein domains. The resulting algorithm was benchmarked on comprehensive peptide and protein test sets, leading to improved, state-of-the-art results with more amyloid forming regions correctly detected at high specificity. The PASTA 2.0 server can be accessed at http://protein.bio.unipd.it /pasta2/.

Full text present (25429 characters)

Docs
http://protein.bio.unipd.it/pasta2/help.html

Genetic variation (Mutation)	narrow_synonym	webpage	0.37%
Small molecules (Peptides)	narrow_synonym	publication_mesh Peptides	0.36%
Pathology	label	publication_abstract	0.33%
Aggregation	label	description	1.10%
Protein secondary structure prediction (Secondary structure prediction (protein))	exact_synonym	publication_fulltext	0.33%
Protein secondary structure prediction (coils)	label	publication_fulltext	0.29%
Protein sequence	label	description	2.98%
Sequence	label	webpage	2.72%
Protein residue (Residue)	exact_synonym	webpage	2.44%
protein	label	webpage	3.09%
Protein secondary structure format	label	publication_abstract	1.76%
FASTA	label	webpage	1.63%

Figure 3: Result of mapping in HTML format for "PASTA".

match composed of several good matches between parts), that green scores are below yellow or red score and yellow scores are below red scores.

3.12 Benchmark: Evaluating performance

As mentioned in the Input section, some input types have manual annotation data available. In some cases, the manual curation is outdated or incomplete or still ongoing, and it can always be debated whether the concrete manually chosen terms are the best ones for a given resource. However, these manual annotations still provide a valuable reference dataset against which to benchmark the automatic mapper.

3.12.1 Benchmark output

We call a true positive (TP) a term that was both found by the automatic mapper and annotated manually with; a false positive (FP) a term that was found by the automatic mapper, but which was not used to annotate the resource; a false negative (FN) a term that was used to manually annotate the resource, but which the automatic mapper failed to find. True negatives (TN) would be the EDAM terms that both the annotator and automatic mapper ignored, but we are not interested in this large set.

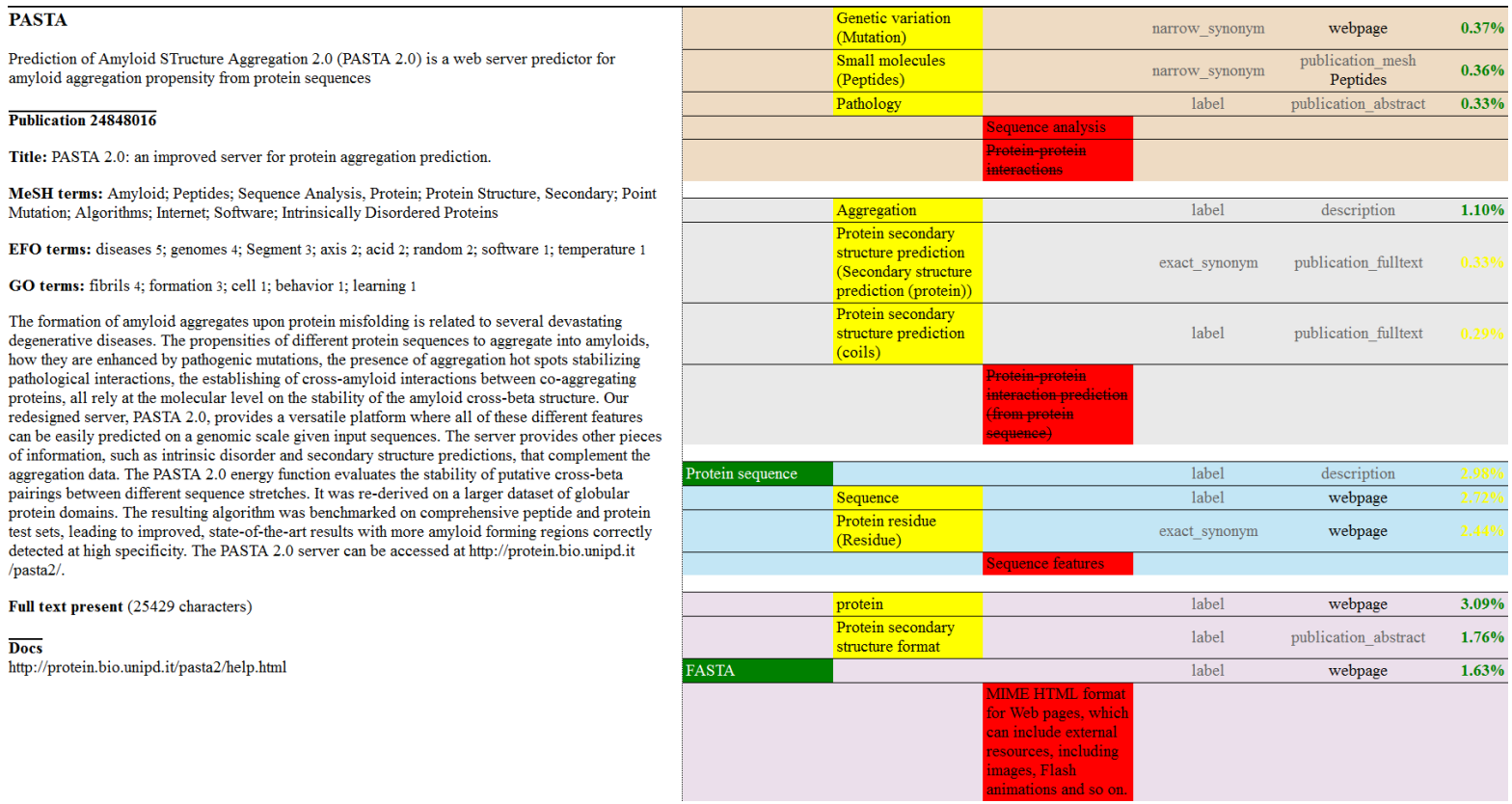
To visually see the performance of the mapper against the manual annotations, we can, for each query, add mapping correctness information for each match. We add this to the HTML output format of the previous section, the result for the same PASTA tool can be seen on Figure 4. TP concepts are green, FPs yellow and red FNs have been added to the end of the list. So, the first match in the data branch and the third match in the format branch are TPs. We can also see, that two manual annotations not found by the automatic mapper are obsolete concepts (as they are stricken through).

3.12.2 Metrics

In addition to be able to see the performance of the automatic mapper at individual query level, we might want to have an overview of the overall performance over all entries. So, a table of mean metrics is also output as part of the benchmark HTML report. An example for the BIO.TOOLS input, with 2402 entries and where top 3 matches were returned, can be seen in Table 6.

The metrics are defined as follows:

Precision $\frac{TP}{TP+FP}$



PASTA

Prediction of Amyloid STructure Aggregation 2.0 (PASTA 2.0) is a web server predictor for amyloid aggregation propensity from protein sequences

Publication 24848016

Title: PASTA 2.0: an improved server for protein aggregation prediction.

MeSH terms: Amyloid; Peptides; Sequence Analysis, Protein; Protein Structure, Secondary; Point Mutation; Algorithms; Internet; Software; Intrinsically Disordered Proteins

EFO terms: diseases 5; genomes 4; Segment 3; axis 2; acid 2; random 2; software 1; temperature 1

GO terms: fibrils 4; formation 3; cell 1; behavior 1; learning 1

The formation of amyloid aggregates upon protein misfolding is related to several devastating degenerative diseases. The propensities of different protein sequences to aggregate into amyloids, how they are enhanced by pathogenic mutations, the presence of aggregation hot spots stabilizing pathological interactions, the establishing of cross-amyloid interactions between co-aggregating proteins, all rely at the molecular level on the stability of the amyloid cross-beta structure. Our redesigned server, PASTA 2.0, provides a versatile platform where all of these different features can be easily predicted on a genomic scale given input sequences. The server provides other pieces of information, such as intrinsic disorder and secondary structure predictions, that complement the aggregation data. The PASTA 2.0 energy function evaluates the stability of putative cross-beta pairings between different sequence stretches. It was re-derived on a larger dataset of globular protein domains. The resulting algorithm was benchmarked on comprehensive peptide and protein test sets, leading to improved, state-of-the-art results with more amyloid forming regions correctly detected at high specificity. The PASTA 2.0 server can be accessed at http://protein.bio.unipd.it /pasta2/.

Full text present (25429 characters)

Docs
http://protein.bio.unipd.it/pasta2/help.html

	Genetic variation (Mutation)		narrow_synonym	webpage	0.37%
	Small molecules (Peptides)		narrow_synonym	publication_mesh Peptides	0.36%
	Pathology		label	publication_abstract	0.33%
		Sequence analysis			
		~~Protein-protein interactions~~			
	Aggregation		label	description	1.10%
	Protein secondary structure prediction (Secondary structure prediction (protein))		exact_synonym	publication_fulltext	0.33%
	Protein secondary structure prediction (coils)		label	publication_fulltext	0.29%
		~~Protein-protein interaction prediction (from protein sequence)~~			
Protein sequence			label	description	2.98%
	Sequence		label	webpage	2.72%
	Protein residue (Residue)		exact_synonym	webpage	2.44%
		Sequence features			
	protein		label	webpage	3.09%
	Protein secondary structure format		label	publication_abstract	1.76%
FASTA			label	webpage	1.63%
		MIME HTML format for Web pages, which can include external resources, including images, Flash animations and so on.			

Figure 4: Benchmarking "PASTA".

Table 6: Mean metrics

	topic	*operation*	*data*	*format*	average
Precision	10.57%	10.28%	13.70%	6.64%	10.30%
Recall	20.59%	25.49%	23.75%	14.18%	21.00%
F1 score	13.17%	14.15%	16.47%	8.54%	13.08%
F2 score	16.44%	19.07%	19.78%	10.99%	16.57%
Jaccard index	8.71%	9.45%	10.84%	5.65%	8.66%
Average precision	15.33%	18.97%	16.68%	8.46%	14.86%
R-Precision	13.48%	15.01%	15.01%	6.06%	12.39%
Discounted cumulative gain	19.92%	24.09%	22.80%	12.93%	19.94%
DCG (alternative)	18.51%	21.47%	21.26%	10.73%	17.99%

Recall $\frac{TP}{TP+FN}$

F1 score $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision}+\text{Recall}}$

F2 score $5 \cdot \frac{\text{Precision} \cdot \text{Recall}}{4 \cdot \text{Precision}+\text{Recall}}$

Jaccard index $\frac{TP}{TP+FP+FN}$

Average precision $\frac{\sum_{k=1}^{n} \text{P}(k)\text{rel}(k)}{TP+FN}$, where k is the rank in the matches list, n is the number of top entries in the matches list, $\text{P}(k)$ is the precision at point k (precision ignoring later entries) and $\text{rel}(k)$ is 1 if match at k is also a manual annotation and 0 otherwise

R-Precision $\frac{r}{TP+FN}$, where r is the number of TPs within the first $TP+FN$ entries

Discounted cumulative gain $\frac{DCG}{IDCG}$, $DCG = rel_1 + \sum_{i=2}^{n} \frac{rel_i}{\log_2(i)}$, where n is the number of entries in the matches list, i is the rank in the matches list, rel_i is 1 if match at ith position is also present in manual annotations and 0 otherwise and $IDCG$ is the maximum possible DCG for n

DCG (alternative) like previous DCG, but $DCG = \sum_{i=1}^{n} \frac{2^{rel_i}-1}{\log_2(i+1)}$ is used instead

These are calculated for each query individually and an average over all queries taken to get the final results per branch. An average of the final branch results is also given.

Precision shows how many of the results in the top list are correct (in the sense that they were also used for manually annotating). Recall shows how many of the correct results the automatic mapper was able to retrieve. In many cases, there are not many manual annotations available. So, for example, if we display the top 5 best matches and there is only 1 manual annotation done, which is present in the top 5, then the precision will be only 20%. But many of the remaining 4 FPs could probably also be considered correct, or at least good hints to the curator. However, we should still require that the one manual annotation is retrieved by the mapper. In the example, this is the case, so recall is 100%.

A better metric to use instead of precision might be the average precision, which takes into account the ranking order of results. So, for example, if the manual annotation was returned as second result by the automatic mapper, then the sum in average precision formula will consist of only the precision calculated at this position. At this rank 2, TP is 1, FP is 1 and thus precision is $\frac{1}{1+1} = 50\%$, thus average precision in this case will be 50%. If the manual annotation would have been returned forth, then average precision would have been $\frac{1}{1+3} = 25\%$. R-Precision is often highly correlated to average precision and DCG is also a measure taking into account the ranking of TPs.

Based on this, we should avoid looking too much at precision and other metrics that don't look at ranking order and contain FPs, and for simplicity, we choose to look at only one of the ranked metrics. So, in the next section, we will look only at recall and average precision when measuring the performance of the automatic mapper.

While doing this, we should still keep in mind, that the benchmark values are based on the assumption, that the manual curation is 100% correct, which is not the case.

4 Results

4.1 Parameter tuning

After having defined an automatic mapper in the previous section, we would like to start getting good results from it. However, at this point, all the parameters defined in the previous section have been given values based on educated guesses, which might not be optimal. We would like to optimise these parameters to give maximally good results on real datasets.

We could try to make an automatic optimiser for the parameters. However, we don't know if the parameters have been well defined and as first approximation would like to get a rough overview of what works and what not, so we will try to manually change parameters and observe changes in performance measures. We do this for each parameter by varying its value while keeping other parameters constant. Note, that this is not entirely correct, as the parameters are not independent.

The majority of tuning was done using the BIO.TOOLS input type, where other inputs would eventually be merged to. However, some experiments were done with other input types as well. In all case, matching of obsolete concepts was enabled, with 5 top results returned and mapping was done in all branches for which manual annotation data was available. Tuning was done, both, when all query parts were enabled, and also for individual query parts alone, to test, if there are differences in behaviour for longer texts (like full-text) and short keywords for example.

The measures looked at where recall and average precision. In case the effects of changing parameters were clearly visible, then these measures were correlated, so it didn't matter which one to take into account. If measure values were changing a little, when parameter values were changed, then there could be differences in the changes of recall and average precision. But it can be argued, that in such cases these fluctuations were too small to be significant. However, it could also be observed sometimes, that the measures peaked at slightly different parameter values. In such case parameter value somewhere between the two peaks could be chosen as final decision.

4.1.1 Approximate matching

First, we look at the effects of approximate matching. Parameter `mismatch multiplier` is set initially to 0.35 and `compound words` is 0.

Parameter `mismatch multiplier` could be set to a larger value than 1, as we may want to make approximate matching a bit more costly. For publication abstracts, for example, increasing the value up to 2, increased

measures by roughly 20%. At higher values, the measures started to drop off very slightly. The same could be observed for keywords, but with smaller increase (10%). Setting the parameter to a higher value means also that less approximate matches happen which means that execution is faster. So we set `mismatch multiplier` provisionally to 2.

As setting `mismatch multiplier` higher can mean that more matches will have a score lower than `match minimum`, then the effects we saw in the previous paragraph could actually be caused by `match minimum`. However, testing revealed, that varying `match minimum` has very little effect on the results. In some branches, enabling approximate matching seemed to make results very slightly worse and in some branches the opposite. But in general, we failed to see any clear pattern.

If anything, making `match minimum` smaller decreased results very slightly in case of long texts. Enabling approximate matching means, that some misspelled words will be correctly picked up, but also some similarly written, but not related words will be picked up. In case of long texts, the concept we are interested in will be mentioned in many places, thus it will not affect results much, if it is sometimes misspelled. On the other hand, the longer the text, the more possibilities we have for approximately match incorrect words.

For shorter input, like keywords, approximate matching might be more useful. In case of SEQWIKI TAGS, we get a slight increase of 2% in case approximate matching is enabled by setting `match minimum` to 0.35. Also, the speed penalty of approximate matching is not an issue if the input is so small. Still, the improvement was too small (only two extra TPs) to draw any conclusions, so more experimenting should be done.

As the effects of approximate matching on matching accuracy seem to be very small, but its effects on execution speed are considerable (see section 3.10.8), then in most cases it should be disabled by setting `match minimum` to 1. We enable it only for very short input in the hope that it sometimes finds a few extra correct matches.

For the `compound words` parameter, our conclusion is similar. Setting it to 1 from 0 only very slightly changes the result, but running time is considerably affected. However, differently from approximate matching, it should generally improve results, as it more likely will correct pre-processing errors, than cause incorrect matches. For example, setting it to 1 for SEQWIKI TAGS would correctly find a mapping from "Split-read" to "Split read mapping" that would otherwise be missing (because "Split-read" is pre-processed to "Splitread" and "Split read" to two words "Split" and "read"). So, we set `compound words` to 1 for short input and unless we have much processing power available, disable it for longer input. Also, currently `compound words`

is set globally for all query parts, we may want to be able to enable in some query parts (like keyword matching) and disable in other (like for full-text).

4.1.2 Proximity matching

There are four proximity matching parameters to optimise: `position loss` (initially 0.5), `position off by 1` (initially 0.7), `position off by 2` (initially 0.3) and `position match scaling` (initially 0.5).

Either disabling proximity matching (setting `position loss` to 0) or requiring all matching to happen in at least partially the right context (setting `position loss` to 1), had noticeable effects on the results. An illustration of a test can be seen on Figure 5. After enabling proximity matching, results get better while `position loss` is increased, until some point when they start gradually getting worse. Tests done for other input types also suggested that a good value is somewhere around the halfway between 0 and 1. As result of the tests, we set the default value of `position loss` to 0.4.

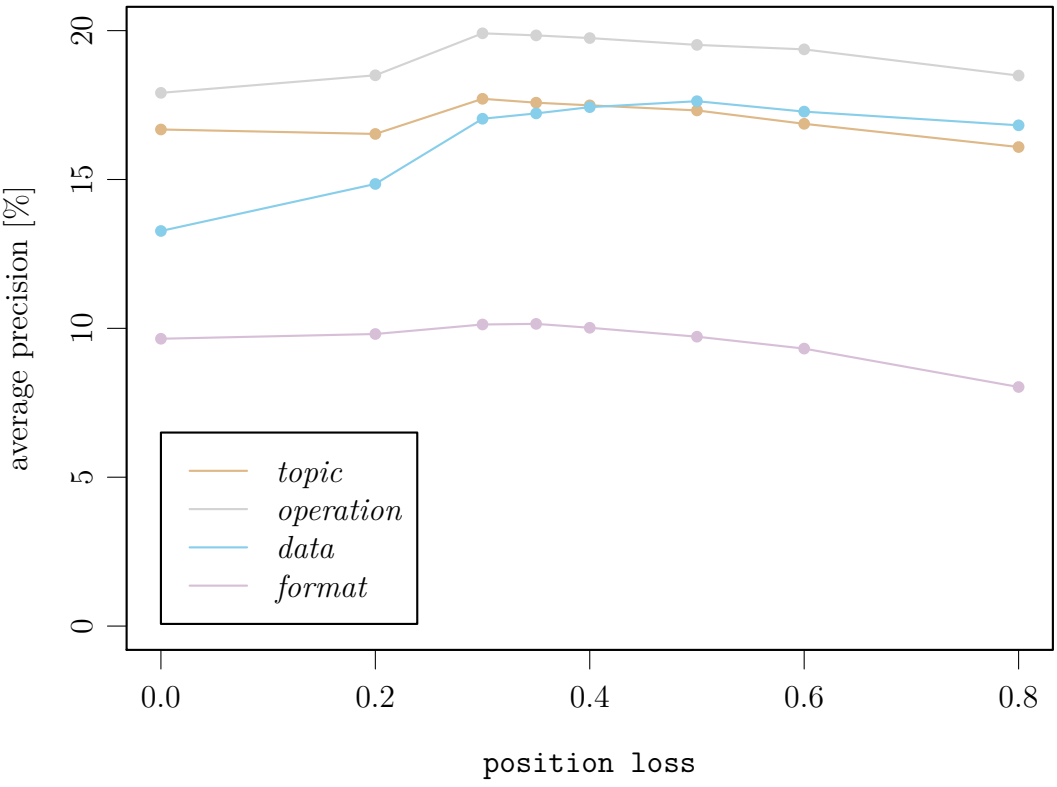


Figure 5: Varying `position loss` in BIO.TOOLS with all query parts enabled

Effects of varying `position off by 1` and `position off by 2`, which

describe proximity matching to further away words, were a lot smaller. In case of `position off by 1` tests suggested a value around 0.35, although this is to be taken with high uncertainty. In case of `position off by 2` changes in measures were hardly noticeable, but results seemed to suggest it should be minimal, so we set it to 0.05. This suggests a `position off by 3` is not necessary.

Parameter `position match scaling` needs approximate matching to be enabled to have any effect, so we had `match minimum` set to 0.35. Its effect on results were also minimal, but setting it to either 0 to 1 seemed to slightly worsen results in both case compared to values around 0.5, so we set it to 0.5 currently.

4.1.3 Inverse document frequency

Using inverse document frequency to weight words, i.e., penalising frequent words, can have considerable effects on the results.

First, we look into `query idf scaling`. So far, we have had bi-directional matching enabled, but we may be interested to see, if IDF has different effects when only matching from concepts to queries is enabled and when only matching from queries to concepts is enabled. A test for the first case, i.e., only matching from concepts to queries, can be seen on Figure 6. We see, that IDF can have big effects, at least in this case, where publication keywords are used. A good value for `query idf scaling` seems to be around 0.5, where average precision can be 2-3 times higher than in the case when IDF score are not used at all (`query idf scaling` is 0). Except in the *format* branch, where `query idf scaling` doesn't seem to have much effect.

For the opposite case, only matching from queries to concepts, this effect is less pronounced, as seen on Figure 7. Moreover, for the *data* branch, enabling `query idf scaling` has clearly negative effects.

When bi-directional matching is enabled and matching is enabled for all query parts, we get results as seen on 8. We see, that results in the *data* and *format* branches drop quite significantly when IDF weighting is enabled and increased. Getting results separately for both matching directions shows, that results go down again when matching is done from queries to concepts. So we may be tempted to disable `query idf scaling` in cases when matching is done in this direction. However, as seen for publication keywords, IDF scoring increases score in the *topic* and *operation* branches even for this direction. And when matching with all parts of the query, then results go down in the *data* and *format* branches for both directions. Also, when working with the MS-UTILS.ORG input, we found that enabling IDF scoring for query words improved results in the *format* branch.

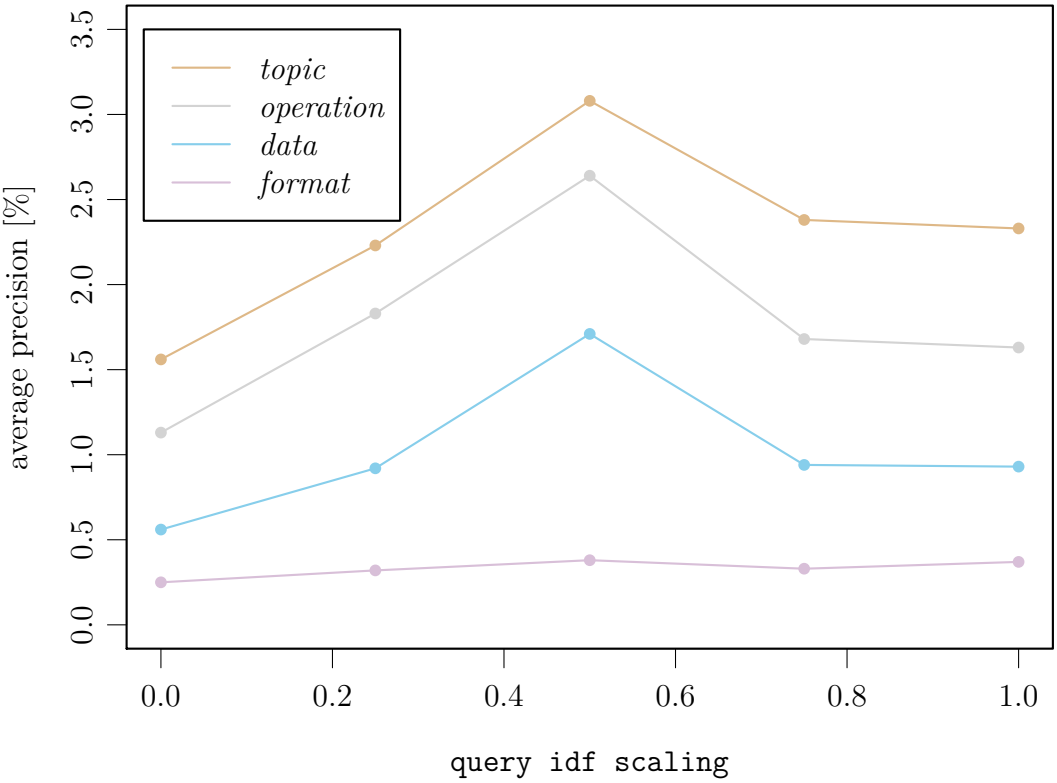


Figure 6: Varying `query idf scaling` in BIO.TOOLS for only publication keywords, when `concept weight` is set to 0. Low scores are caused by the fact, that only some entries in BIO.TOOLS have publications attached and of these, only some have keywords available in the publication.

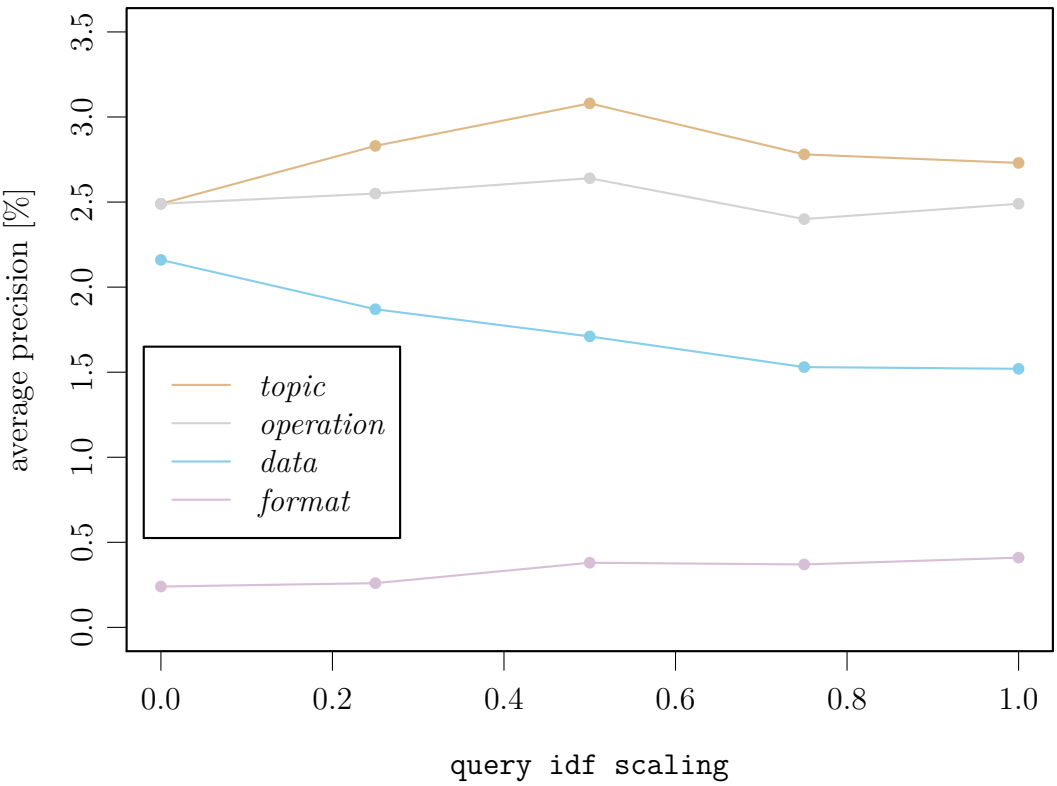


Figure 7: Varying `query idf scaling` in BIO.TOOLS for only publication keywords, when `query weight` is set to 0

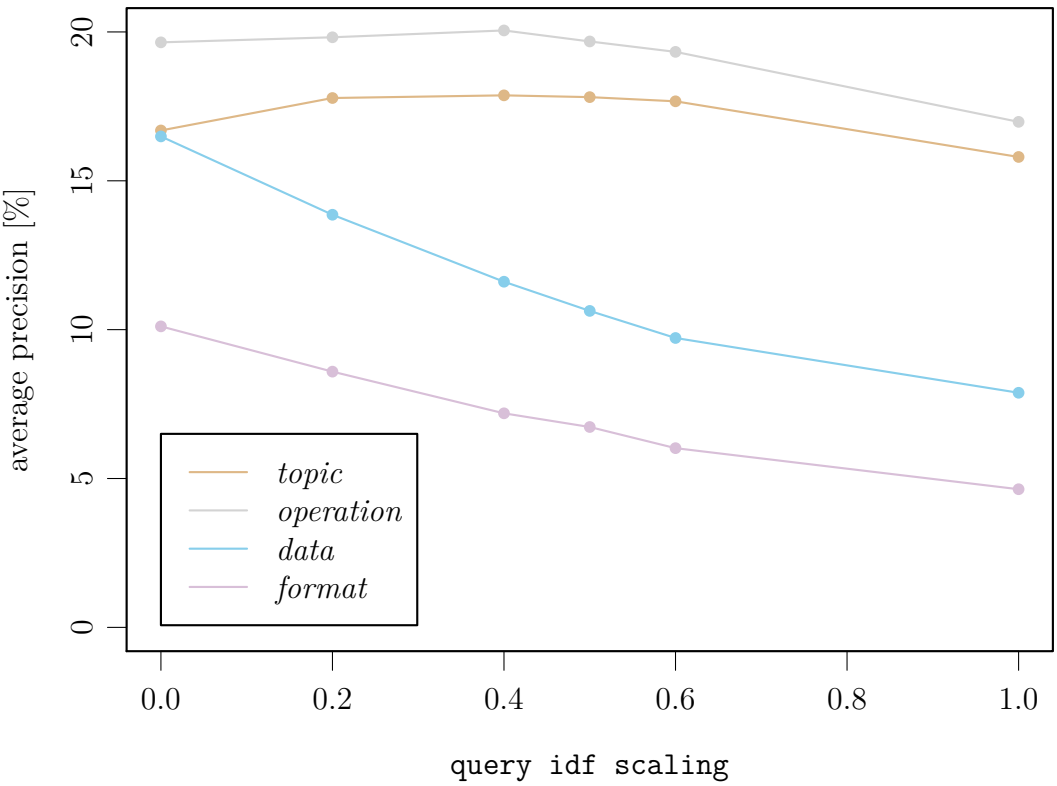


Figure 8: Varying `query idf scaling` in BIO.TOOLS, when all query parts are enabled

So, the effects of IDF seems to be a more complex issue and necessitates more thorough investigation, not only by looking at the overall metrics, but by going through individual results. For example, the reason why enabling IDF weighting has such negative effects in the *data* branch is probably the following. In the *data* branch, there are 1480 concepts (in EDAM version 1.14), but some of them are used much more often than others. For example, in `bio.tools`, a lot of tools are annotated with concepts "Sequence" and "Data". However, as seen from Table 3, these are words that occur very often in queries and thus have low scores (0.0414 and 0.0317 respectively). This means, that these two words must occur in one query more often, than some other word with a higher IDF weight, to get a similar match score. Thus, often some other words will be chosen over these and we get many false negatives containing "Sequence" and "Data". But the idea seems to work in principle, as IDF scoring helped for the *topic* and *operation* branches for example.

So we could measure performance, without looking at these two words, and maybe some others as well (as the curator might not need suggestions to "Sequence" and "Data" if these are very common). And more thoroughly investigate the relationship of IDF scoring to bi-directional matching and maybe some other parameters as well. But currently, as a simple workaround, we introduce the following parameter: `disable-query-idf-branches`. This enables us to disable query words IDF scoring selectively in some branches. So for BIO.TOOLS, we use it to disable query IDF scoring in the *data* and *format* branches. In *topic* and *operation* branches, IDF weighting is still done, with `query idf scaling` having the default chosen value of 0.5.

Also, we introduce the parameters `disable name keywords idf`, `disable description idf`, etc, to disable query IDF scaling for only some query parts. The rational, although Figures 6 and 7 seem to contradict this, is that IDF scoring should be mainly useful for longer texts.

We also tested the effects of IDF scores for the words in concepts by varying the `concept idf scaling` parameter. We found, that this parameter was best kept at 0, as increasing it decreased performance in all branches. However, when IDF scaling is disabled for words in concept labels and synonyms, then it has a slight positive effect, with best performance around 0.5. Thus we only allow IDF scaling to happen in concept definitions and comments and introduce the parameter `enable label synonyms idf` if we really wish to have IDF scoring for concept labels and synonyms.

In conclusion, IDF weights have a strong effect on performance as they influence greatly which words are preferred over which words. So, the match choices made when varying IDF weights should be studied more carefully.

4.1.4 Bi-directional matching

Next, we test if the idea of doing bi-directional matching was a valid one. Results can be seen on Figure 9. We see, that when doing only concept to query matching (`concept weight` is 0) or when doing only query to concept matching (`concept weight` is $infty$), then results are noticeably worse than for bi-directional matching. The exception is the *format* branch, where it seems that only queries to concept matching should be done. It's not clear, if it is a rule or an exception, like maybe a manifestation of the fact that parameters are not independent from each other. For example, when doing the same test for only publication abstracts, then bi-directional matching had higher performance than only queries to concept matching also in the *format* branch. So, currently we keep it enabled for all branches. As for the values of the weights, we set both `concept weight` and `query weight` to 1, in essence taking the mean of both direction matches.

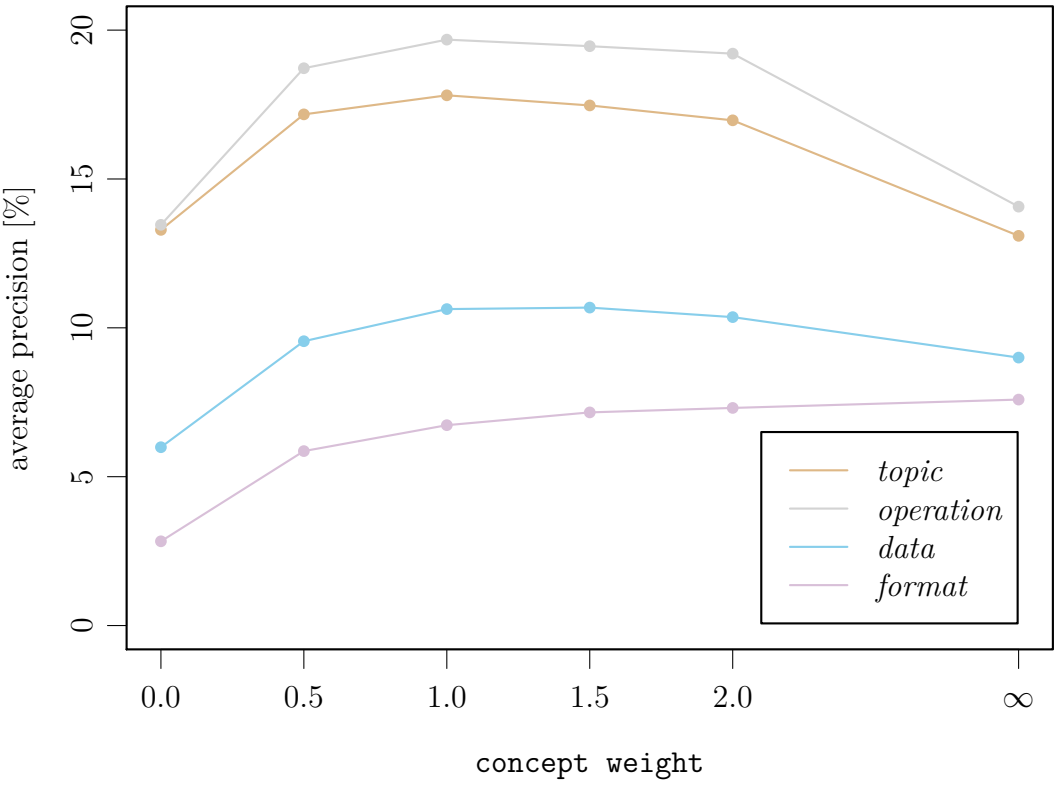


Figure 9: Varying `concept weight`, while `query weight` is 1, in BIO.TOOLS, when all query parts are enabled

4.1.5 Score scaling

As seen on Figure 10, `score scaling` seems to also be a valid concept, except for the *format* branch again. We set its default value to 0.2.

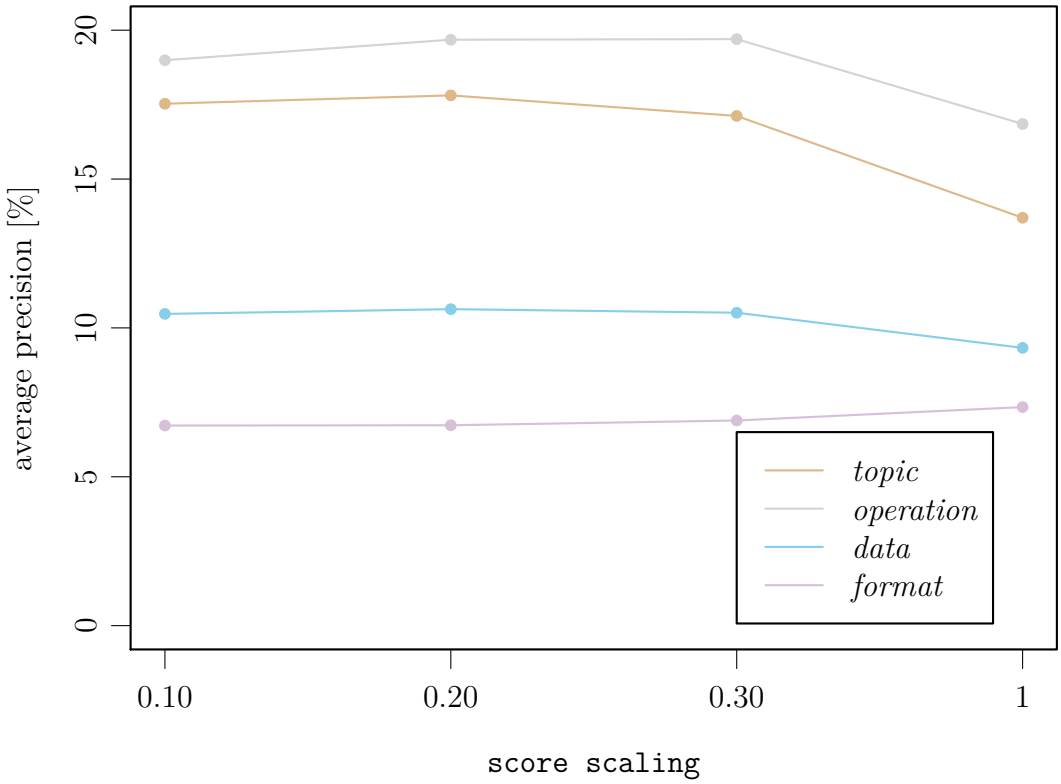


Figure 10: Varying `score scaling` in BIO.TOOLS, when all query parts are enabled

4.1.6 Multipliers, normalisers and weights

In this section, we will see, how the different multiplier, normaliser and weight parameters of concept and query parts should be set in order to best combine the scores of these parts.

First, we have a look at the multipliers of concept parts, that can be used to set the relative importance between concept label, synonyms, definition and comment. The concept's label is the most important part of a concept, thus we set its multiplier to 1. We expect that matching of exact synonyms should have a slightly lesser score, and that matching of narrow and broad synonyms a bit lesser still. However, testing revealed, that setting the exact synonyms's

multiplier to a smaller value than the label's multiplier was not justified, as performance started to drop. In case of narrow and broad synonyms, the same could be observed. Or to be more precise, the optimal value of the multiplier seemed to be somewhere around 0.97–0.98. However, the difference of performance to having the value at 1 was very small. So we set this multiplier also to 1.

As for `definition` and `comment multipliers`, their values had the best effect at around 0.5–0.75. But only when concept IDF weights were disabled. If concept IDF weights are enabled, then `definition` and `comment multipliers` should also be set to 1 for best performance. And in that case, performance was better than when concept IDF weighting was disabled and `definition` and `comment multipliers` were around 0.5–0.75.

So, in the end, we set all concept multipliers to 1. But the options to change these parameters will remain available, as we may still decide that the matches provided through matching narrow synoyms for example are not as good as matches provided through labels.

Next, we look at the publication part normalisers. Their goal is to make the scores of matching short query parts (like name) more comparable with scores of matching long query parts (like web page content).

Currently, setting these normalisers has to be done manually. First, we set `mapping strategy` to "best". Then, we set the normaliser value of a query part to 1, and the normaliser value of all other query parts to 0. From the mapping result, we find the usual score or score range of good matches, discarding the few possible outliers. We repeat this for every query part. Then, for the query part that had the lowest score, we set the normaliser to 1. For other parts we set it so, that multiplying their good match score range with their normaliser gets as result a score comparable to the score of the part that had the lowest scores.

For example, we set `publication fulltext normaliser` and `webpage normaliser` to 1, `publication abstract normaliser` to 0.985, `description normaliser` to 0.92 and `publication keyword normaliser` to 0.77. In general, the shorter the content of the query part, the more its score needs to be reduced.

Next, we look at changing the relative importance between query parts, by changing their weights. We set `name weight` and `description weight` to 1, as these are succinct and good sources for getting concept matches. Setting the weight of the description even higher was found not to be justified. The different publication keywords seemed to change the result a little, but it could be seen that the weight for the mined terms and MeSH terms should be lower than normal — these were set to 0.25. The user-assigned keywords are a better source, as they are more close to the ideas that the author

intended to express, we set `publication keyword weight` to 0.75. The abstract influenced the results more, a good weight value for it seemed to be around 0.75. The publication title is often highly correlated with the abstract and full-text and performance measures clearly showed its weight should be less than 1 – it was set to 0.25. It should be noted, that the different publication parts are often correlated, so we should try to make sure that their cumulative weight does not get too big when compared to the other query parts. As for the parts with long texts – publication full-text, web pages and documentation – a good value seemed to be between 0.5.

Some choices we made were somewhat deliberate, as sometimes performance measures did not change enough to decide on the best value (we could see that the value should be somewhere between 0 and 1, but it was hard to see where exactly). Also, for the documentation link we may get slightly better results if we disable matching against it entirely. However, it may find or reinforce some interesting matches, that most other parts are missing. For example, documentation (but also web pages) often contain the used data formats in the instructions on how to use the tool – this is more often missing in other query parts. By looking through the results, we may notice that some query parts pick up bad matches more often, so we could tune the weights further then.

When computing the weighted average of the query parts, we use the `average scaling` parameter. We mentioned in section 3.10.6, that its value should be set to at least $\frac{1}{\texttt{score scaling}}$ to cancel out the effects of score scaling. This can be seen on Figure 11. In that test we had `score scaling` set to 0.2, so `average scaling` should be set to at least 5. From the plot, we see that we should set it even higher – the effect of this will be, that the scores of mediocre matches will be diminished more than the scores of good match, so mediocre matches will become relatively less important. A good value for `score scaling` seemed to generally be around 10 up to 15.

Last, we compare the "best" and "average" strategies of combining the results of query parts. From Table 7, it can be seen that, except for the *format* branch, the use of the "average" strategy is well justified.

4.1.7 Pre-processing parameters

We haven't yet looked at changing the parameters defined in the following steps of preprocessing (section 3.7):

7 — set by `remove numbers`

12 — set by `stopwords`

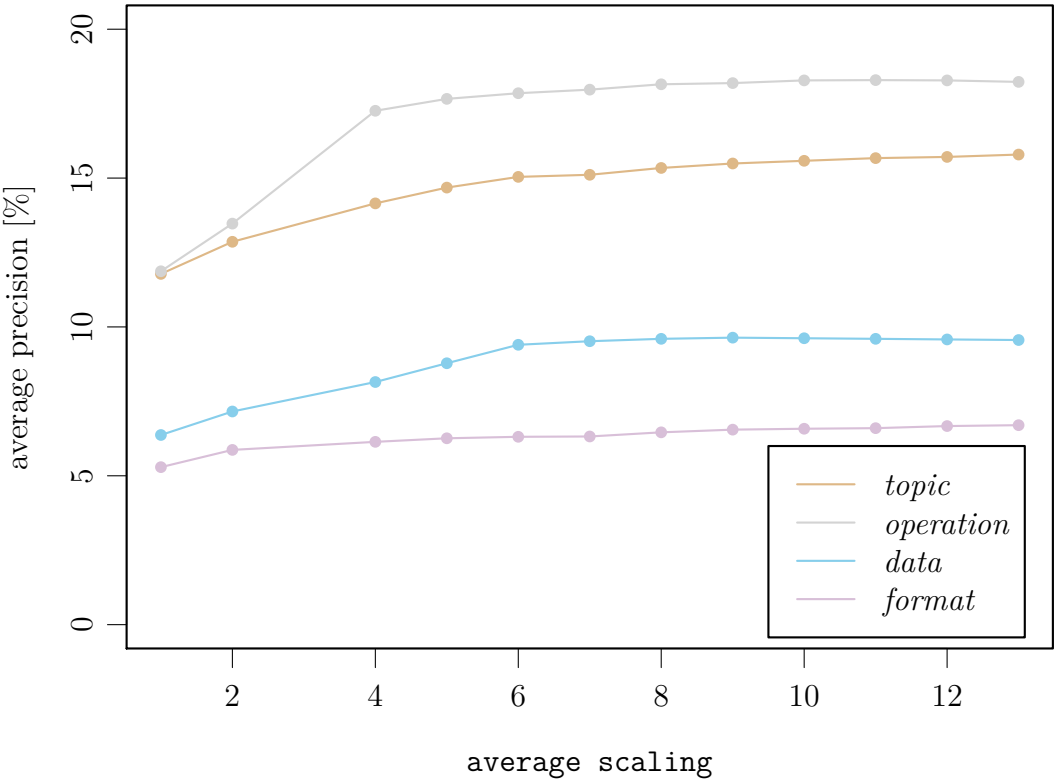


Figure 11: Varying `average scaling` in BIO.TOOLS, while `score scaling` is 0.2 and only the description query part is enabled

Table 7: Comparing "best" and "average" mapping strategies in bio.tools, when all query parts have been enabled and top 5 matches per branch are taken into account

(a) Metrics for the "best" strategy

	topic	*operation*	*data*	*format*	average
Recall	21.74%	28.29%	28.66%	19.62%	24.58%
Average Precision	13.69%	17.55%	14.54%	10.77%	14.14%

(b) Metrics for the "average" strategy

	topic	*operation*	*data*	*format*	average
Recall	26.25%	31.59%	31.11%	19.42%	27.09%
Average Precision	16.84%	20.45%	18.61%	9.68%	16.39%

13 — set by `no stemming`

14 — set by `short words`

Removing free-standing numbers had almost no effect. So we decide to keep them.

Same for removing short words. Setting `short words` to 1 (removing all words of length 1) had little effect, while setting it to 2 slightly decreased performance.

The more interesting parameters where those controlling stopwords removal and stemming. For short texts, stemming had a clear positive effect. But when tried on publication full-texts, it actually caused performance to drop (except in the *topic* branch, where it increased). While removing stop words had less impact on the results, it had a similar pattern: it worked better for short texts and decreased performance for longer text (except in *topic* branch). Also, the shorter the stop word list, the better – so we set as default the "lucene" list, which has only 33 words.

As stopwords removal and stemming had still an overall positive influence (except in the *format* branch), then these operations will be done. Also, removing stop words has a noticeable positive effect on the execution speed of the mapper.

But in general, the effect of these two pre-processing parameters should be studied more thoroughly. We may, for example, want to be able to disable them for longer texts, while keeping them enabled for shorter texts. Note, that the content of the query IDF file also depends on these parameters, so if we implement such feature, then multiple query IDF files might be needed at once.

4.1.8 Conclusions

In this section we got a rough overview how well different ideas of the mapper algorithm worked and tried to find optimal parameters controlling the behaviour of the algorithm. As result, we have set these parameters to reasonable defaults, which can be seen in Appendix B.

However, these should not be considered final. We should still look more deeply into optimising some of them, for example parameters concerning IDF or pre-processing. Also, some parameters, like multipliers and weights, can be changed also later according to the task or input type at hand or according to suggestions by curators.

We could also be interested in controlling the parameters individually for each branch and individually for each query or concept part, or at least based on if the part generally contains short of long text. However, while allowing

more control, this also means that the parameter space will increase many times.

Increasing the number of parameters can be a problem if we want to attempt any automated optimisation. However, a brute-force approach could be prohibitively expensive anyway. Using some genetic algorithm for parameter optimisation might be possible. To make it more feasible, we could also look at only optimising a few, more important parameters. Also, before such endeavour is attempted, we should also try to increase the quality of the manual annotation.

4.2 Results of automatic mapping

In this section, we provide an example result for each input type. While a curator would be better in judging the mapping quality, we can point to some more obvious shortcomings of the algorithm and some more obvious good matches.

In each case, the default parameters named in Appendix B were used with minor modifications. These modifications are brought out under each input type.

The ontology file used was EDAM version 1.14. Also, for each input type, the query IDF file used was the file based on entries of BIO.TOOLS, as it was found that it gives the best performance. For example, in case of using the query IDF file generated from MS-UTILS.ORG entries when mapping MS-UTILS.ORG entries, the performance was significantly worse than when using the BIO.TOOLS query IDF file. The reason might be, that the BIO.TOOLS file is based on larger input, but this has to be investigated more thoroughly.

Also, where applicable, a table with Recall and Average Precision metrics is given for each input type. While example outputs show top 3 matches per branch, the metrics values are based on a mapper run where top 5 matches were output.

4.2.1 SEQwiki

No manual annotation data was available for SEQWIKI, so no metrics were calculated.

Modification to parameters where the following:

- `name weight` to 0.5,
- `description weight` to 1,
- `keyword weight` to 1.5,

AB Large Indel Tool Identifies deviations in clone insert size that indicate intra-chromosomal structural variations compared to a reference genome. **Domain** Indel detection; Sequencing **Method** Mapping	Mapping	label	webpage	0.87 %
	DNA structural variation (Structural variation)	exact_synonym	description	0.54 %
	Clone library	label	webpage	0.28 %
	Indel detection	label	keyword Indel detection	1.12 %
	Mapping	label	keyword Mapping	0.58 %
	Structural variation discovery	label	description	0.29 %

Figure 12: Results for "AB Large Indel Tool"

AGE AGE is a tool that implements an algorithm for optimal alignment of sequences with SVs. **Domain** Structural variation **Method** Sequence alignment	Geriatric medicine (Aging)	broad_synonym	name	1.77 %
	DNA structural variation (Structural variation)	exact_synonym	keyword Structural variation	0.45 %
	Sequence analysis (Sequences)	exact_synonym	webpage	0.25 %
	Alignment	label	webpage	0.46 %
	Sequence alignment	label	webpage	0.41 %
	Local sequence alignment	label	webpage	0.37 %

Figure 13: Results for the "AGE" tool

- `webpage weight` to 1,

because the keywords (biological domains and bioinformatics method) are the most accurate part in the query and tool names seemed to cause many strange matches.

For "AB Large Indel Tool" (Figure 12), we can make the following observations:

- We can see a Domain/Method mixup, as the Domain "Indel detection" has matched the *operation* "Indel detection". This means, that "Indel detection" has to be moved under Method in SEQwiki.
- As "Sequencing" has a low IDF score, we have not matched to *topic* "Sequencing", but to other *topics* we consider more relevant.
- One such is "Clone library", the word "clone" appears once in the description and is mentioned multiple times in the web page, where also the phrase "clones from each library" appears (which gets a little boost from position matching).

For "AGE" (Figure 13), we can make the following observations:

- The top *topic* suggestion with a good score (matched through the stemmed version of the broad synonym "Aging") is clearly wrong. The tool name "AGE" does not relate to human age.
- We have matched the *operation* "Alignment", which is a parent of the matched "Sequence alignment", which is a parent of the matched "Local sequence alignment". Instead of receiving three parent-children terms, we may find more useful to get some unrelated suggestions instead of some of these terms.
- "Local sequence alignment" is matched, because "local end alignments" appears in the wiki page of the tool.

4.2.2 SEQwiki tags

As mapping of the short SEQwiki tags to concepts can be done by simple string comparison, then the original mapper by Rabie Saidi could be used here. Using it for concepts in the *topic* and *operation* branches and returning top 3 entries, we measured its performance and got as result Recall 66.6% and Average precision 57.7%.

Then, we used the new mapper with default parameters, with additionally setting `compound words` to 1 and `match minimum` to 0.35. Recall increased to 74.8% and Average precision to 68.9%.

This shows, that some of the features developed for matching free-texts have also been useful in the simplest case – direct keyword-to-keyword matching.

Most of the remaining concepts will be hard to map against using only string comparisons. For example, from "Viewer" to "Visualisation". The matching could be achieved however, if additional synonym information was available.

But there are still some more promising examples, for example from "Assembly" to "Sequence assembly". There are many concepts with their labels containing "assembly" and we have chosen as top 3 the following: "Assembly", "Assembly QC" and "EST assembly". It could be argued, that just the word "Assembly" is more related to "Sequence assembly" than "Assembly QC", however we can't say this based on the characters in "Sequence" and "QC" alone, but need additional input about the meaning or common usage of words in some context.

4.2.3 ms-utils.org

Table 8: Metrics for MS-UTILS.ORG

	topic	*operation*	*format*	*average*
Recall	58.97%	46.34%	51.83%	52.38%
Average Precision	36.49%	34.78%	40.27%	37.18%

Metrics for MS-UTILS.ORG (Table 8) were quite good, at least when compared to other input types.

Modifications to parameters were the following: matching of obsolete concepts was enabled and `query-idf-scaling` was set to 0.6 (as it increased metric by around 0.5% points).

For the "GenePattern" tool (Figure 14), we can observe the following:

- The top *topic* "Proteomics" is a true positive that occurs in the description and in every part of the publication.
- Conversely, we have the FN "Functional genomics", because no "functional" or "genomics" appear in any part of the query.

Figure 14: Results for "GenePattern"

GenePattern

platform for integrative genomics and proteomics (includes PEPPeR and other tools for proteomics)

Publication 16857664

Title: PEPPeR, a platform for experimental proteomic pattern recognition.

MeSH terms: Animals; Mice, Inbred C57BL; Mice; Peptides; Biological Markers; Calibration; Normal Distribution; Proteomics; Algorithms; Models, Theoretical; Pattern Recognition, Automated

Quantitative proteomics holds considerable promise for elucidation of basic biology and for clinical biomarker discovery. However, it has been difficult to fulfill this promise due to over-reliance on identification-based quantitative methods and problems associated with chromatographic separation reproducibility. Here we describe new algorithms termed "Landmark Matching" and "Peak Matching" that greatly reduce these problems. Landmark Matching performs time base-independent propagation of peptide identities onto accurate mass LC-MS features in a way that leverages historical data derived from disparate data acquisition strategies. Peak Matching builds upon Landmark Matching by recognizing identical molecular species across multiple LC-MS experiments in an identity-independent fashion by clustering. We have bundled these algorithms together with other algorithms, data acquisition strategies, and experimental designs to create a Platform for Experimental Proteomic Pattern Recognition (PEPPeR). These developments enable use of established statistical tools previously limited to microarray analysis for treatment of proteomics data. We demonstrate that the proposed platform can be calibrated across 2.5 orders of magnitude and can perform robust quantification of ratios in both simple and complex mixtures with good precision and error characteristics across multiple sample preparations. We also demonstrate de novo marker discovery based on statistical significance of unidentified accurate mass components that changed between two mixtures. These markers were subsequently identified by accurate mass-driven MS/MS acquisition and demonstrated to be contaminant proteins associated with known proteins whose concentrations were designed to change between the two mixtures. These results have provided a real world validation of the platform for marker discovery.

Full text present (61668 characters)

Proteomics			label	description	**0.87%**
	~~Proteome~~		label	description	**0.65%**
	Biomarkers		label	publication_abstract	**0.36%**
		Functional genomics			
		~~Mass spectrometry~~			
	Quantification		label	publication_abstract	0.28%
	Deisotoping		label	publication_fulltext	**0.27%**
	iTRAQ		label	publication_fulltext	**0.25%**
		Clustering			
	~~Experimental data (proteomics)~~		label	publication_title	0.17%
	~~Proteomics experiment report~~		label	publication_mesh Proteomics	**0.12%**
	Concentration		label	publication_abstract	0.11%
	match		label	publication_abstract	0.11%
GCT/Res format (Tab-delimited text files of GenePattern that contain a column for each sample, a row for each gene, and an expression value for each gene in each sample.)			definition	name	**0.06%**
	ppm		label	publication_fulltext	0.06%
		MAGE-TAB			
		Gene expression report format			

- The false positive *topic* "Biomarkers" can be a correct suggestion, as the keyword appears in the MeSH terms list. But unfortunately as "Biological Markers", so it will not be matched there. Fortunately, it also appears as "biomarker" in the abstract and full-text, thus we still pick it up.

- The *operations* "Deisotoping" and "iTRAQ" appear only once in the full-text and not in a context describing the tool itself, but they are probably chosen over "Clustering", that occurs multiple times and also occurs in the abstract, because "Clustering" is a quite frequent word. This is probably not the correct behaviour here.

- We see that the correct *format* "GCT/Res format" is picked, because it contains the tool name (GenePattern) it its description. So here we see an example, when matching query name and matching concept definition is useful.

- The third entry – "ppm" – does appear in the full-text multiple times. But what is meant there, is the parts-per-million unit, not the PPM image file.

4.2.4 BioConductor

Table 9: Metrics for BioConductor

	topic	*operation*	*average*
Recall	40.99%	31.47%	36.23%
Average Precision	25.82%	22.31%	24.07%

Modification to parameters where the following:

- include obsolete concepts,

- `webpage normaliser` to 0,

- `name normaliser` to 0.89,

- `description normaliser` to 0.95,

- `keyword normaliser` to 0.79,

- `doc normaliser` to 1,

Figure 15: Results for "canceR"

canceR : A Graphical User Interface for accessing and modeling the Cancer Genomics Data of MSKCC.

The package is user friendly interface based on the cgdsr and other modeling packages to explore, compare, and analyse
all available Cancer Data (Clinical data, Gene Mutation, Gene
Methylation, Gene Expression, Protein Phosphorylation, Copy
Number Alteration) hosted by the Computational Biology Center
at Memorial-Sloan-Kettering Cancer Center (MSKCC).

biocViews
GUI; Gene Expression; Software

Docs
http://bioconductor.org/packages/release/bioc/manuals/canceR/man/canceR.pdf
http://bioconductor.org/packages/release/bioc/vignettes/canceR/inst/doc/canceR.pdf

Gene expression (Transcription)			narrow_synonym	doc	1.60%
	Oncology (Cancer)		narrow_synonym	description	1.16%
	Epigenetics		label	doc	0.99%
		Nucleic acid structure analysis			
		Software engineering			
	Gene expression profiling		label	doc	0.87%
	Gene expression data analysis (Gene expression (microarray) data processing)		exact_synonym	doc	0.85%
~~Gene expression analysis~~			label	keyword Gene Expression	0.79%
		Protein secondary structure comparison			
	Gene expression data (Microarray data)		narrow_synonym	doc	0.90%
	Gene expression matrix (Normalised microarray data)		exact_synonym	doc	0.89%
	Gene expression profile		label	doc	0.87%
	Gene expression report format (Gene expression data format)		exact_synonym	doc	0.69%
	~~Gene expression data format~~		label	doc	0.69%
	Textual format (txt)		exact_synonym	doc	0.33%

- `name weight` to 0.5,
- `description weight` to 1,
- `keyword weight` to 1.5,
- `doc weight` to 0.75,
- `query idf scaling` to 0.4,

as among other things, we wanted emphasis more the biocViews keywords, which are a good source for the mapper.

For the "GenePattern" tool (Figure 14), we can observe the following:

- The top *topic*, which is a TP, is matched through both "Gene expression" (in description, biocViews and docs) and narrow synonym "Transcription" (in docs), strengthening the tie between query and concept.
- The second *topic* "Oncology (Cancer)" is probably a good suggestion, as it seems to describe what the package is about. In this case, matching the tool name – "canceR" – does the correct thing (compare to matching "AGE" above).
- The concept "Software engineering" could be third, but is probably not higher, because it contains quite common words (IDF score is relatively low) and it only occurs in biocView, and not as "Software engineering", but "Software".
- The rest of the matches seem to lack in variety as mostly different variations of "Gene expressions" are suggested.
- The *format* "txt" could be a good match, as it is mentioned several times in the documentation. Here we see a case, when documentation is quite useful, as mentioning this format is not important enough to include in the description or keywords, but we would still like to know, with what formats the tool works with.

The "Software engineering" keyword is an interesting case, as in BioConductor, it is almost always used to annotate the tool. However, very often we miss it. So one way to increase average precision in the *topic* branch, from around 26% (as seen in Table 9) to around 38%, or even to 42% if IDF weighting is disabled, is to only take the biocViews keywords as source. However, even if it would considerably raise performance according to metrics, we decide not to do it, as other useful concepts, like "Oncology (Cancer)",

would not be suggested then. For the curator, it's also easier to manually find concepts from the list of keywords than by going through documentation for example, from where the automatic mapper can quickly find and suggest potential matches.

4.2.5 bio.tools

Modification to parameters were the following:

- include obsolete concepts,
- `disable query idf branches` to "data, format",
- `good score data` to 0.77,
- `good score format` to 0.70,
- `bad score data` to 0.65,
- `bad score format` to 0.63,

where score limits had to be changed as scores have been altered by disabling IDF weighting in *data* and *format* branches (done because of reasons explained in section 4.1.3).

We have used an example for BIO.TOOLS output previously (Figure 4), so we will reuse it here. We can observe the following:

- The top *topic*, a FP picked through narrow synonym "Mutation", is picked because of stemming ("Mutations" in abstract, "Mutate" in web page, "mutated" in full-text).
- The top *operation*, the FP "Aggregation", is about aggregation of data items, which is not the same aggregation as mentioned multiple times in different query parts.
- The correctly found *format* "FASTA" is only present in longer texts (full-text and web page).
- Every time proteins are talked about in the query parts, we pick the "protein" *format* with high confidence. Quite often, this would not be a valid suggestion probably.

Metrics for BIO.TOOLS can be seen in Table 7. Scores may seem quite low, however, this is not necessarily caused only by the shortcoming of the automatic mapper or mistakes in the manual annotations. Namely, many

tools in BIO.TOOLS are badly described (have only a short description and no publication), and for many queries, for example all from a software suite called EMBOSS, have the same general publication attached, causing false positive results. So, improving the quality of the query could increase scores.

5 Discussion

In this word we tried to maximise the performance of the automatic mapper by looking at metrics that show how well its output compares to available manual annotations. However, the best metric is actually the tool's usefulness to the curator.

Manual accession of the results by experienced annotators is still ongoing. The first indications from people doing the curation show, that the tool can be used to find and make useful annotations. Most false positive terms make sense, which shows that manual curation might have missed many useful annotations, that we can now potentially add. Also, catching all false negatives should not be a goal in itself, as some of them could be less relevant concepts, or even mistakes, than the ones that are labelled as false positive. Another positive aspect about the mapper, is its flexibility – we can map both very short and long texts and map queries made up of many parts, each potentially providing valuable additional input to the mapper.

However, as seen in the example results, the automatic mapper does make many mistakes. In some cases, less relevant concepts are ranked above more relevant concepts. We could try to rectify this by tuning parameters or by disabling/enabling some features or matching of some query parts. But the suggestions can also be outright mistakes and these could be harder to correct. For example, from texts attached to the tools, we can pick up terms from sentences that are used to describe something not related to the tool itself. And, as we saw frequently, many mistakes are caused by the fact, that the mapper does simple character comparison and doesn't know anything about the meaning of the words.

To reduce the number of mistakes and make the tool more useful, we will continue to do incremental updates. Many of these were already discussed in this thesis, for example:

- take into account the hierarchy and relationships within the EDAM ontology,
- more intelligently extract content from web pages,
- differentiate between primary and other publications,
- investigate more the effects of using features such as IDF, stop words removal and stemming,
- also, experiment with different sources for calculating IDF scores,
- automatically find normalisation parameters or score limits,

- provide different output according to curators' wishes.

Moreover, the addition of following features or concepts could be useful:

On-line suggestions That is, integration of the mapping tool into the Registry and other portals to simplify upload of new material. The automatic mapper could act as a web server, answering to query requests with mapping responses.

Annotating training materials Another useful resource to annotate, for better discoverablity, is training materials of tools included in the Registry, which are usually in PPT or PDF format.

Concept search Concept searching means that ideas, not words, of the query are matched against ideas of the concepts. We could try to add some techniques used there, for example word-sense disambiguation, which means finding the correct meaning for words having multiple meanings. For this we could use the WordNet lexical database for English[1], which is around 50 MB in size while uncompressed, and has free libraries available to work with it.

Discovering new tools One potentially useful extension would be the ability to find new interesting software that is still not annotated in BIO.TOOLS. This could be achieved by trying to map a collection of publications and finding the ones that have the highest score or largest number of good matches to EDAM concepts. For example, over 1 million Open Access articles could be downloaded from PubMed Central[2] or Europe PMC[3]. The test case would be Nucleic Acids Research journal as each year hundreds of bioinformatics web applications and databases are published in special issues of Nucleic Acids Research journal.

Extracting new EDAM concepts As an opposite application, we could also discover new EDAM concepts. For this, we need a tool to extract keywords from the source full-texts. One such possible tool is Maui[4]. Then, keywords which are extracted with high enough confidence, but which we can't map to EDAM with a good enough score, can be suggested as new EDAM concepts.

[1] http://wordnet.princeton.edu/
[2] https://www.ncbi.nlm.nih.gov/pmc/tools/openftlist/
[3] http://europepmc.org/FtpSite
[4] https://github.com/zelandiya/maui

6 Conclusions

In the background section (2) we saw, that available tools not satisfy our needs. Thus, in the methods section (3), we set out to program and optimise a tool for automatically reading in free text, adding content from the Internet, and mapping it against EDAM ontology terms, giving as output the best matches. In results (4), we optimised parameters for the program and saw that it can suggest many correct terms. According to experienced curators, we have developed a useful tool to make their job easier. Development and improvement, in cooperation with curators, will continue.
Source code of the automatic mapper is available at:

- https://github.com/edamontology/edammap

with documentation at:

- https://github.com/edamontology/edammap/wiki

References

[1] J. Ison, M. Kalaš, I. Jonassen, D. Bolser, M. Uludag, H. McWilliam, J. Malone, R. Lopez, S. Pettifer and P. Rice, 'Edam: An ontology of bioinformatics operations, types of data and identifiers, topics and formats', *Bioinformatics*, vol. 29, no. 10, pp. 1325–1332, 15th May 2013, ISSN: 1367-4803, 1460-2059. DOI: 10.1093/bioinformatics/btt113.

[2] R. Saidi. Initial EDAM mapper, [Online]. Available: https://github.com/edamontology/edammap/commit/a685bd9bd8c1c95363d063cb26d84794526508d1 (visited on 02/02/2016).

[3] Samples, Phenotypes and Ontologies Team at the EBI. Zooma, [Online]. Available: http://www.ebi.ac.uk/spot/zooma/ (visited on 29/02/2016).

[4] J.-W. Li, K. Robison, M. Martin, A. Sjödin, B. Usadel, M. Young, E. C. Olivares and D. M. Bolser, 'The SEQanswers wiki: A wiki database of tools for high-throughput sequencing analysis', *Nucleic Acids Research*, vol. 40, pp. D1313–D1317, D1 2012, ISSN: 0305-1048, 1362-4962. DOI: 10.1093/nar/gkr1058. [Online]. Available: http://seqanswers.com/wiki/SEQanswers (visited on 12/05/2016).

[5] SEQwiki : Software list, [Online]. Available: http://seqanswers.com/wiki/Software/list (visited on 18/04/2016).

[6] SEQwiki : Mapping report, [Online]. Available: http://seqanswers.com/wiki/Ontology:EDAM#tab=Mapping_report (visited on 28/02/2016).

[7] M. Palmblad and V. Schwämmle. Ms-utils.org, [Online]. Available: http://www.ms-utils.org/wiki/pmwiki.php/Main/SoftwareList (visited on 12/05/2016).

[8] R. C. Gentleman, V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, K. Hornik, T. Hothorn, W. Huber, S. Iacus, R. Irizarry, F. Leisch, C. Li, M. Maechler, A. J. Rossini, G. Sawitzki, C. Smith, G. Smyth, L. Tierney, J. Y. Yang and J. Zhang, 'Bioconductor: Open software development for computational biology and bioinformatics', *Genome Biology*, vol. 5, R80, 2004, ISSN: 1474-760X. DOI: 10.1186/gb-2004-5-10-r80. [Online]. Available: http://bioconductor.org/ (visited on 12/05/2016).

[9] J. Ison, K. Rapacki, H. Ménager, M. Kalaš, E. Rydza, P. Chmura, C. Anthon, N. Beard, K. Berka, D. Bolser, T. Booth, A. Bretaudeau, J. Brezovsky, R. Casadio, G. Cesareni, F. Coppens, M. Cornell, G. Cuccuru, K. Davidsen, G. D. Vedova, T. Dogan, O. Doppelt-Azeroual, L. Emery, E. Gasteiger, T. Gatter, T. Goldberg, M. Grosjean, B. Grüning, M. Helmer-Citterich, H. Ienasescu, V. Ioannidis, M. C. Jespersen, R. Jimenez, N. Juty, P. Juvan, M. Koch, C. Laibe, J.-W. Li, L. Licata, F. Mareuil, I. Mičetić, R. M. Friborg, S. Moretti, C. Morris, S. Möller, A. Nenadic, H. Peterson, G. Profiti, P. Rice, P. Romano, P. Roncaglia, R. Saidi, A. Schafferhans, V. Schwämmle, C. Smith, M. M. Sperotto, H. Stockinger, R. S. Vařeková, S. C. E. Tosatto, V. de la Torre, P. Uva, A. Via, G. Yachdav, F. Zambelli, G. Vriend, B. Rost, H. Parkinson, P. Løngreen and S. Brunak, 'Tools and data services registry: A community effort to document bioinformatics resources', *Nucleic Acids Research*, vol. 44, pp. D38–D47, D1 4th Jan. 2016, ISSN: 0305-1048, 1362-4962. DOI: 10.1093/nar/gkv1116. [Online]. Available: https://bio.tools/ (visited on 12/05/2016).

[10] biotoolsXSD : Resource description model for bioinformatics, [Online]. Available: https://github.com/bio-tools/biotoolsXSD (visited on 28/04/2016).

[11] Bio.tools REST API : All the resources in the registry, [Online]. Available: https://bio.tools/api/tool (visited on 02/05/2016).

[12] Edam: Ontology of bioinformatics operations, types of data, formats, and topics, [Online]. Available: http://edamontology.org/page (visited on 19/05/2016).

[13] M. Horridge and S. Bechhofer, 'The OWL API: A Java API for OWL ontologies', *Semantic Web Journal*, vol. 2, no. 1, pp. 11–21, 2011, ISSN: 1570-0844.

[14] The DOI® Handbook, [Online]. Available: https://www.doi.org/hb.html (visited on 02/05/2016).

[15] Medical Subject Headings, [Online]. Available: https://www.ncbi.nlm.nih.gov/mesh/ (visited on 14/05/2016).

[16] J. Malone, E. Holloway, T. Adamusiak, M. Kapushesky, J. Zheng, N. Kolesnikov, A. Zhukova, A. Brazma and H. Parkinson, 'Modeling sample variables with an Experimental Factor Ontology', *Bioinformatics*, vol. 26, no. 8, pp. 1112–1118, 15th Apr. 2010, ISSN: 1367-4803, 1460-2059. DOI: 10.1093/bioinformatics/btq099.

[17] T. G. O. Consortium, 'Gene Ontology Consortium: Going forward', *Nucleic Acids Research*, vol. 43, pp. D1049–D1056, D1 28th Jan. 2015, ISSN: 0305-1048, 1362-4962. DOI: 10.1093/nar/gku1179.

[18] Pubmed, [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed (visited on 14/05/2016).

[19] PubMed Central, [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/ (visited on 14/05/2016).

[20] Entrez Programming Utilities Help, [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK25501/ (visited on 28/04/2016).

[21] T. E. P. Consortium, 'Europe PMC: A full-text literature database for the life sciences and platform for innovation', *Nucleic Acids Research*, vol. 43, pp. D1042–D1048, D1 28th Jan. 2015, ISSN: 0305-1048, 1362-4962. DOI: 10.1093/nar/gku1061.

[22] Jsoup : Class Selector, [Online]. Available: https://jsoup.org/apidocs/org/jsoup/select/Selector.html (visited on 05/05/2016).

[23] Yaml: YAML Ain't Markup Language, [Online]. Available: http://yaml.org/ (visited on 28/04/2016).

[24] The Dublin Core Metadata Initiative, [Online]. Available: http://dublincore.org/ (visited on 01/05/2016).

[25] Adobe XMP Developer Center, [Online]. Available: https://www.adobe.com/devnet/xmp.html (visited on 01/05/2016).

[26] M.F. Porter, 'An algorithm for suffix stripping', *Program*, vol. 14, no. 3, pp. 130–137, 1st Mar. 1980, ISSN: 0033-0337. DOI: 10.1108/eb046814.

[27] Karen Sparck Jones, 'A statistical interpretation of term specificity and its application in retrieval', *Journal of Documentation*, vol. 28, no. 1, pp. 11–21, 1972, ISSN: 0022-0418. DOI: 10.1108/eb026526.

[28] E. Ukkonen, 'Algorithms for approximate string matching', *Information and Control*, vol. 64, no. 1, pp. 100–118, 1985, ISSN: 0019-9958. DOI: 10.1016/S0019-9958(85)80046-2.

[29] H. Berghel and D. Roach, 'An extension of Ukkonen's enhanced dynamic programming ASM algorithm', *ACM Trans. Inf. Syst.*, vol. 14, no. 1, pp. 94–106, 1996, ISSN: 1046-8188. DOI: 10.1145/214174.214183.

A Used libraries

Library	version	URL
JCommander	1.48	http://jcommander.org/
OWL API [13]	5.0.1	http://owlcs.github.io/owlapi/
opencsv	3.7	http://opencsv.sourceforge.net/
jsoup	1.9.1	https://jsoup.org/
SnakeYAML	1.16	https://bitbucket.org/asomov/snakeyaml
Apache PDFBox	2.0.1	https://pdfbox.apache.org/
MapDB	3.0.0-M6	http://www.mapdb.org/
Porter stemmer [26]	Release 4	http://tartarus.org/martin/PorterStemmer/java.txt

B Program parameters

```
--average-scaling
   Scaling for the average strategy
   Default: 10.0
--bad-score-data
   Final scores under this are considered bad (in data branch)
   Default: 0.57
--bad-score-format
   Final scores under this are considered bad (in format branch)
   Default: 0.57
--bad-score-operation
   Final scores under this are considered bad (in operation branch)
   Default: 0.57
--bad-score-topic
   Final scores under this are considered bad (in topic branch)
   Default: 0.57
-k, --benchmark-report
   File to write HTML benchmark report to. It will contain metrics and
   comparisons to the manual mapping specified in the input query file.
   Default: <empty string>
-b, --branches
   Branches to include. Space separated from list [topic, operation, data,
   format].
   Default: [topic, operation]
--comment-multiplier
   Score multiplier for matching a concept comment. Set to 0 to disable
   matching of comments.
   Default: 1.0
--compound-words
   Try to match words that have accidentally been made compound (given
   number is maximum number of words in an accidental compound minus one)
   Default: 0
--concept-idf-scaling
   Set to 0 to disable concept IDF. Setting to 1 means linear IDF weighting.
   Default: 0.5
--concept-weight
   Weight of matching a concept (with a query). Set to 0 to disable matching
   of concepts.
   Default: 1.0
-d, --database
   Use the given database for getting and storing webpages, publications and
   docs
```

```
   Default: <empty string>
--definition-multiplier
   Score multiplier for matching a concept definition. Set to 0 to disable
   matching of definitions.
   Default: 1.0
--description-normaliser
   Score normaliser for matching a query description. Set to 0 to disable
   matching of descriptions.
   Default: 0.92
--description-weight
   Weight of query description in average strategy. Set to 0 to disable
   matching of descriptions in average strategy.
   Default: 1.0
--disable-abstract-idf
   Disable IDF weighting for publication abstract
   Default: false
--disable-description-idf
   Disable IDF weighting for query description
   Default: false
--disable-name-keywords-idf
   Disable IDF weighting for query name and keywords
   Default: false
--disable-query-idf-branches
   Branches to disable query IDF in. Space separated from list [topic,
   operation, data, format].
   Default: []
--disable-title-keywords-idf
   Disable IDF weighting for publication title and keywords
   Default: false
--doc-normaliser
   Score normaliser for matching a query doc. Set to 0 to disable matching
   of docs.
   Default: 1.0
--doc-weight
   Weight of query doc in average strategy. Set to 0 to disable matching of
   docs in average strategy.
   Default: 0.5
* -e, --edam
   Path of the EDAM ontology file
--enable-label-synonyms-idf
   Enable IDF weighting for concept label and synonyms
   Default: false
--exact-synonym-multiplier
```

```
   Score multiplier for matching a concept exact synonym. Set to 0 to
   disable matching of exact synonyms.
   Default: 1.0
--fetching-disabled
   Disable fetching of webpages, publications and docs
   Default: false
--good-score-data
   Final scores over this are considered good (in data branch)
   Default: 0.63
--good-score-format
   Final scores over this are considered good (in format branch)
   Default: 0.63
--good-score-operation
   Final scores over this are considered good (in operation branch)
   Default: 0.63
--good-score-topic
   Final scores over this are considered good (in topic branch)
   Default: 0.63
-h, --help
   Print this help
   Default: false
--keyword-normaliser
   Score normaliser for matching a query keyword. Set to 0 to disable
   matching of keywords.
   Default: 0.77
--keyword-weight
   Weight of query keyword in average strategy. Set to 0 to disable matching
   of keywords in average strategy.
   Default: 1.0
--label-multiplier
   Score multiplier for matching a concept label. Set to 0 to disable
   matching of labels.
   Default: 1.0
--mapping-strategy
   Choose the best or take the average of query parts matches
   Default: average
   Possible Values: [best, average]
-m, --match
   Number of best matches per branch to output
   Default: 3
--match-minimum
   Minimum score allowed for approximate match. Set to 1 to disable
   approximate matching.
```

```
   Default: 1.0
--mismatch-multiplier
   Multiplier for score decrease caused by mismatch
   Default: 2.0
--name-normaliser
   Score normaliser for matching a query name. Set to 0 to disable matching
   of names.
   Default: 0.81
--name-weight
   Weight of query name in average strategy. Set to 0 to disable matching of
   names in average strategy.
   Default: 1.0
--narrow-broad-synonym-multiplier
   Score multiplier for matching a concept narrow or broad synonym. Set to 0
   to disable matching of narrow and broad synonyms.
   Default: 1.0
--no-stemming
   Don't do stemming as part of pre-processing
   Default: false
--obsolete
   Include obsolete concepts
   Default: false
-o, --output
   File to write results to. If not specified or invalid, will be written to
   standard output.
   Default: <empty string>
--position-loss
   Maximum loss caused by wrong positions of matched words
   Default: 0.4
--position-match-scaling
   Set to 0 to not have match score of neighbor influence position score.
   Setting to 1 means linear influence.
   Default: 0.5
--position-off-by-1
   Multiplier of a position score component for the case when a word is
   inserted between matched words or matched words are switched
   Default: 0.35
--position-off-by-2
   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
   Default: 0.05
--publication-abstract-normaliser
```

Score normaliser for matching a publication abstract. Set to 0 to disable matching of abstracts.
Default: 0.985

--publication-abstract-weight
Weight of publication abstract in average strategy. Set to 0 to disable matching of abstracts in average strategy.
Default: 0.75

--publication-fulltext-normaliser
Score normaliser for matching a publication fulltext. Set to 0 to disable matching of fulltexts.
Default: 1.0

--publication-fulltext-weight
Weight of publication fulltext in average strategy. Set to 0 to disable matching of fulltexts in average strategy.
Default: 0.5

--publication-keyword-normaliser
Score normaliser for matching a publication keyword. Set to 0 to disable matching of keywords.
Default: 0.77

--publication-keyword-weight
Weight of publication keyword in average strategy. Set to 0 to disable matching of keywords in average strategy.
Default: 0.75

--publication-mesh-normaliser
Score normaliser for matching a publication MeSH term. Set to 0 to disable matching of MeSH terms.
Default: 0.75

--publication-mesh-weight
Weight of publication MeSH term in average strategy. Set to 0 to disable matching of MeSH terms in average strategy.
Default: 0.25

--publication-mined-term-normaliser
Score normaliser for matching a publication mined term (EFO, GO). Set to 0 to disable matching of mined terms.
Default: 1.0

--publication-mined-term-weight
Weight of publication mined term (EFO, GO) in average strategy. Set to 0 to disable matching of mined terms in average strategy.
Default: 0.25

--publication-title-normaliser
Score normaliser for matching a publication title. Set to 0 to disable matching of titles.
Default: 0.91

```
--publication-title-weight
   Weight of publication title in average strategy. Set to 0 to disable
   matching of titles in average strategy.
   Default: 0.25
* -q, --query
   Path of file containing queries
--query-idf
   Use the given query IDF file; if not specified, weighting of queries with
   IDF scores will be disabled
   Default: <empty string>
--query-idf-scaling
   Set to 0 to disable query IDF. Setting to 1 means linear IDF weighting.
   Default: 0.5
--query-weight
   Weight of matching a query (with a concept). Set to 0 to disable matching
   of queries.
   Default: 1.0
--remove-numbers
   Remove free-standing numbers (i.e., that not part of a word) as part of
   pre-processing
   Default: false
-r, --report
   File to write a HTML report to. In addition to the usual output, but with
   formatting in a browser.
   Default: <empty string>
--score-scaling
   Score is scaled before appyling multiplier and weighting with other
   direction match. Setting to 0 or 1 means no scaling.
   Default: 0.2
--short-word
   When all pre-processing steps are done, tokens with length less or equal
   to this length are removed
   Default: 0
-s, --stopwords
   Do stopwords removal as part of pre-processing, using the chosen
   stopwords list
   Default: lucene
   Possible Values: [off, corenlp, lucene, mallet, smart, snowball]
--threads
   How many threads to use for mapping (one query is processed by one
   thread)
   Default: 4
-t, --type
```

```
   Specifies the type of the query and how to output the results
   Default: generic
   Possible Values: [generic, SEQwiki, SEQwikiTags, SEQwikiTool, msutils,
   biotools, BioConductor]
--webpage-normaliser
   Score normaliser for matching a query webpage. Set to 0 to disable
   matching of webpages.
   Default: 1.0
--webpage-weight
   Weight of query webpage in average strategy. Set to 0 to disable matching
   of webpages in average strategy.
   Default: 0.5
```

Non-exclusive licence to reproduce thesis and make thesis public

I, Erik Jaaniso,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

1.1 reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2 make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

Automatic mapping of free texts to bioinformatics ontology terms,

supervised by Hedi Peterson.

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 19.05.2016