OCEAN CERTAIN
FP7-ENV-2013.6.1-1
Project number 603773

Deliverable 1.5
Source Code and Interface of LBKD software
WP's leader: GEOMAR

Delivery date:
December 20th, 2016

Authors:

Erwin Marsi
Dept. of Computer and Information Science (IDI), NTNU
emarsi@idi.ntnu.no

Pinar Øzturk
Dept. of Computer and Information Science (IDI), NTNU
pinar@idi.ntnu.no

Principal investigators:
Pinar Øzturk, **NTNU** (NO)
Erwin Marsi, **NTNU** (NO)

Non-permanent students and collaborators:
Biswanath Barik, **NTNU**

Project's coordinator:
Yngvar Olsen, **NTNU** (N)

1 (Popular) Description of deliverable

The goal of literature-based knowledge discovery (LBD) is to build software system to support researchers in discovering new knowledge from scientific literature. It is based on the premise that -- in addition to other methods for gaining knowledge such as experiments and modelling -- new knowledge can be inferred by combining existing knowledge written down in the literature. For example, if one publication states that event A causes event B, and another publication states that event B causes event C, those two pieces of information can be combined to suggest the plausible hypothesis that A causes C. However, such knowledge often remains undiscovered simply because individual researchers can only manage to read a relatively small part of the literature, typically mostly in the narrow field of their own expertise. Therefore we need software that can help researcher dealing with the ever growing scientific literature. This is even more so the case for "big problems" in science, such as climate change, which require a cross-disciplinary approach. LBKD goes beyond literature keyword-based literature search engines, aiming for a deeper understanding of the meaning of a text (semantic search), uncovering hitherto unknown associations and relations, and possibly even suggesting new hypotheses.

The work reported here is part of Task 2 in WP1: Development and application of an LBKD system, also referred to as "text mining of marine science literature". The objective of Task 2 is thus to build a LBKD system for marine science literature, in particular, journal articles which are relevant to the research questions addressed in the Ocean-Certain project. The original plan was that the system output would provide input to WP2. Whether this objective has been realized is a matter of discussion we will address in deliverable D1.4. The same goes for assessment of the system output. The current deliverable is described in the DOW as:

> "**D1.5) Source code and interface of LBKD-software**: The source code and resources developed for the LBKD system will be made publicly available."

The work carried out primarily corresponds to subtasks 1.2.4 and 1.24 in the DOW, repeated below:

> "**Subtask 1.2.4: Algorithms for literature-based discovery**
> Starting from the results of the preceding subtasks (i.e. relevant methods and resources for LBKD, annotated data and domain ontologies), the next objective is to develop algorithms for discovering relations and hypotheses that are implicit in the textual data. The first step is generation of candidates. The result is likely to be both large and noisy in several respects, e.g. containing trivial or irrelevant suggestions, duplicates, etc. The second step is therefore to rank and filter candidates, keeping only those most promising for further inspection by humans. These developments require substantial input from domain experts."

> "**Subtask 1.2.5 User interface and visualization**
> In order to be useful to the targeted audience, the algorithms for LBKD must be supplied with a graphical user interface. A major part of this concerns visualization of the results in the form of graphs, allowing users to explore concepts, relations and their textual sources through networks of links. A second major part concerns interactive behavior, where

> users can zoom in on the particular aspects of their interest through gradual refinement of search criteria."

It should be noted that according to the DOW the deliverable consists of *publicly released open source software*. It was not the intention to write extensive textual reports or full documentation, since this would require substantially more time than available in WP1. The software libraries can be used as-is by competent programmers and can serve as a good basis for future work on text mining in marine science and natural sciences in general. The interface is available as an online web application, which can be freely used by anyone interested in exploring marine science literature.

2 Summary of contribution of involved partners to deliverable

Design and implementation of all software was carried out by NTNU at IDI. Development of software specifications, early prototypes and final implementation was guided by advice and feedback from several project partners. From NTNU, Murat V. Ardelan served as our main advisor and has played a crucial role in our work. He has helped us (computer scientists) understanding the basics of marine science and the goals of Ocean-Certain. He has guided us in many aspects, including the selection of relevant journals and publications for text mining, examples and case studies for hypothesis discovery (e.g. the "iron hypothesis") and assessment of several forms of system output. Earlier in the project, we have also benefitted from input and feedback during meetings with other NTNU collaborators: Jennifer Bailey, Yngvar Olsen, Rachel Tiller and Olav Vadstein. In addition, several meetings with UiB collaborators Frede Thingstad and Tanya Tsagaraki about hypotheses in marine science were most valuable is shaping our ideas about literature-based knowledge discovery in marine science and scenarios for system evaluation. Finally, we received generous help from many other partners in the form of feedback on our presentations at general assemblies and online questionnaires.

3 Algorithms for literature-based knowledge discovery

3.1 Design

The overall goal of this component is to turn unstructured data (text) into structured data (graph database) through a process of information extraction. Information is extracted following the conceptual model defined earlier on in the project, which essentially consists of events of changing variables (increase/decrease/change) and the relations among these events (co-occurrence/causal/correlation). The extraction process is basically a pipeline consisting of a sequence of processing steps. The input is the raw text of journal articles (abstracts and/or full-text). This text results from information retrieval activities carried out under subtask 1.2.2, Collection and preprocessing of textual data. These involve crawling of the websites of selected journals, extraction of the article text from the HTML code, identification of text segments (title, authors, abstract, body text, references, etc.) and filtering of out-of-domain articles (i.e. non-marine science related).[1] Currently the title, abstract, and body text (which includes paragraph boundaries and section headings) are processed, whereas other parts are skipped (references, tables, figures, metadata). The main steps in the processing are[2]:

1. **Linguistic analysis**
   The first step is automatic linguistic analysis of the input text. This involves detection of sentence boundaries, lemmatisation (e.g. the verbs *increase, increases,increased* and increasing all have lemma *increase*), part-of-speech tagging (e.g. *CO2* is a proper noun, *changed* is a verb and *in* is a preposition) and syntactic parsing (e.g. *CO2* is the head of the noun phrase *an increase in CO2*). Linguistic analysis provides essential information that is required in subsequent processing such as variable extraction by pattern matching against syntactic parse trees.

2. **Variable and event extraction**
   Extraction of variables (e.g. *carbon dioxide*) and events (e.g. *decreasing carbon dioxide*) is performed simultaneously. We use pattern matching for this, more specifically tree pattern matching, where manually written patterns are matched against lemmatised syntax trees of sentences in order to identify events types (increase/decrease/change) and their variables.

3. **Filtering and generalisation of variables**
   Some of the extracted variables are filtered out, because they are meaningless without a wider context (e.g. referring expressions like *it* or *that*). Other variables are very long and complex and therefore unlikely to occur more than once. These variables are generalised (abstracted) by removing non-essential words or splitting them into atomic variables. For example, the variable *the annual , Milankovitch and continuum temperature* is split into three parts, one of which is *annual temperature*, which is ultimately itself generalised to *temperature*.

[1] Subtask 1.2.2 originally also included linguistic preprocessing of text, but this turned out to fit better under the information extraction activities of subtask 1.2.4, described here.

[2] This is a high-level, conceptual presentation of the pipeline. The actual implementation consists of more fine-grained steps and handles many more details.

this is accomplished through progressive pruning of a variable's syntactic tree, using a combination or tree pattern matching and tree operations, which are written by hand.

4. **Relation extraction** Once variables and events are identified, the next step is to extract the causal relations between events. Relation extraction again relies on tree pattern matching using hand-written patterns.

5. **Conversion to graph**
All extracted variables, events and relations are subsequently converted to a single huge graph, which is stored and indexed in graph database to facilitate fast search and retrieval. The graph is a property graph consisting of nodes and relations, both of which can carry properties. It has nodes for variables, generalised variables, event instances, event relations, sentences and articles. It has relations between e.g. event instance nodes and the nodes of the sentences they occur in. Properties hold information like the sentence number and text on sentence nodes, or character offsets for event instances.

6. **Graph post-processing**
The initial graph is further processed in a number of ways. Event instance nodes are aggregated in event type nodes. likewise, causal relation instances are aggregated in causal relation between event types. Furthermore, co-occurrence counts of event pairs occurring in the same sentence are computed and added as co-occurrence relations between event type nodes. Post-processing also includes addition of metadata and citation information to articles nodes in the graph.

The final output is a huge knowledge graph (millions of nodes) containing all information extracted from the input text. The graph can be searched in innumerable different ways, depending on interest, using a graph query language. The user interface described in section 4 offers a more user-friendly way of searching for certain patterns, i.e., relations between changing variables.

3.2 Implementation

The code base consists of a number of libraries. At the base is baleen-python[3], which is implemented in Python and forms the backbone of a generic system for text mining. It provides the pipeline architecture, command line interface, configuration facilities and all high-level processing steps. It depends on range of other Python libraries to carry out specialized tasks. In addition, it depends on other executable code running in different processes , where communication takes place through command line interface (subprocesses, e.g. CoreNLP) or over socket connections with local servers (e.g. Lucene and Neo4j) or remote web APIs (e.g. Crossref API). A second library, baleen-java, is implemented in Java, because it depends on a number of Java libraries for tree matching and transformation. It provides services to baleen-python; see further below for more details. A third Python library, called megamouth[4], provides the specific text mining system for extracting variables, events and relations from marine science articles. It depends on baleen-python for most of its tasks. Libraries have an API

[3] https://github.com/OC-NTNU/baleen-python
[4] https://github.com/OC-NTNU/megamouth

as well as command line interfaces for most tasks. Many library functions are documented in the source code. Command line interfaces include online help; see Appendix.

The megamouth system is basically a pipeline consisting of a sequence of processing steps. A processing step reads input from files and writes output to files. Each processing step can be run separately from the command line. Alternatively a sequence of processing steps can be started. Each processing step usually has a number of obligatory arguments (e.g. input and output files/directories, paths to executables, URLs, etc) as well as options (e.g. for caching/resuming or task specific flags). Their values are read from configuration files in .ini format. Configuration files have default sections providing default values, which can overridden in more specific sections or by other more specific configuration files. These values can in turn be overridden by arguments and options provided to command line scripts. This combination provides a convenient and flexible way to tweak pipelines.

Text of journal articles resulting from the information retrieval phase is stored and indexed in Apache Lucene[5] server, using different cores for different publishers. Articles are obtained by querying the server with a DOI, which returns a structured text in Json format.

Linguistic analysis relies on the Stanford CoreNLP tools, a Java library for natural language processing[6]. Annotators used are tokenization, sentence splitting, lemmatization, POS tagging and constituency parsing. CoreNLP is used through the command line, executed as a subprocess from the main Python script. CoreNLP's XML format is parsed to extract the required information such as character offsets of words and sentences, lemmas and parse trees in labeled-brackets format. Some post-processing is performed, e.g., words in parse trees are replaced by their lemma.

Extraction of variables and causal relations is based on tree pattern matching. We implemented a Java library called baleen-java[7] for information extraction through tree pattern matching. It depends on two existing libraries, Tregex and Tsurgeon[8]. Tregex is a Java library for matching patterns in trees based on tree relationships and regular expression matches on nodes. Tsurgeon is a closely related library transforming trees through sequences of tree operations. Both tools employ a declarative language for defining tree patterns and tree operations in a concise way. The baleen-java library takes as input files with trees in labeled-bracket format as well a files with definitions of patterns/operations for extraction. It outputs extractions in Json format, including information such as the name of matched pattern, tree and node numbers and unique identifiers.

The tree patterns and tree operations for extraction are partly manually written and partly automatically generated.

[5] https://lucene.apache.org/
[6] http://stanfordnlp.github.io/CoreNLP/
[7] https://github.com/OC-NTNU/baleen-java
[8] http://nlp.stanford.edu/software/tregex.shtml

Generalisation of variables by progressive pruning of their syntax trees is also implemented in baleen-java, relying on Tsurgeon to do the heavy lifting. The patterns and operations for e.g. removing premodifiers or picking apart coordinated structures are again manually written.

A property graph is used for storing and indexing the extracted information, which consists of variables, events and relations, together with meta-info regarding the source sentences and articles. For this the Community Edition of the Neo4j graph database engine[9] is used. Creation and management of graph databases from within Python relies on the neokit abstraction layer from the py2neo[10] toolkit. First, extracted information in Json format in converted to CSV format, after which neo4j-import, the command line tool for fast massive imports, is used to create an initial graph with nodes for variables, event instances, causal relation instances, sentences and articles, as well as their properties and relations. Next, the Cypher[11] query language is used to postprocess the graph in a number of ways: (1) aggregation of event instances in event types nodes; (2) computation of variable co-occurrence relations; (3) aggregation of causal relations in causal relation between event types; (4) imposing constraints and indexes to speed up search. Use of Cypher from within Python code depends on the Neo4j Bolt driver for Python[12].

Finally, metadata for articles (authors, title, year, journal, volume, pages, url) and formatted citations are obtained through the Crossref metadata API[13] via lookup of the article's DOI.

The latest stable version of own source code is maintained in a version control system (Git) and is freely available from the Github website. All software dependencies have permissive open source licenses and no paid software is required.

3.3 Discussion

The LBKD system turns unstructured data (text) into structured data (graph database) through a process of information extraction. It takes text as input, performs information extraction and outputs a knowledge graph. The system is flexible and configurable in many ways. Text input currently consist of journal articles, but it can process other text sources, e.g., text books or wikipedia pages. Currently information is extracted following the conceptual model defined earlier on in the project. However, the system can accommodate extraction of other -- or more specific -- types of entities, events and relations, which would require writing tree patterns matching them. The resulting graph can be searched efficiently using Cypher queries, which opens up many interesting opportunities for knowledge discovery. The graph can also be modified and extended, e.g., with knowledge regarding synonyms or ISA relations (hypernyms).

The software libraries can be used as-is by competent programmers and can serve as a good basis for future work on text mining in marine science and natural sciences in general. Still there are many ways in which the software can be improved. One aspects that is still under active

[9] https://neo4j.com/
[10] http://py2neo.org/
[11] https://neo4j.com/developer/cypher/
[12] https://github.com/neo4j/neo4j-python-driver
[13] https://github.com/CrossRef/rest-api-doc

development is the integration of a new and better algorithm for extraction of causal relations which is based on machine learning from manually annotated data.

4 User interface and visualisation

4.1 Design

All structured data that is extracted from journal articles is stored in a huge graph database. This database can be searched for information of interest by writing search patterns in the Cypher query language, akin to SQL for relational databases. For example, the following Cypher query (automatically generated by the "model" component of the user interface) initiates a search for a relations between an increase of the variable CO2 and events with variable temperature, including any more specific instances of these variables:

```
MATCH
        (v1:VariableType) <-[:TENTAILS_VAR*0..3]- (vs1:VariableType)
        <-[:HAS_VAR]- (e1:EventType),
        (v2:VariableType) <-[:TENTAILS_VAR*0..3]- (vs2:VariableType)
        <-[:HAS_VAR]- (e2:EventType),
        (e1) -[r:CAUSES]-> (e2)
WHERE
        ( v1.subStr = "co2" AND e1.direction = "increase" ) AND
        v2.subStr = "temperature" AND
        r.n > 0
WITH DISTINCT
        vs1 as varType1,
        e1 as eventType1,
        vs2 as varType2,
        e2 as eventType2,
        r
WITH
        varType1.subStr AS variable1,
        eventType1.direction as event1,
        eventType1.n AS eventCount1,
        id(eventType1) AS nodeId1,
        varType2.subStr AS variable2,
        eventType2.direction as event2,
        eventType2.n AS eventCount2,
        id(eventType2) AS nodeId2,
        type(r) AS relation,
        r.n AS relationCount,
        id(r) AS relationId
RETURN
        relationCount,
        relation,
        event1,
        variable1,
        event2,
        variable2,
        nodeId1,
        nodeId2,
        relationId,
        eventCount1,
        eventCount2
        ORDER BY relationCount DESC
```

```
                LIMIT 500;
```

The response from the database engine is in the form of a table:

```
+----------------------------------------------------------------------------------------------------------------------------------------------------------+
| relationCount | relation | event1     | variable1 | event2     | variable2                     | nodeId1 | nodeId2 | relationId | eventCount1 | eventCount2 |
+----------------------------------------------------------------------------------------------------------------------------------------------------------+
| 1             | "CAUSES" | "increase" | "co2"     | "increase" | "July temperature"            | 991646  | 1010567 | 1573990    | 96          | 1           |
| 1             | "CAUSES" | "increase" | "co2"     | "change"   | "equilibrium temperature"     | 991646  | 972182  | 1574386    | 96          | 1           |
| 1             | "CAUSES" | "increase" | "co2"     | "increase" | "surface air temperature"     | 991646  | 1001596 | 1573434    | 96          | 9           |
+----------------------------------------------------------------------------------------------------------------------------------------------------------+
```

Although such queries can be written by hand, it takes time, effort and a considerable amount of expertise. In addition, the output format leaves a lot to be desired. Large tables are difficult to read and navigate. There is no easy way to browse the results, e.g. looking up the source sentences and articles for the extracted events. Moreover, its assumes that users have a local installation of all required software and data, e.g. the Neo4j database engine and the graph database.

The user interface is intended to solve these problems. Its main task is to enable non-expert users (marine scientists) to easily search the graph database in an interactive way and to present search results in a browsable and visual way. It is a web application that runs inside the browser and communicates with a remote web server over the Internet, which has the advantage that users do not need a local installation of software and data. Moreover, it runs on any modern platform with a browser (Linux, Mac OS, Windows, Android, iOS, etc). It is a graphical user interface, which relies on familiar input components such as buttons and selection lists to compose queries. It uses interactive tables and graphs to present search results. Hyperlinks are used for navigation and to connect related information, e.g. a search result and the webpage of the source journal article.

4.2 Interface

The Marine Variable Linker[14] (MVL) helps to find related events in marine science literature. The current demo system searches in article abstracts from a large number of journals. It can find three kinds of events:

- variables that are **increasing**
- variables that are **decreasing**
- variables that are **changing** into an unspecified direction

For example, the following sentence contains two events:

> The increase of greenhouse gasses causes a decline of Arctic sea ice.

The first event is an *increase* of the variable *greenhouse gasses* (marked in red). The second event is a *decrease* of the variable *Arctic sea ice* (marked in blue). Technically each event is a combination of a semantic **predicate** (increase, decrease or change) and a particular variable.

[14] http://baleen.idi.ntnu.no/demos/megamouth-abs

Events can be related to each other. In the example above, the two events occur together in the same sentence, so there exists a **co-occurrence relation** between them. If two events tend to co-occur frequently in text, this usually indicate that there exists some meaningful connection between them. In the above example, there is in fact a **causal relation** between the two events, as indicated by the use of the verb *causes*.

The current version of the Marine Variables Linker allows searching for both co-occurrence and causal relations between events.

4.2.1 Searching for events

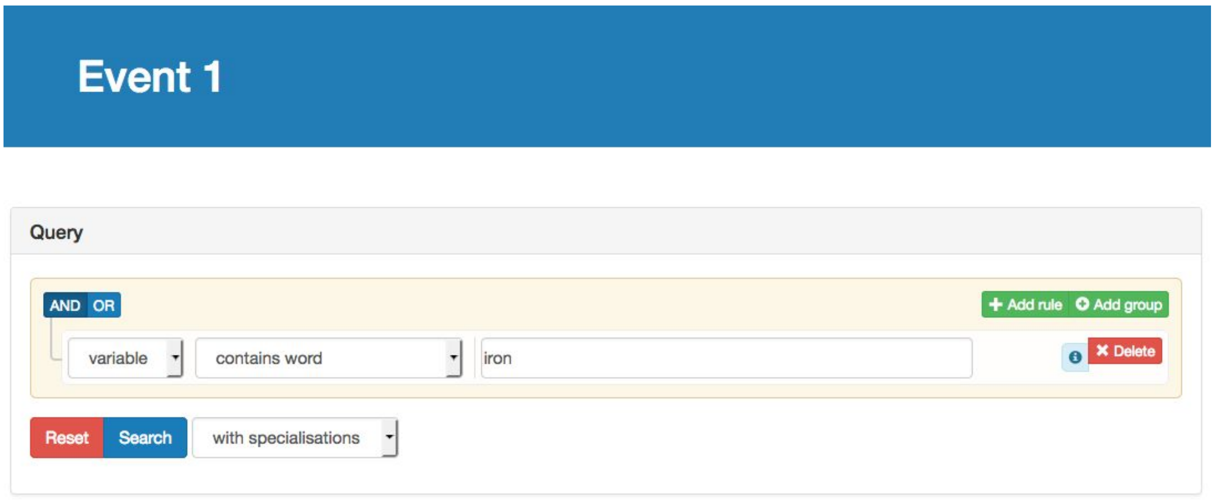
Clicking on Event1 in the menu bar at the top brings up the section for Event 1.

4.2.1.1 Query

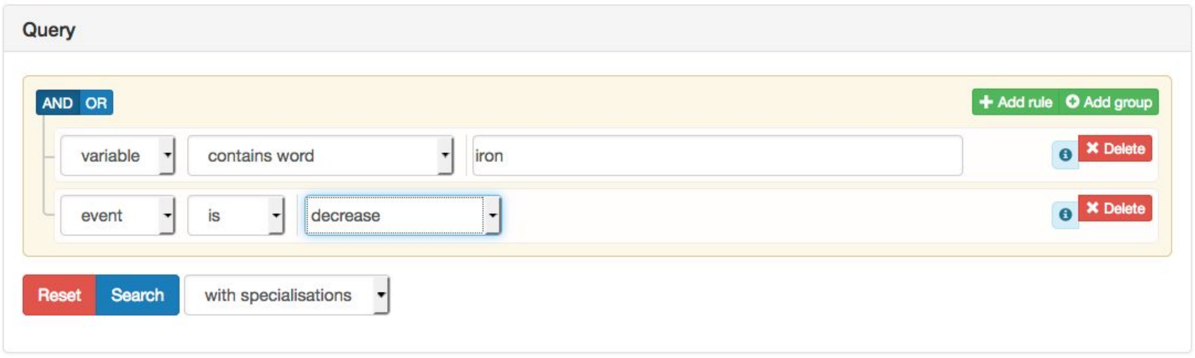
In the panel titled *Query* one can compose a query to search for events. Each query consists of one or more *rules*. A rule has a combination of drop-down lists and text boxes for its three parts:

1. subject
2. assertion
3. value

For example, one can select *variable* as the subject, *contains word* as the assertion and *iron* as the value, to search for all events involving a variable containing the word *iron*.



Clicking on the green *Add rule* button will add more rules in order to narrow down the search. For example, to further restrict events to only those with predicate *decrease*.


Query

AND OR

+ Add rule | Add group

variable | contains word | iron | Delete

event | is | decrease | Delete

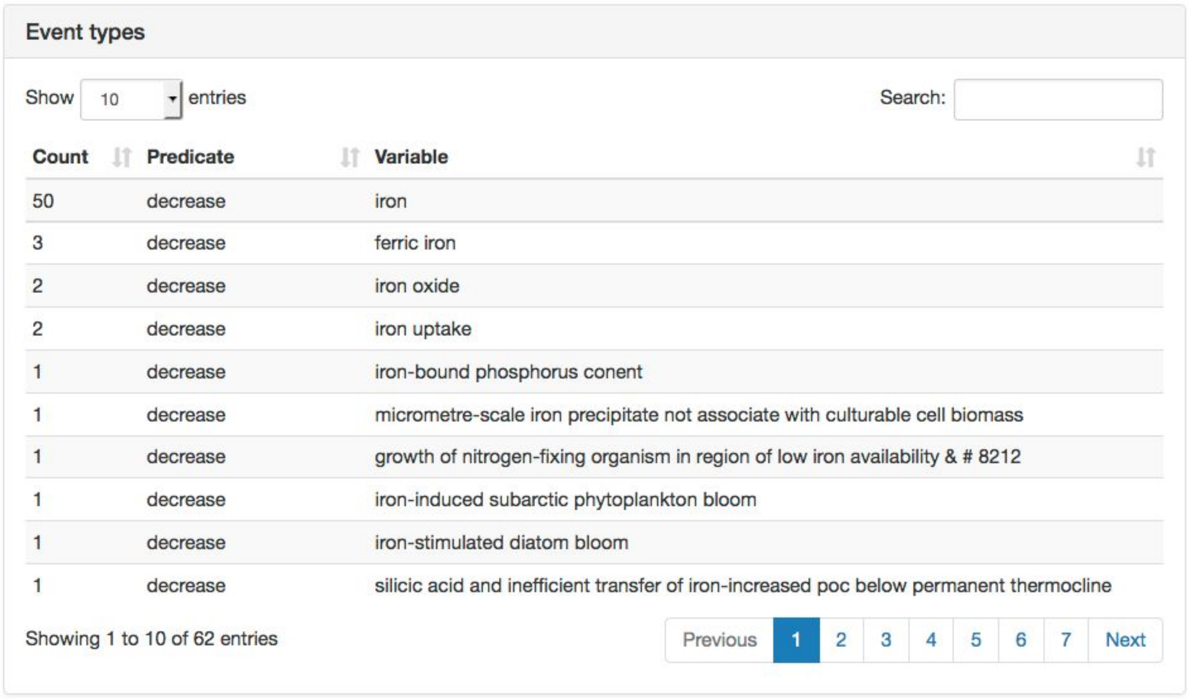
Reset | Search | with specialisations

The red *Delete* button can be used to remove rules from a query.

By default, rules are combined by *conjunction* (that is, rule1 **and** rule 2 must match). To combine rules by *disjunction* (that is, rule1 **or** rule 2 must match), click on the blue OR button in the upper left corner.

4.2.1.1 Event types

Once the query is finished, clicking on the *Search* button will bring up a new panel titled *Event types* that shows all types of event matching the query. It contains a table showing the instance counts, predicates and variables. One can browse through multiple pages (if the table is long enough) or sort rows on a particular column. Note that there is maximum to the number of event types shown (currently 500).


Event types

Show 10 entries   Search:

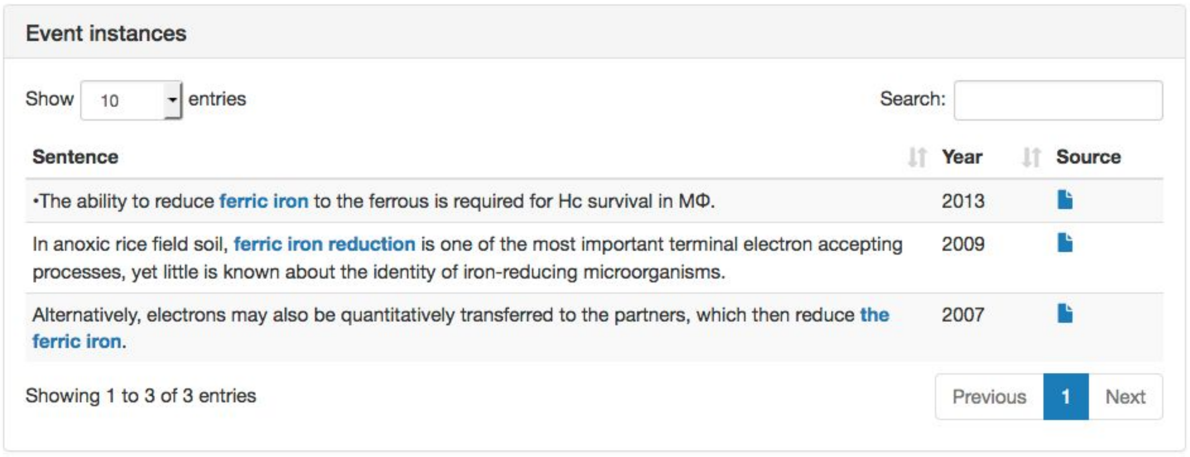
Count	Predicate	Variable
50	decrease	iron
3	decrease	ferric iron
2	decrease	iron oxide
2	decrease	iron uptake
1	decrease	iron-bound phosphorus conent
1	decrease	micrometre-scale iron precipitate not associate with culturable cell biomass
1	decrease	growth of nitrogen-fixing organism in region of low iron availability & # 8212
1	decrease	iron-induced subarctic phytoplankton bloom
1	decrease	iron-stimulated diatom bloom
1	decrease	silicic acid and inefficient transfer of iron-increased poc below permanent thermocline

Showing 1 to 10 of 62 entries

Previous 1 2 3 4 5 6 7 Next

4.2.1.2 Event instances

Clicking on a row in the *Event types* table will bring up the *Event instances* panel that shows all the actual mentions of this event in journal articles. Each row in the table shows a sentence, year of publication and source. The event is marked in color; red for increasing, blue for decreasing and green for changing. Hovering the mouse over the icon will show a citation for the source article. Clicking on this icon will open the article containing the sentence in a new window. Note that in the case of a full text article, your need to have access rights to view it.


Event instances

Show 10 entries Search:

Sentence	Year	Source
•The ability to reduce **ferric iron** to the ferrous is required for Hc survival in MΦ.	2013	
In anoxic rice field soil, **ferric iron reduction** is one of the most important terminal electron accepting processes, yet little is known about the identity of iron-reducing microorganisms.	2009	
Alternatively, electrons may also be quantitatively transferred to the partners, which then reduce **the ferric iron**.	2007	

Showing 1 to 3 of 3 entries

Clicking on the *Reset* button to start a new search.

4.2.1.3 Searching with specialisations and generalisations

The drop down next to the *Search* button offers three choices:

- *exact only*: This means that only events exactly matching the search criteria will be returned. This is the default setting.
- *with specialisations*: This means that also more specific types of events will be returned. For example, if the variable is *sea ice*, then the search result includes more specific types such as *winter sea ice*, *Artic sea ice* and *frost, snow and baltic sea ice.*
- *with generalisations*: This means that also more general types of events will be returned. For example, if the variable is *summer Arctic sea ice*, then the search result includes more general types such as *Arctic sea ice*, *sea ice* and ice.

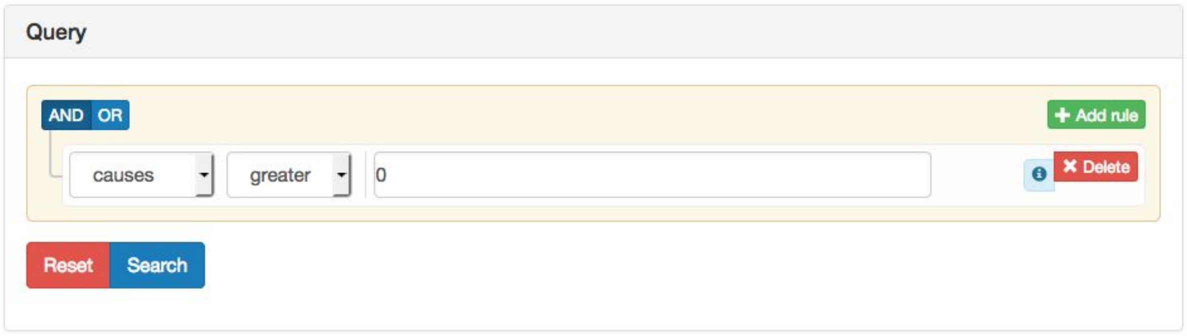
4.2.2 Searching for relations

In order to search for relations between events, one first needs to define queries for Event 1 and Event 2. Then clicking on *Relation* in the menu bar at the top brings up the section for relations.

4.2.2.1 Query

In the panel titled *Query* one can compose a query to search for relations. This works in a similar way as for events. A query typically consist of a single rule, although it is possible to use multiple rules where each one specifies conditions on the relation.

Relation

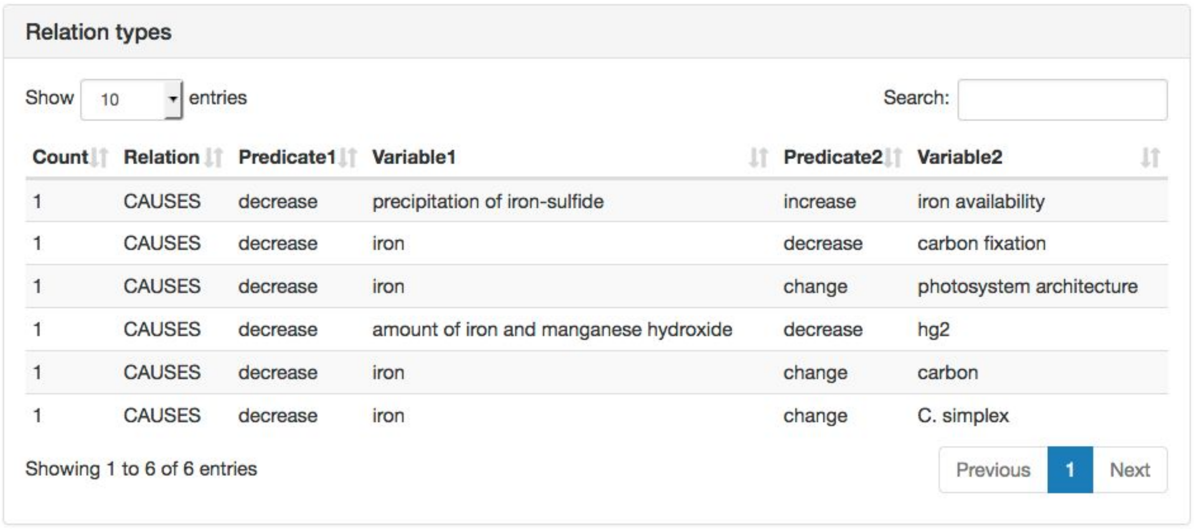


The first kind of relation is *cooccurs*. This means that the two events co-occur in the same sentence. When two events often co-occur in a single sentence, they tend to be associated in some way. This may indicate a correlation between the two types of events. However, it may also indicate a meaningless coincidence or -- especially in case of low frequency -- the co-occurrence may be purely by chance.

The second kind of relation is *causes*. This means that the two events in a sentence are causally related, where one event is the cause and other the effect. That is, causal relations are directed. Causality must be described in the sentence, for example, by words like *causes*, *therefore* or *leads to*, etc.

4.2.2.2 Relation types

Clicking on the *Search* button will bring up two new panels. The first panel is titled *Relation types* and contains a table with all pairs of event types in the given relation(s), specifying counts, relations, event predicates and event variables.

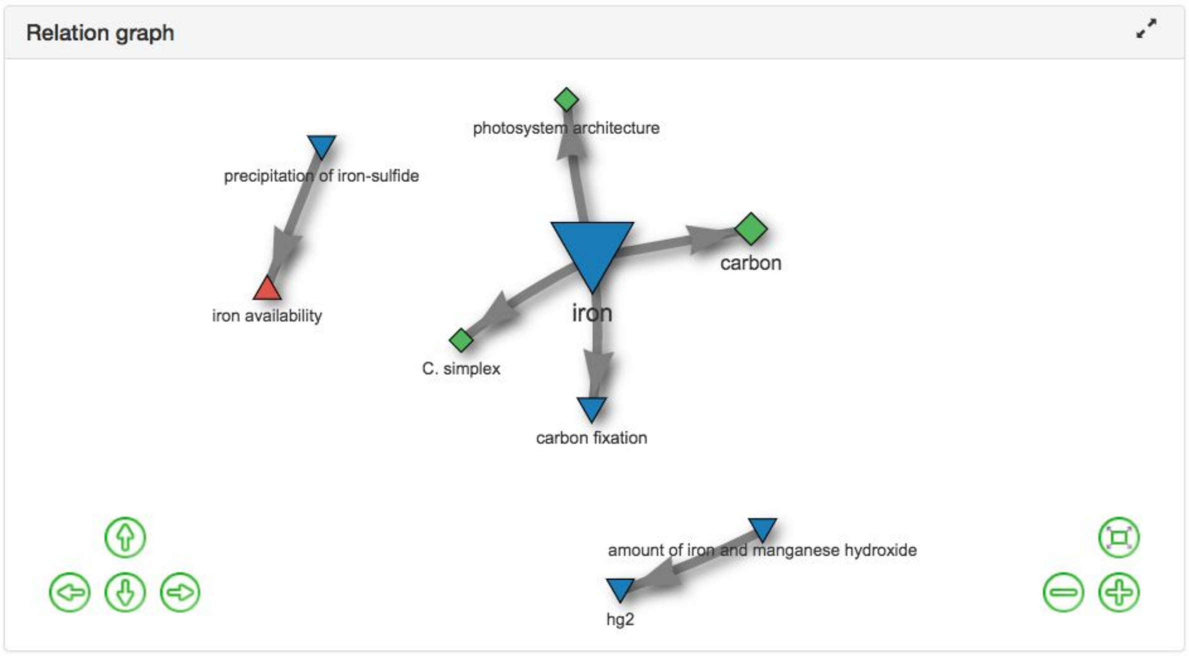


Count	Relation	Predicate1	Variable1	Predicate2	Variable2
1	CAUSES	decrease	precipitation of iron-sulfide	increase	iron availability
1	CAUSES	decrease	iron	decrease	carbon fixation
1	CAUSES	decrease	iron	change	photosystem architecture
1	CAUSES	decrease	amount of iron and manganese hydroxide	decrease	hg2
1	CAUSES	decrease	iron	change	carbon
1	CAUSES	decrease	iron	change	C. simplex

Showing 1 to 6 of 6 entries

4.2.2.3 Relation graph

The second panel, called *Relation graph*, displays the same event types in the form of a graph.



The nodes are event types with

- red triangles for increasing variables,
- blue triangles for decreasing variables and
- green diamonds for changing variables.

Edges indicate relations between events, where the thickness of the edge represents the relative frequency of the relation. Causal relations have an arrow, pointing from cause to effect. The size of a node indicates the relative frequency of the event.

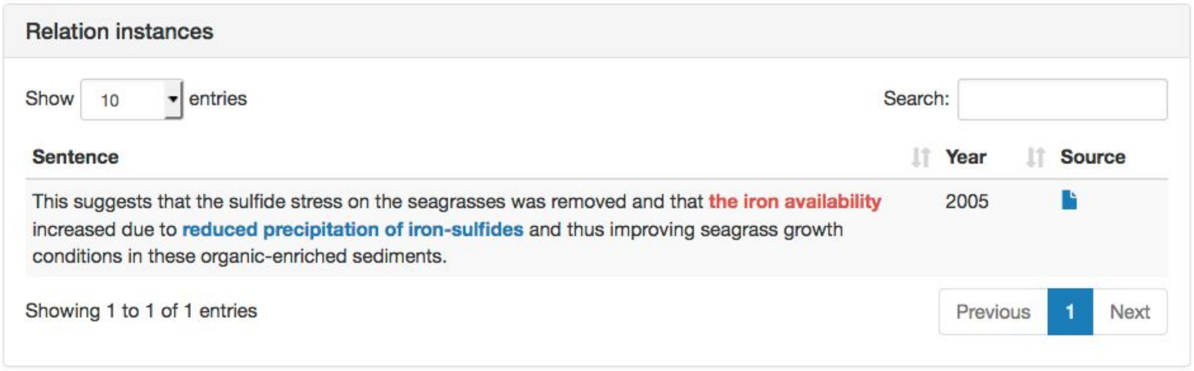
The view on the graph can be changed using the green navigation icons at the bottom of the graph or by using the mouse:

- Click and drag on nodes to move them around.
- Click and drag on the canvas to move the whole graph.
- Use the mouse wheel to zoom in or out.

Clicking on the icon in the upper right corner of the graph panel will show the graph in full screen mode, while clicking the icon will return it in its original size.

4.2.2.4 Relation instances

To see the corresponding relation instances, either click on a row in the relation types table or click on an edge in the relation graph. This will open up a new panel called *Relation instances*, showing sentences, years and citations of articles containing the given relation. The events are marked in color: red for increasing, blue for decreasing and green for changing. Hovering the mouse over the icon will show a citation for the source article. Clicking on this icon will open the article containing the sentence in a new window. Note that in the case of a full text article, one needs to have access rights to view it.


Relation instances

Show 10 entries Search:

Sentence	Year	Source
This suggests that the sulfide stress on the seagrasses was removed and that the iron availability increased due to reduced precipitation of iron-sulfides and thus improving seagrass growth conditions in these organic-enriched sediments.	2005	🗎

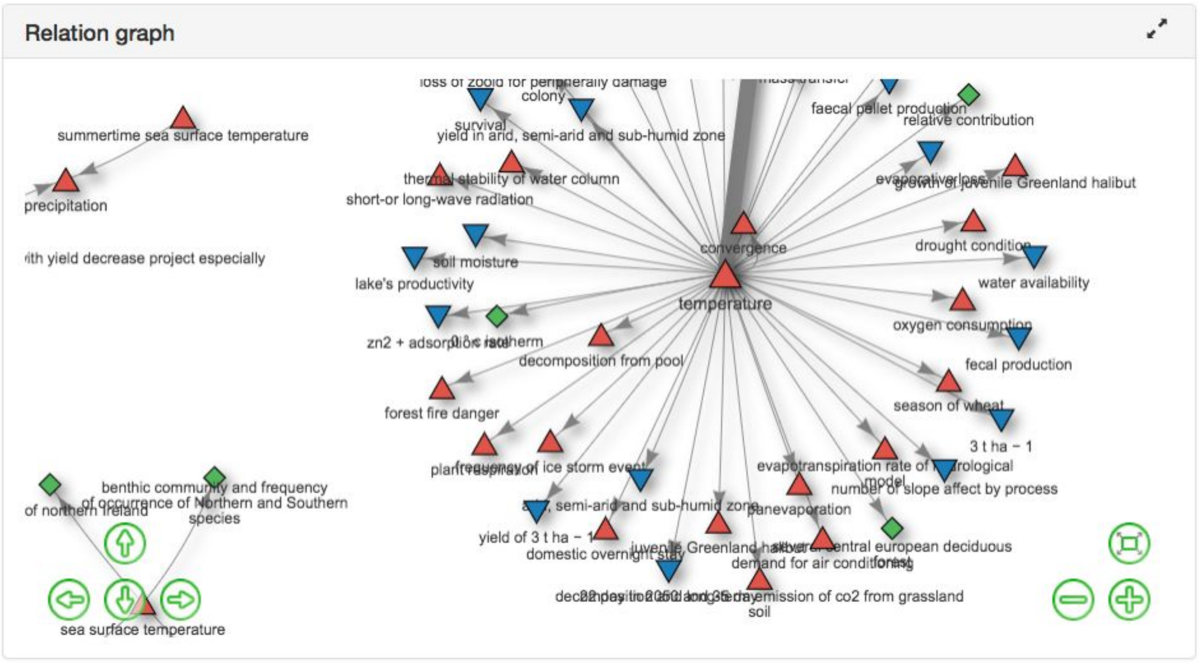
Showing 1 to 1 of 1 entries

Previous 1 Next

Clicking on the *Reset* button starts a new search.

4.2.2.5 Open-ended relation search

Suppose the user is interested in the effects of increased temperature. In that case, one has to define Event 1 as an increase of the variable *temperature*, while leaving Event 2 open, and then search for causal relations. Note that all rules for Event 2 have to be deleted to leave it completely open. This search will show all events caused by increased temperature.



Relation graph
summertime sea surface temperature
precipitation
ith yield decrease project especially
benthic community and frequency
of northern ireland
of occurrence of Northern and Southern species
sea surface temperature
loss of zooid for peripherally damage colony
survival
yield in arid, semi-arid and sub-humid zone
thermal stability of water column
short-or long-wave radiation
soil moisture
lake's productivity
zn2 + adsorption isotherm
decomposition from pool
forest fire danger
frequency of ice storm event
plant respiration
semi-arid and sub-humid zone
yield of 3 t ha − 1
domestic overnight stay
juvenile Greenland halibut
decomposition of 2050 and 35 day
soil
convergence
temperature
faecal pellet production
relative contribution
evaporative loss
growth of juvenile Greenland halibut
drought condition
water availability
oxygen consumption
fecal production
season of wheat
3 t ha − 1
evapotranspiration rate of hydrological model
number of slope affect by process
panevaporation
central european deciduous forest
demand for air conditioning
emission of co2 from grassland

4.3 Implementation

The implementation has two major parts: a front end and a back end.

The front end runs in the web browser (no specific requirements, except support for javascript enabled) at the client side. It communicates with the back end through Ajax.. It is implemented in a combination of HTML, CSS and Javascript. HTML and CSS formatting relies on the Twitter's well-known Bootstrap framework[15]. Several Javascript libraries are employed:

- Jquery[16] for HTML document traversal and manipulation, event handling and Ajax
- DataTables[17] for rendering of interactive tables
- QueryBuilder[18] for UI components to compose queries
- Vis.js Network for rendering interactive event type graphs

On top of these, there is custom Javascript code for responding to interface events and communicating with the back end.

The back end runs on a remote server, currently a virtual machine running Ubuntu Linux at NTNU, handles http requests from the front end. The main tasks of back end are:

- translate incoming request parameters in Json format to database queries
- pass queries to graph database and retrieve result tables
- translate result tables to response parameters in Json format, including formatting of results (e.g. highlighting of events)

The back end is implemented in Python using the Flask[19] web micro-framework and Jinja2[20] template engine, running behind a Apache HTTP server[21] through WSGI[22]. The graph database is a Neo4j server instance running on the same server[23]. The back end communicates with the graph server through Cypher[24] queries over the Bolt protocol.

The latest stable version of own source code is maintained in a version control system (Git) and is freely available from the Github website[25]. All software dependencies have permissive open source licenses and no paid software is required.

4.4 Discussion

The current graphical user interface -- the Marine Variable Linker -- enables non-expert users (marine scientists) to easily search the graph database resulting from information extraction on marine science literature.. Queries can be refined in an interactive way and the results are presented in a visual and browsable way using tables and graphs. The web application runs inside the browser, so users do not need a local installation of software and data. It runs on any modern platform with a browser (Linux, Mac OS, Windows, Android, iOS, etc).

[15] http://getbootstrap.com/
[16] https://jquery.com/
[17] https://datatables.net/
[18] http://querybuilder.js.org/
[19] http://flask.pocoo.org/
[20] http://jinja.pocoo.org/
[21] https://httpd.apache.org/
[22] http://www.wsgi.org/
[23] https://neo4j.com/
[24] https://neo4j.com/developer/cypher-query-language/
[25] https://github.com/OC-NTNU/MVL

The disadvantage of the current user interface is that it allows only one category of searches, namely, that of events of changing variables and the relations between them. However, there are many other search types and/or options that are potentially of interest. For instance, it is possible to search for chains of causal events or positive/negative feedback loops. Literature searches can be restricted to certain year ranges, certain journals, impact factor thresholds, etc. Such alternative searches can be expressed in Cypher queries, but are currently not supported by the user interface, as this would render it extremely complicated from a user perspective and hard to debug and maintain from a programming perspective. A promising direction for future research is therefore to find out exactly which search types are most interesting to marine science researchers, and to build dedicated user interfaces to accommodate their information needs.

5 Dissemination & exploitation

Work on literature-based knowledge discovery in WP1 is disseminated through our website "Text Mining in Marine Science", which includes a brochure targeted a general public, as well as publications and demos:

Website: http://baleen.idi.ntnu.no/

Parts of this deliverable have been presented at conferences and published in proceedings:

Erwin Marsi and Pinar Öztürk, "Extraction and generalisation of variables from scientific publications", Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (Lisbon, Portugal), Association for Computational Linguistics, September 2015, pp. 505–511.
Paper: http://aclweb.org/anthology/D15-1057
Poster: https://drive.google.com/open?id=0B6_H_KFDE9oMc1NBbk5XM2hQbHM

Erwin Marsi and Pinar Øzturk, "Text mining of related events from natural science literature", Proceeding of Workshop on Semantics, Analytics, Visualisation: Enhancing Scholarly Data (SAVE-SD 2016) (Montreal, Canada), 2016.
Paper: http://cs.unibo.it/save-sd/2016/papers/pdf/marsi-savesd2016.pdf
Poster: https://drive.google.com/open?id=0B6_H_KFDE9oMd3lYc2pjTmZOUEk

The open source software libraries can be used by competent programmers and can serve as a basis for future work on text mining in natural sciences.

The interface is available as an online web application, which can be freely used by anyone interested in exploring marine science literature.

Marine Variable Linker webapp: http://baleen.idi.ntnu.no/demos/megamouth-abs/

A slide presentation shows a number of step-by-step examples on how to use the MVL interface.

Slides: https://drive.google.com/open?id=0B6_H_KFDE9oMZTVMSGlHeHZvZEk

6 Appendix

6.1 Sample of megamouth top-level command line interface for processing abstracts

```
./megamouth-abs.py -h
usage: megamouth-abs.py [-c CONFIG_FILE] [-s SECTION] [-h]

{get-abs,core-nlp,lemma-trees,ext-vars,offsets,prep-vars,prune-vars,tocsv,setup-server,toneo,
ppgraph,tag-trees,ext-rels,add-rels,add-cit,add-meta,get-inp,remove-server,start-server,stop-
server,clean,clean-cache,report,run-all}
                      ...

positional arguments:

{get-abs,core-nlp,lemma-trees,ext-vars,offsets,prep-vars,prune-vars,tocsv,setup-server,toneo,
ppgraph,tag-trees,ext-rels,add-rels,add-cit,add-meta,get-inp,remove-server,start-server,stop-
server,clean,clean-cache,report,run-all}
       get-abs              get text of abstracts
       core-nlp             Run Stanford CoreNLP
       lemma-trees          Extract lemmatized parse trees
       ext-vars             Extract variables in change/increase/decrease events
       offsets              Add character offsets to extracted variables
       prep-vars            Preprocess variables
       prune-vars           Prune variables in change/increase/decrease events
       tocsv                Transform extracted variables to csv tables that can
                            be imported by neo4j
       setup-server         Setup and run a new neo4j server
       toneo                Create a new Neo4j database from data in CSV files
       ppgraph              Post-process graph after import.
       tag-trees            Tag variable nodes in tree
       ext-rels             Extract relations between events
       add-rels             Add extracted causal relations to graph
       add-cit              Add citation string to Article nodes
       add-meta             Add article metadata to Article nodes
       get-inp              get input (Solr core & DOI pairs)
       remove-server        remove neo4j server
       start-server         start neo4j server
       stop-server          stop neo4j server
       clean                Clean output
       clean-cache          Remove records with None values from metadata cache
       report               Report on graph database
       tag-trees            Tag variable nodes in tree
       run-all              Run complete pipeline: get-abs --> core-nlp --> lemma-
                            trees --> ext-vars --> offsets --> prep-vars -->
                            prune-vars --> tocsv --> setup-server --> toneo -->
                            ppgraph --> tag-trees --> ext-rels --> add-rels -->
                            add-cit --> add-meta

optional arguments:
  -c CONFIG_FILE, --config CONFIG_FILE
                      configuration file; option can be repeated where later
                      configurations override earlier ones (default:
                      ['megamouth.ini', 'local.ini'])
  -s SECTION, --section SECTION
```

```
                      section in config file (default: 'ABS')
  -h, --help          show this help message and exit
```

6.2 Sample of megamouth command line interface for core-nlp command

```
./megamouth-abs.py core-nlp -h
usage: megamouth-abs.py core-nlp [-h] [--out-dir OUT_DIR] [-a ANNOTATORS]
                              [-c CLASS_PATH] [-v VERSION] [-m MEMORY]
                              [-t THREADS] [--replace-ext]
                              [--output-ext OUTPUT_EXT] [--options OPTIONS]
                              [-s] [--resume] [-u]
                              [input]

Run Stanford CoreNLP

positional arguments:
  input               '/Users/work/Projects/OCEAN-CERTAIN/megamouth/ocean-
                      local/local/out/abs/text'

optional arguments:
  -h, --help          show this help message and exit
  --out-dir OUT_DIR   '/Users/work/Projects/OCEAN-CERTAIN/megamouth/ocean-
                      local/local/out/abs/corenlp'
  -a ANNOTATORS, --annotators ANNOTATORS
                      'tokenize,ssplit,pos,lemma,parse'
  -c CLASS_PATH, --class-path CLASS_PATH
                      '/Users/work/local/src/corenlp'
  -v VERSION, --version VERSION
                      '3.5.1'
  -m MEMORY, --memory MEMORY
                      '3g'
  -t THREADS, --threads THREADS
                      1
  --replace-ext       True
  --output-ext OUTPUT_EXT
                      '.xml'
  --options OPTIONS   '-ssplit.newlineIsSentenceBreak always'
  -s, --stamp         True
  --resume            toggle default for resuming process (default: False)
  -u, --use-sr-parser   True
```

6.3 Sample of megamouth configuration file for processing abstracts

```
###############################################################################
[DEFAULT]
###############################################################################

log_level = INFO

base_dir = .

in_dir = %(base_dir)s/local/inp
out_dir = %(base_dir)s/local/out
cache_dir = %(base_dir)s/local/cache


# Override in your local config file:

core_nlp.class_path = %($HOME)s/local/src/corenlp
core_nlp.version = x.y.z

baleen_java_home = %(home_dir)s/local/src/baleen-java

solr_url = http://localhost:8983/solr


###############################################################################
[ABS]
###############################################################################

abs_out_dir = %(out_dir)s/abs


#-----------------------------------------------------------------------------
# get_inp
#-----------------------------------------------------------------------------
get_inp.xml_files = %(in_dir)s/*.xml
get_inp.in_dir = %(in_dir)s

#-----------------------------------------------------------------------------
# get_abs
#-----------------------------------------------------------------------------
get_abs.doi_files = %(in_dir)s/*.tsv
get_abs.solr_url = %(solr_url)s
get_abs.text_dir = %(abs_out_dir)s/text
get_abs.resume = False

#-----------------------------------------------------------------------------
# core_nlp
#-----------------------------------------------------------------------------
core_nlp.input = %(get_abs.text_dir)s
core_nlp.out_dir = %(abs_out_dir)s/corenlp
# newlines are always sentence breaks
core_nlp.options = -ssplit.newlineIsSentenceBreak always

#-----------------------------------------------------------------------------
# lemma_trees
```

```
#-------------------------------------------------------------------------------
lemma_trees.scnlp_dir = %(core_nlp.out_dir)s
lemma_trees.out_dir = %(abs_out_dir)s/lemtrees

#-------------------------------------------------------------------------------
# ext_vars
#-------------------------------------------------------------------------------
ext_vars.extract_vars_exec = %(baleen_java_home)s/bin/extract-vars
ext_vars.trees_dir = %(lemma_trees.out_dir)s
ext_vars.vars_dir = %(abs_out_dir)s/vars
#ext_vars.resume = True

#-------------------------------------------------------------------------------
# offsets
#-------------------------------------------------------------------------------
offsets.vars_dir = %(ext_vars.vars_dir)s
offsets.scnlp_dir = %(core_nlp.out_dir)s
#offsets.resume = True

#-------------------------------------------------------------------------------
# prep_vars
#-------------------------------------------------------------------------------
prep_vars.trans_exec = %(baleen_java_home)s/bin/transform
prep_vars.trans_file = %(baleen_java_home)s/src/main/resources/tsurgeon/preproc/preproc.tfm
prep_vars.in_vars_dir = %(offsets.vars_dir)s
prep_vars.out_vars_dir = %(abs_out_dir)s/prep
#prep_vars.resume = True

#-------------------------------------------------------------------------------
# prune_vars
#-------------------------------------------------------------------------------
prune_vars.prune_vars_exec = %(baleen_java_home)s/bin/prune-vars
prune_vars.in_vars_dir = %(prep_vars.out_vars_dir)s
prune_vars.out_vars_dir = %(abs_out_dir)s/prune
#offsets.resume = True

#-------------------------------------------------------------------------------
# tag_trees
#-------------------------------------------------------------------------------
tag_trees.vars_dir = %(prep_vars.out_vars_dir)s
tag_trees.trees_dir = %(lemma_trees.out_dir)s
tag_trees.tagged_dir = %(abs_out_dir)s/tagtrees

#-------------------------------------------------------------------------------
# ext_rels
#-------------------------------------------------------------------------------
ext_rels.class_path = %(core_nlp.class_path)s
ext_rels.tagged_dir = %(tag_trees.tagged_dir)s
ext_rels.pattern_path = %(patterns_dir)s
ext_rels.rels_dir = %(abs_out_dir)s/rels

#-------------------------------------------------------------------------------
# add_rels
#-------------------------------------------------------------------------------
add_rels.rels_dir = %(ext_rels.rels_dir)s
add_rels.warehouse_home = %(setup_server.warehouse_home)s
add_rels.server_name = %(setup_server.server_name)s
```

```
#-------------------------------------------------------------------------------
# tocsv
#-------------------------------------------------------------------------------
tocsv.vars_dir = %(prune_vars.out_vars_dir)s
tocsv.scnlp_dir = %(core_nlp.out_dir)s
tocsv.text_dir = %(core_nlp.input)s
tocsv.nodes_dir = %(abs_out_dir)s/csv/nodes
tocsv.relations_dir = %(abs_out_dir)s/csv/relations

#-------------------------------------------------------------------------------
# toneo
#-------------------------------------------------------------------------------
toneo.warehouse_home = %(setup_server.warehouse_home)s
toneo.server_name = %(setup_server.server_name)s
toneo.nodes_dir = %(tocsv.nodes_dir)s
toneo.relations_dir = %(tocsv.relations_dir)s
toneo.options =

#-------------------------------------------------------------------------------
# ppgraph
#-------------------------------------------------------------------------------
ppgraph.warehouse_home = %(setup_server.warehouse_home)s
ppgraph.server_name = %(setup_server.server_name)s
#ppgraph.password = %(setup_server.password)s

#-------------------------------------------------------------------------------
# add_cit
#-------------------------------------------------------------------------------
add_cit.warehouse_home = %(setup_server.warehouse_home)s
add_cit.server_name = %(setup_server.server_name)s
add_cit.cache_dir = %(cache_dir)s/citations
add_cit.resume = true
#add_cit.password = %(setup_server.password)s

#-------------------------------------------------------------------------------
# add_meta
#-------------------------------------------------------------------------------
add_meta.warehouse_home = %(setup_server.warehouse_home)s
add_meta.server_name = %(setup_server.server_name)s
add_meta.cache_dir = %(cache_dir)s/metadata
add_meta.resume = true
#add_cit.password = %(setup_server.password)s

#-------------------------------------------------------------------------------
# setup_server
#-------------------------------------------------------------------------------
setup_server.warehouse_home = %(abs_out_dir)s/neokit
setup_server.server_name = abs_graph

#-------------------------------------------------------------------------------
# start_server
#-------------------------------------------------------------------------------
start_server.warehouse_home = %(setup_server.warehouse_home)s
start_server.server_name = %(setup_server.server_name)s

#-------------------------------------------------------------------------------
# start_server
#-------------------------------------------------------------------------------
```

```
remove_server.warehouse_home = %(setup_server.warehouse_home)s
remove_server.server_name = %(setup_server.server_name)s

#-------------------------------------------------------------------------------
# stop_server
#-------------------------------------------------------------------------------
stop_server.warehouse_home = %(setup_server.warehouse_home)s
stop_server.server_name = %(setup_server.server_name)s

#-------------------------------------------------------------------------------
# report
#-------------------------------------------------------------------------------
report.warehouse_home = %(setup_server.warehouse_home)s
report.server_name = %(setup_server.server_name)s
#report.password = %(setup_server.password)s

#-------------------------------------------------------------------------------
# clean
#-------------------------------------------------------------------------------
clean.dir = %(out_dir)s

#-------------------------------------------------------------------------------
# clean_cache
#-------------------------------------------------------------------------------
clean_cache.cache_dir = %(add_meta.cache_dir)s
```